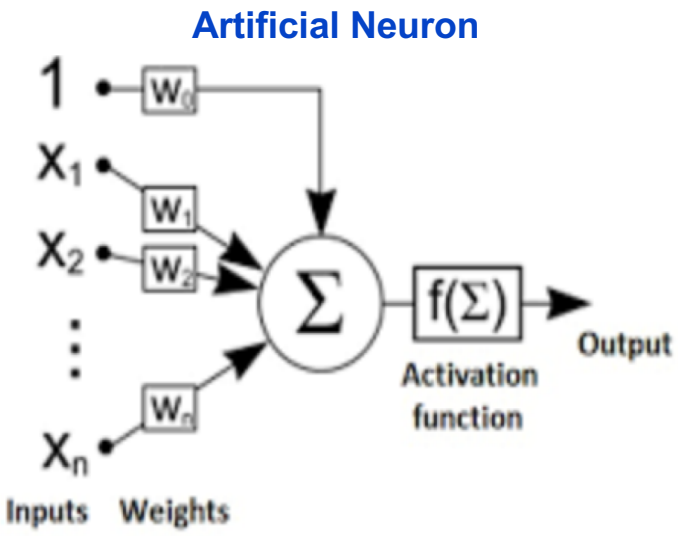
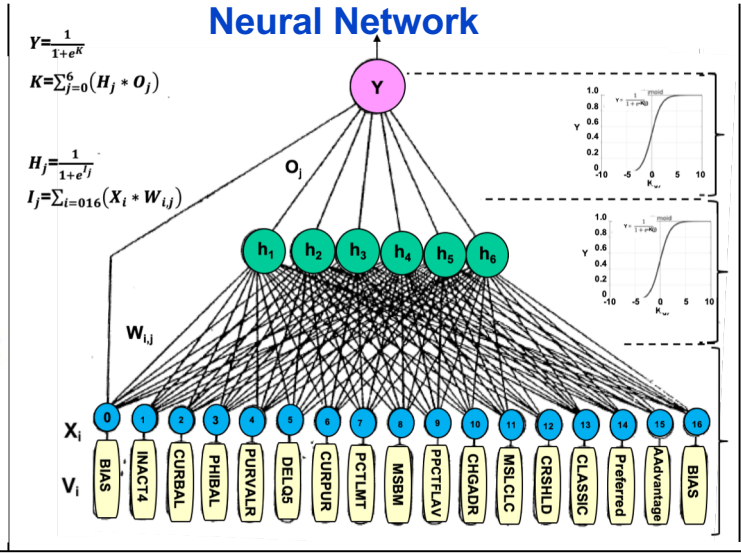
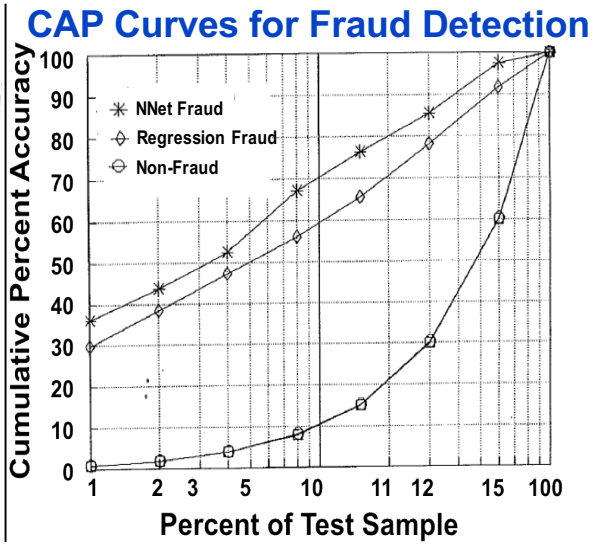
Benzschawel Scientific, LLC

β – ψ







Financial Applications of Machine Learning

IAQF/Thalesians Seminar

April 8, 2019

Terry Benzschawel
Benzschawel Scientific, LLC
tbenzschawel@benz-sci.com
+1 646-599-1854

Financial Applications of Machine Learning

1. Overview, Introduction and Some Early History
   a) Artificial Neurons
   b) The Perceptron and Perceptron Networks
   c) Neuron Activation Functions
   d) Backpropagation
2. Early Applications
   a) Why Do Neural Networks Work?
   b) Credit Card Fraud
   c) Credit Card Attrition
   d) Interpreting Neural Network Decisions
   e) US Treasury Trading
3. Predicting Market Moves from Customer Trading Patterns
4. Deep Learning Models
   a) Corporate Bond Relative Value and OAS Changes
   b) Predicting Market Moves from Trading Data
   c) Using Sentiment Data to Predict Market Moves
5. How AI/ML is Transforming Bond Markets

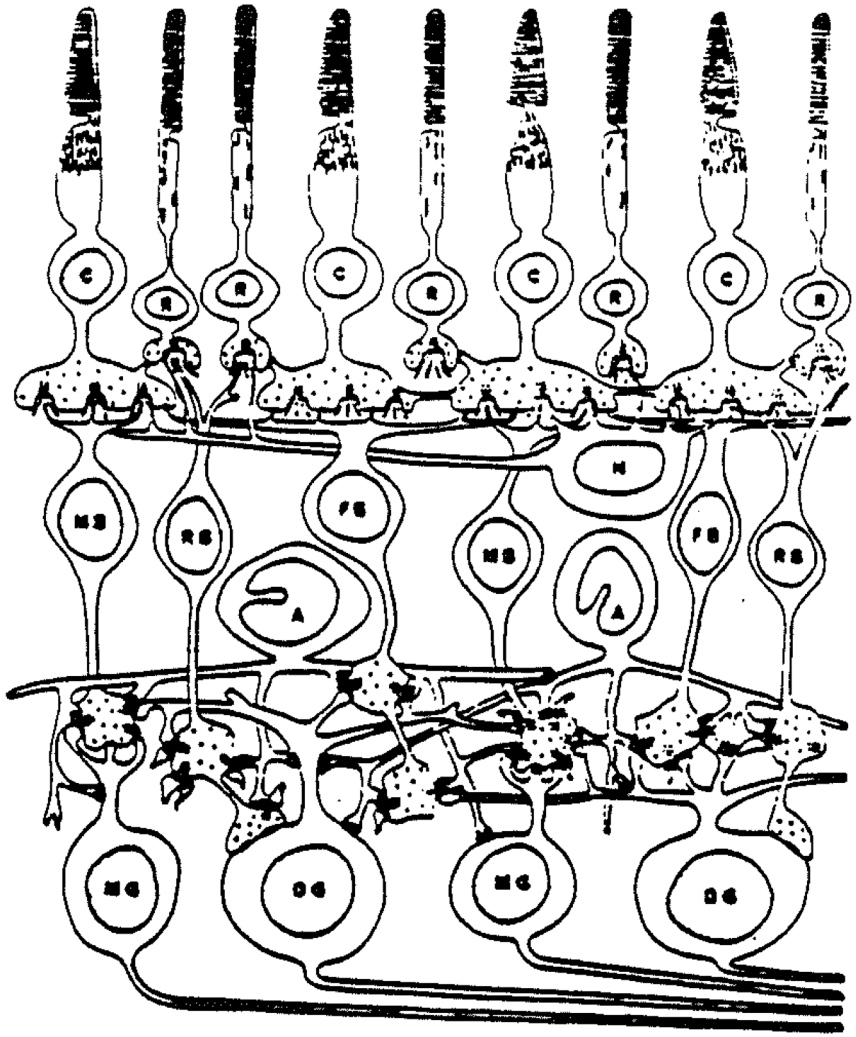
Some Financial Applications of Machine Learning

Overview, Introduction and Some Early History

A Neural Network

Schematic diagram or a section through the peripheral retina. The layers or the retina are indicated on the right.

A Neural Network



Schematic diagram of a section through the peripheral retina

Photoreceptors

Inner Retinal Layer

Horizontal Cells

Bipolar Cells

Amacrine Cells

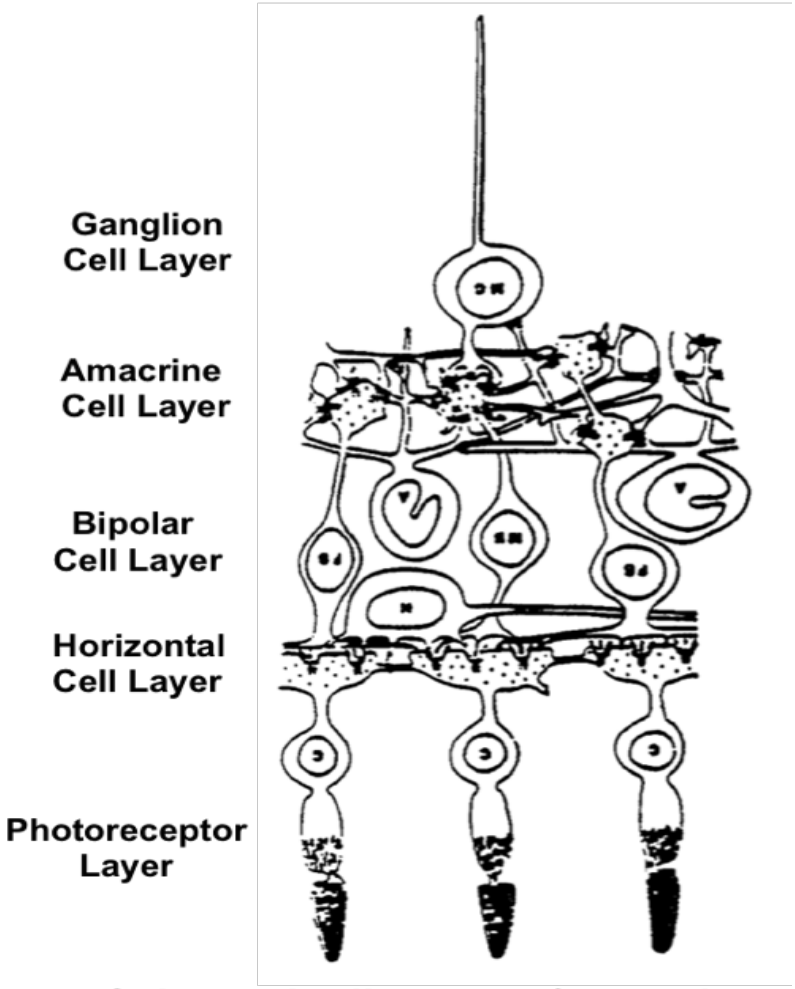
Outer Retinal Layer

Ganglion Cells

Neural Networks and Artificial Neural Networks

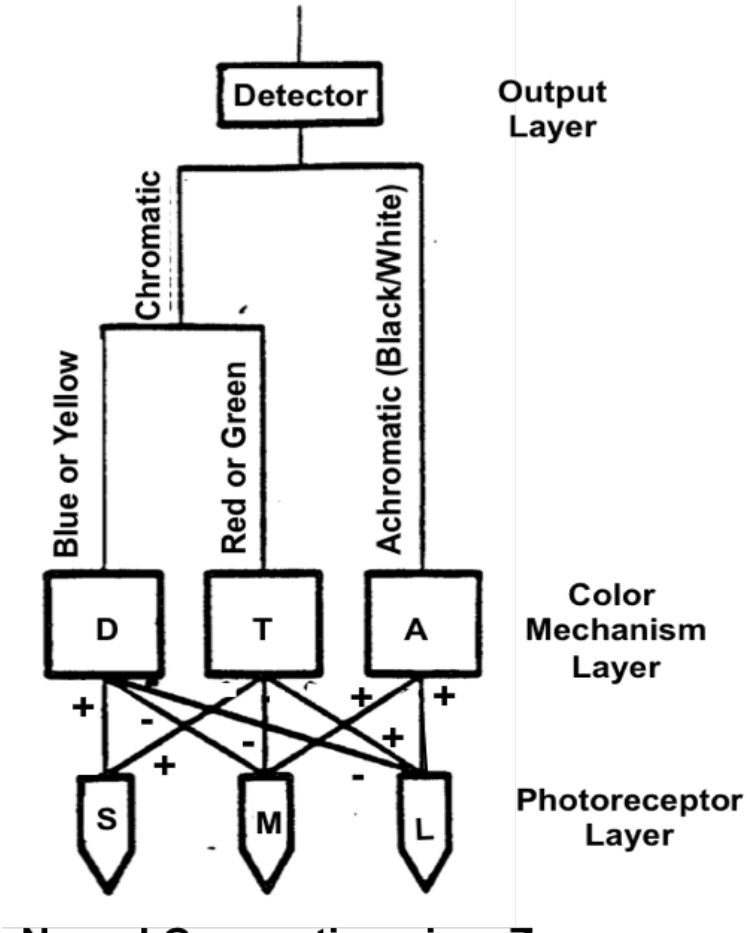
An artificial neural network (ANN) is a mathematical computing system inspired by studies of the brain.

Neural Network



Schematic diagram of a section through the retina of the eye

Artificial Neural Network

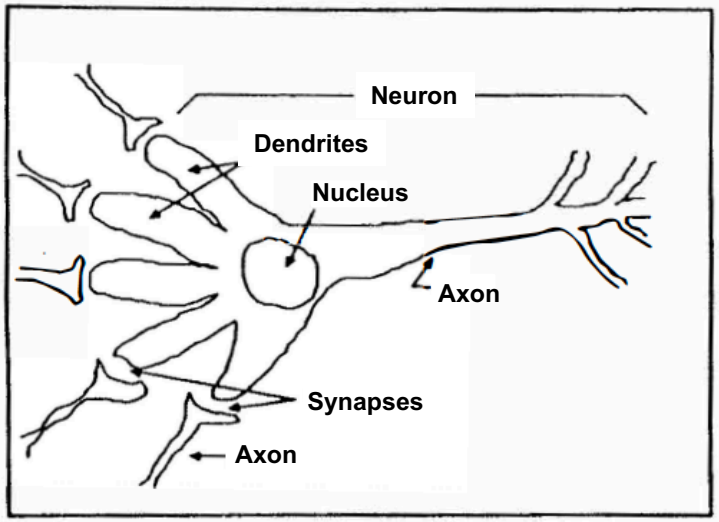


Neural Connections in a Zone Theory of Color Vision (Guth, Massof & Benzschawel , 1980)

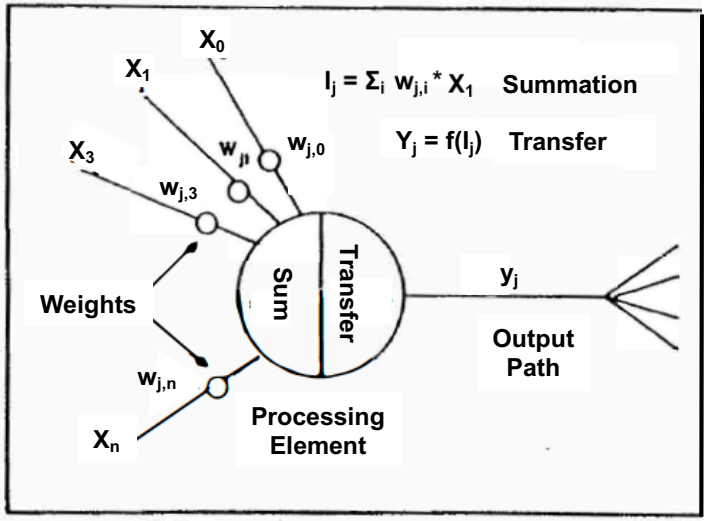
Artificial Neuron

An artificial neural network is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain.

Biological Neuron



Artificial Neuron



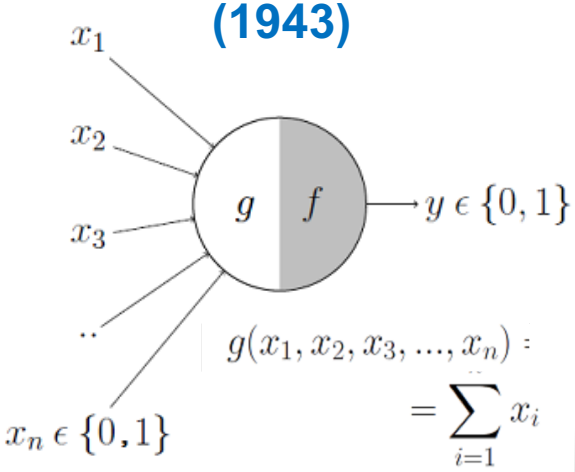
- **The analogy between biological neurons and artificial neurons is straightforward**
  - **The inputs to the artificial neurons correspond to axons of incoming neurons**
  - **The weights on the inputs to the artificial neuron correspond to the strength of the connection between the axons of the incoming neuron to the dendrites of target neuron**
  - **The summation and transfer functions of the artificial neural network correspond to the cell body of the neuron. It has two parts:**
    - **The first part takes the input (analogous to the dendrite) and performs a summation**
    - **Based on the aggregated value, the second part, the transfer makes a decision**
  - **The output of the artificial neuron is analogous to the axon of the neuron**

Benzschawel Scientific, LLC

Evolution of the Artificial Neuron

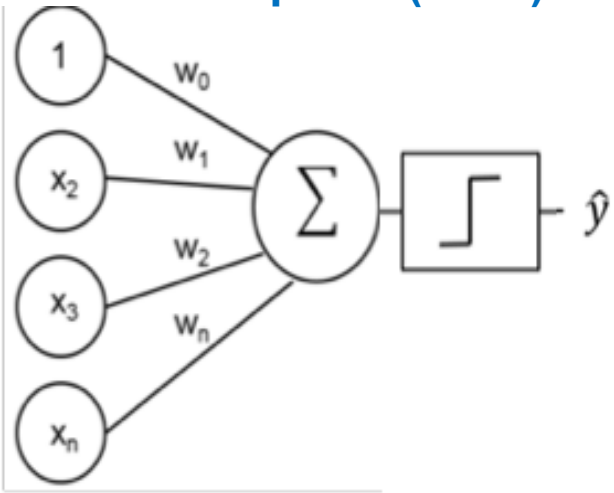
The artificial neuron has undergone several iterations to result in the neurons that are used in neural networks today.

McCulloch-Pitts Neuron (1943)



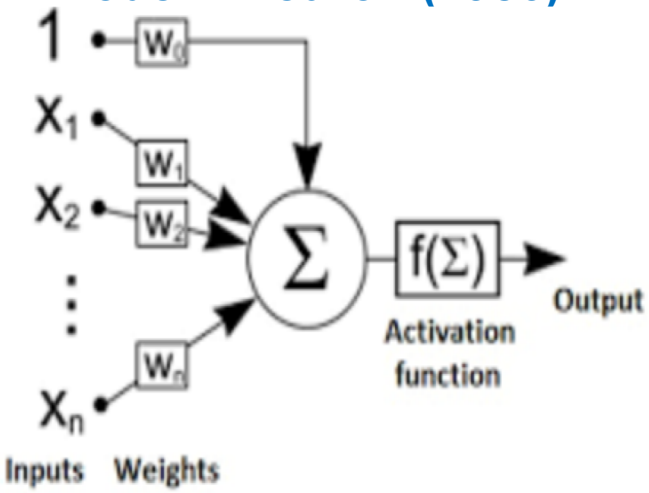
- The McCulloch-Pitts neuron takes as input Boolean values of x_i, either 0 or 1, and has a Boolean output y (0:1)
- There is no learning involved in McCulloch-Pitts neuron model
- The McCulloch-Pitts neuron can not input real values (only 0 and 1)

Perceptron (1957)



- Rosenblatt's perceptron can take continuous values as inputs, but has Boolean output
- The weights can be adjusted over time (the perceptron can "learn")
- Single layer perceptrons are only capable of learning linearly separable patterns (can not solve XOR problem)

Modern Neuron (1986)



- This neuron can have a non-linear activation function and a bias
- The differentiability of the activation function enables application of the gradient descent error back propagation
- Still, the single layer perceptron can not solve the XOR problem

Perceptrons and the XOR Problem

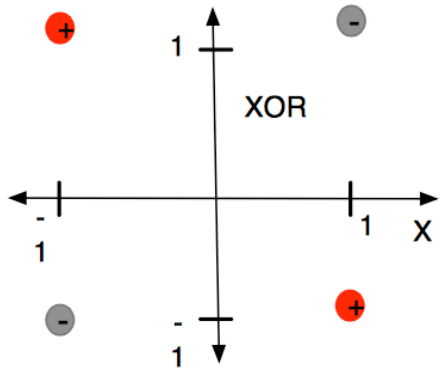
Minsky and Papert demonstrated that a one-layer perceptron could not solve the XOR (exclusive OR) problem. This set the field back for over a decade.

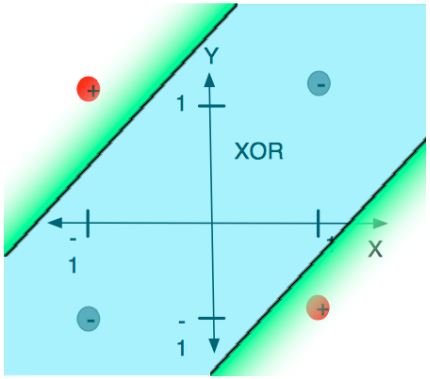
- **The perceptron is an algorithm for supervised learning of binary classifiers**
  - **A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class**
  - **A single layer perceptron at the output node is a linear combination of its inputs**
  - **This means it can classify input-output space only if you can draw one linear line which will clearly separate them**
  - **Since the XOR function is not linearly separable, it is impossible for a single hyperplane to separate it**

	X	Y	Class
Desired i/o pair 1	0	0	0
Desired i/o pair 2	0	1	1
Desired i/o pair 3	1	0	1
Desired i/o pair 4	1	1	0

$$
\begin{aligned}
0 \times w_1 + 0 \times w_2 + w_0 \leq 0 &\iff w_0 \leq 0, \\
0 \times w_1 + 1 \times w_2 + w_0 > 0 &\iff w_0 > -w_2, \\
1 \times w_1 + 0 \times w_2 + w_0 > 0 &\iff w_0 > -w_1, \\
1 \times w_1 + 1 \times w_2 + w_0 \leq 0 &\iff w_0 \leq -w_1 - w_2.
\end{aligned}
$$

- **The solution to the problem is the multi-layer perceptron**

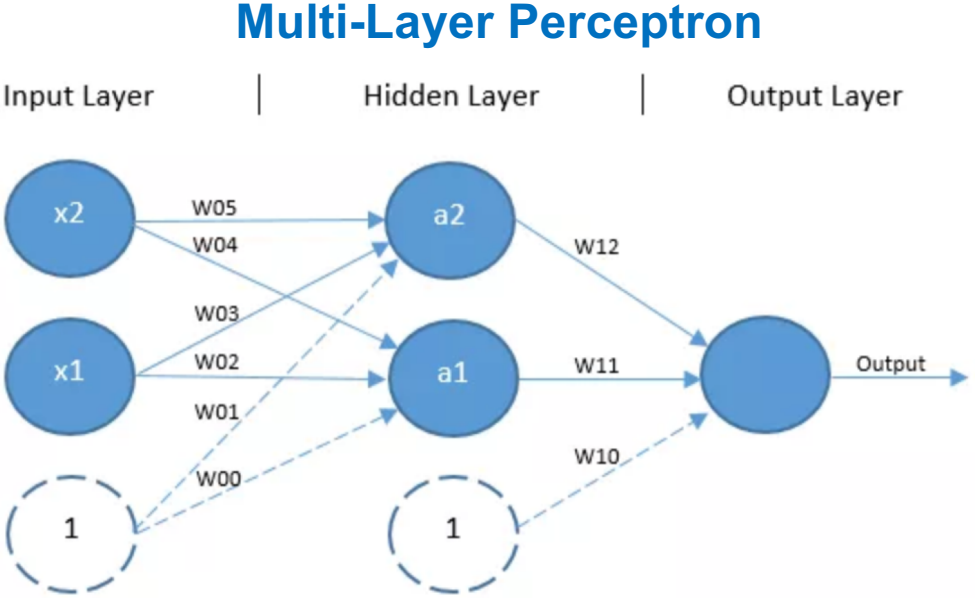




The Multi-Layer Perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural networks consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer.

- The solution to the XOR problem is to add an additional layer of units without any direct access to the outside world, known as a hidden layer
  - This architecture, while more complex than the classic perceptron network, is capable of achieving non-linear separation

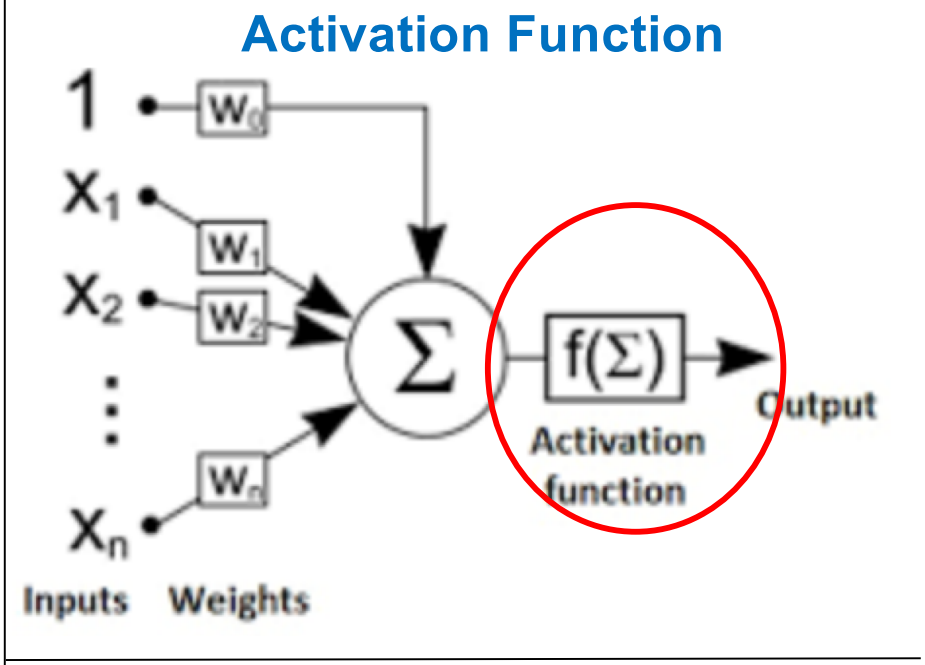


- The most noticeable difference from Rosenblatt's model to the multi-layer perceptron is the differentiability of the activation function
  - Except for the input nodes, each node is a neuron that uses a nonlinear activation function
  - Recall the the Rosenblatt perceptron had a binary output activation function
- David Rumelhart and Geoffrey Hinton (1986) changed the history of neural networks research by introducing the error backpropagation algorithm

Neuron Activation Function

In artificial neural networks, the activation function of a node defines the output of that node, or "neuron," given an input or set of inputs.

- **The basic operation of an artificial neuron involves summing its weighted input signal and applying an activation function**
  - **In biologically inspired neural networks, the activation function is usually an abstraction representing the rate of action potential firing in the cell**
  - **In its simplest form, this function is binary—that is, either the neuron is firing or not. This was true for the perceptron.**



- **Typically the same activation function is used for all neurons in a particular layer of the network, although this is not required**
- **In most cases, a non-linear activation function is used**
  - **Historically, the most common function used in multilayer perceptrons is a sigmoidal activation function, but this has been replace by the ReLU function**
  - **Two forms of the sigmoid function are commonly used: $\phi(v_i) = tanh(v_i)$ whose range is normalized from -1 to 1, and $\phi(v_i) = (1+ exp(-v_i))^{-1}$ is vertically translated to normalize from 0 to 1**

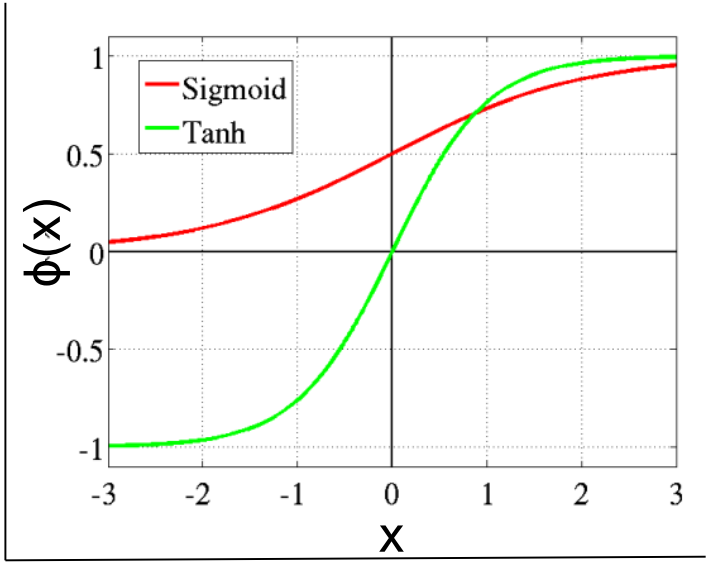
The Sigmoid and Hyperbolic Tangent Functions

The most common activation functions in artificial neural networks are the sigmoid and hyperbolic tangent activation functions.

Sigmoid Function

- **The sigmoid function curve looks like an S-shape (see figure). Its activation function is:**

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



- **The main reason the sigmoid function is used because it exists between 0 to 1**
  - **It is especially useful for models where we have to predict the probability as an output**
  - **Since the probability of anything exists only between the range of 0 and 1, the sigmoid is the right choice**

Hyperbolic Tangent Function

- **The hyperbolic tangent (tanh) function has become popular as it is less likely to get "stuck" during training**
- **Its activation function is:**

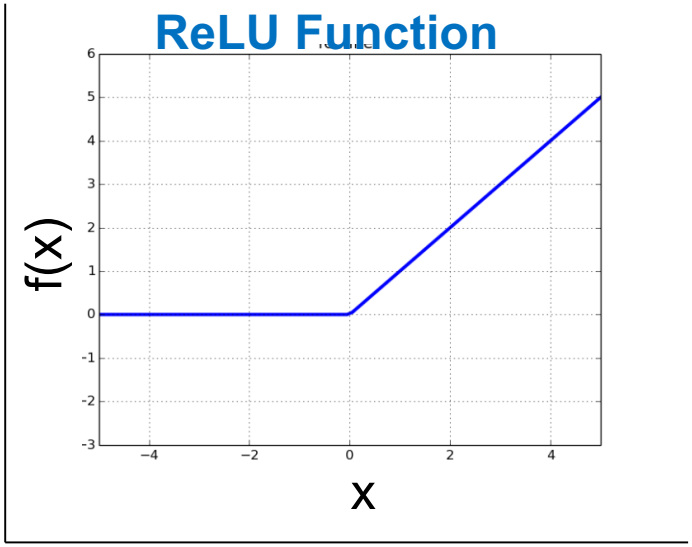
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

- **The tanh function produced output in the range between -1 and 1**
  - **The tanh has stronger gradients (derivatives) around zero than the sigmoid which is preferable for optimization**

The Rectified Linear Unit (ReLU) Function

In recent years, the rectified linear unit (ReLU) function has become popular, particularly for deep learning networks.

- **The rectified linear unit (ReLU) is an activation function defined as the positive part of its argument:** $f(x) = x^+ = \max(0, x),$
- **This activation function was introduced to a dynamical network by Hahnloser et al. in 2000**
- **Demonstrated to enable better training of deeper networks, compared to the widely-used sigmoid and tanh functions**
- **As of 2018, ReLU is the most popular activation function for deep neural networks**



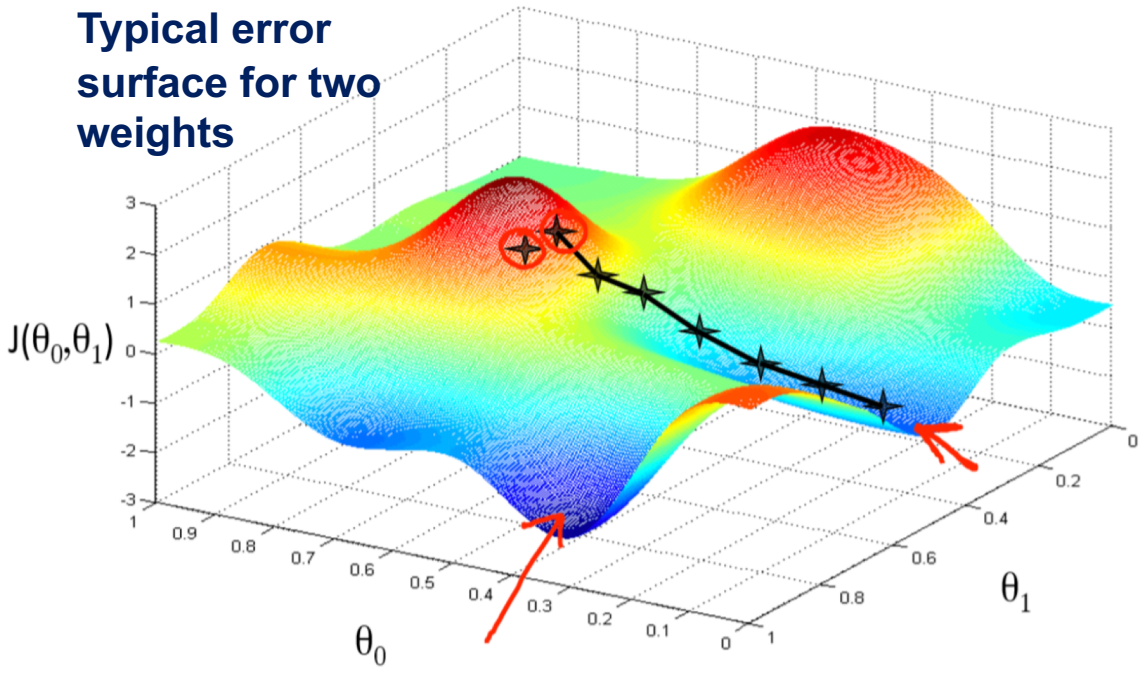
Error Backpropagation - Overview

Backpropagation is used to describe gradient descent optimization algorithm which adjusts weights of neurons by calculating the gradient of the loss function.

- **Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network's layers**
- **Backpropagation requires the derivative of the loss function with respect to the network output to be known**

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Gradient Descent Method



- **The derivative of this with respect to the network weights is**

$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}\,(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^T\cdot\mathbf{x}^{(i)} - y^{(i)})\,x_j^{(i)}$$

- **Then increase or decrease the value of each weight to produce the maximal decrease in network error for that input**

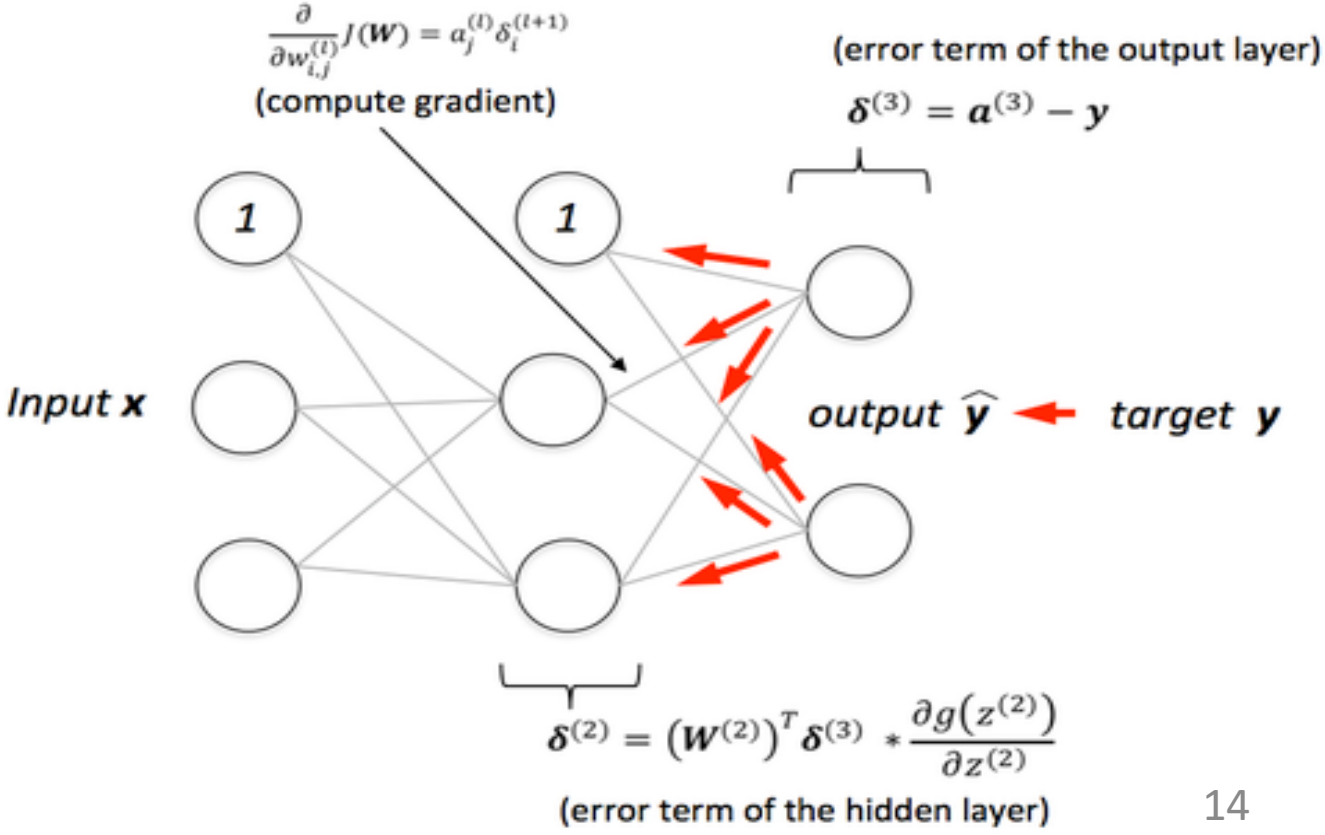
Error Backpropagation

Backpropagation in multi-layered feedforward networks, is made possible by using the chain rule to iteratively compute gradients for each layer

The Learning Rate

- **The problem for most models is how to set the learning rate**
- **The update expression for each weight is:** $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
  - ***j* ranges from 0 to the number of weights**
  - **θ_j is the j^{th} weight in a weight vector, and**
  - **α is the learning rate**
- **We're computing $dJ/d\theta_j$ (the gradient of weight θ_j) and then taking a step of size alpha in that direction**

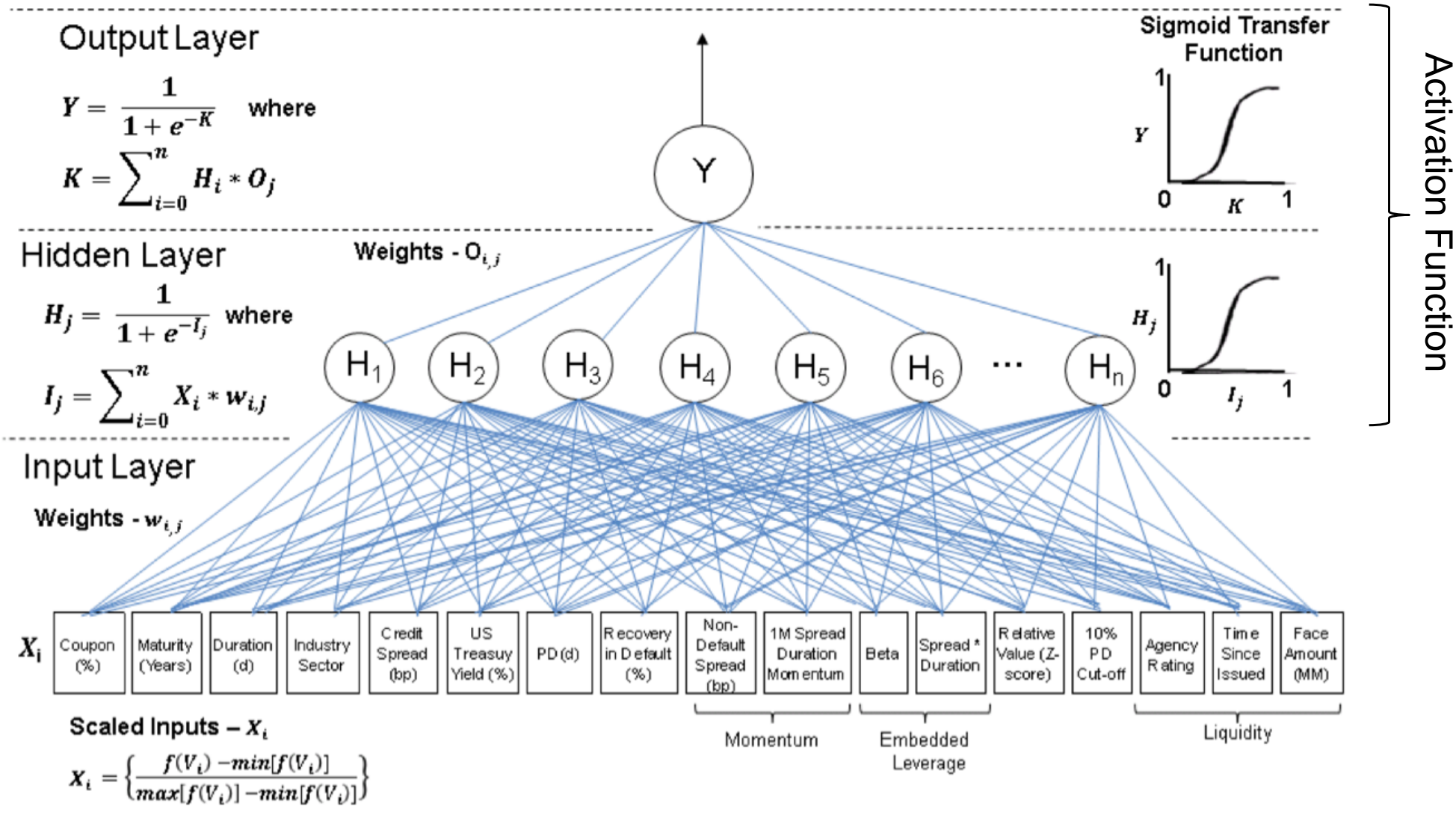
Error Backpropagation



Putting it All Together: The Neural Network

A popular form of neural network is the feed-forward network (below) which is typically trained using backpropagation.

Feed Forward Neural Network



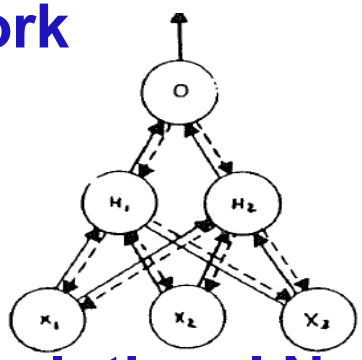
Machine Learning and Neural Networks in Finance

Early Applications

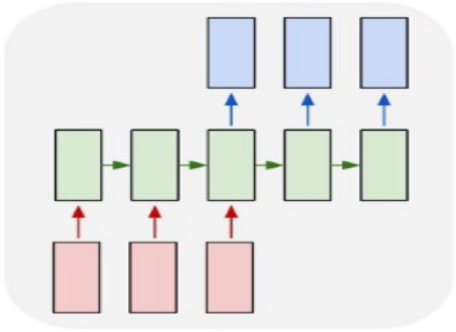
Types of Neural Networks

There are many types of neural networks. See examples of popular ones below.

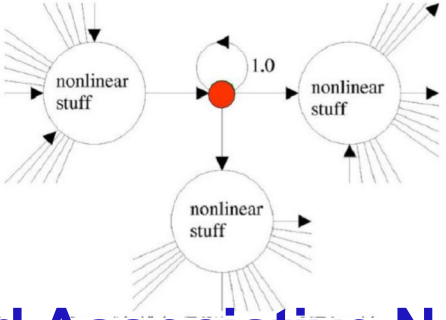
Back-Propagation Network



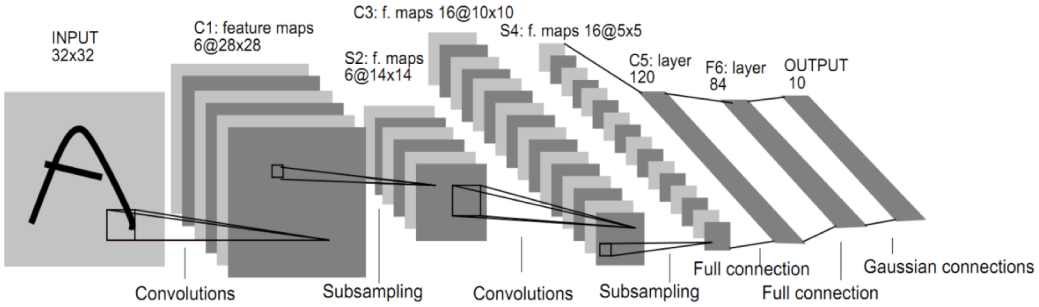
Recurrent Network



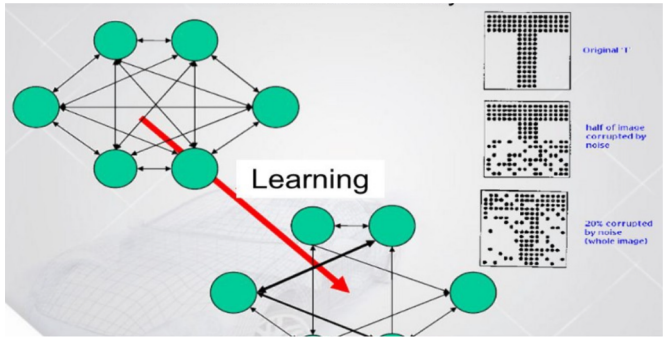
Long- Short-Term Memory



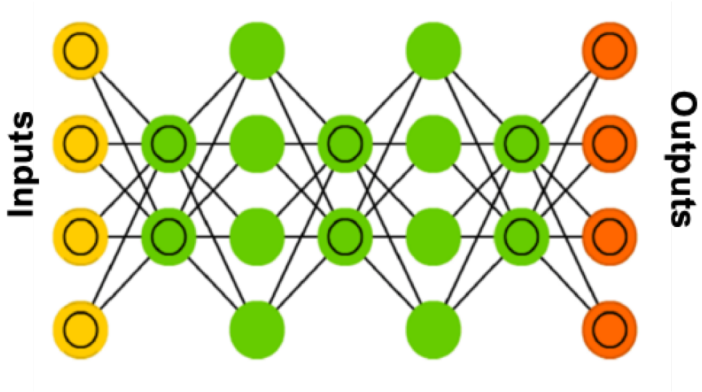
Convolutional Neural Network



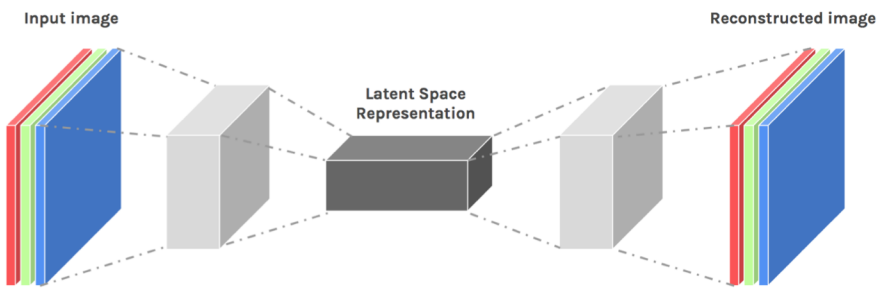
Hopfield Associative Network



Deep Learning Network



Deep Auto Encoders

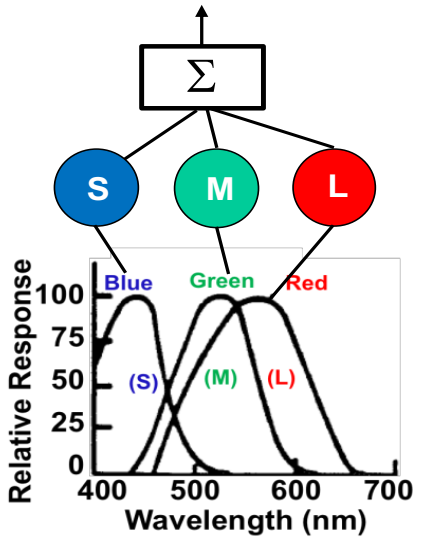


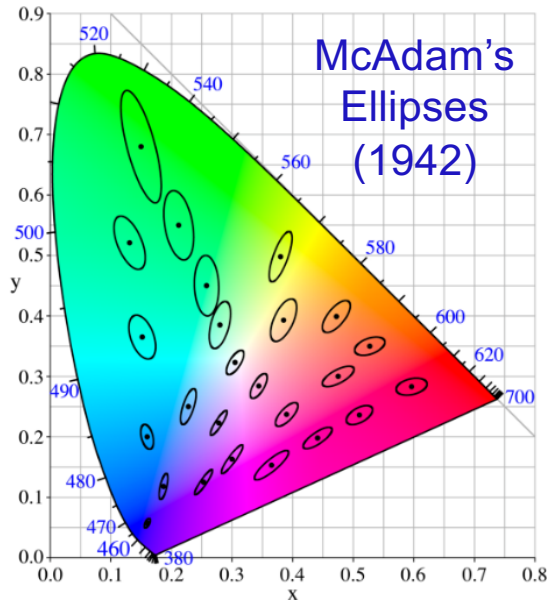
What Can Neural Networks Do? The Linear Model

A fundamental problem in visual psychophysics is to construct a geometry in which equal distances represent equal changes in sensation.

- **An early theory of color vision was based on the idea of three types of photoreceptor whose outputs combined linearly**
  - **If that were the case, it should be possible to construct a color diagram in which just discriminable color differences would plot as circles of equal size**
- **MacAdam (1942) measured the loci of just-noticeable differences in color for various starting colors**
  - **However, as shown in the diagram below, that was not possible as those just noticeable differences were not only elliptical, but differed in size as well**
- **An important problem in color science was to create a geometry in which those ellipses plot as circles of equal size**
  - **This only became possible using a neural network as described below**

Trichromatic Theory of Human Color Vision





What Can Neural Networks Do?

An initial attempt to construct a more uniform color space was to put nonlinear intensity-response functions on the photoreceptors.

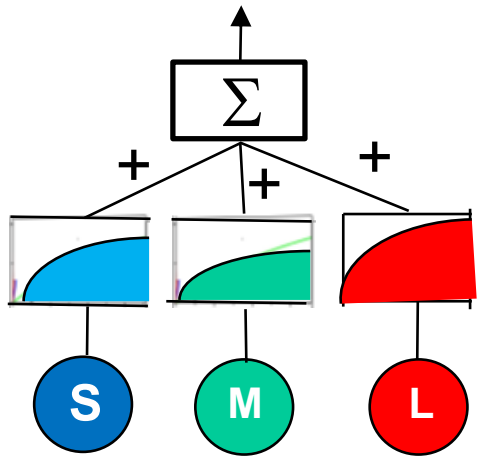
- In 1976, an attempt was made to construct a more uniform color diagram by putting saturating intensity-response functions on the photoreceptors
  - In fact, physiologists suggested that photoreceptors had saturating responses of stimulus intensity *I* of the form:

$$R(I) = R_{max}\frac{I^n}{I^n + k}$$

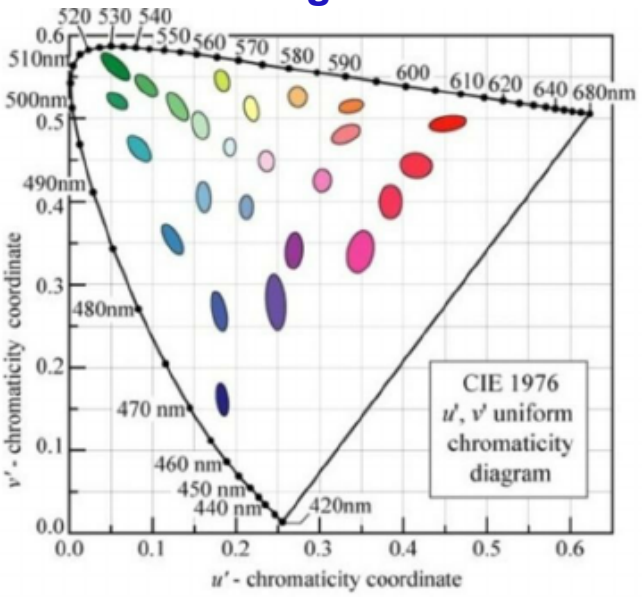
where *R* is the response, *k* is a constant (usually ½ max) and *n* is proportional to the slope of the curve

- Although the ellipses are more uniform that in the earlier diagram, they are far from circular and far from uniform in diameter
  - This is in spite of the fact that the diagram was called the "Uniform Chromaticity Diagram"

Non-Linear Model of Color Vision



1976 CIE Uniform Chromaticity Diagram



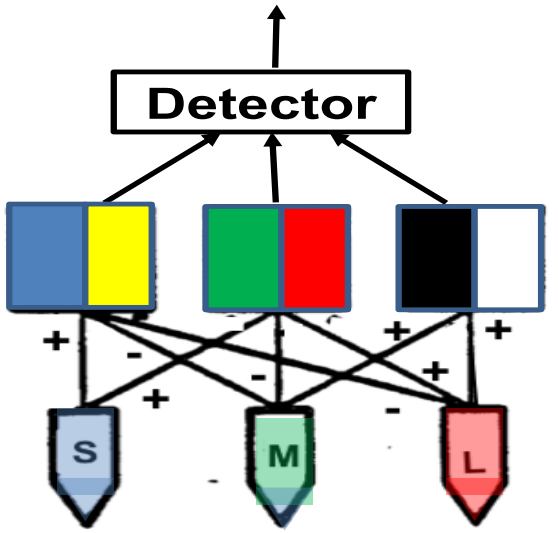
What Can Neural Networks Do?

A neural network imposed on the nonlinear photoreceptor responses fit by backpropagation resulted in a uniform color space.

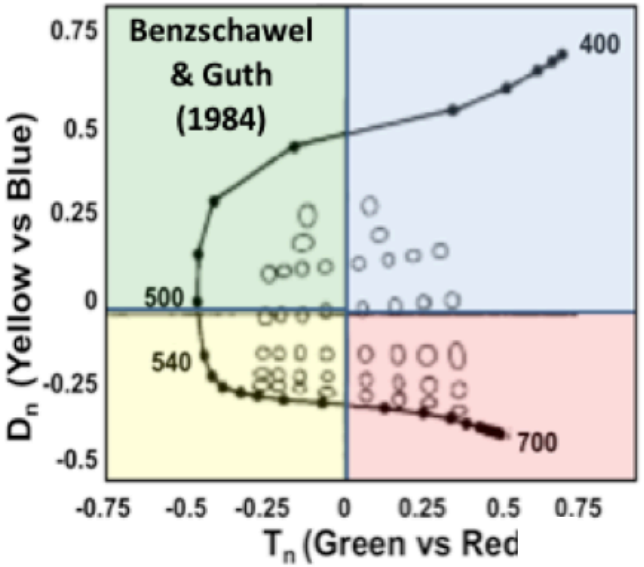
- **Many lines of evidence suggested that photoreceptor outputs do not just sum at the post-receptor level, but also cancel**
  - **In fact, the zone theory of color vision was constructed based on evidence from other studies of color vision**

- **We decided to see if our version of a back-propagation method applied to the neural network zone theory could solve the uniform color diagram.**
  - **There were no backpropagation algorithms available at the time, so we devised our own**
  - **We adjusted the weights in a direction that would make the ellipses more circular and uniform in size (akin to gradient descent)**
  - **Those resulting loci of just-noticeable differences are close to circles of similar size**

Zone Theory of Color Vision



ATDN Uniform Color Space



Neural Network for Credit Card Fraud Detection

The problem of fraudulent use of lost/stolen credit cards is a long-standing problem for credit card issuers.

The Credit Card Fraud Problem

- **Credit card fraud occurs when someone uses another's credit card to make unauthorized transactions**
  - **It could be that the credit card had been lost or stolen and the holder begins charging on the account**
- **Typically, in cases of credit card fraud, the perpetrator attempts to make as many charges as possible in a short period of time**
  - **Thus, one must act quickly in order to stem large losses**
- **In general, the credit card holder is not liable for fraudulent use of the card, so the risk of fraud falls on the issuing bank**
  - **Thus, banks have set up "fraud early warning" units to call customers to verify if they have their cards and, if not, will close the account**
  - **More recently, banks have become even more proactive, closing the card accounts even before calling the customer**

Neural Network for Credit Card Fraud Detection

The job was to use a neural network to improve upon a model that sent questionably fraudulent transaction to the early warning queue.

- **Because of my background with neural network, Citibank asked me to build a neural network to discriminate charges that result from fraudulent activity from legitimate charges**
  - **The problem was that a traditional statistician had tried to implement a neural network and it didn't work**

- **Fortunately, when I arrived, Citibank already had a dataset of transactions resulting from fraudulent activities and non-fraud transactions**
  - **The dataset was prepared by my predecessor who failed to find improvement of a neural network model over a logistic regression**
  - **The input variable set (see next slide) was chosen to include items and charge features that were generally associated with fraudulent activity**

- **<u>MODELING OBJECTIVE</u>: Build a neural network to discriminate fraudulent transactional activity from legitimate charges using information from transactional records and account histories**
  - **In particular, the objective was to build a model that would outperform the existing regression model**

Neural Network for Credit Card Fraud Detection

Given a small sample (by today's standards), we found an optimal net to consist of our given 25 input variables, 7 nodes in the single hidden layer and an output.

- A list of 25 variables thought to be useful in classifying fraud/non-fraud charges were selected along with a bias
  - The variables are coded for security purposes, but some of these include:
    - Number of charges in a given day
    - Amount of open-to-buy on the credit card
    - Number of purchases in "high risk" fraud categories
    - Has there been other recent activity on the card, and so forth
    - The type of credit card: Classic, Preferred, AAdvantage

The Sample

- The characteristics of the sample for training and test (holdout) appear at the right
  - Both training and test samples had about 4,000 fraud and non-fraud cases (i.e., a 50/50 split both ways)
  - More recent practice is about 66/33 split between training and test, but still a 50/50 (fraud/non-fraud) category split

Sample Statistics

	Training Sample	
Brand	N(Fraud)	N(Valid)
Classic	2,663	2,524
Preferred	1,054	1,055
AAdvantage	369	369
Total	4,086	3,948

	Holdout Sample	
Brand	N(Fraud)	N(Valid)
Classic	2,664	2,503
Preferred	1,054	1,055
AAdvantage	182	185
Total	4,086	3,948

Variables

0	BIAS
1	AMTOTB1
2	AMTOTB5
3	AMT1LN
4	UTIL
5	OTB
6	INACT2
7	INACT3
8	INACT9
9	CH5AD1
10	DELQ1
11	HG2DY1
12	LOGAM1
13	AMTCL1
14	HG2DY1
15	MOB1
16	HGHAM1
17	NCLMT
18	GGAMS1
19	HSCDY1
20	LOGMN1
21	HGISC1
22	HILMT
23	Classic
24	Preferred
25	Advantage

Building Neural Networks

Although neural networks can take many forms, there are certain general activities that are commonly required to construct most networks.

Procedure for Constructing Back-Propagation Neural Networks

1. **Assemble Data Set (Most Difficult Part)**
2. **Define Variables and Scale Them for Input to Network**
3. **Extract Training, Cross-Validation, and Test Samples**
4. **Choose Initial Network Parameters: Type, Nodes, Levels, Activation Functions**
5. **Train Network with Intermittent Testing on Cross-Validation Sample until No Further Improvement**
6. **Test Network using Test Sample (Note Accuracy of Prediction)**
7. **Depending on Test Results, Change Network (by Adding a Node) and Repeat Steps**
8. **When Finished, Run Software to Determine Importance (i.e., interpret) of Network Input Variables**
9. **Begin Programming Real-Time Implementation**

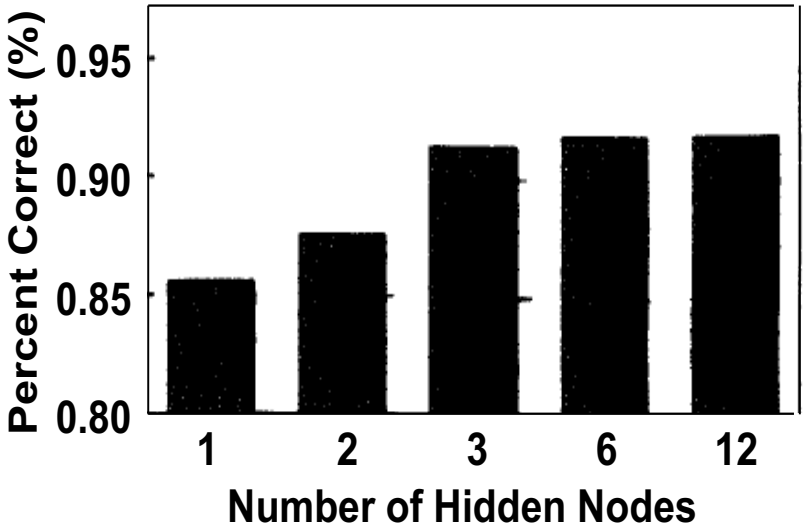
Determining the Network Structure

There is no reliable algorithm for determining the number of hidden nodes in a network, so the process is one of trial-and-error.

Some Rules of Thumb:

- Choose a number of hidden neurons between 1 and the number of input variables
- The number of hidden nodes should be somewhere between the size of the input and output layer, potentially the mean
- The number of hidden nodes shouldn't exceed twice the number of input nodes, as you are probably grossly overfitting

Number of Hidden Nodes versus Probability Correct Classification



My Procedure:

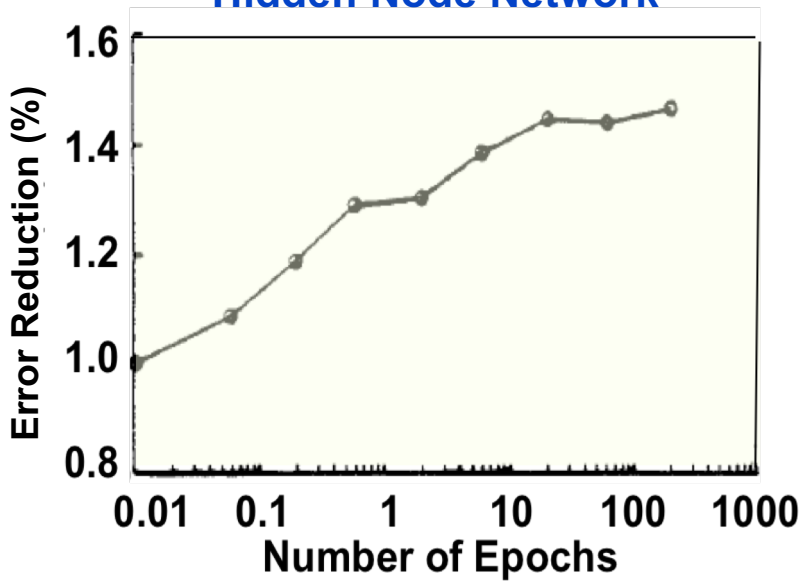
- Start with one neuron in the hidden layer
  - Train the model to asymptote (described further in the next slide)
  - Model accuracy is measured a maximum probability of correct classification on a holdout sample (equal fraudulent and non-fraud)
- Now add another node and repeat procedure until performance asymptotes
  - When performance no longer improves, back off one neuron
- We found that seven nodes was optimal for the fraud model

Training the Model to Asymptote

In addition to setting the number of nodes, each attempt at a model for a given number of nodes will need a learning rate and number of training epochs.

- For each of the models with different numbers of hidden nodes, it was necessary to specify the learning rate and train for the number of epochs
  - As for number of hidden layers and nodes, there is no analytical method for specifying the learning rate in backpropagation
    - A traditional default value for the learning rate is 0.1 or 0.01, but this is only a starting point
    - One should try a range of values and settle on the best one
    - Configuring the learning rate is challenging and time consuming, but also critical

Classification Error Reduction versus Number of Epochs for 7 Hidden Node Network



- Determining the number of epochs for training is an empirical process
  - The figure shows how the errors are reduced as number of epochs increase for the seven hidden-node network
  - Performance on the test sample stabilized after about 30-50 epochs
- Note that while it is often preferable to do these tests on the holdout sample, there is the risk of overfitting

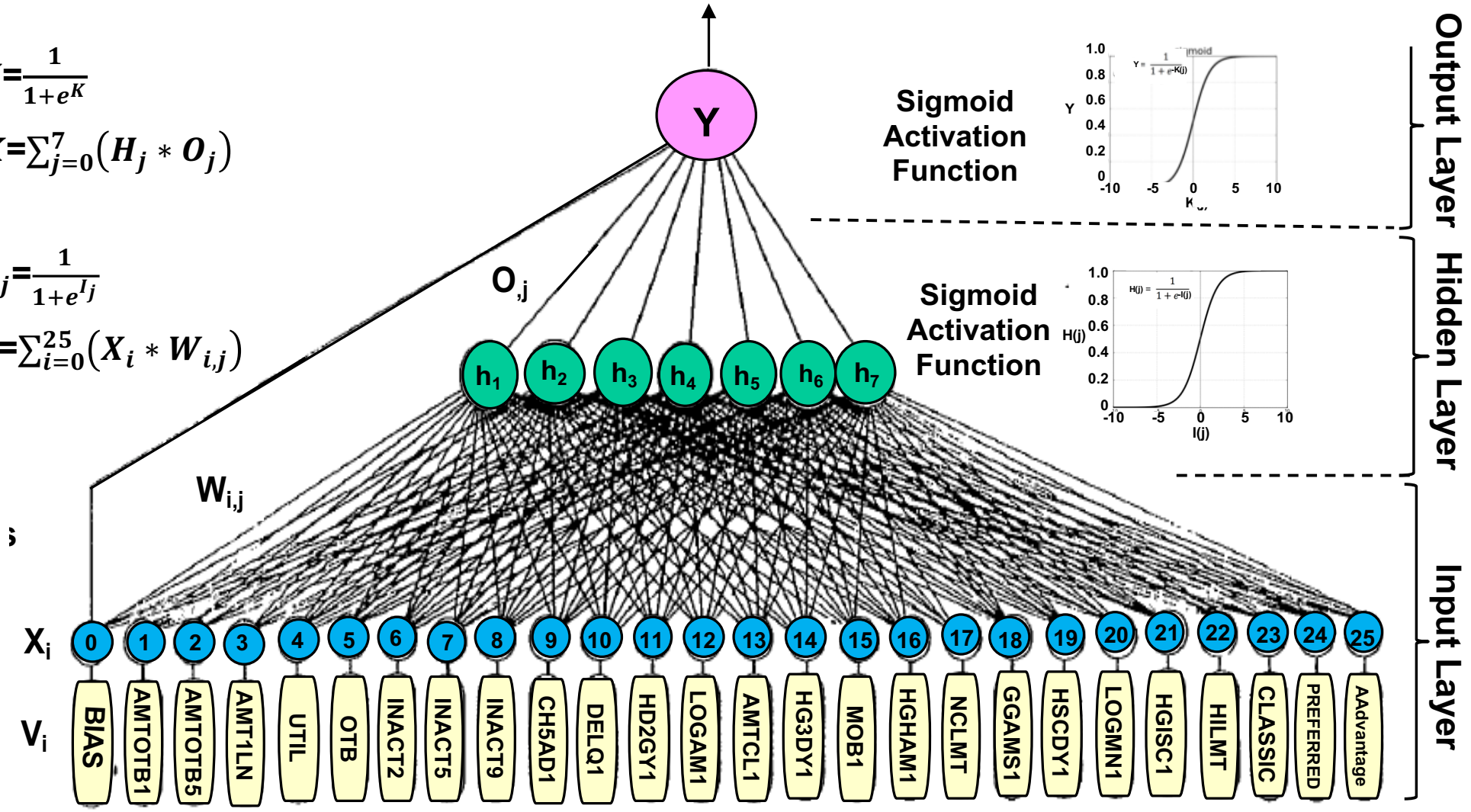
NNet Architecture for Credit Card Fraud Detection

$$Y=\frac{1}{1+e^{K}}$$

$$K=\sum_{j=0}^{7}\left(H_j * O_j\right)$$

$$H_j=\frac{1}{1+e^{I_j}}$$

$$I_j=\sum_{i=0}^{25}\left(X_i * W_{i,j}\right)$$



Input Variables: V_i **– Raw Inputs**

$$X_i=\left\{\frac{f(V_i)-min[f(V_i)]}{max[f(V_i)]-min[f(V_i)]}\right\}$$

Evaluating Classification Models: CAP Curves

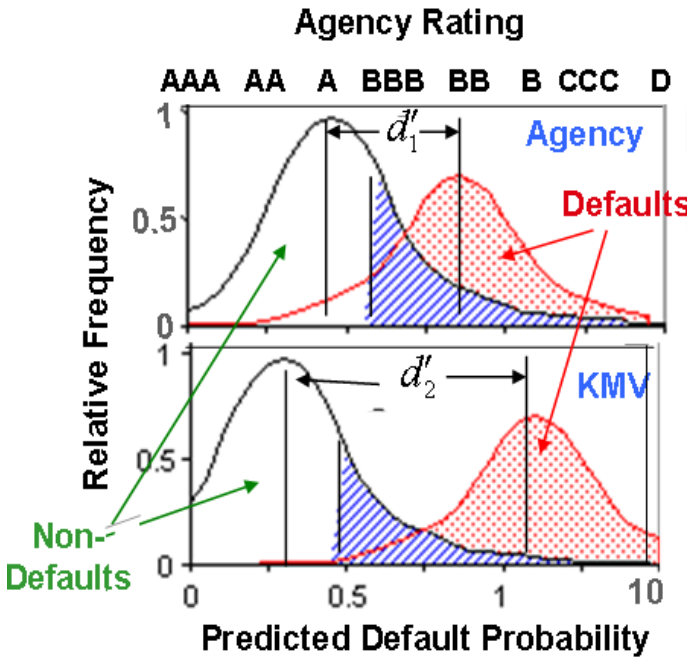
The receiver operating characteristic (ROC) and Cumulative Accuracy Profiles (CAP) are used to evaluate models' abilities to classify events into different categories.

Order Observations by MODEL Score

	Percentile	Rating	D or N	PD(%)	D or N
Highest	0	CCC	D	0.40	D
	1	CCC	D	0.36	D
	2	CCC	N	0.35	D
	3	CCC	D	0.29	D
	4	CCC	N	0.27	N
	5	CCC	N	0.25	D
	6	B	D	0.20	D
	7	B	N	0.19	N
	8	B	N	0.16	D
Risk	87	AA+	N	0.100	N
	88	AA+	N	0.090	N
	89	AA+	N	0.008	N
	90	AA+	N	0.070	N
	91	AA+	D	0.007	N
	92	AA+	N	0.006	N
	93	AAA	N	0.004	D
	94	AAA	N	0.004	N
	95	AAA	N	0.003	N
	96	AAA	N	0.003	N
	97	AAA	N	0.003	N
	98	AAA	N	0.002	N
	99	AAA	D	0.001	N
Lowest	100	AAA	N	0.001	N

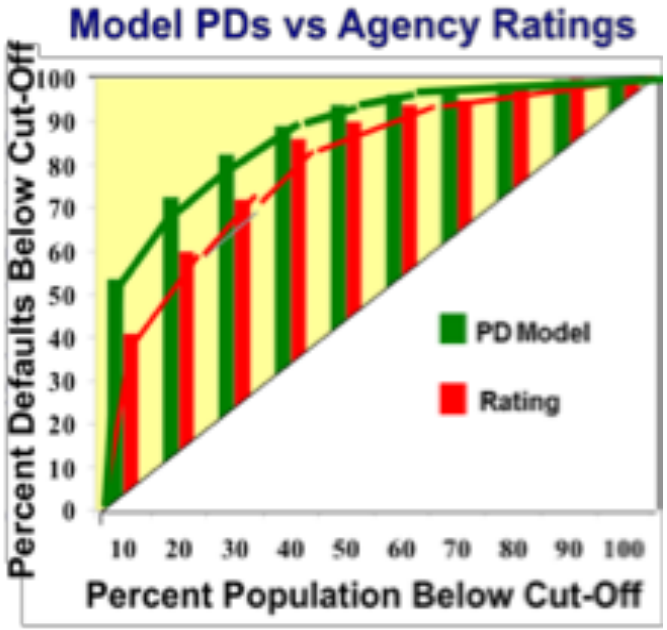
Distributions of Defaults and Non-Defaults

The analysis plots "hits" (defaults called "defaults") versus either the population percentile cut-offs for CAP curves or "false alarms" (calling non-defaulters "defaults") for ROCs



To Plot a CAP Curve:

1. Rank all firms by scores from both models (EDF and credit rating)
2. For each decile in each ranked population (by EDF or credit rating) calculate the percentage of the total number of defaulted firms in or above that decile.
3. Plot that value for each model



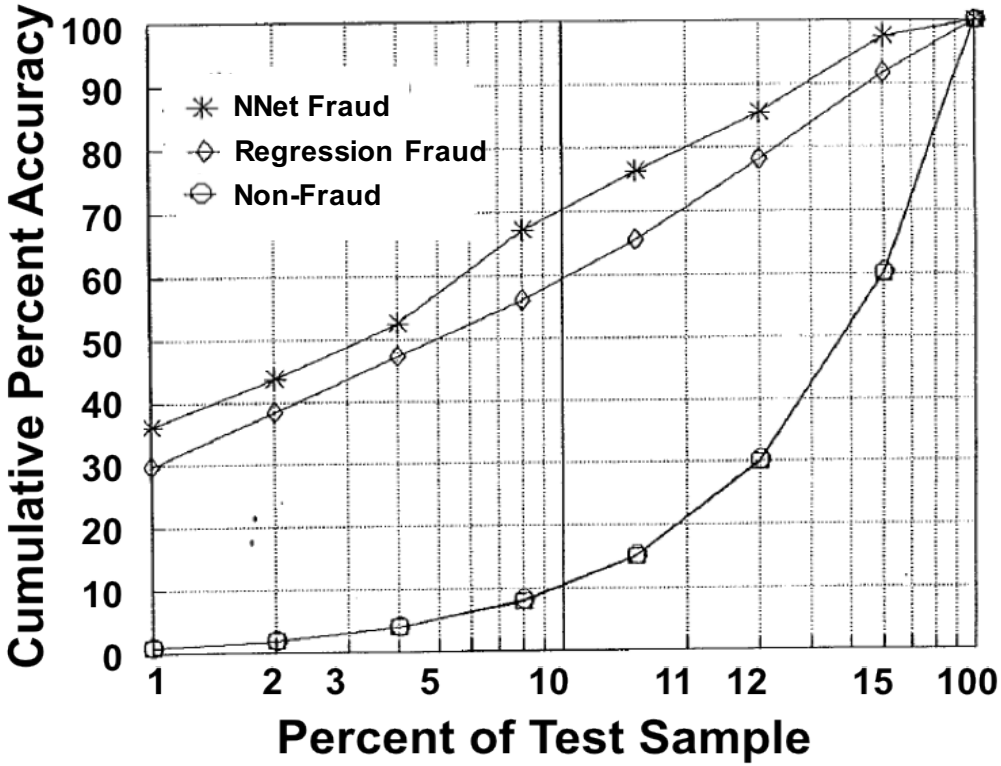
Evaluating Fraud Network Performance

One method for evaluating classification models are cumulative accuracy profiles which plot correct classifications as a function of ranked population scores.

Computing Cumulative Accuracy Profiles

- **To generate a cumulative accuracy profile for a given model:**
  1. **Generate a ranked list of the scores from the model (i.e., likelihood of fraud) for all transactions and note whether or not each transaction was fraudulent or not**
  2. **Then, starting with the highest ranked scores, go down the population and calculate for each percentile the cumulate percentage of fraudulent transactions at or above that percentile**
  3. **Plot the results in a graph like that on the right**

CAP Curves for Fraud Detection Models



- **The graph shows "hit" rates (percent of fraudulent transactions falling above a given ranked population percentile) for the neural network and regression models**
  - **The neural network model consistently outperforms the regression one**

Neural Network for Credit Card Attrition

We built a neural network to predict which credit card customers would cancel their accounts when assessed their annual fee. This was used for marketing.

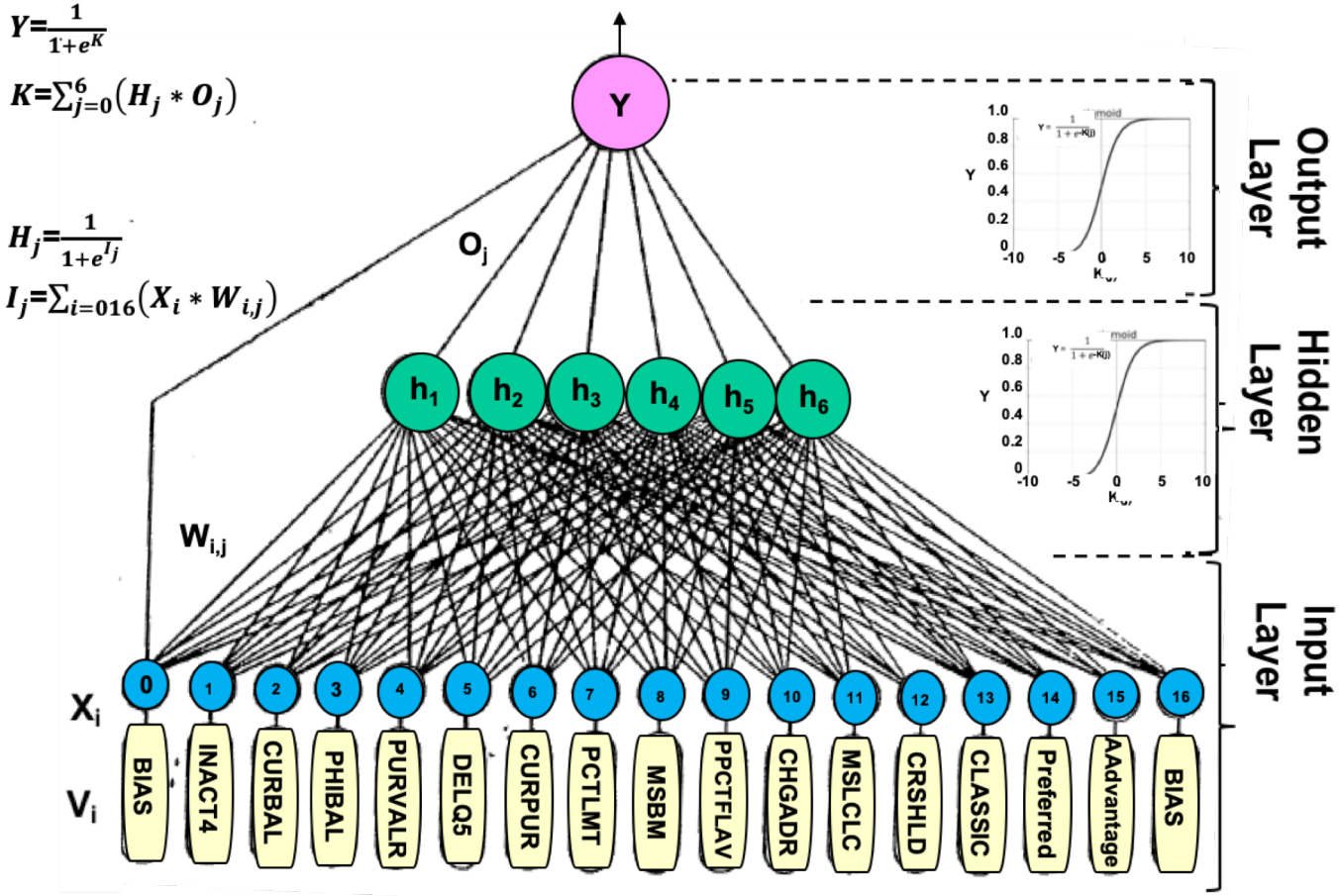
- **OVERALL GOAL: Develop quantitative tools for use in anti-attrition marketing strategies (companion certificate for American Airlines)**
  - **These tools will identify potential customer cancellations and suggest optimal marketing strategies on an individual account basis**
- **A time-series database of account activity from the period 3/88 to 4/90 was extracted and used to build neural network models to predict fee-based attrition**
- **As a first step, we built a model for identifying fee-based attrition using variables currently computed for an attrition model sold by Fair Isaac Company (FICO)**
  - **This enabled us to evaluate the gain provided by network modeling tools**
  - **It also made it easier to implement the neural network model and to integrate its results into the targeting system used by marketing**
- **We demonstrated that a network, model provides significant improvement over the FICO model**
  - **We implemented that model for targeting potential attritions, and we evaluated its effectiveness as well as that of anti-attrition strategies in the following months**

Neural Network for Credit Card Attrition

The "optimal" neural network was found to have six nodes in a single hidden layer

- A total of 16 variables are input to the network along with a constant BIAS of 1.0
- Each variable is scaled in term of its minimum and maximum

Model for Credit Card Attrition



V_i – Raw Inputs

$$X_i=\left\{\frac{f(V_i)-min[f(V_i)]}{max[f(V_i)]-min[f(V_i)]}\right\}$$

Variable Names, Minimum and Maximum Values

No.	Variable	Minium	Maximum
1	BIAS	1	1
2	INACT4	0	4
3	CURBAL	-747	12060
4	PHIBBAL	-610	104
5	PURBALR	0	100
6	DELQ12	0	100
7	CURPUR	0	1
8	PCTLMT	0	419
9	MSBM	1	255
10	PPCTLFAV	-168	67800
11	CHGADDR	0	9
12	MSLCLC	0	9
13	CRSHLD	0	1
14	CLASSIC	0	1
15	PREFFER	0	1
16	ADVANTG	0	1
17	COLLEGE	0	1

Interpreting Neural Network Decisions

Neural Networks have often been criticized as being "black boxes", but in fact, there are several methods for reliably determining neural network decisions.

1. List of Variables
2. Univariate Relationships
3. Variable Exclusion Method
4. Garson's Method
5. Analysis of Derivatives

List of Variables

- The simplest and most straightforward, but least informative method for imputing network decision making is to consider the list of input variables
- A list of variables is useful in that one can at least see the full spectrum of information available to the network
- Of course, the usefulness of this method is limited as there is no measure of the relative importance of each input variable in the network

Variable Names

Variable
BIAS
INACT4
CURBAL
PHIBBAL
PURBALR
DELQ12
CURPUR
PCTLMT
MSBM
PPCTLFAV
CHGADDR
MSLCLC
CRSHLD
CLASSIC
PREFFER
ADVANTG
COLLEGE

Interpreting Neural Network Decisions (cont.)

Univariate Relationships

- A more informative analysis of individual variables can be gained by performing regression tests between each input variable and the dependent variable in question
  - For example, assume that we have 20 input variables thought to be related to the item that we wish to predict
  - Then, for each candidate input variable, we build a logistic regression model by selecting values of a_i and β_i for each variable that is most highly correlated with the target variable
- We can then rank the variables with respect to their relative univariate predictive power
  - Presumably, the variable with the greatest univariate predictive power is the most important, and so forth
- One limitation of this method is that relationships among input variables are not captured and multicolinearity is not assessed
  - Still, the method reveals just as much as revealed by simple regression models

Interpreting Neural Network Decisions (cont.)

Variable Exclusion

- A direct way to determine a variable's importance is to first build a predictive model with a given set of variables and, once constructed, remove each variable in turn and retrain the model without that given variable
  - The variable exclusion models need not be of the same structure as the original model; they should be the best models one can build with and without that variable

- It is hard to see how the variable exclusion method does not capture the importance of each individual variable
  - However, this method does not capture the importance of bi-variate or multi-variate interactions in multi-layer models
  - Like the univariate relationship method, the variable exclusion method reveals at least as much about variable contributions as is revealed by simple regression models
  - Furthermore, variable exclusion provides information about multi-collinearity

Interpreting Neural Network Decisions (cont.)

Garson's Method

- A quick and effective way to determine univariate contributions to multi-layered networks and decision trees was proposed by Garson (1991)
- Essentially, Garson's method is to compute for each normalized (e.g., 0 to 1 scaled, Gaussian distributed, etc.) input variable the sum of the absolute values of its weights throughout the network or tree
- That is, Garson's score for a given input variable, *i*, is calculated as

$$G(i) = \log\left(\frac{\sum_{j=1}^{k} |w(i,j) * o(j)|}{\sum_{i=1}^{n}\left(\sum_{j=1}^{k}|w(i,j) * o(j)|\right)}\right)$$

where

n = number of input variables
k = number of hidden nodes
w(i,j) = weight of the i^{th} variable into the j^{th} hidden node
o(j) = weight of the j^{th} hidden node into the output node

Interpreting Neural Network Decisions (cont.)

Analysis of Derivatives

- **Currently, the best way I know of to reveal the decision-making properties of a complex system is to perturb slightly its inputs both individually and in combinations with all other inputs to the desired degree of complexity**
- **Assuming that one has trained a version of the network or decision tree to a given level of accuracy, one version of the par procedure is as follows:**
  1. **Present a single training input vector to model and measure the output of network**
     - **The resulting value is the benchmark for that vector**
  2. **For each of the *n* variables in that input vector**
     - **Adjust the value of that variable up by a given amount (say 5% for convenience) and measure the change in output; do the same for a small perturbation downward**
     - **Compute the slope of best fit line through the three outputs (up, down and benchmark) as a function of value of the input variable to approximate change in output with respect to input (i.e., the compute the derivative)**

Interpreting Neural Network Decisions (cont.)

Analysis of Derivatives

3. **Repeat steps 1-3 for all cases in the sample used for training the model and calculate the summary statistics of the resulting changes for each individual variable**
   - **Be careful: it may be that a single variable have only positive or negative effects relative to the benchmark. This is why examination of the distribution is critical**

- **Proceed to analysis of bi-variate and multi-variate contributions**

4. **Beginning again with a single n-valued input vector from the training set, select two variables for joint perturbation.**
   - **As in Step 1, present a input vector to the model, and let the resulting output value be the benchmark for that vector**
   - **Then perturb by 5% each of the two variables simultaneously, up together, both down, and each pair in opposite directions, up-down, down-up, respectively**
   - **As before present the entire vector to the model, with the resulting value be designated as the benchmark for that vector**

Interpreting Neural Network Decisions (cont.)

Partial Derivative Analysis

- Compute the slope of best fit line through the three outputs (up, down and benchmark) as a function of value of the input variable to approximate change in output with respect to input (i.e., the compute the derivative)

5. Repeat Step 4 for all cases in the sample used for training the model and calculate the summary statistics of the resulting changes for each set of variables
   - Be careful: it may be that a single variable have only positive or negative effects relative to the benchmark. This is why examination of the distribution is critical
6. Repeat Steps 4 and 5, but this time perturbing three variables at a time and calculating its nine associated derivatives, for each trio and compute their averages and standard deviations (for each of the nine cases for each trio.

Neural Network for Ranking Mutual Funds 3/3/92

We trained a neural network to select the mutual fund that had the best chance of being profitable over the next month. We then analyzed the variable contributions.

- We built a model to predict one month returns on mutual funds
  - The model had 18 inputs, 4 hidden nodes and 1 output
  - 500K learning trials (~100 Epochs)
  - Binary dependent variable (0:1)
  - N(Train) = 2,175; N(Test) = 2,253
- Variable importance was assessed using partial derivative analysis and Garson's method
  - There is reasonable consistency between the two methods
  - This is particularly true at the extremes where largest positive and negative contributing variables are the same

Analysis of Variable Contributions

Analysis of Small (5%) Perturbations				Garson's
Variable	Mean	Deriv.	Std.Dev.	Method
R(t-1)	0.498	3.56	2.31	0.253
R(t-3)	0.505	1.03	0.77	0.046
dR(t-12)	0.499	0.68	0.84	0.059
R(t-12)	0.502	0.43	0.47	0.040
Group 4	0.138	0.41	0.43	0.021
Class-II	0.250	0.25	0.40	0.250
Group 1	0.362	0.25	0.46	0.027
Class-III	0.250	0.25	0.47	0.035
Class-I	0.363	0.01	0.40	0.023
LOAD	0.206	-0.01	0.21	0.007
R(t-6)	0.498	-0.07	0.73	0.049
Group 2	0.251	-0.08	0.44	0.031
dR(t-6)	0.510	-0.14	0.51	0.035
Class-IV	0.138	-0.20	0.44	0.038
LOG($)	0.762	-0.21	0.45	0.036
Group 3	0.249	-0.24	0.37	0.023
dR(t-1)	0.513	-0.99	1.16	0.110
dR(t-3)	0.492	-2.56	1.30	0.142

Neural Network for US Treasury Yield Changes

We developed a model to predict short-term movements in US Treasury yields and evaluated the model's performance in simulated and actual trading.

- Our objective was to predict the near-term directional change of the U.S. Treasury bonds
  - The model was trained to predict changes in 30-year US Treasury bond yields
  - We chose as inputs to the model a set of 26 variables consisting of raw market bond price series, equity indices, foreign exchange rates and technical stochastic indicators
  - In addition, the ratios of short-term changes in variables and other statistics were computed on some of these quantities
  - We were able to get data on all these quantities since 2Q84, so the first market moves that we predicted were for 2Q92, trained on the data between 1984 and 1992
- The neural network model is really a series of quarterly models with a new set of network weights developed for each subsequent quarter over the 11-year sample period (walk-forward method)
  - Each successive period incorporated an additional three months of data
  - Thus, although we began predictions for 1992 with eight years of trailing data for training, by 2000, we had doubled the training set by including all the data from 1992 to 2000

We were granted a patent on the model in 2009: T. Benzschawel, C. Dzeng, and G. Berman, Method and System for Artificial Neural Networks to Predict Price Movements in Financial Markets, US Patent: 7529703, May 5, 2009

The US Treasury Bond Model

One feature of development of the US Treasury Bond Model was that we only trained the on largest 1/3 up and 1/3 down moves in US Treasury yields.

- **The network consists of:**
  - **Input layer (represents the daily market values used for prediction)**
  - **Hidden layer (sums weighted signals from each of the inputs prior to a nonlinear transformation)**
  - **Output layer (sums the weighted outputs of the hidden layer and produces the response).**
- **During training, values of variables for a given day are presented to the network**
  - **The difference between the network output and its 'desired' output (i.e., the actual market move) is backpropagated**

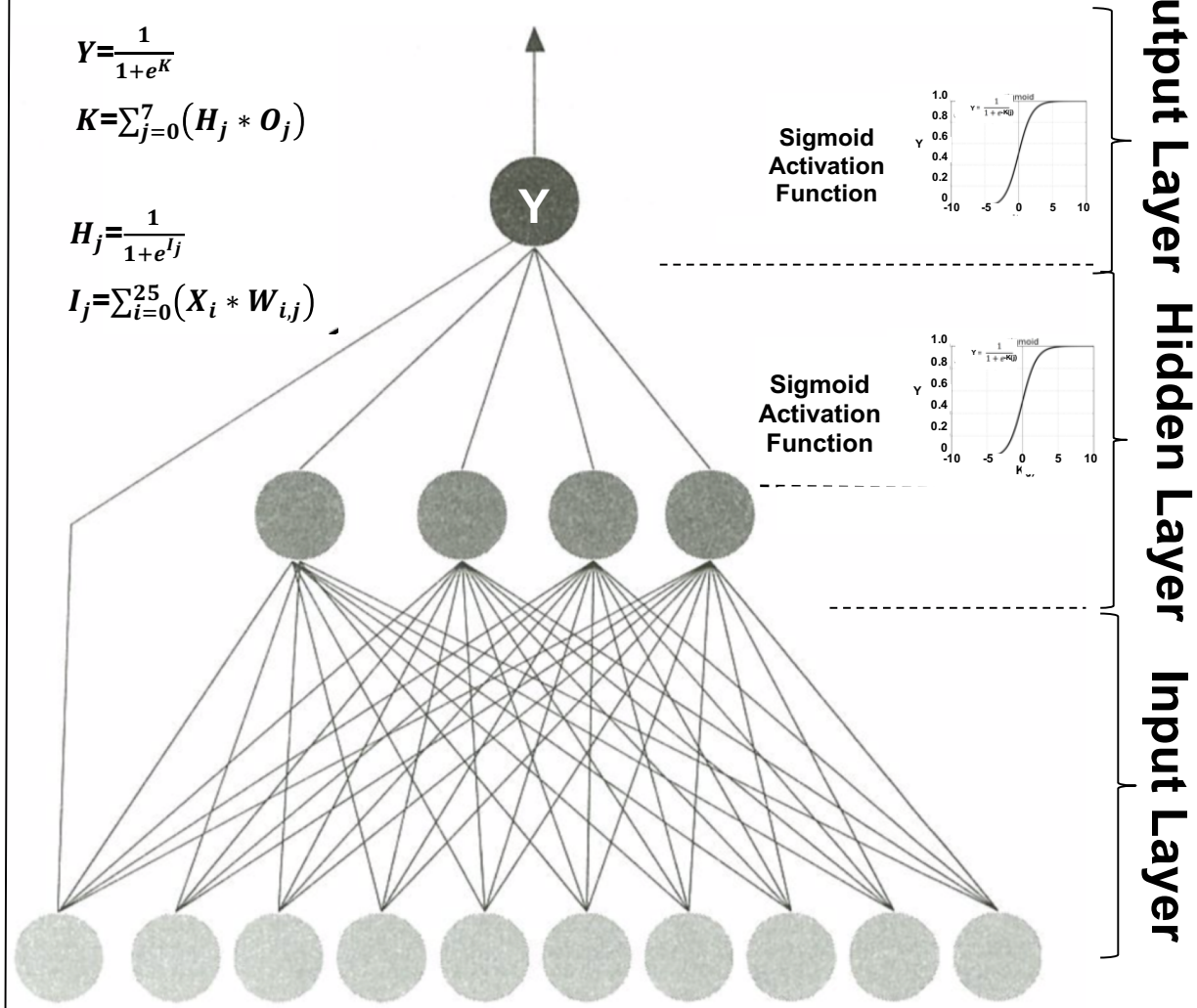
Network for US Treasury Bond Yields

$$Y=\frac{1}{1+e^{K}}$$

$$K=\sum_{j=0}^{7}(H_j * O_j)$$

$$H_j=\frac{1}{1+e^{I_j}}$$

$$I_j=\sum_{i=0}^{25}(X_i * W_{i,j})$$



Input Variables: V_i – Raw Inputs

$$X_i=\left\{\frac{f(V_i)-min[f(V_i)]}{max[f(V_i)]-min[f(V_i)]}\right\}$$

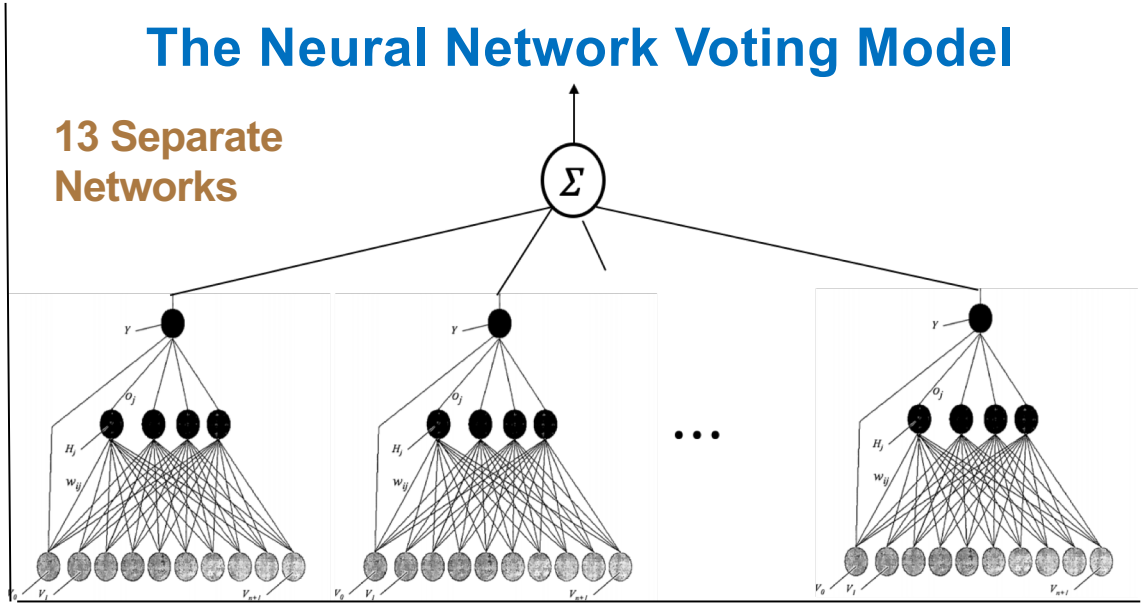
The actual networks had 27 inputs and 7 hidden nodes.

The US Treasury Bond Model (cont.)

- Backpropagation consists of adjusting slightly each of the weights in the network in proportion to its ability to reduce the output error.
- The entire training set is presented repeatedly in random order until network performance stabilizes
- The technique is called the gradient descent method of error backpropagation.

- We used this method to set the weights of the network using as training data daily technical indicators and the resulting future price change over a minimum sample period of eight years prior to the test dates

- Our actual trading model consisted of not just one, but 13 of the structures similar (but not identical) to that shown in the previous slide
  - The idea behind having multiple models is to avoid the "local minima" problem



- The weights for each of the 13 networks were generated using the same structure of nodes, learning data number of trials, etc., but the initial random weights assigned to the connections for each network were generated using a different random seed

The US Treasury Bond Model (cont.)

- Although each of the 13 models produces a continuous output score, that output goes through a series of transformations that ultimately result in the generation of one of three signals: BUY(B), SELL (S), or NEUTRAL (N)

Daily Trading Procedure

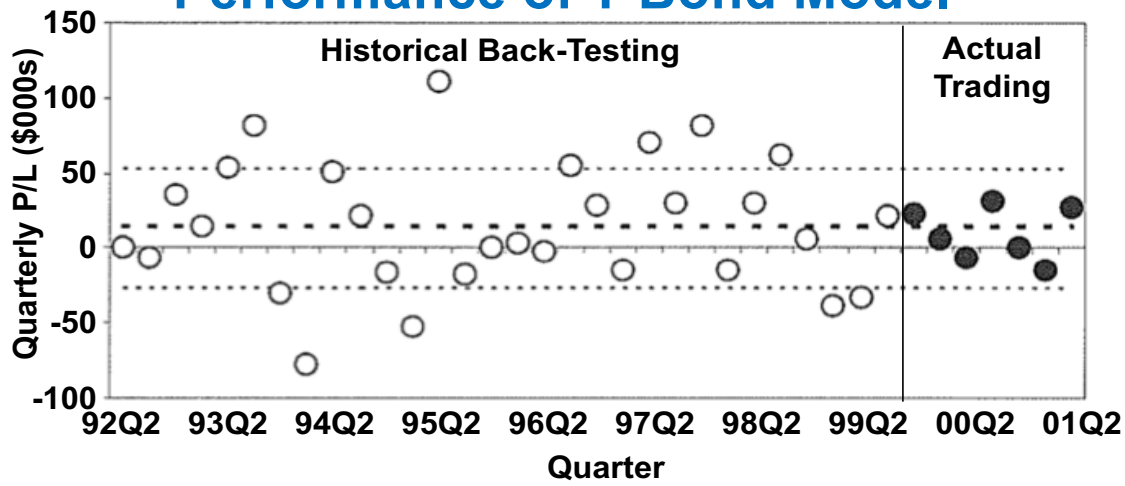
- At the close of a given trading day, input variable values are fed into the 13 networks and the signal is calculated
  - If 7 or more networks signal B (output > 0.66), a BUY is initiated, if 7 or more networks signal S (output < 0.33), a SELL, and if neither gets at least 7, the signal is N, NEUTRAL
  - We call a trade of one-unit the equivalent of $100,000 face of US T-Bonds
- If the signal is B, we add one unit to our position, if it is S, we subtract one unit, and if the signal is N, we do nothing that day
- However, if any trades were done on the 12th-previous business day, that trade is unwound at the current day's close
- On any given day, the maximum number of units that one can trade is two: one new position, and an unwinding of a previous one
- Similarly on any given day, the maximum size of our position could be long or short 12 units

To my knowledge, this was the first usage of a voting model. This has now become commonplace.

T-Bond Network Performance

We evaluated the model's performance in out-of-sample simulated trading from 2Q92 to 4Q99, and in actual market trading beginning in 4Q99.

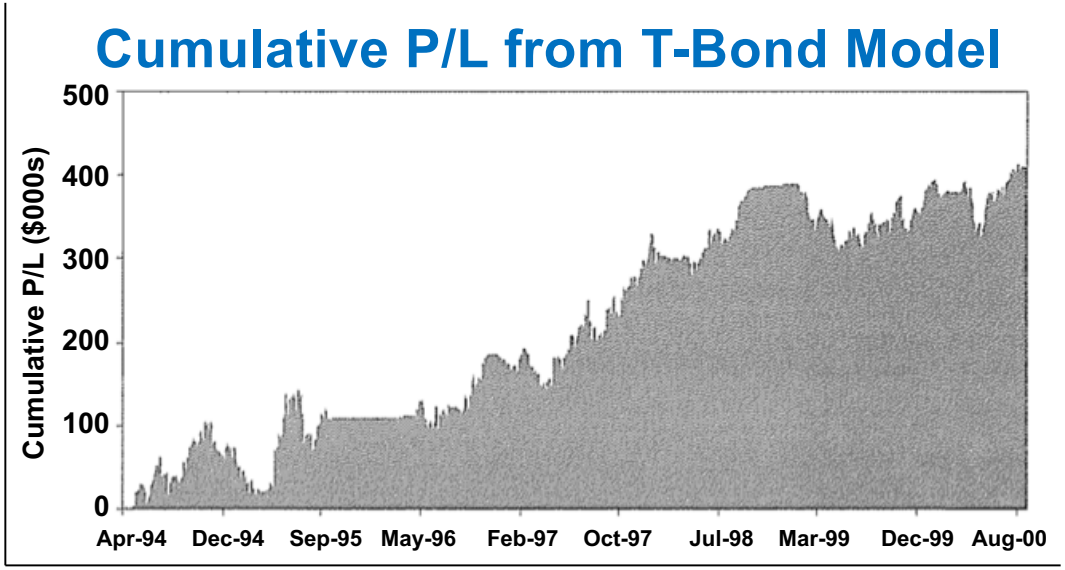
- We conducted historical backtesting using this method, while recalculating the model parameters each quarter with additional data from the prior one



- The figure presents the results of our backtesting and trading
  - The solid line in is the zero-profit axis, the heavy dashes show the average monthly P/L over the sample period, and the light dashes denote +/-1 standard deviation in monthly P/L
  - Of the 37 quarters tested, 22 are profitable, 13 show losses and 2 have no P/L
  - Profitable quarters have a mean P/L of $38,000, whereas losing quarters have an average P/L of -$25,000
  - The largest quarterly gain over the 1992-2001 period is $111,000 and the largest quarterly loss is $78,000
- Variability in P/L is sometimes high, but has decreased in later years
  - This may be due to the fact that each quarter we are adding cases to the training set such that the predictions for 2001 are generated from networks that have nearly twice the learning cases as those for 1992

T-Bond Network Performance (cont.)

- The figure shows cumulative P/L from 2Q94 through September 9, 2000
  - The model generated $0 P/L in the first sample quarter (2Q92) and -$7,000 in 3Q92 (not shown in the cumulative plot),"
  - The model has not had negative cumulative P/L since then
  - From April 1994 through September 2000, it had a cumulative P/L of $413,604, with a mean of $256 per business day and a standard deviation of $5,151.
  - This gives an annualized Sharpe Ratio of 0.79 for the period.



- The historical probability of correct decision (i.e., Market goes up I BUY and Market goes down | SELL) for each unit-trade is 54% and this has been replicated in actual trading from 3Q99 through June 2001

Machine Learning and Neural Networks in Finance

Predicting Market Moves from Customer Trading Patterns

Predicting Bond Price Changes from Client Trade Flow

We used data from client trades, bond indicative data, and client type to predict moves in bonds prices.

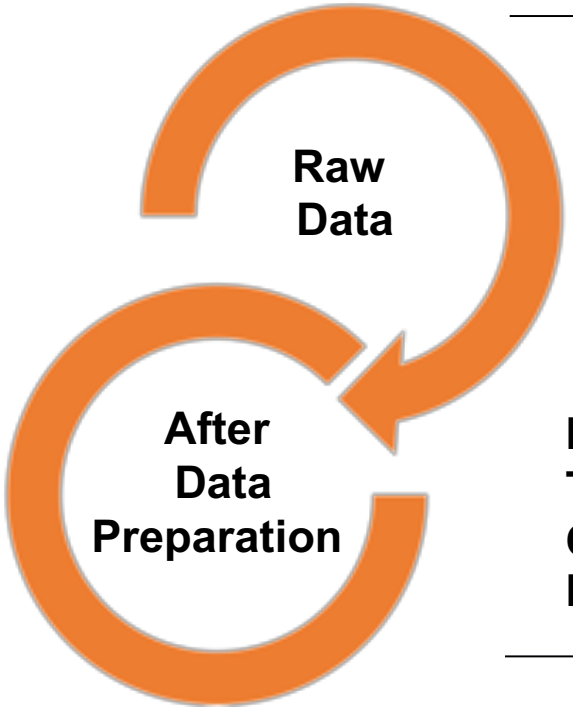
- **We used three types of information to predict bond price movements:**
  - **Trade information: Quantity and Direction**
  - **Bond Indicative Data: ID, Industry, OAS, Investment-Grade and High Yield**
    - **We collected both pre-trade and post-trade OAS spreads. The key observation time points as 1, 5, 10, 20 days before and after the trade and the trade date**
  - **Client Information: ID and Type**
    - **The groups are Insurers, Corporates, Banks, Asset managers, Hedge funds, Public sector, Pension funds, Other.**

Time Frame for Model Development and Testing

To remove the market impact from the data, all trades were analyzed relative to percent changes in the US Investment-grade bond index.

Data Preparation

- The time frame is from Jan 2014 to Aug 2015, 20 months in total
- There are 109,890 trades, 1,908 clients and 6,100 bonds
- Data cleaning issues:
  - For newly issued bonds which did not have 20 days of past data we just set them to NA and excluded them from training
  - We removed clients with less than 50 trades over the observation period
  - We removed bond OAS outliers with more than +/- 2.7σ from the mean



Period: 1/14 to 8/15
Trades: 109,890
Clients: 1,908
Bonds: 6,100

Period: 1/14 to 8/15
Trades: 88,394
Clients: 1,406
Bonds: 5,105

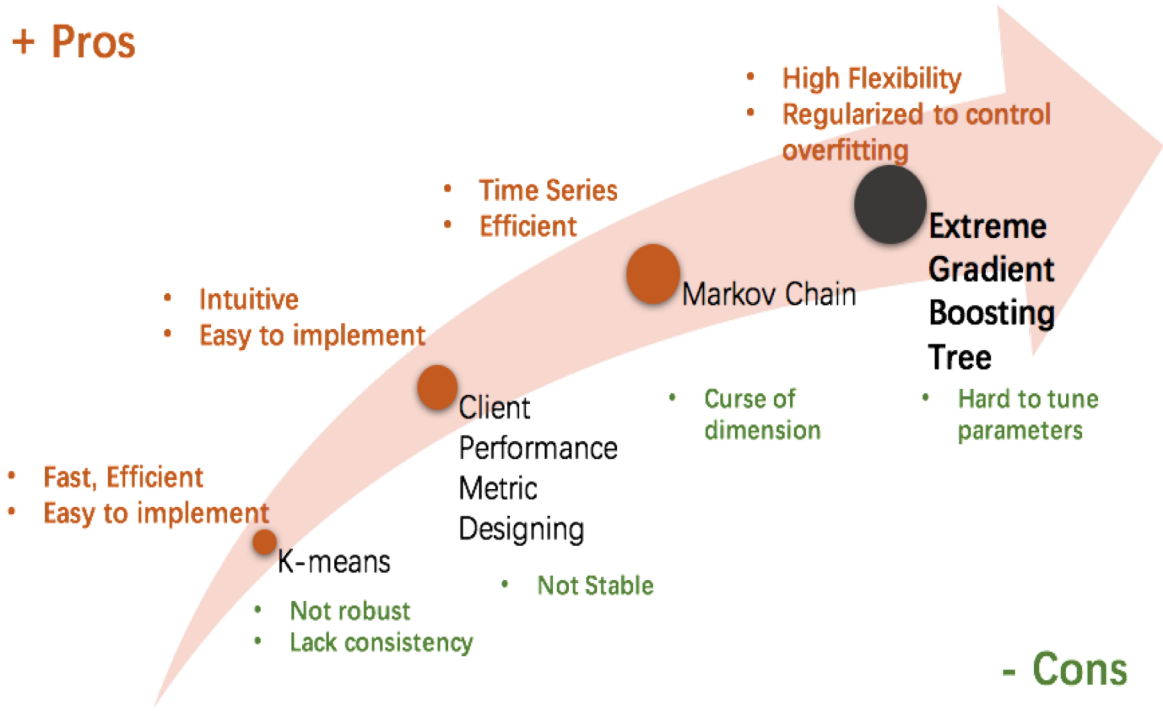
- After data cleansing, we have 88,394 trades, 1406 clients and 5105 bonds for further study

Model Selection

We tested several methods in order to accurately predict corporate bond price moves over 1, 2, 5, 10 and 20 days.

- **K-Means:**
  - **Advantages: fast, efficient, easy to implement**
  - **Disadvantages: not robust, lack consistency**
- **Client Performance Metric Designing:**
  - **Advantages: intuitive, easy to implement**
  - **Disadvantages: not stable**
- **Markov Chain:**
  - **Advantages: time series, efficient**
  - **Disadvantages: curse of dimension**
- **Extreme Gradient Boosting Tree:**
  - **Advantages: high flexibility, regularized to control over fitting**
  - **Disadvantages: Hard to tune the parameters**

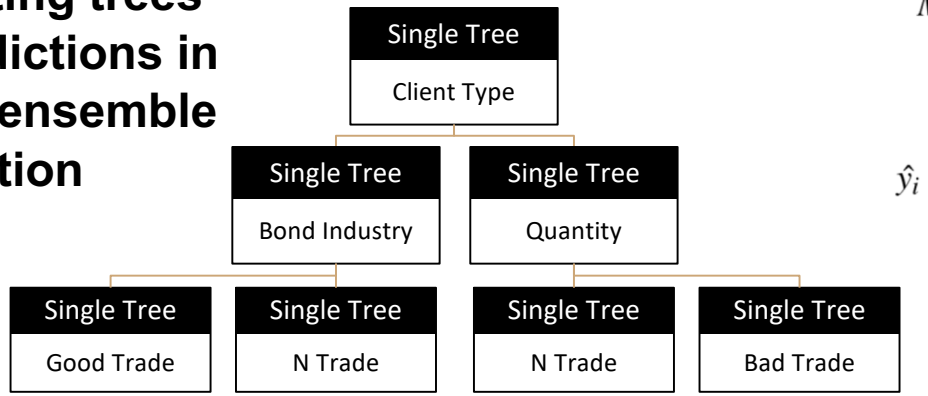
Taxonomy of Models Tested



Extreme Gradient Boosting Tree

The extreme gradient boosting tree yields the best results and will be the focus of the report.

- **The Extreme Gradient Boosting Tree is a widely used supervised learning method which can be applied to regression, classification and ranking problems**
  - **Gradient boosting trees produce a predictions in the form of an ensemble of weak prediction decision trees**



GBT Loss Function

$$MinObj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

n = number of training trades

y_i = actual score

$\hat{y}_i$ = predicted score = sum of leaf scores

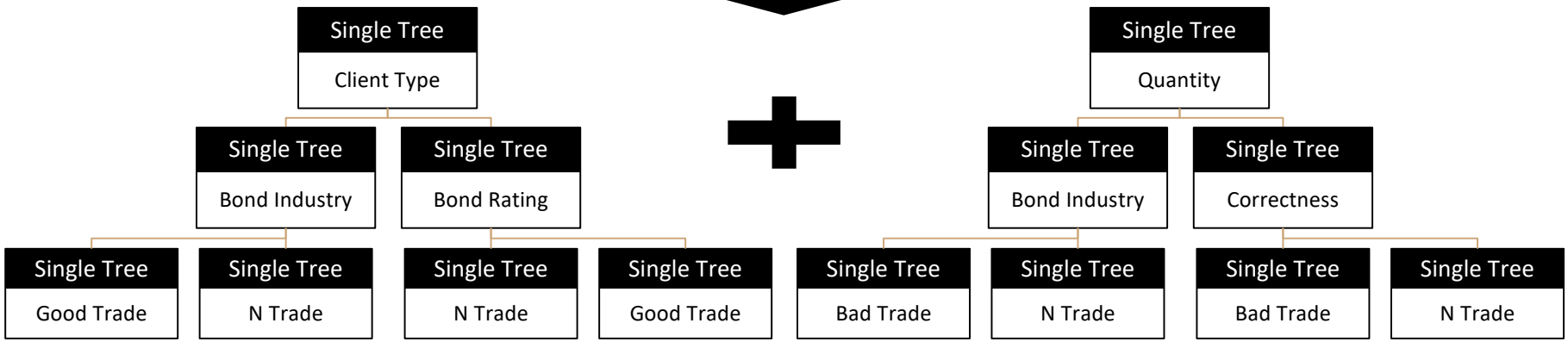
l: logistic lost function

Ω = complexity penalty function

$= \gamma T + \frac{1}{2}\lambda \sum_{j=i}^{T} \omega_j^2$

T: number of leaves

ω_j:leaf score



Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models.

- **Most boosting algorithms consist of iteratively learning weak classifiers and adding them to a final strong classifier**
  - **They are typically weighted in relation to the weak learners' accuracy.**
  - **After a weak learner is added, the data weights are readjusted,**
  - **Misclassified input data gain a higher weight and examples that are classified correctly lose weight**
  - **Thus, future weak learners focus more on the examples that previous weak learners misclassified.**
- **The main variation between many boosting algorithms is their method of weighting training data points and hypotheses**
  - **AdaBoost is very popular and the most significant historically as it was the first algorithm that could adapt to the weak learners**
    - **AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier**
    - **When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.**

Training the Model

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of models

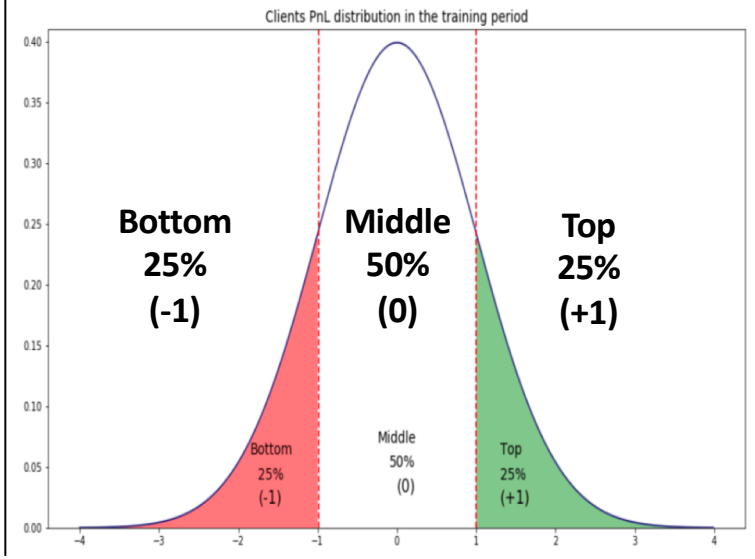
- **Walk-Forward Method**
  - In order to incorporate changes in the corporate bond market yet have enough information to train the model, we used a rolling training period of 3 months
  - For each training period, we used the following 1 month of data to test the model performance
  - We then moved the training and test samples one month in time and repeated the training and test procedure

- **Variables**
  - For each modelling period, we input the client type, bond industry, bond grade, past return as independent variables
  - The dependent variable is the post trade P/L calculated using the following formula:

$$\text{PnL} = \sum_{i=1}^{n_t c} [\text{quantity} \times duration \times (\Delta_{\%} s_c - \Delta_{\%} b_m)]$$

  - Then, we assign the P/L to one of three classes (-1, 0 ,+1) as shown on the right

Transformed Post-Trade P/L



Training the Model (cont.)

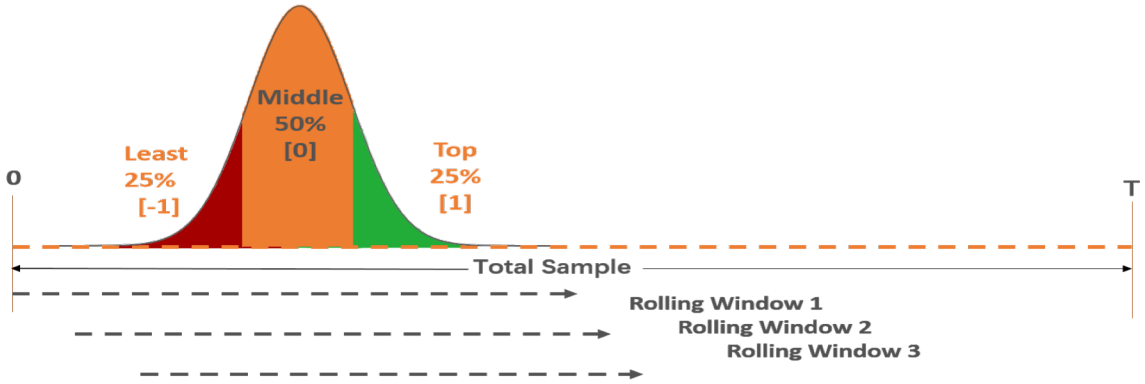
- **Data Segmentation**
  - To improve the predictive power of the model, trades were segmented into buys and sells
  - This is because we noticed that buy and sell trades have fundamental difference in trade behavior
  - As we have interest in both short-term prediction and long-term prediction, we also set different lags (1, 2, 5, 10, 20 days) of P/L as dependent variables

- **Portfolio Construction and Evaluation**
  - In our test set, we construct the portfolio in the following three steps:
    1. Based on the model prediction, if the result is -1 we do the oppose of the trade direction. If the result is 0, we abandon the trade. If the result is 1, we follow the trade
    2. Normalize our resulting trade P/L to ensure we use same capital for model and benchmark
    3. Compare our constructed trade portfolio with the benchmark portfolio

Modeling Paradigm Summary

- **Training Paradigm**
  - **We trained the model on a 3-month rolling window and tested 1-month out**
  - **We divided our signal range into three regions for executing trades**

- **Training 3 months and Testing 1 month**
- **Features: client type/bond industry/bond grade/past return**
- **Signal Strength: Quantile**
- **Classify by signal strength: Post Trade P/L -> +1 / 0 / -1**

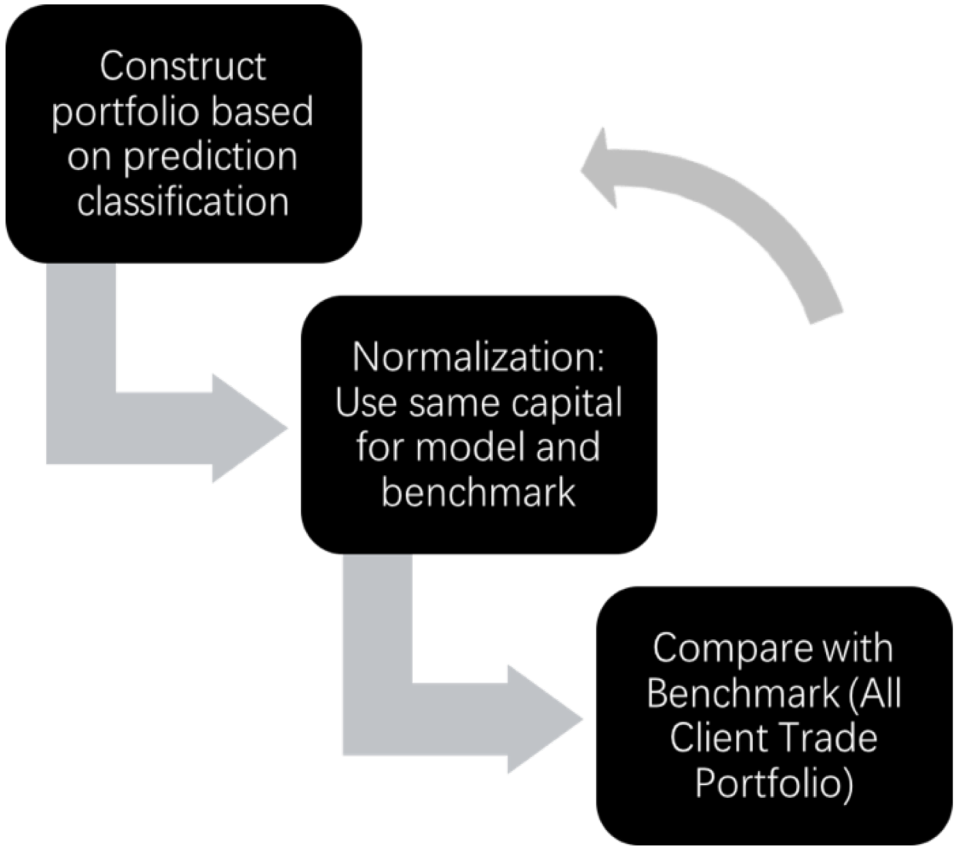


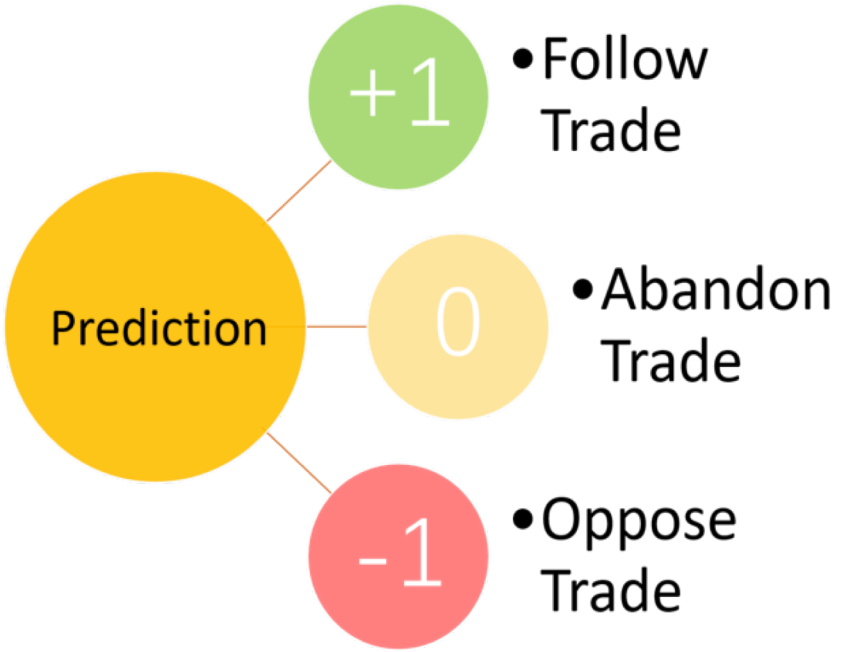
- **Generating Predictions**
  - **We examined client buy and sell trades for holding times of 1, 5, 10 and 20 days**

Portfolio Construction

We constructed portfolios for each client based on the model predictions and compared that with the clients' performance.

When the client traded, the model could either go along with the client, against the client or stay out of the trade.





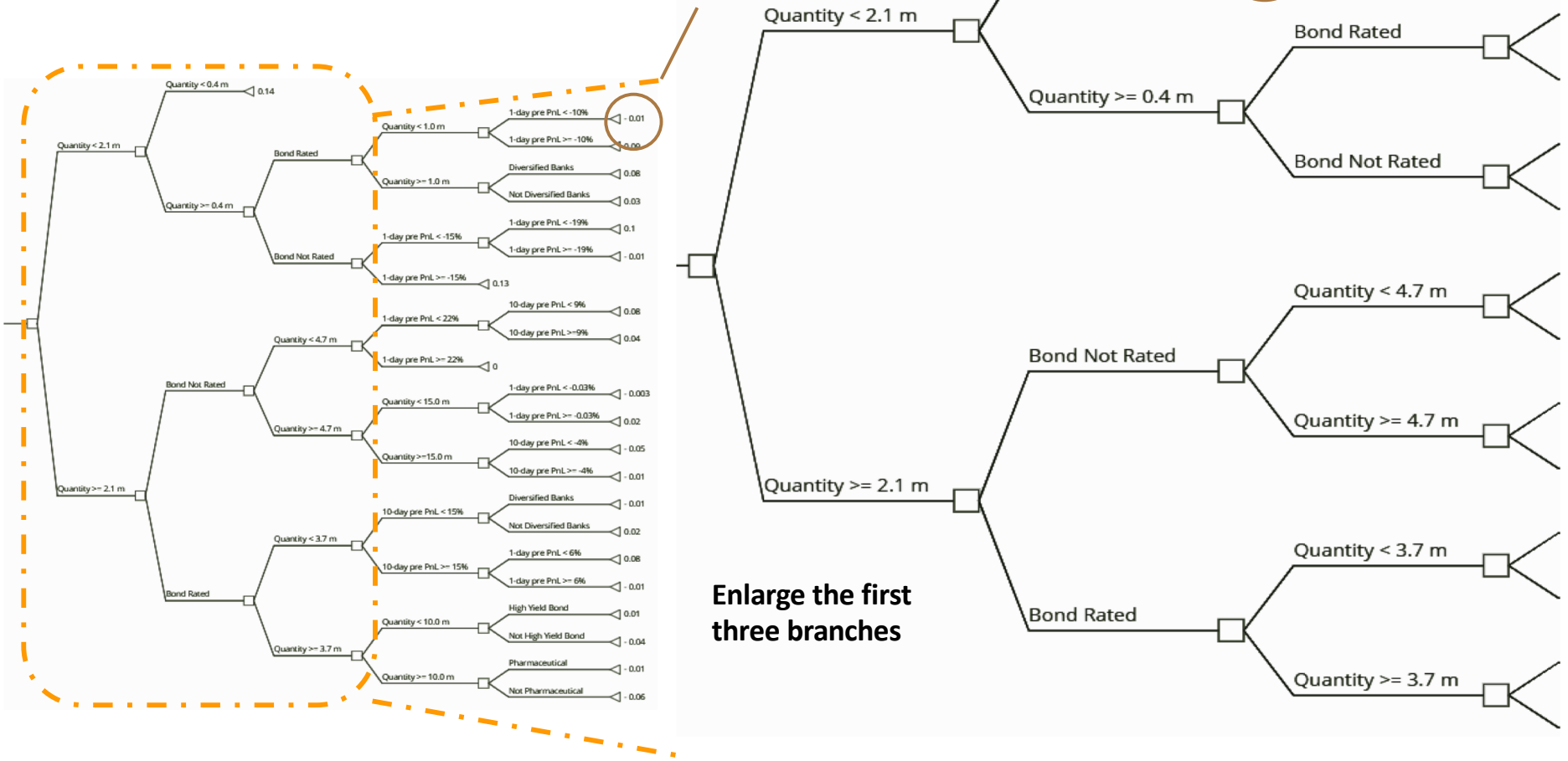
XGBoost Tree Demonstration

The final prediction in the form of -1, 0 and 1 will be calculated as the sum of score given by all individual trees.

Given the full tree structure (below) is complicated, partial tree graph is presented on the right

A rated bond trade with quantity between 0.4 million and 1 million dollars and past 1 day P/L less than -10% will be assigned to the second ending node with score -0.01

A trade with quantity less than 0.4 million dollars will be assigned to the first ending note with score 0.14



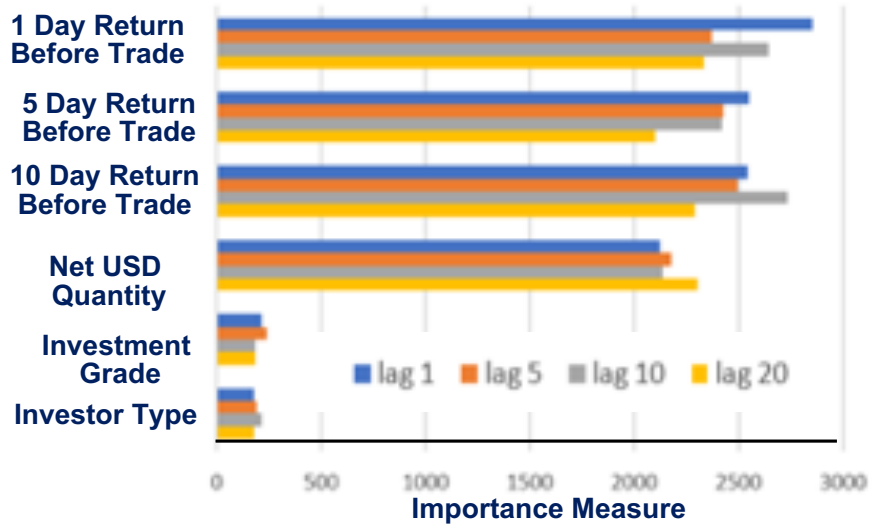
Enlarge the first three branches

XGBoost Tree Model Feature Analysis

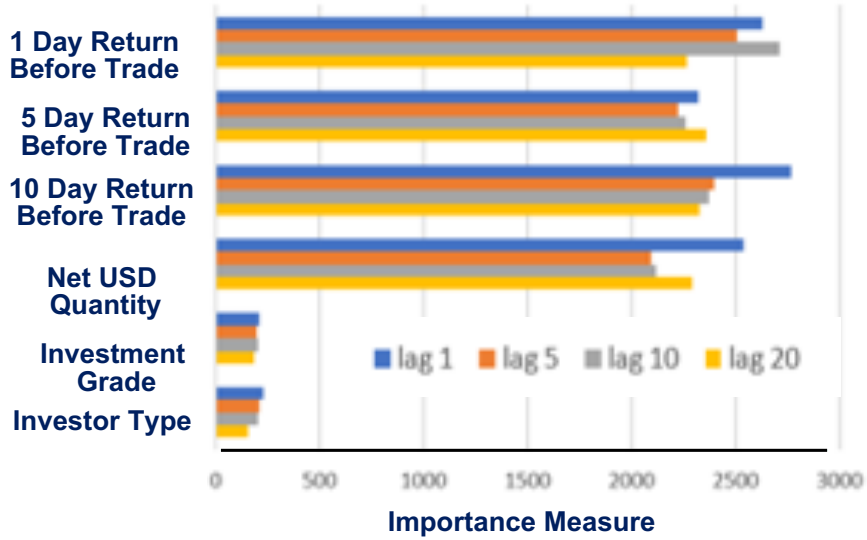
The Boosting Tree Model provides a decision path predicting each trade's profitability, and gives some insights about features of good and bad trades.

- **The pattern of variable importance is similar for buy and sell trades**
- **The most important feature in predicting returns from buy and sell orders is recent P/L**
  - **This is true even for 10-day prior P/L**
  - **Thus, the model is mainly a momentum model**
- **Besides past performance, order quantity, corporate bond rating, and investors' type (e.g., bank, asset management and hedge fund) also play important roles in this model.**

Variable Importance for Buy Trades



Variable Importance for Sell Trades



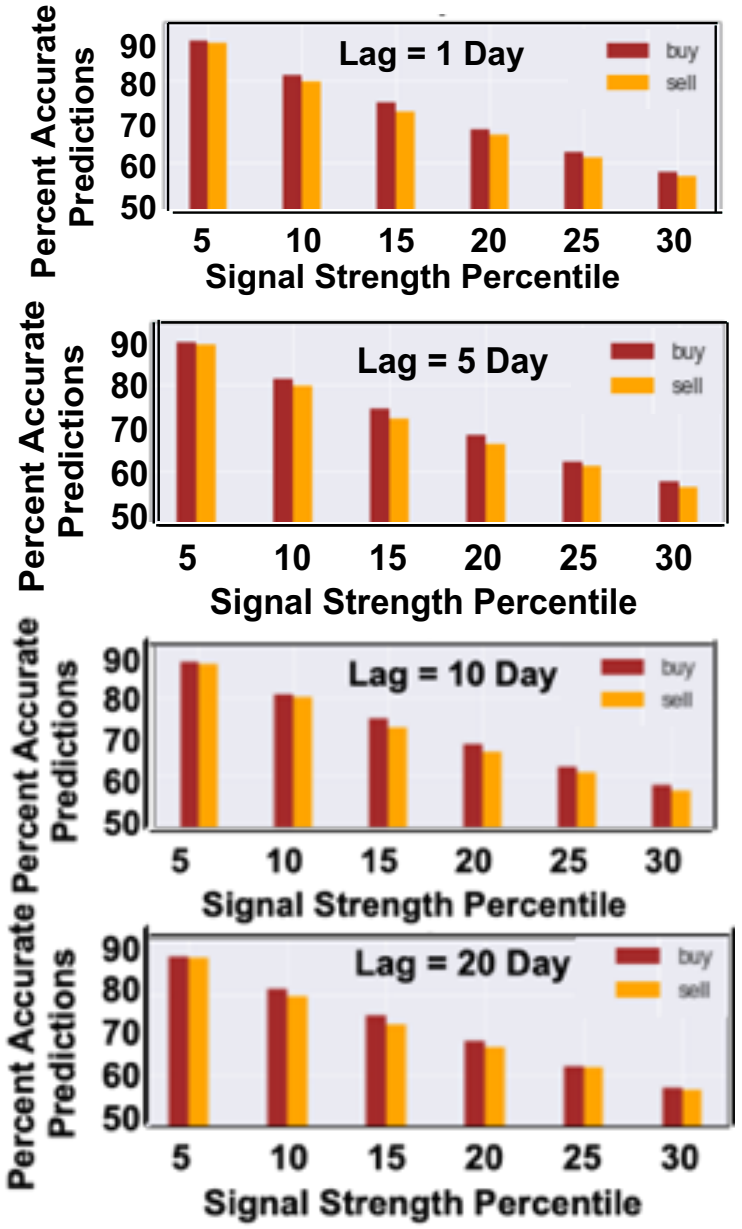
XGBoost Model Accuracy

We measure model accuracy by percent correct over various time periods (1, 5, 10 and 20 days).

Probability Correct

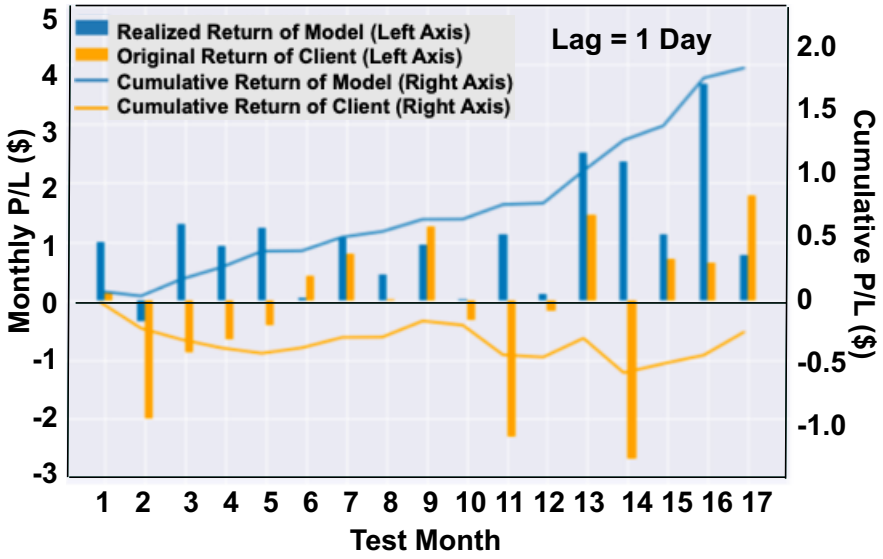
- One measure of accuracy is the percentage of correct predictions (buy – bond up; sell – bond down)
- The figures plot probability correct as a function of the signal strength
  - Signal strength percentile means that all signals at or stronger than that percentile are included in the analysis
- Performance is above 50% for signals greater than the 30% quantile
  - This means that for 30%-40% of the signals, model performance is at chance
- Accuracy decreases with decreasing signal strength for all lags
  - in order to archive a high accuracy score such as 90%, a criterion of 5% is required

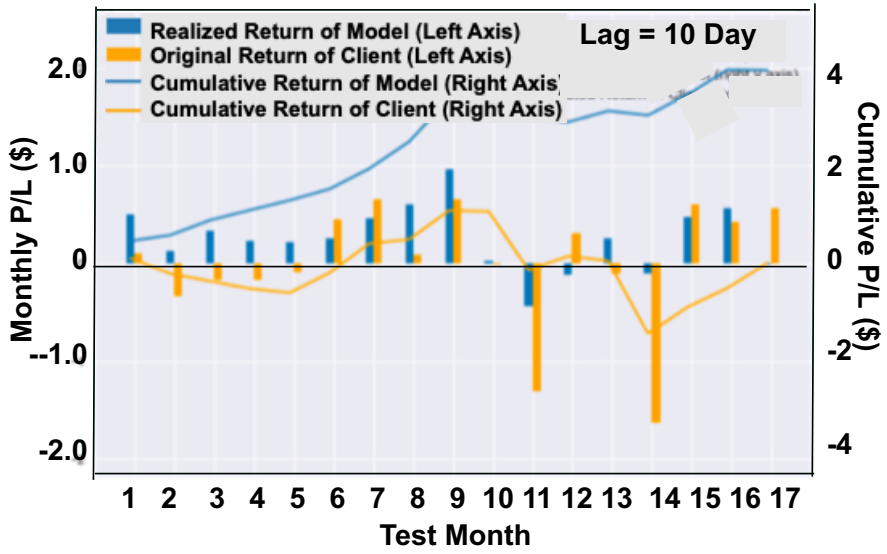
P(Correct) by Percentile Criterion

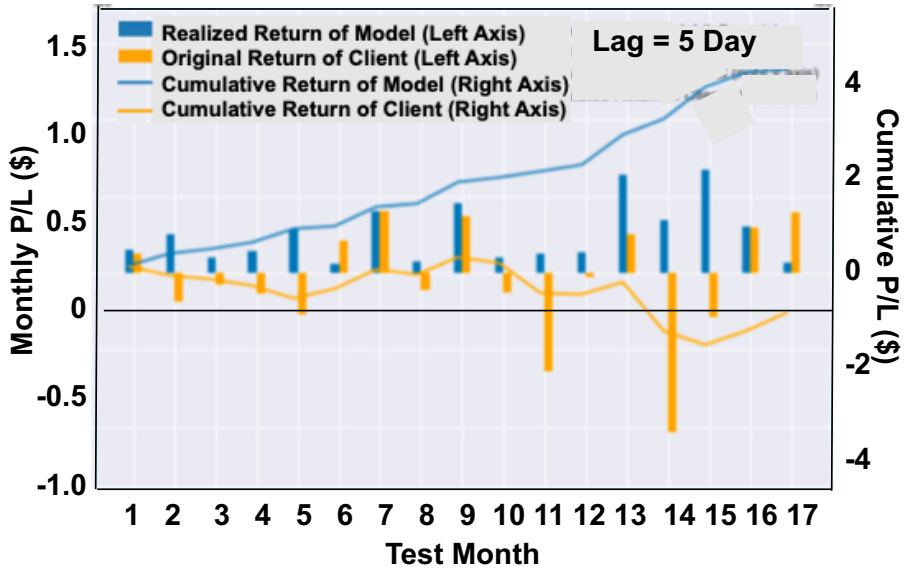


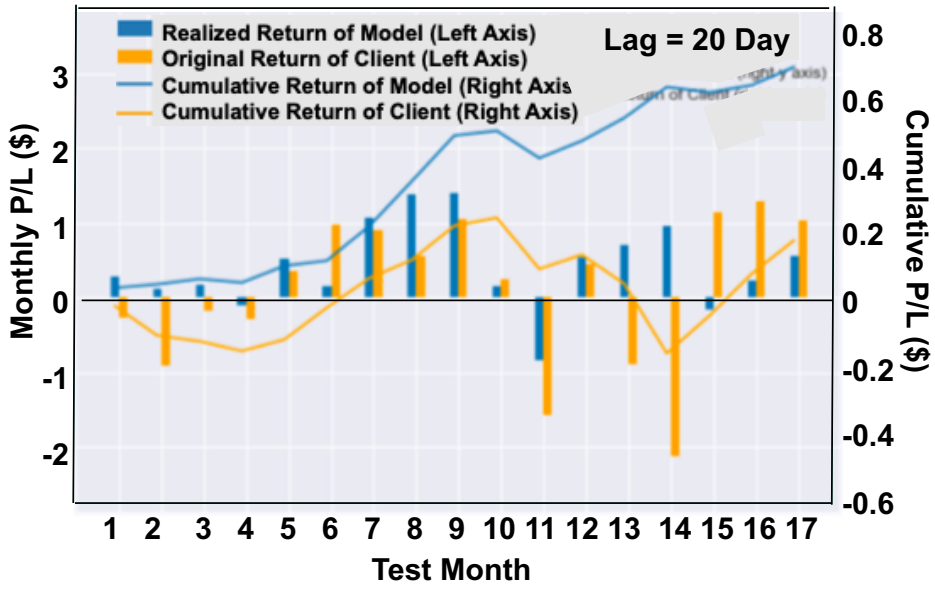
Analysis of Trade Portfolios – Buy Trades

On average, the model outperforms our clients for bond buy trades. This is true over all tenors.



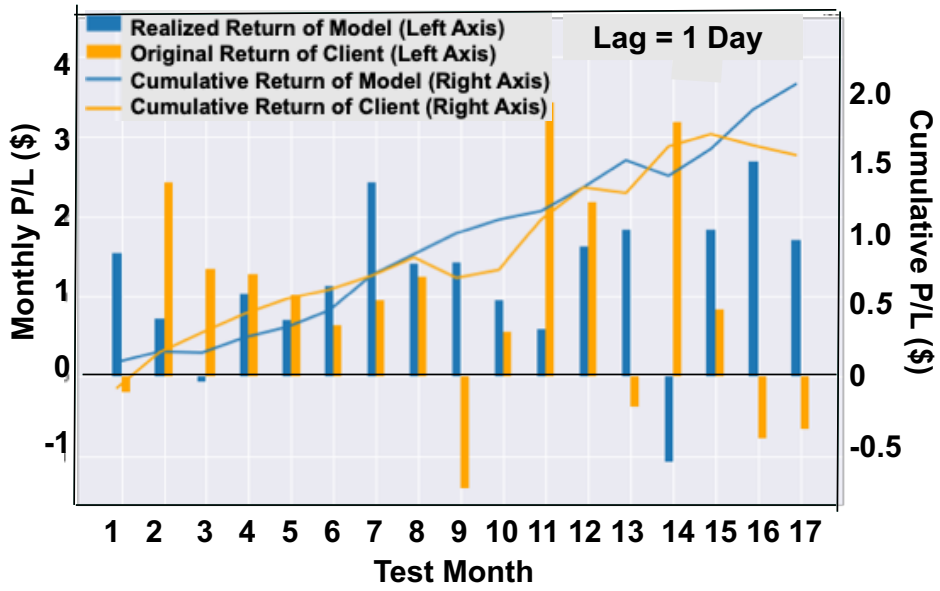


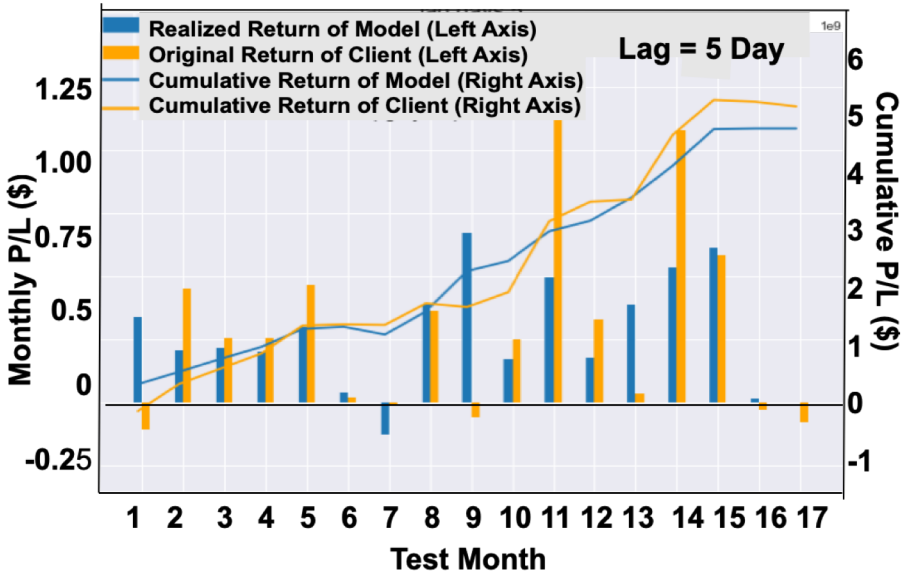


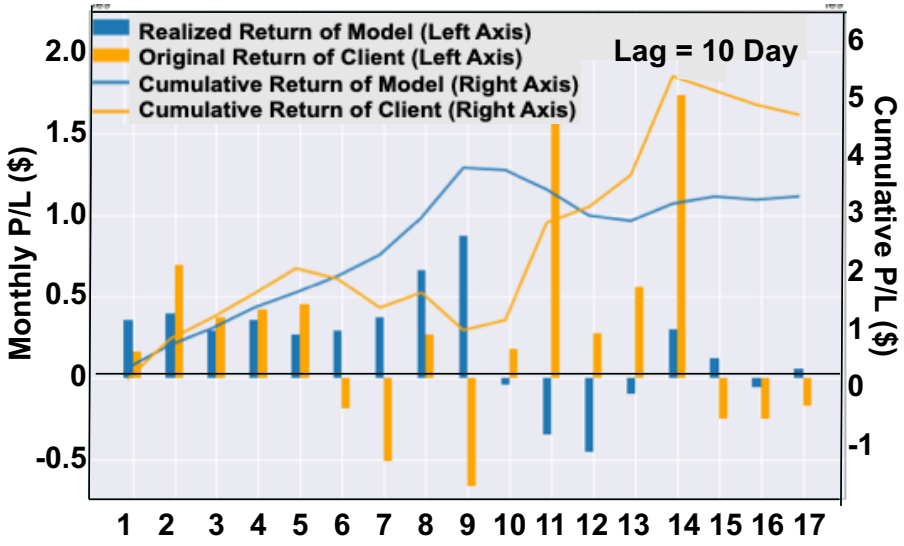


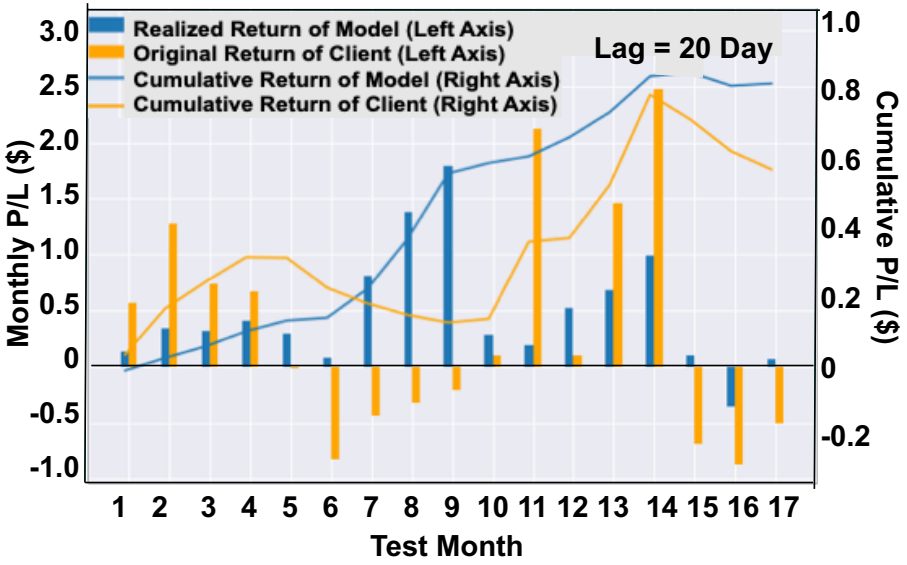
Analysis of Trade Portfolios – Sell Trades

On average, the model performs similar to clients for bond sell trades. This is true over all tenors.







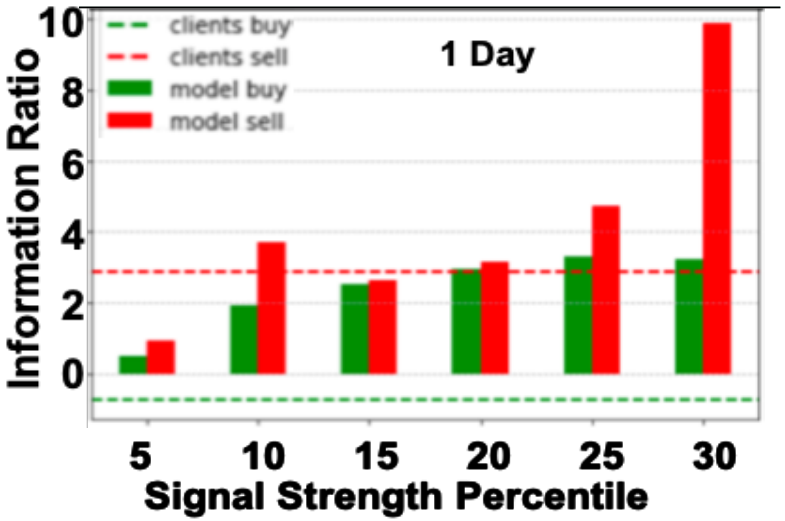


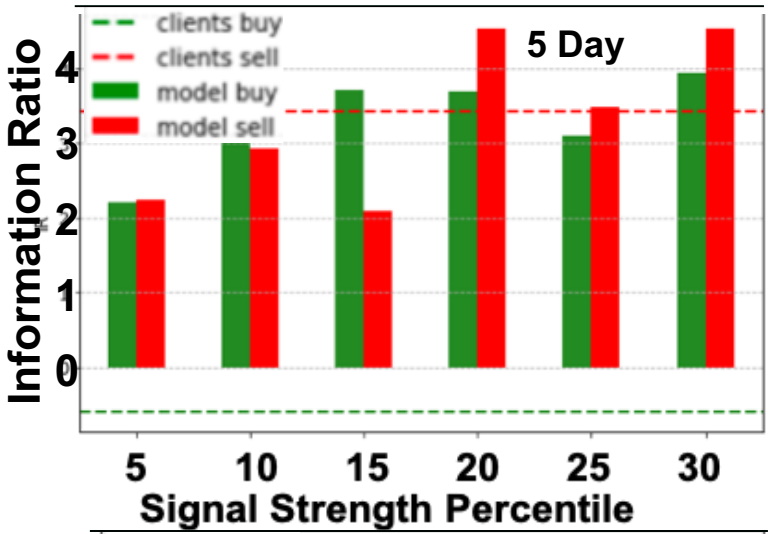
Analysis of Trade Portfolios – Summary

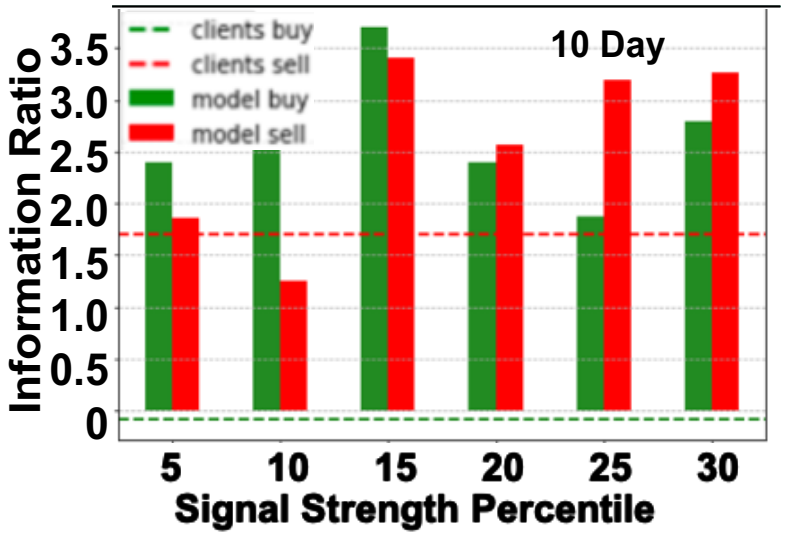
- For both buy and sell trades, the model produces steadily growing cumulative returns as represented by the blue lines in the Figures
- For buy orders, the model outperforms the benchmark portfolio on all time frames (1, 5, 10 and 20 days)
  - Credit may partially give to the fact that clients with buy orders did not perform very well
  - During the monitoring period, the accumulated return for client portfolio is almost stays below zero
- Although the model performs better for client "buy" trades, it does not do much better for "sell" trades
  - The client portfolio performance on sell orders is much better than for buy orders
  - The model may not always beat the clients (e.g., 5 and 10 day)
  - Still, the volatility of returns from the model is lower and this is reflected in better information ratios

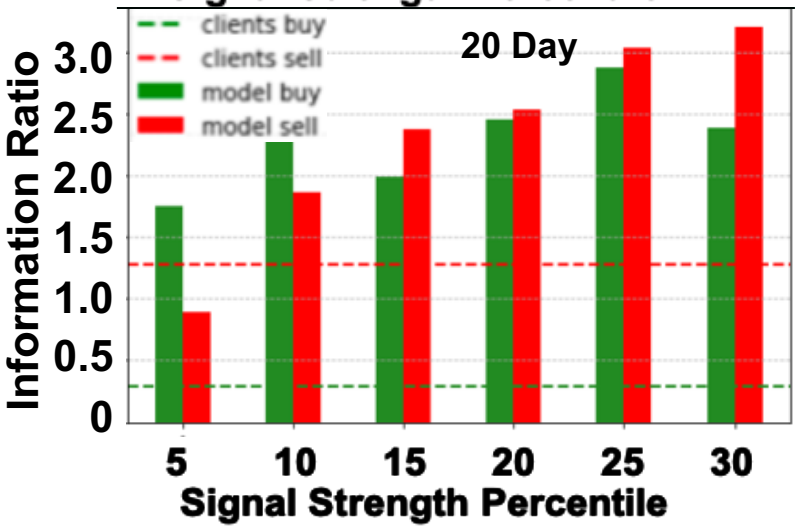
Analysis of Trade Portfolios – Information Ratios

- For buys and sells at most tenors results are significant
  - Results for 5% high signal strength are often not significant owing to the small number of cases in that bucket









Project Summary

- In this project, we trained XGBoost decision trees to differentiate profitable buy and sell trades over time periods of 1, 5, 10 and 20 day holding periods
  - We chose to analyze client trades as we were able to obtain the exact prices at which trades were executed
  - We trained the XGBoost model on rolling 3-month trades, with each model predicting bond spread moves relative to the market over the next month
  - We used data from client trades, bond indicative data, and client type to predict moves in bonds prices
- Applying XGBoost model and using client trade flow, the resulting portfolios are able significantly more profitable than imputed client portfolios and generates more stable and increasing cumulative returns

I thank Wenyu Chen, Xiaoyi Li, Jie Sheng, Zhuolu Xu for their important contributions to this project.

Machine Learning and Neural Networks in Finance

Deep Learning Models

What is Deep Learning?

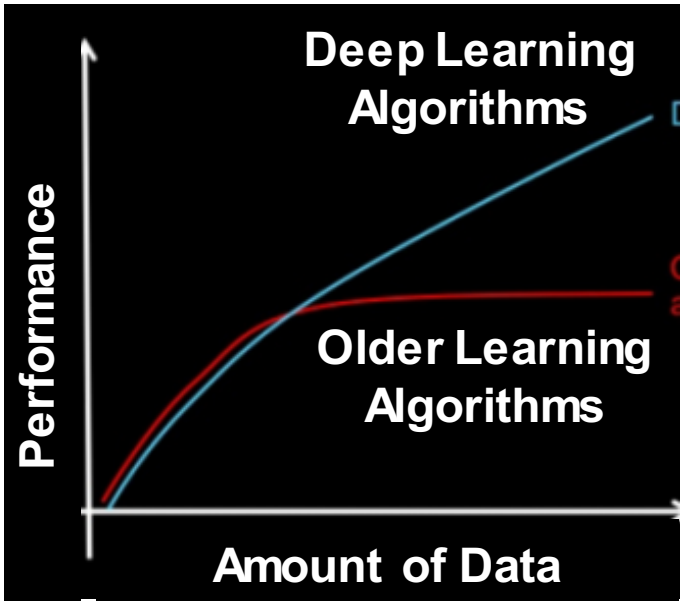
Deep learning is just very big neural networks on a lot more data, requiring bigger computers – J. Brownlee (2016)

- **Leaders and experts in the field have various ideas of what deep learning is and we consider some of these**
- **Some common aspects of their thoughts on deep learning are:**
  - **Deep Learning Involves Large Neural Networks**
  - **Deep Learning is Hierarchical Feature Learning**
  - **Why Call it "Deep Learning"?; Why Not Just Call it "Artificial Neural Networks"?**

Deep Learning is Large Neural Networks

- **Andrew Ng has described deep learning as:**
  - **" . . . for most flavors of the old generations of learning algorithms … performance will plateau. … deep learning … is the first class of algorithms … that is scalable. … performance just keeps getting better as you feed them more data**

Deep Learning



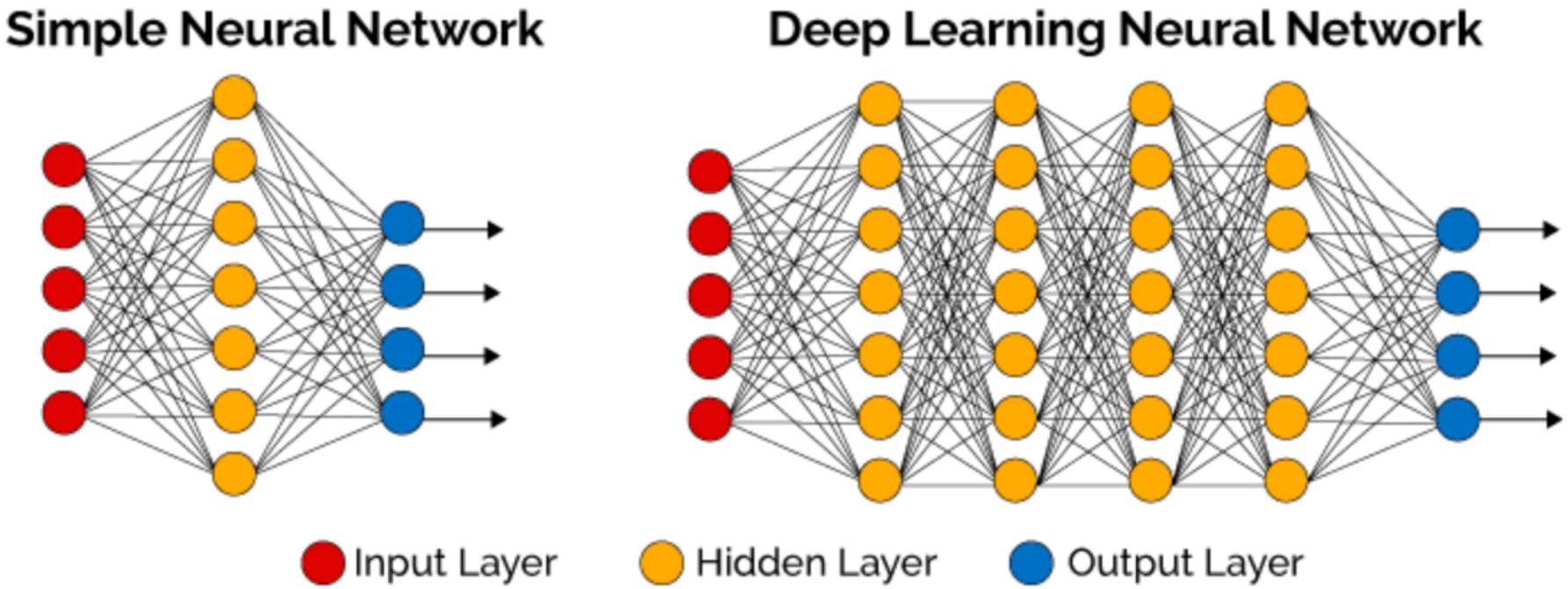
How do data science techniques scale with the amount of data?

Brownlee, J. *What is Deep Learning?* DEEP LEARNING, August 16, 2016

Deep Learning Neural Networks

Deep learning is just very big neural networks on a lot more data, requiring bigger computers – J. Brownlee (2016)

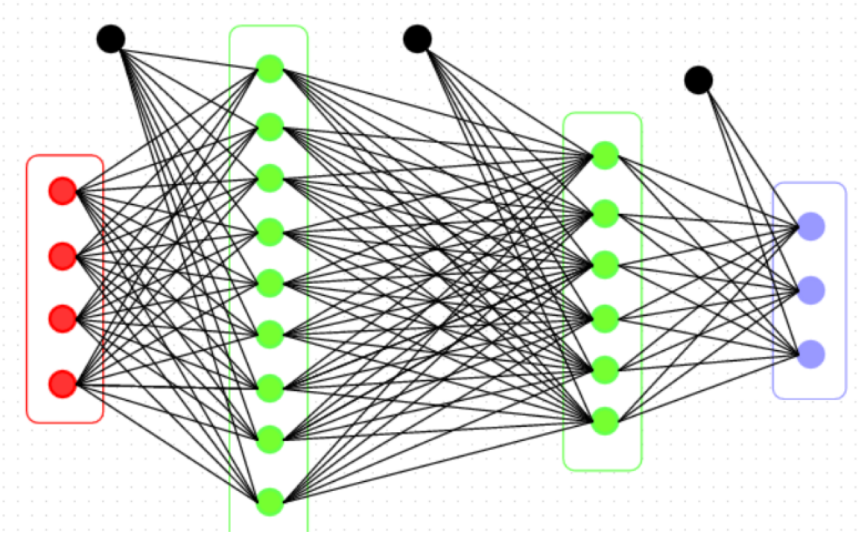
- **A deep neural network is a neural network with more than two layers**
  - **Deep neural networks use sophisticated mathematical modeling to process data in complex ways**
  - **It is the added complexity of deep learning neural networks that makes optimization and regularization particularly important**
  - **Most definitions include multiple layers of non-linear transformations**



Examples of Deep Learning Neural Networks

There are many types of deep learning neural networks. The most successful have taken place in the domain of image processing and speech recognition.

Classic Multi-Layer Network



Bi-Directional Recurrent Network

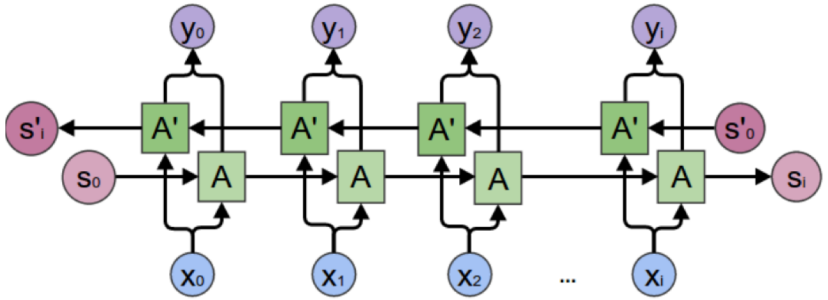
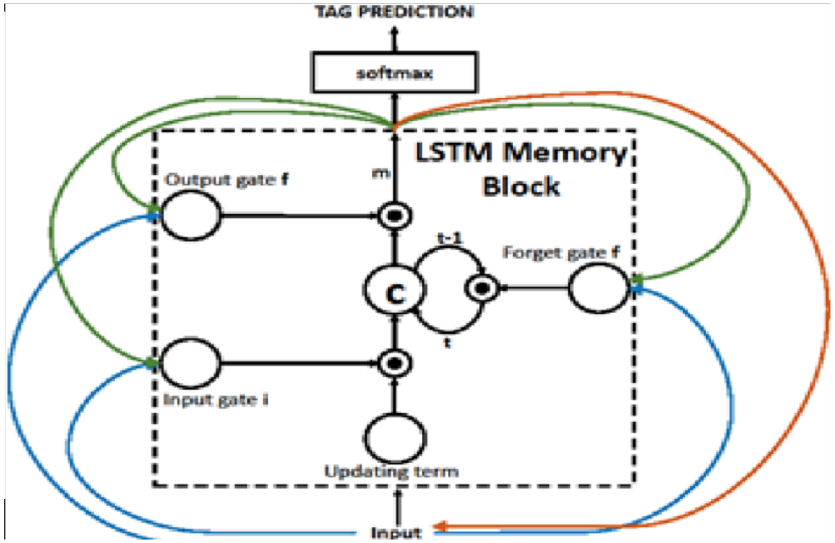
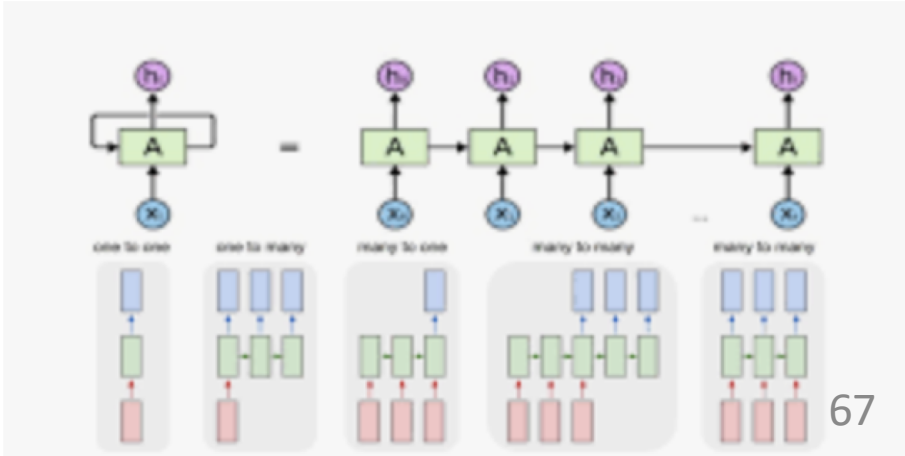


Image Classification Network

Convolutional & Recurrent Network



Recurrent Neural Network



What is Deep Learning?

Why Did Backpropagation (i.e. "Deep Learning") Not Take off in the 1990s?

- **Hinton says that people drew the wrong conclusions about why deep learning failed. The real reasons were**
  1. **Our labelled datasets were thousands of times too small**
  2. **Our computers were millions of times to slow**
  3. **We initialized weights in a stupid way**
  4. **We used the wrong type of non-linearity in the activation function**
- **Early "deep learning" approaches published by Hinton and collaborators focused on layerwise training and unsupervised methods like autoencoders**
- **Modern state-of-the-art deep learning is focused on training deep (many layered) neural network models using the backpropagation algorithm**
  - **Multilayer Perceptron Networks.**
  - **Convolutional Neural Networks**
  - **Long Short-Term Memory Recurrent Neural Networks**

An autoencoder learns to compress data from the input layer into a short code, and then uncompress that code into something that closely matches the original data.

Machine Learning and Neural Networks in Finance

Machine Learning Models of Corporate Bond Relative Value

Corporate Bond Relative Value

The objective of this project is to improve upon our current method for beating corporate bond indexes by adding variables and applying machine learning techniques.

OVERALL GOAL: Outperform the cut-and-rotate method at beating corporate bond benchmarks

- In 2004, we developed a strategy that consistently outperforms global corporate bond indexes and have been testing it out-of-sample since then
- The strategy takes as input bonds' model-based expected default probabilities and recovery values in default along with credit spreads and applies rules for portfolio construction based on those inputs
- We wanted to use additional explanatory variables in a non-linear model to more accurately determine fair spread and anticipate spread change/convergence
  - Use neural networks to model non-linear relationship
- Using those network-based relative value numbers, test the performance of the new relative value numbers in our cut-and-rotate strategy
  - Predict 1 month change in OAS directly

Bond Pricing – Yield Spreads to Treasuries

The yield spread to Treasuries is the market standard for quoting and evaluating the relative riskiness among different credits and/or maturities.

- **To isolate the price of credit risk, corporate bonds are typically quoted on a yield *spread-to-Treasury* basis**
  - **The credit risk of a bond is the yield spread over the yield of a Treasury bond of similar maturity**
  - **To compute the present value of a bond with maturity, T:**

for US Treasuries:

$$PV = \left[\sum_{t=1}^{2T-1} \frac{c/2}{\left(1+\frac{r_{0.5t}}{2}\right)^{t}}\right] + \frac{c/2+100}{\left(1+\frac{r_T}{2}\right)^{2T}}$$

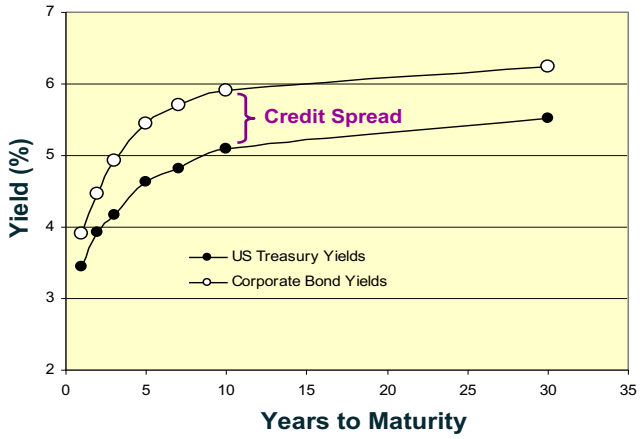
Credit Spread

for Corporates:

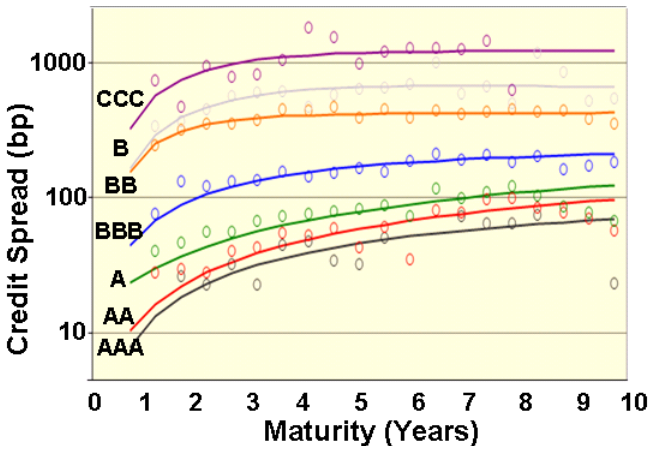
$$PV = \left[\sum_{t=1}^{2T-1} \frac{c/2}{\left(1+\frac{r_{0.5t}+s}{2}\right)^{t}}\right] + \frac{c/2+100}{\left(1+\frac{r_T+s}{2}\right)^{2T}}$$

Where *PV* is the price of bond with coupons (c_t), r_t is the term structure of US Treasury spot yields at 0.5 year intervals, and *s* is the yield spread of the credit curve to US Treasuries. Spread is often *Basis Points* where 1bp is 1/100 of 1%

Yield Curves for US Treasuries and for Single-A Corporate Bonds



Yield Spread by Agency Rating



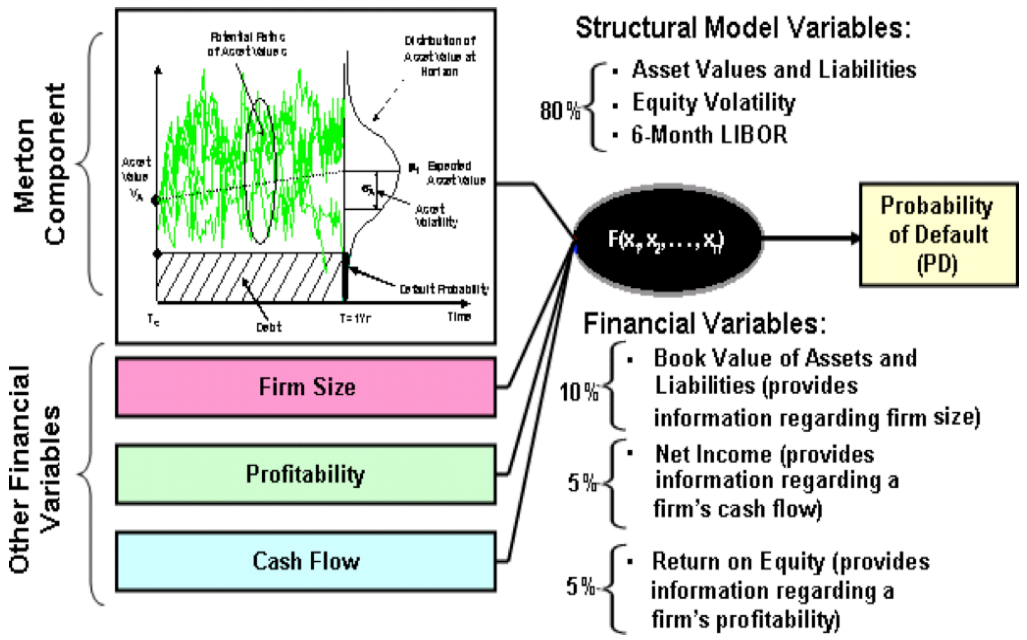
Yield spreads to Treasuries increase with maturity and decreasing credit quality.

Calculating Bond Relative Value - Default Risk

The success of our current "cut-and-rotate" strategy depends on having accurate estimates of bonds' expected probabilities of default and recovery values. We consider first our model for predicting bond defaults; Citi's Hybrid Probability of Default (HPD) model.

- **We use the hybrid probability of default (HPD) model to estimate firms' probabilities of default**
  - **The model is called a "hybrid" because it combines a "Merton-type" structural model with statistical variables on firms' size, profitability and cash flow**
  - **This model is the best we know of commercial and industrial firms**

Hybrid Probability of Default Model



- **We use the Merton model framework because idiosyncratic risk in the equity market appears to lead the bond market**
  - **The equity market is larger, has more strategist coverage and is more liquid**
  - **It is cheapest to put on a view of credit in the equity market**

Bond Relative Value – Recovery Value

Expected losses on corporate bonds depend on both the likelihood of default and recovery value in default. We use a decision-tree model of recovery value in default.

- **The decision-tree model embeds known determinants of recovery value in default**
  - **Credit cycle, seniority, industry sector, credit quality and geography**

The tree begins by assigning a recovery rate of 40% for all securities. If one has no other information, the tree will output 40%

The first decision point is the adjustment for credit cycle dependency. The next step in the decision tree concerns seniority in the capital structure

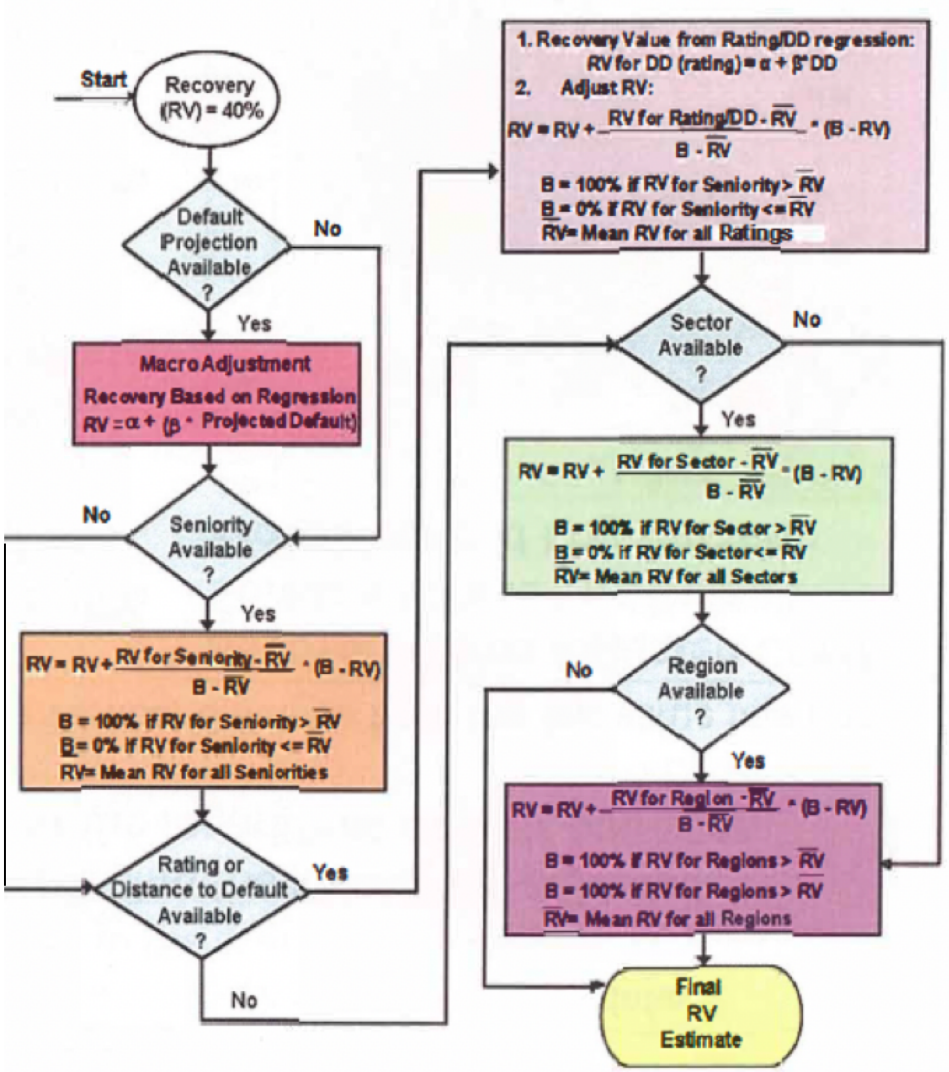
Following the hierarchy shown in the figure, we assign the firm's seniority to one of the following six categories

The next adjustment in the model is for credit quality just prior to default. If nothing is input at this stage, the analysis advances to the sector adjustment stage

The recovery value is then adjusted for industry sector

The final adjustment is for geographical region as different regions have different bankruptcy regimes, legal procedures and precedents

Model for Recovery Value in Default



Corporate Bond Relative Value

We calculate bonds' relative values as the amount of credit spread for their default probability, recovery value, and duration relative to the average of all bonds.

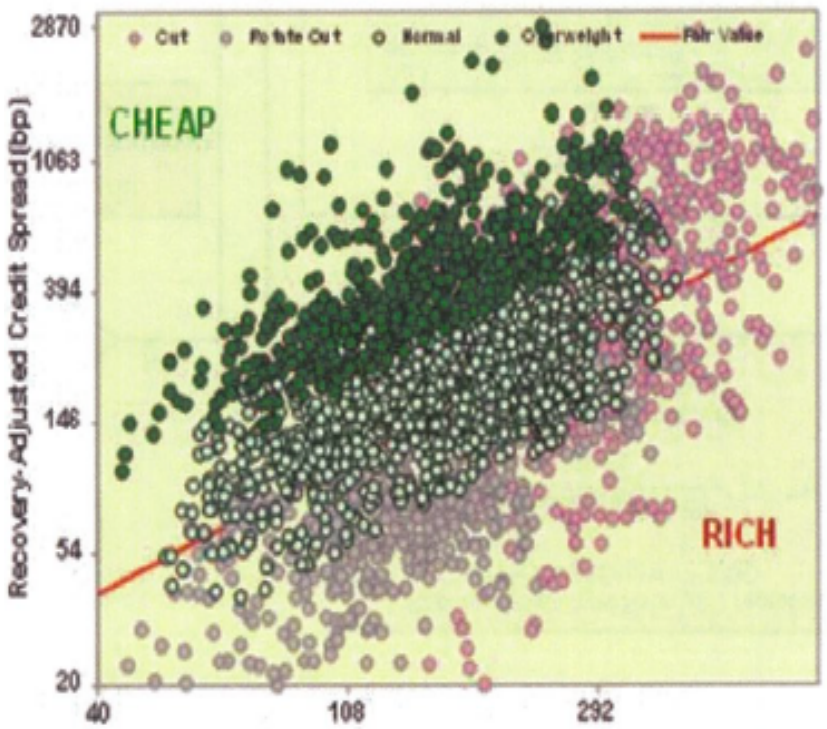
- **We calculate bonds' relative values by plotting their recovery-adjusted spreads to Treasuries versus the logarithms of their one-year default probabilities and durations**
  - **Relative value is z-score vertical distance from the fair value line (the red line in the figure)**
- **The recovery-adjusted spread puts all bonds on a 40% recovery value basis and is calculated as:**

$$s_i^{c=40} = -\frac{1}{duration} * \ln\left[1 - \frac{1-e^{-(spread*duration)}}{1-recovery\ rate}\right]$$

- **Thus, our model for yield spreads to Treasuries for bond *i* is:**

$$\log\left(s_i^{c=40}\right) = \gamma_0 + [\gamma_1 * log(PD_i)] + [\gamma_2 * log(Dur_i)] + \varepsilon_i$$

Credit Spreads vs Log Default Probability and Log Duration



$$\log(s_i^{c=40}) = \gamma_0 + [\gamma_1 * \log(PD_i)] + [\gamma_2 * \log(Dur_i)] + e_i$$

The pink circles are the 10% riskiest bonds, the gray are 10% riches and dark read are 10% cheapest

- **The relative value measure adjusts for effects of duration and recovery value on spreads**

Predicting 1-Month OAS Changes

We decided to test if we could predict one-month changes in option-adjusted credit spreads.

- **We used our relative value measure and other variables as input to regression and neural network models to predict one-month changes in bond spreads to US Treasuries**
- **In addition to bonds' relative values, we added the variables:**
  - **OAS Momentum (1M an 3M)**
  - **Relative Value Momentum (1M and 3M)**
  - **Spread-Times-Duration Momentum (1M and 3M)**
  - **Sector Relative Value Momentum (1M and 3M)**

Input Variables for 1-Month OAS Change Models

Features	Type	Description
OAS Momentum - 1M	Percentile (0 ---> 1.0)	OAS (to Treasury) Momentum - 1M
OAS Momentum - 3M	Binary (0-1)	OAS (to Treasury) Momentum - 3M
Relative Value	Percentile (0 ---> 1.0)	Bond Relative Value
Relative Value Momentum - 1M	Percentile (0 ---> 1.0)	Bond Relative Value Momentum - 1M
Relative Value Momentum - 3M	Percentile (0 ---> 1.0)	Bond Relative Value Momentum - 3M
Spread-Times-Duration Momentum - 1M	Percentile (0 ---> 1.0)	Spread-Times-Duration Momentum - 1M
Spread-Times-Duration Momentum - 3M	Percentile (0 ---> 1.0)	Spread-Times-Duration Momentum - 3M
Sector Relative Value Momentum - 1M	Percentile (0 ---> 1.0)	Bond Industry Sector Momentum - 1M
Sector Relative Value Momentum - 3M	Percentile (0 ---> 1.0)	Bond Industry Sector Momentum - 3M

Input Variable Definitions

Because several of the variables are normalized with respect to changes in the market, they are defined explicitly below.

- **OAS Momentum N-Month:**

$$\log(OAS2TSY_i / OAS2TSY_{i-n}) - avg(\log(OAS2TSY_i / OAS2TSY_{i-n}))$$

where $OAS2TSY_i$ is OAS yield - Treasury yield for month i

- **Relative Value Momentum N-Month:**

$$\log(RV_i / RV_{i-n}) - avg(\log(RV_i / RV_{i-n}))$$

where RV_i is the bond relative value for month i

- **Spread Duration Momentum N-Month:**

$$\log(STD_i / STD_{i-n}) - avg(\log(STD_i / STD_{i-n}))$$

where STD_i is Spread times Duration for month i

- **Sector Relative Value Momentum N-Month:**

$$\log(RV_i / RV_{i-n}) - sctavg(\log(RV_i / RV_{i-n}))$$

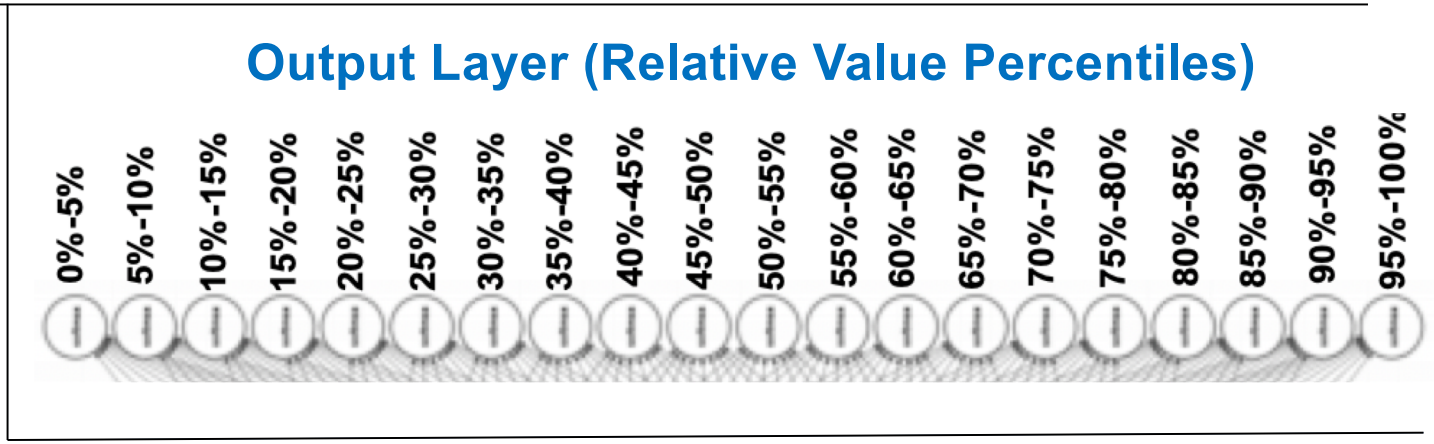
where RV_i is the bond relative value for month i

and sctavg takes the average of the corresponding month and sector

Network Architecture – The Output Layer

We chose a 20-bin output layer, scaled in units of 5% of the ranked population of one-month spread changes and applied the Softmax function to normalize the distribution.

- **We rank all the relative value numbers in the training sample and convert them to percentiles as the dependent variable for each bond**



- **Then for each training case output, we apply the Softmax function which takes an un-normalized vector of density across the 20 bins and normalizes it into a probability distribution**
- **The standard (unit) Softmax function is given by the standard exponential function on each coordinate, divided by the sum of the exponential function applied to each coordinate**
  - **The sum of the exponential function acts as a normalizing constant, so the output coordinates sum to 1:**

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Choosing the Loss Function and Network Architecture

We chose the categorical cross entropy measure as the error function and used it as a criterion for choosing the network architecture.

- **We used categorical cross entropy as the error function**

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{1}_{y_i \in C_c}\log p_{model}[y_i \in C_c]$$

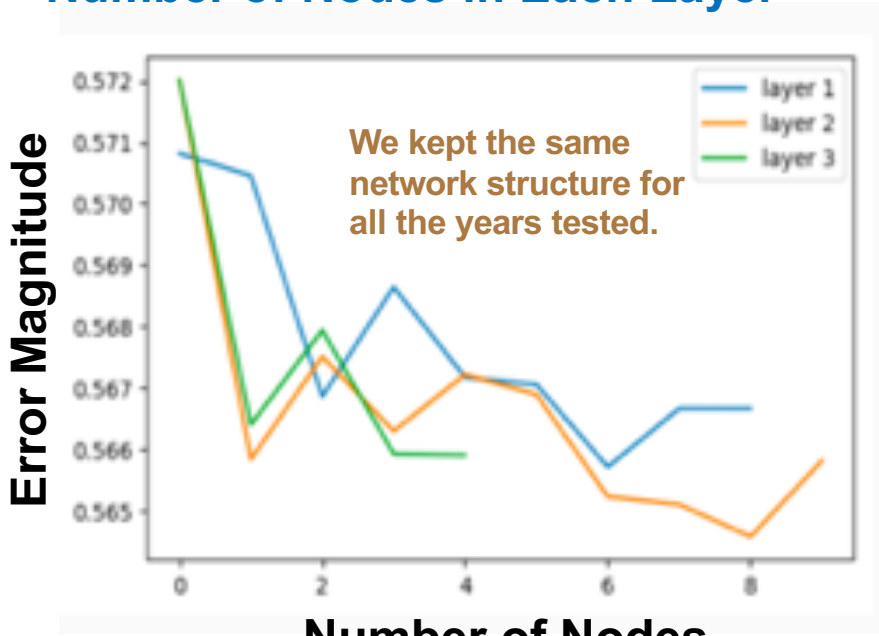
- **The double sum is over the observations *i*, whose number is *N*, and the categories *c*, whose number is *C***
- **The term $[y_i \in C_c]$ is the indicator function of the i^{th} observation belonging to the c^{th} category. The $p_{model}[y_i \in C_c]$ is the model probability for the i^{th} observation to belong to the c^{th} category.**
- **The network outputs a vector of *C* probabilities, each giving the probability that the network input should be classified as belonging to the respective category**
- **We used the cross entropy error function as a measure for deciding on the structure of the network (see next slide)**

Neural Network and Regression Model

We trained both a regression model and neural network model using the same input variables using the same walk-forward procedure used previously.

- The regression model was an ordinary least squares regression
- The overall approach to the neural network architecture was similar to that for the relative value network. That is,
  - We used a standard Softmax function with 20 OAS bins as the output layer of the network
  - We used categorical cross entropy as the error function
- As before, to determine the optimal neural network architecture, we began with a single node in a single hidden layer, and add nodes and layers until performance fails to improve
  - For example, the chart on the right shows that a two-layer network is preferable to a single layer
  - We decided on a network with 2 hidden layers with 7 and 9 nodes, respectively

Categorical Cross Entropy Error vs Number of Nodes in Each Layer

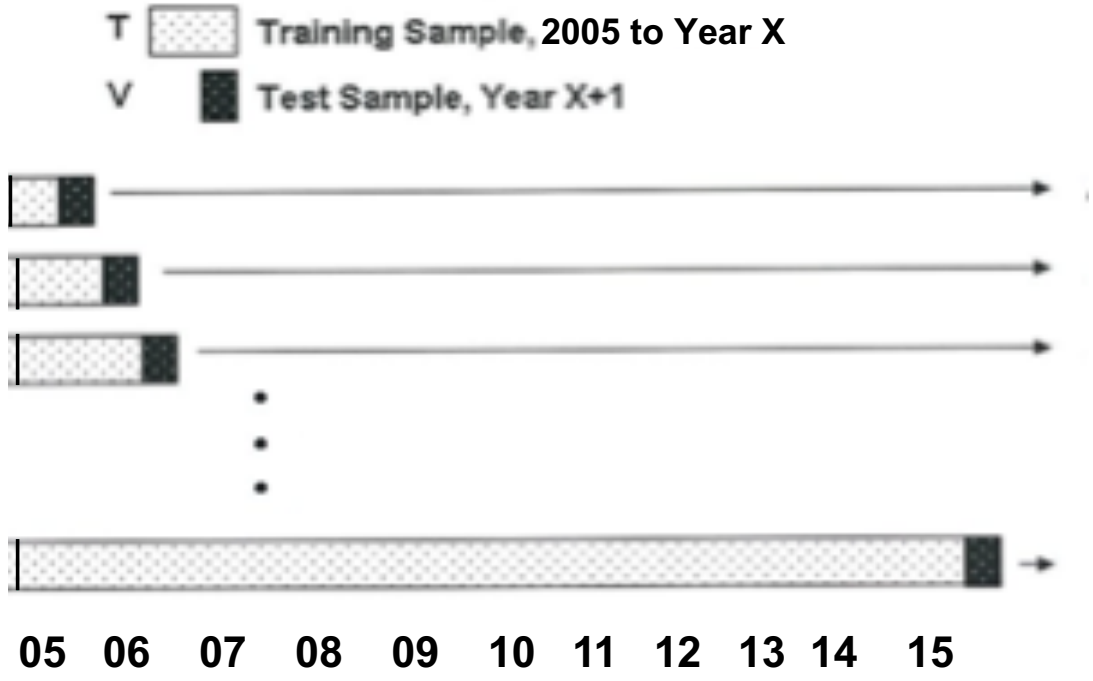


Training the Network – The Walk Forward Procedure

We used a walk-forward procedure to train each network, each year adding the data from the previous year.

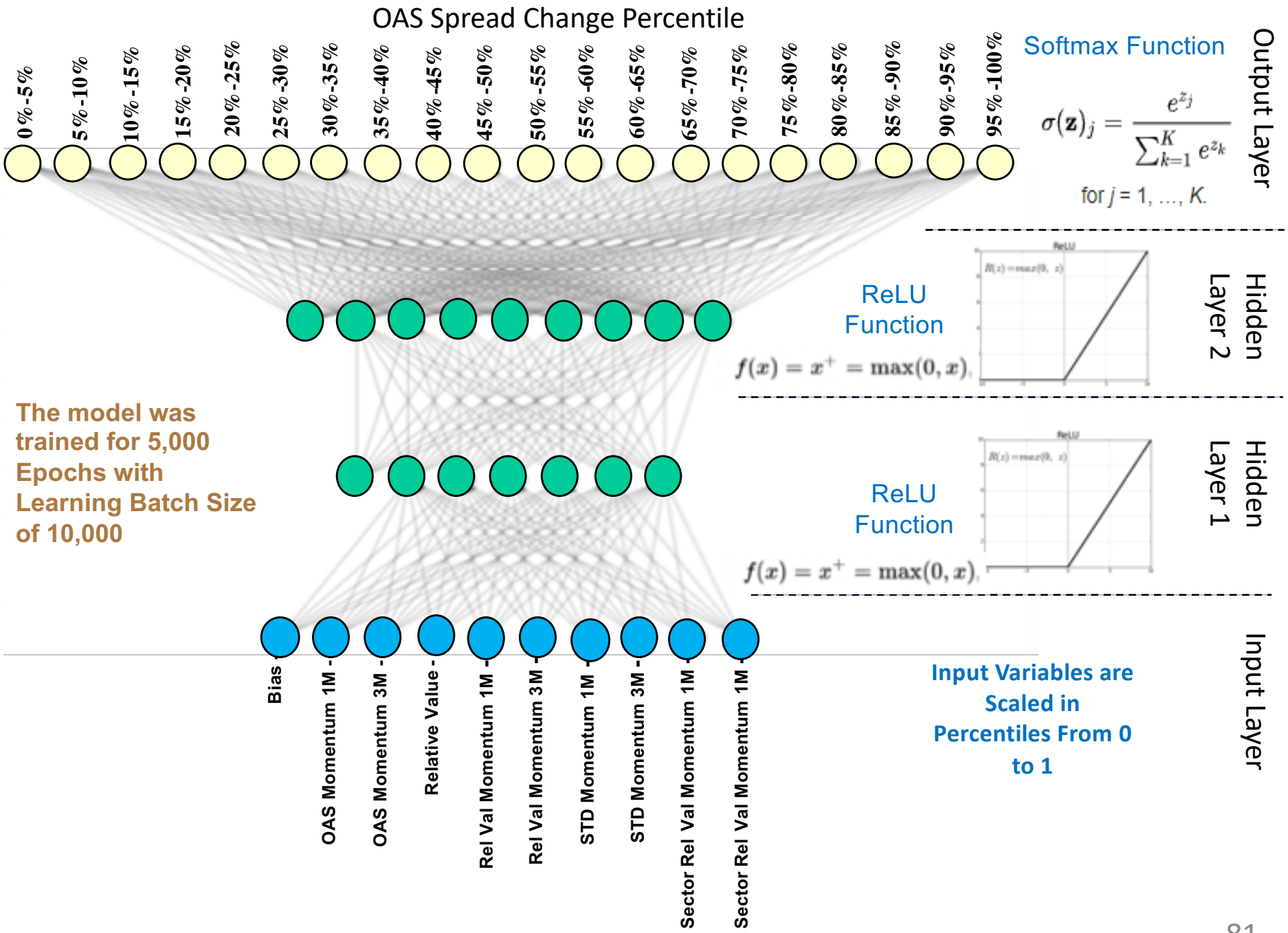
- We used a ”walk forward” procedure to train a series of annual neural network models
  - For example, the chart on the right shows that the first network model was trained only on the data from 2005
  - That model was used to generate relative value numbers for 2006
- Then data from 2006 were added to the training sample of 2005 and used to train the model to test on data from 2007

Illustration of Walk-Forward Network Training Procedure



- This process continued until the final model in 2015 which trained on data from 2005 through 2015 was used to generate predictions for 2016
  - Thus, each successive annual model was trained on an increasing amount of data

Corporate Bond 1 Month OAS Change Network



One Month OAS Change Network Performance

Both the regression and neural net models of one-month OAS changes were designed to select bonds on relative OAS change only.

- **The 1-Month OAS change models predicted spreads directly, so did not go through the cut-and-rotate paradigm.**
- **The OLS and neural net models had information ratios of 1.3 and 1.7, respectively, mainly by reducing the volatility of returns**
  - **These are far superior to the 0.6 and 1.1 information ratios of the benchmark and neural network cut-and-rotate strategies**

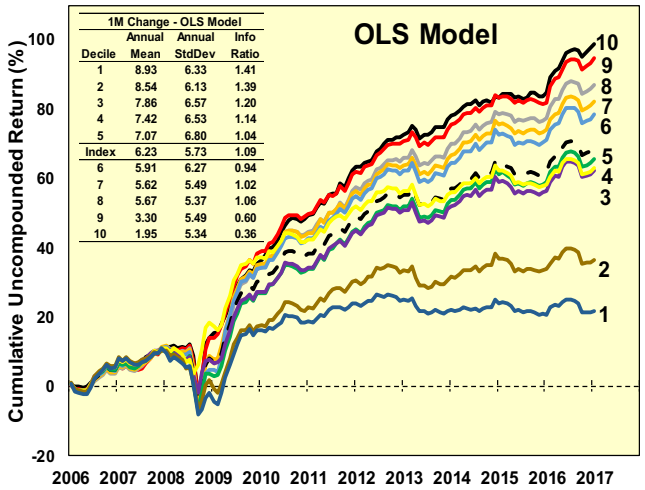
Annual Returns and Summary Statistics from Relative Value and 1-Month OAS Models

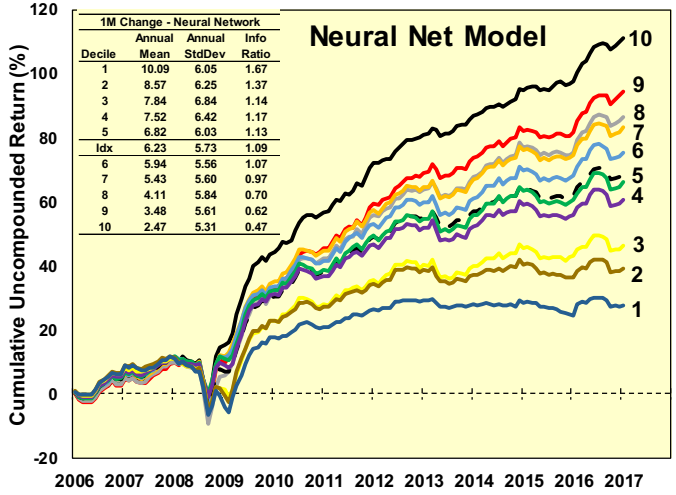
	OAS		Relative Value Models						1 Month Change in OAS					
	Average	Annual	Benchmark			Neural Network			OLS			Neural Network		
Year	Spread	Change	Cut	Rotate	CnR	Cut	Rotate	CnR	Cut	Rotate	CnR	Cut	Rotate	CnR
2006	95	-5	4	30	38	-4	21	14	-5	-19	-22	-5	-16	-40
2007	196	111	-4	-15	-17	13	-18	4	11	-4	13	11	7	28
2008	676	477	56	-10	8	134	-92	62	134	180	291	134	63	252
2009	212	-403	-71	614	553	-188	386	147	-190	18	-126	-190	135	-39
2010	171	-18	-6	129	122	-4	79	73	-3	98	91	-3	88	96
2011	237	65	18	30	46	21	-2	20	19	76	96	19	83	124
2012	152	-85	5	121	137	-6	103	98	-8	94	93	-8	92	94
2013	131	-29	-11	27	20	-16	47	31	-15	83	62	-15	67	39
2014	126	12	7	72	93	15	81	104	15	57	71	15	73	82
2015	165	46	47	-15	39	63	-6	64	66	56	129	66	78	127
2016	136	-42	-64	116	35	-82	55	-27	-80	117	35	-80	107	22
Sum	2297	129	-19	1099	1074	-54	654	590	-56	756	733	-56	777	785
Mean	209	12	-2	100	98	-5	59	54	-5	69	67	-5	71	71
Std Dev	153	195	37	171	151	77	116	49	77	54	98	77	41	80
Ratio			0.0	0.6	0.6	-0.1	0.5	1.1	-0.1	1.3	0.7	-0.1	1.7	0.9

1-Month OAS Change Model Performance (cont.)

We also analyzed models' performance using monthly returns by relative value decile.

- We analyzed monthly returns by decile for the OLS and neural network 1-month change models
- Both models perform well at ranking absolute and risk adjusted returns by decile
- Returns from decile 10 versus decile 1 are 50bp per annum greater for the neural network model





1M Change - OLS Model			
Decile	Annual Mean	Annual StdDev	Info Ratio
10	8.93	6.33	1.41
9	8.54	6.13	1.39
8	7.86	6.57	1.20
7	7.42	6.53	1.14
6	7.07	6.80	1.04
Index	6.23	5.73	1.09
5	5.91	6.27	0.94
4	5.62	5.49	1.02
3	5.67	5.37	1.06
2	3.30	5.49	0.60
1	1.95	5.34	0.36

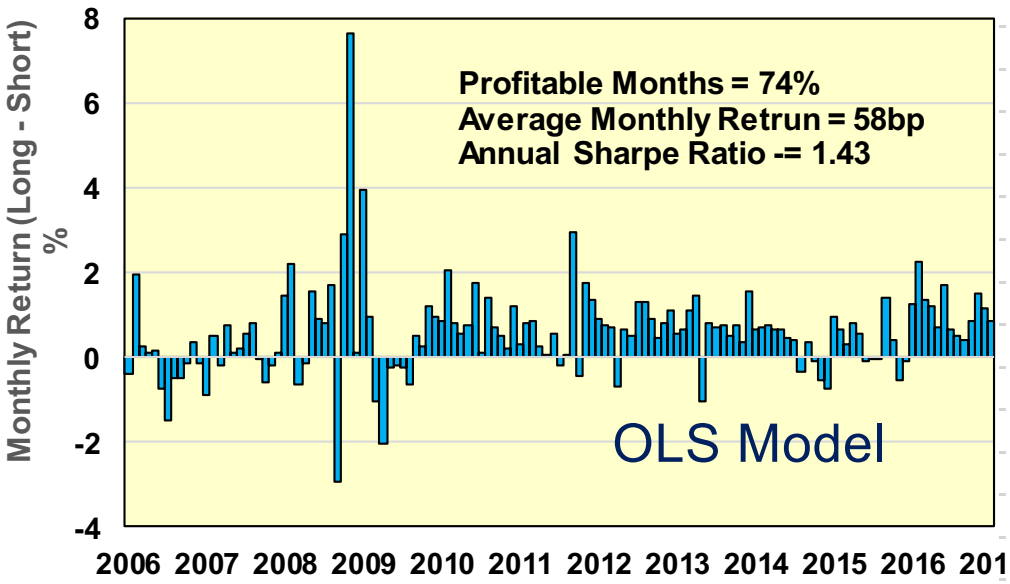
1M Change - Neural Network			
Decile	Annual Mean	Annual StdDev	Info Ratio
10	10.09	6.05	1.67
9	8.57	6.25	1.37
8	7.84	6.84	1.14
7	7.52	6.42	1.17
6	6.82	6.03	1.13
Index	6.23	5.73	1.09
5	5.94	5.56	1.07
4	5.43	5.60	0.97
3	4.11	5.84	0.70
2	3.48	5.61	0.62
1	2.47	5.31	0.47

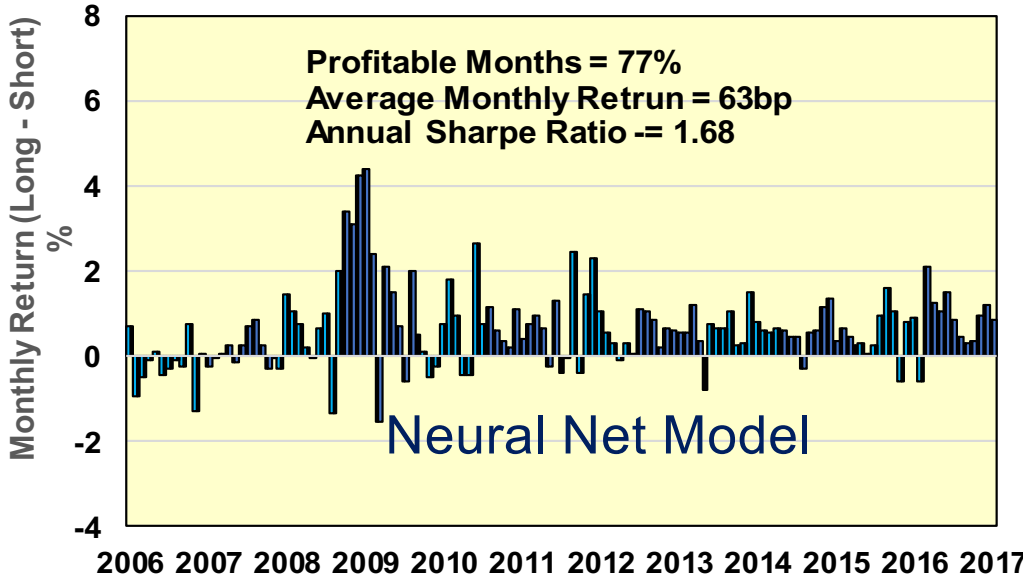
1-Month OAS Change Model Performance (cont.)

We analyzed profitability from each model of going long the bonds in decile 10 and short the bonds in decile 1

- **Both OLS and Neural Net models perform well in decile 10 versus decile 1 long/short trades**
  - **The OLS model has 74% profitable months with an average return of 56bp (6.72% per annum) and an information ratio of 1.4**
  - **The neural network was profitable 77% of months with an average return of 63bp (7.56% per annum) and an information ratio of 1.7**
- **The Neural Network model performs best**

Monthly Returns and Summary Statistics from 1-Month OAS Models



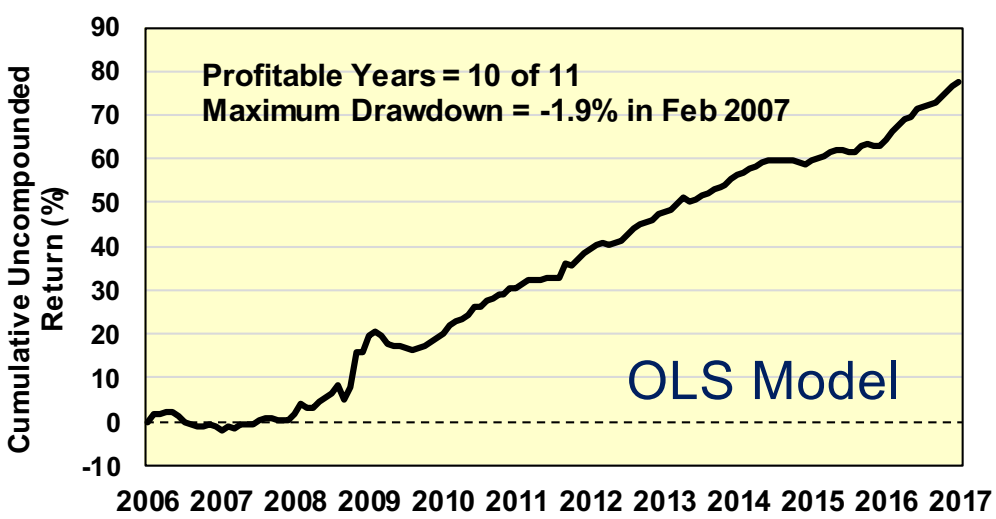


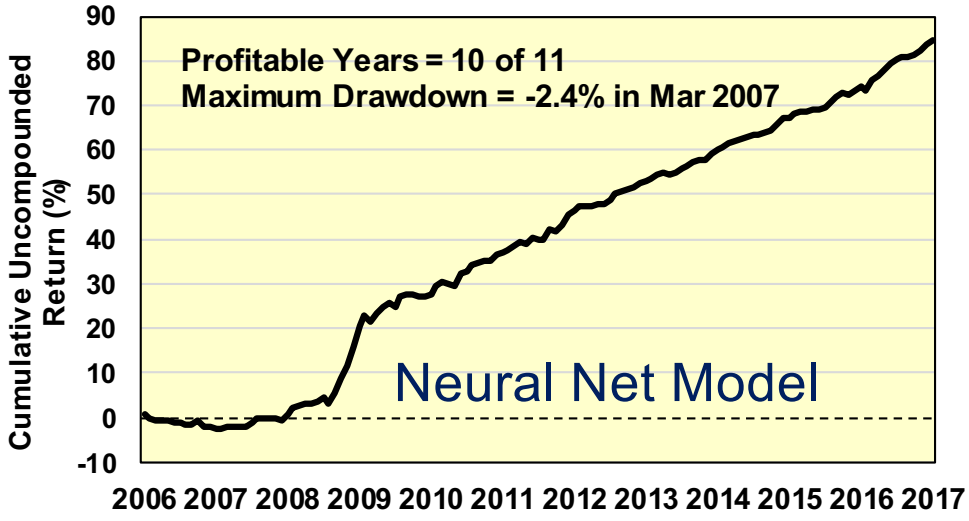
1-Month OAS Change Model Performance (cont.)

We analyzed cumulative profitability from each model of going long the bonds in decile 10 and short the bonds in decile 1

- **We analyzed cumulative uncompounded returns from the OLS and Neural Network Models**
- **Models perform similarly as regards profitable years and drawdowns, but the Neural Network had higher overall returns**
  - **Both are profitable 10 of 11 years and drawdowns are -1.9% and -2.4%, but the neural network has higher absolute returns**
- **Both models have steady returns after the credit crisis of 2008-2009**
  - **Recall the because of the "walk-forward" procedure, the samples for the models increase as time goes on**

Monthly Returns and Summary Statistics from 1-Month OAS Models





Analysis of Variable Contributions – 1M OAS Change

For the 1-month OAS change network, there is greater dispersion of variable importance rankings among analysis methods.

- **Relative value momentum is important in variable exclusion and univariate analyses**
  - **However, they are relatively unimportant using Garson's method**
  - **Garson's method ranks relative value and 1-month OAS momentum as most important**

Importance of Variables in OAS Change Neural Network

Input Variable	Variable Exclusion Method	Univariate Relationship	Garson's Method
RelValMom_3M	0.18	0.27	0.05
RelValMom_1M	0.13	0.20	0.03
OASMom_3M	0.10	0.17	0.06
RelVal	0.18	0.13	0.14
OASMom_1M	0.05	0.12	0.21
STD Mom_3M	0.06	0.05	0.08
STD Mom_1M	0.06	0.05	0.05
SecRelValMom_1M	0.12	0.01	0.33
SecRelValMom_3M	0.12	0.01	0.04

- **Sector relative value momentum at 1M and 3M is important in variable exclusion and Garson's method, but have only a weak univariate relationship to OAS changes**

Variable Contributions - Derivative Analysis

The analysis of derivatives confirms the importance of relative value momentum in the neural network model, but relative value and STD momentum are important

- As for the exclusion and univariate methods, the derivative method assigns greatest importance to 1-month relative value momentum
  - 3-Month relative value momentum is also important
- Consistent with Garson's method, relative value is tied with relative value momentum as the third most important variable
- Average values of sector relative value momentum and OAS momentum indicate little directional bias in the effects of those variables.

Summary Statistics from Derivative Analysis

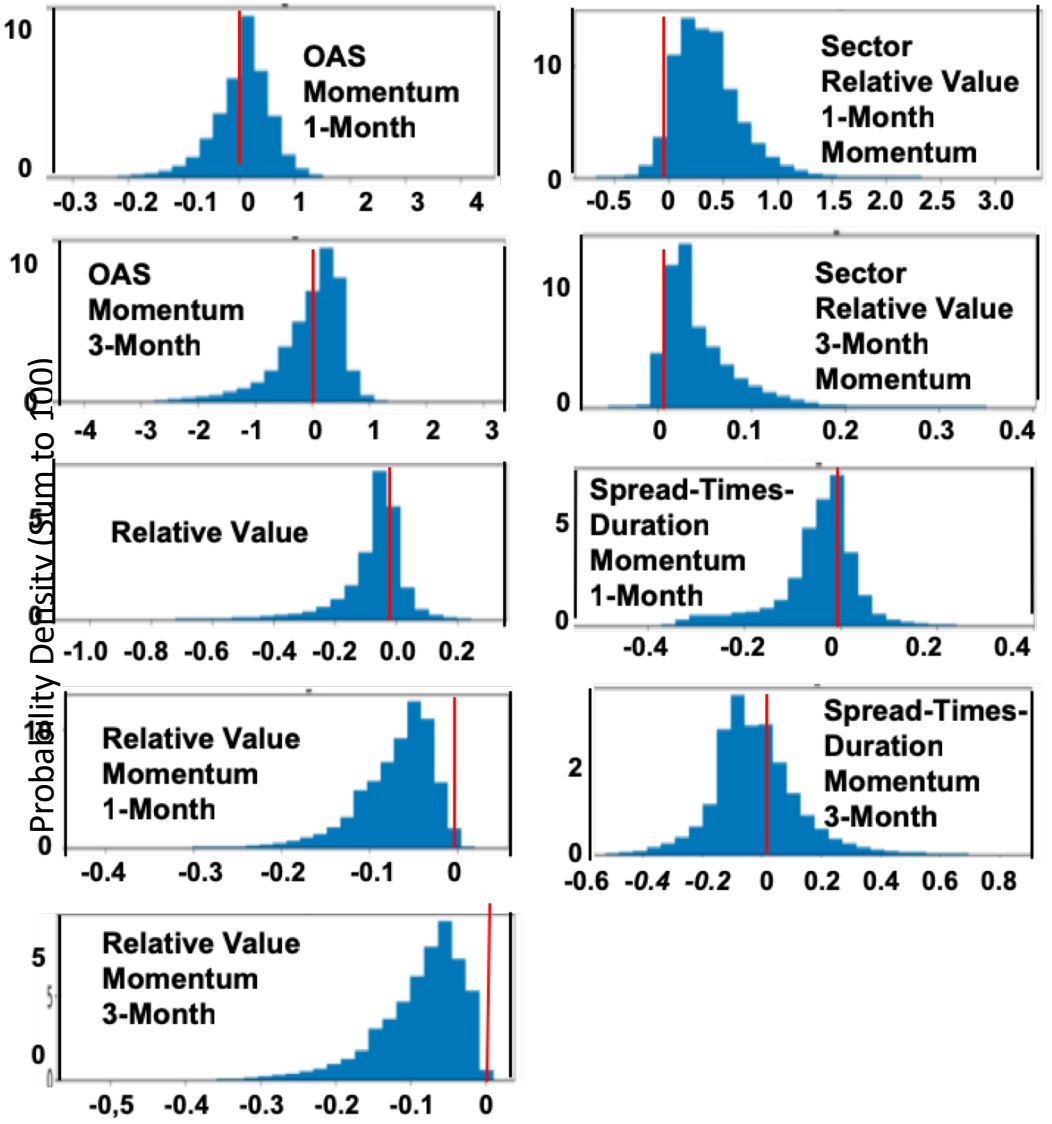
Input Variable	Mean	Std Deviation	Skew	Kurtosis
OASMom_1M	0.03	0.05	0.71	3.58
OASMom_3M	0.00	0.05	-0.75	3.91
SecRelValMom_1M	0.02	0.02	1.27	3.52
SecRelValMom_3M	0.02	0.05	0.32	7.59
RelVal	-0.09	0.07	-1.58	5.24
STD Mom_1M	-0.07	0.15	-0.72	3.49
STD Mom_3M	0.01	0.12	1.86	6.87
RelValMom_3M	-0.06	0.04	-1.71	5.26
RelValMom_1M	-0.09	0.06	-1.69	5.22

Variable Contributions – Distribution of Derivatives

The derivative method provides information regarding the strength of each variable, but the direction and consistency of its contribution.

- **The distributions of derivatives show that variables whose mean derivatives are close to zero can have large influences on network responses**
  - **For example, OAS momentum (1 and 3 month) derivatives have broad influences, but in both directions**
- **Increases in relative value and relative value momentum consistently lead the model to predict tighter spreads**

Distribution of Derivatives for Input Variables



Machine Learning and Neural Networks in Finance

Predicting Market Moves from News Headlines

Project Objectives

The objective of this project was to generate sentiment scores from news headlines and use those scores to predict credit spread moves.

- **In this project, we focused on using Natural Language Processing (NLP) techniques to build trading strategies for 1-day horizons for the credit market using news headlines.**
- **Using data scraped from multiple financial sources, we employ machine learning approaches of varying complexities.**
- **We find that pure sentiment prediction does not require models of very high complexity, but the link between sentiments and predictability of returns is not straight-forward.**
- **We also find that approaches using the latest advances in NLP are better suited to predict future returns in credit indices, by using news headlines directly as inputs, instead of news headline sentiments**

Introduction and Background

The objective of this project was to generate sentiment scores from news headlines and use those scores to predict credit spread moves.

Natural Language Processing (NLP)

- **Natural language processing (NLP) is a branch of artificial intelligence that is being used more and more for both business and financial applications**
  - **Financial institutions, on both the buy- and sell-sides, are adopting the technology for tasks like robo-advisories, credit checks, employee surveillance, and investment strategies**
  - **Financial literature on NLP has focused on metrics related to corporate governance, competitive dynamics, management quality, etc. that would be useful for longer-term investment signals in equity markets**
- **Traditional approaches have generally focused on conventional NLP methods like bag of words, TF-IDF scores to rank firms based on the frequency of occurrences of pre-defined "relevant" words in the firms' 10-Ks, analyst reports, earnings call transcripts or news**
  - **There has also been progress in parsing through high-frequency information sources like news, employing the information gleaned in high frequency trading**

Bag of Words

- **Bag of Words (BoW) is an algorithm that counts how many times a word appears in a document**
  - **Those word counts allow us to compare documents and gauge their similarities for applications like search and document classification**
- **BoW lists words paired with their word counts per document**
  - **In the table where the words and documents that effectively become vectors are stored, each row is a word, each column is a document, and each cell is a word count**
  - **Each of the documents in the corpus is represented by columns of equal length**
  - **Those are wordcount vectors, an output stripped of context**

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

- **Before they're fed to the neural network, each vector of wordcounts is normalized such that all elements of the vector add up to one**
  - **Thus, the frequency of each word is effectively converted to represent the probabilities of those words' occurrence in the document**
  - **Probabilities that surpass certain levels will activate nodes in the network and influence the document's classification**

Term Frequency-Inverse Document Frequency (TF-IDF)

Term-frequency-inverse document frequency (TF-IDF) is another way to judge the topic of an article by the words it contains.

- **With TF-IDF, words are given weight – TF-IDF measures relevance, not frequency**
  - **Wordcounts are replaced with TF-IDF scores across the whole dataset**
- **TF-IDF measures the number of times that words appear in a given document (that's "term frequency").**
  - **Because words such as "and" or "the" appear frequently in all documents, those must be discounted**
  - **That's the inverse-document frequency part. The more documents a word appears in, the less valuable that word is as a signal to differentiate any given document**
  - **That's intended to leave only the frequent and distinctive words as markers. Each word's TF-IDF relevance is a normalized data format that also adds up to one.**

$$w_{i,j}=tf_{i,j} * log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

  - **Those marker words are then fed to the neural net as features in order to determine the topic covered by the document that contains them**

Introduction and Background (cont.)

One common problem is that old approaches (like bag of words) lead to highly sparse predictor matrices.

- **A standard way of dealing with the sparse matrix problem is creating vector representations for each word, called the "word2vec" algorithm (Mikolov, et al., 2013)**
  - **Word2vec is a step towards transfer learning - for instance, Google has trained a freely-downloadable word2vec model outputting word representations using over a 100 billion words from a Google news dataset**

- **There are some issues with using off-the-shelf word2vec models**
  - **One problem is that the data on which the model is trained is generic - it encompasses non-financial data, leading to meanings which might not make much sense in our context**
  - **For instance, the closest words to "bull" in the Google word2vec model would be other animals like "cow" and "dog", whereas we want to see similar words like "bear", "rally" and so on**

- **Li and Shah (2017) used finance-relevant text data from micro-blogging sites (StockTwits, Twitter) to train their word2vec models**
  - **They created sentiment-specific embeddings so as to be able to predict text sentiment of any given text for any firm**

Introduction and Background (cont.)

- **Moore and Rayson (2017) used news headlines to train a word2vec model***
  - Moore et al. try to predict sentiments in headlines concluding that deep learning approaches like Bidirectional Long-Short Term Models (BiLSTM) beat simpler approaches like Support vector Regressions (SVR) by 4-6%
- **Schumaker and Chen (2007) take a linguistic approach to predicting intraday stock movements based on financial news**
  - They extract specific phrases from all documents split by sector, leading to a less sparse predictor set than a simple Bag of Words approach
  - They conclude that firm- / sector-specific training for their models leads to better performance
- **Recent work by Velay and Daniel (2018) used top 25 news headlines to predict the end-of-day value of the DJIA index**
  - They tried both statistical and deep learning models, but find that deep learning algorithms had difficulty figuring out the link between the headlines and the index trend

* They have kindly open-sourced their model which we use for one of our approaches

Introduction and Background – Our Approach

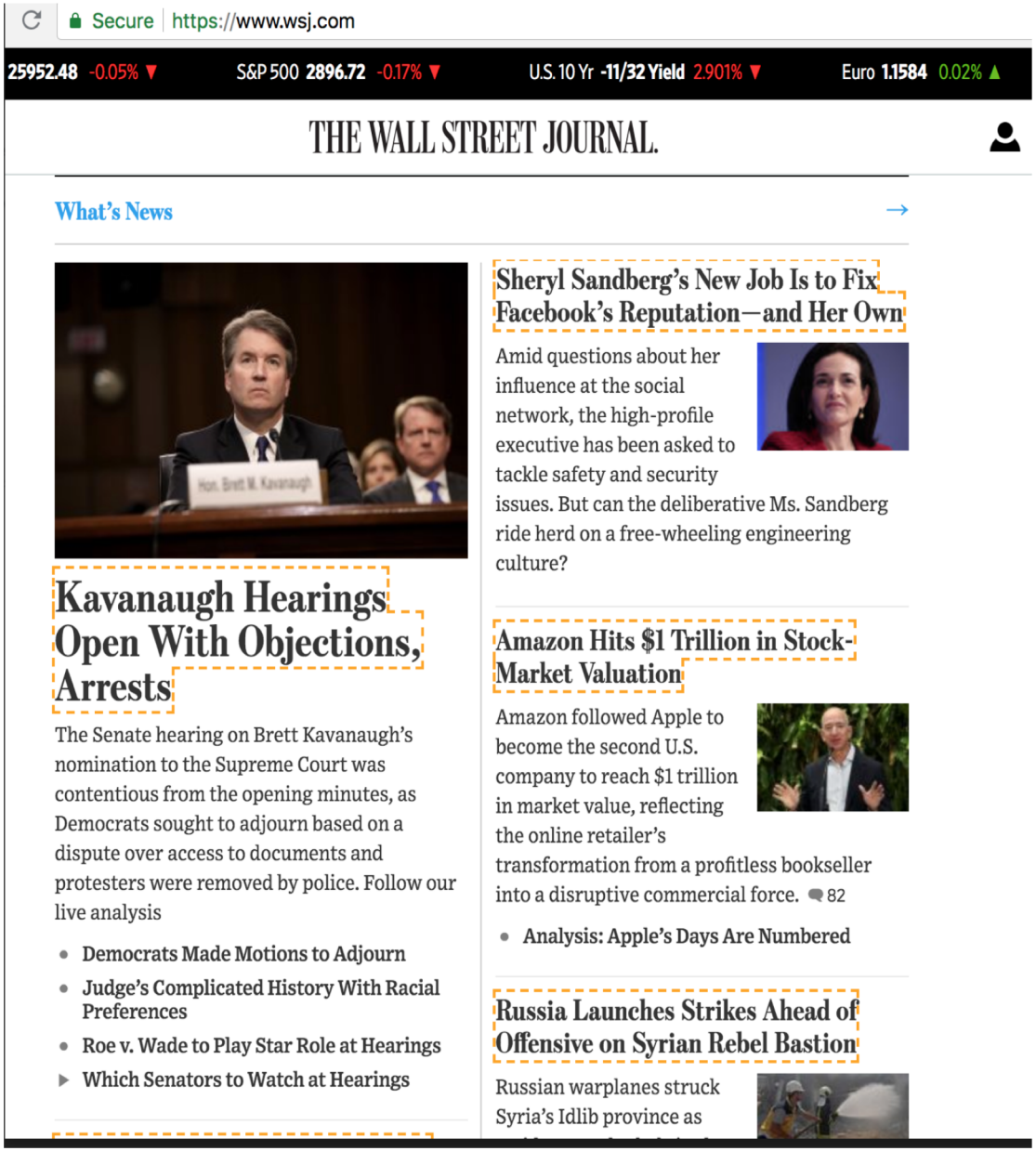
In this project, we incorporate lessons from the above cited literature (among others) as we attempt to use sentiment data to predict corporate bond prices.

- **Most previous approaches approaches have focused either on pure sentiment prediction, or on its effects in liquid markets like equities, with little attention being given to credit markets.**
- **This study explores different approaches to predicting corporate bond price moves over a 1-day horizon, using a self-sourced dataset of news headlines**
- **We focus on both single-name credits as well as the investment grade and high yield credit index ETFs**
  - **These different levels of aggregations carry their own benefits and challenges.**
  - **One obvious benefit for single-name credits is the better one-to-one correspondence between a news item and the firm**
  - **However, there are fewer headlines for individual firms than the market and trading single-name credits involves relatively high transaction costs**

The Data

Headline data were collected from major finance news sources such as Wall Street Journal, Yahoo Finance, Washington Post and PR News

- **Data were collected using the Wayback machine**
  - **The Wayback Machine was launched in 2001 to address the problem of website content vanishing whenever it gets changed or shut down**
  - **The service enables users to see archived versions of web pages across time, which the archive calls a "three dimensional index".**
  - **Kahle and Gilliat created the Wayback machine hoping to archive the entire Internet and provide "universal access to all knowledge."**



Data - Sources

In addition to the data from the Wayback machine, we were able to find sentiment scores from the Thomson-Reuters 2 Sigma data.

- **Wayback Machine Data (Crawl Data)**
  - **The raw html files contain the date, headline, content, and link for each news item**
  - **There is no sentiment data associated with the news items**
  - **Although there are headlines, there is typically very little additional news**

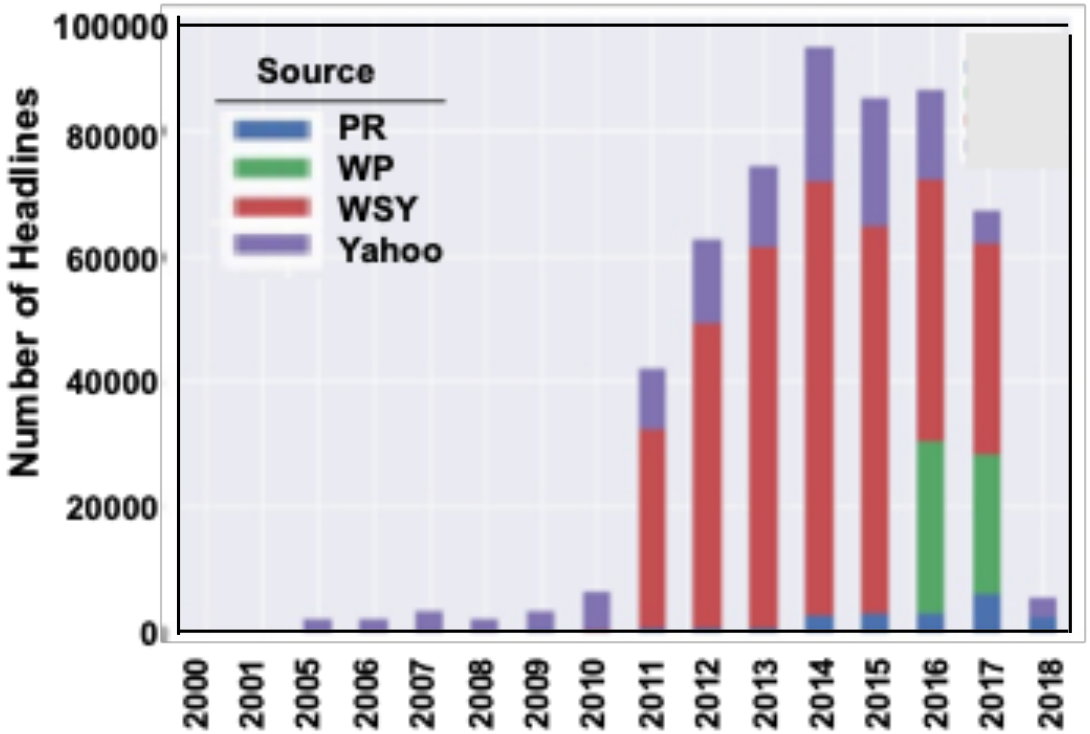
Data Source Properties

Source:	Total	Features
Thomson Reuters(2 sigma data)	9,328,750 (8902 companies covered)	35(Time/ Headline/ Assetcodes/ SentimentClass,etc)
Wayback Machine	534,467	4 (Date/ Headline/ Content/ URL)

- **2 Sigma Data**
  - **The participants were given access to organized and comprehensive finance data provided by 2 sigma (from 2007-01-01 to 2018-07-31, total 9 million rows).**
  - **This dataset contains information at both article level and asset level and includes article details, sentiment and other commentary**
  - **The sentiment class in the news data indicates the predominant sentiment class for this news item regarding to the specific asset**
  - **Sentiment for each item is divided into three classes: Negative, Neural and Positive, then selected the class with the highest probability**

Wayback Data by News Source

We extracted a total of over 500,000 news headlines from the Wayback machine beginning in 2000.



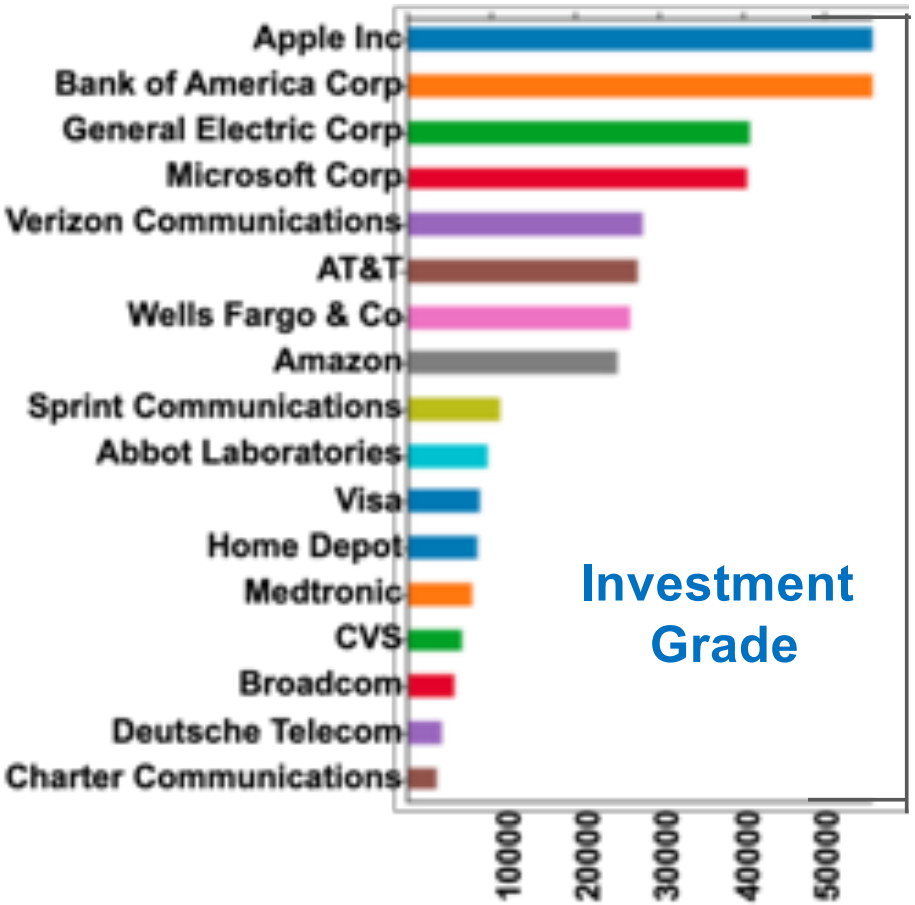
Sources	Number
Bloomberg	0
Yahoo	118,542
PR	17,882
WSJ	348,717
Google	0
WP	49,325
Total	534,467

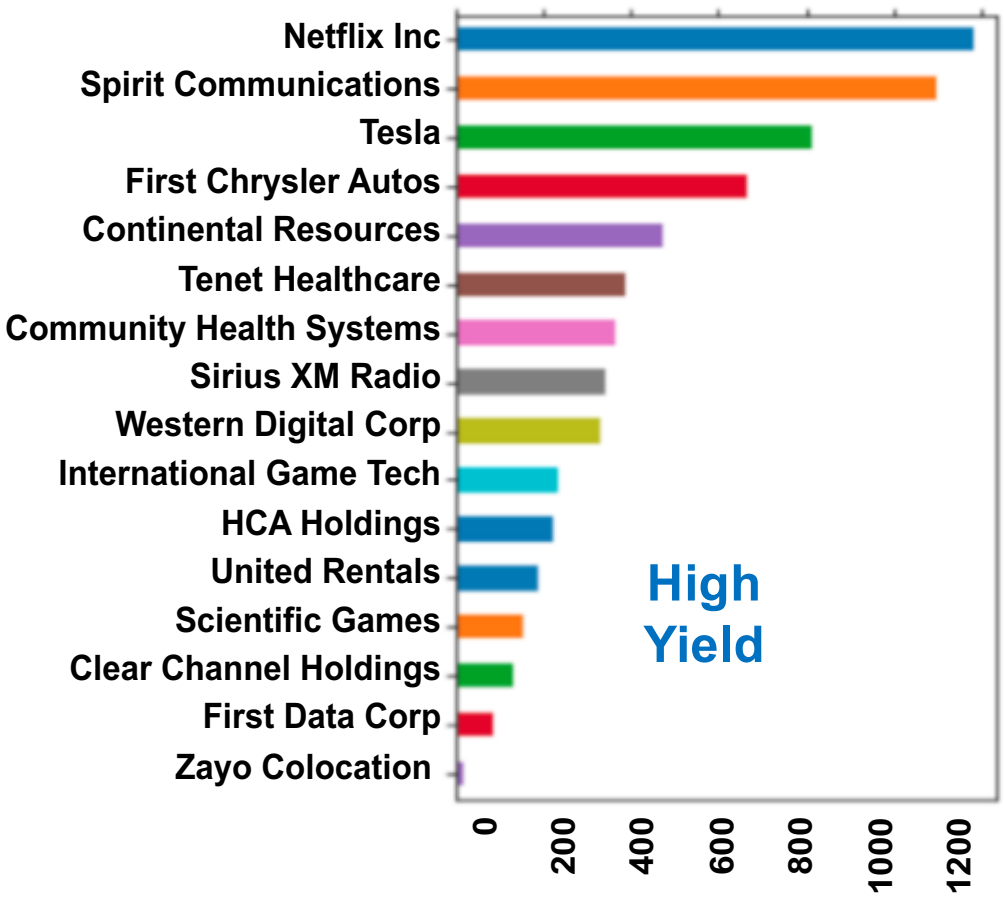
- The aggregated level data was obtained by crawling the web
- Duplicate headlines were dropped from the dataset
- Over half of the data comes from the Wall Street Journal (WSJ)

Company Coverage

In addition to the data from the Wayback machine, we were able to find sentiment scores from the Thomson-Reuters 2 Sigma data.

- **We chose two corporate bond index: LQD (investment-grade) and HYG (high yield) as our analysis targets**
  - **The charts below show the number of news stories for each company covered in the 2-sigma data**



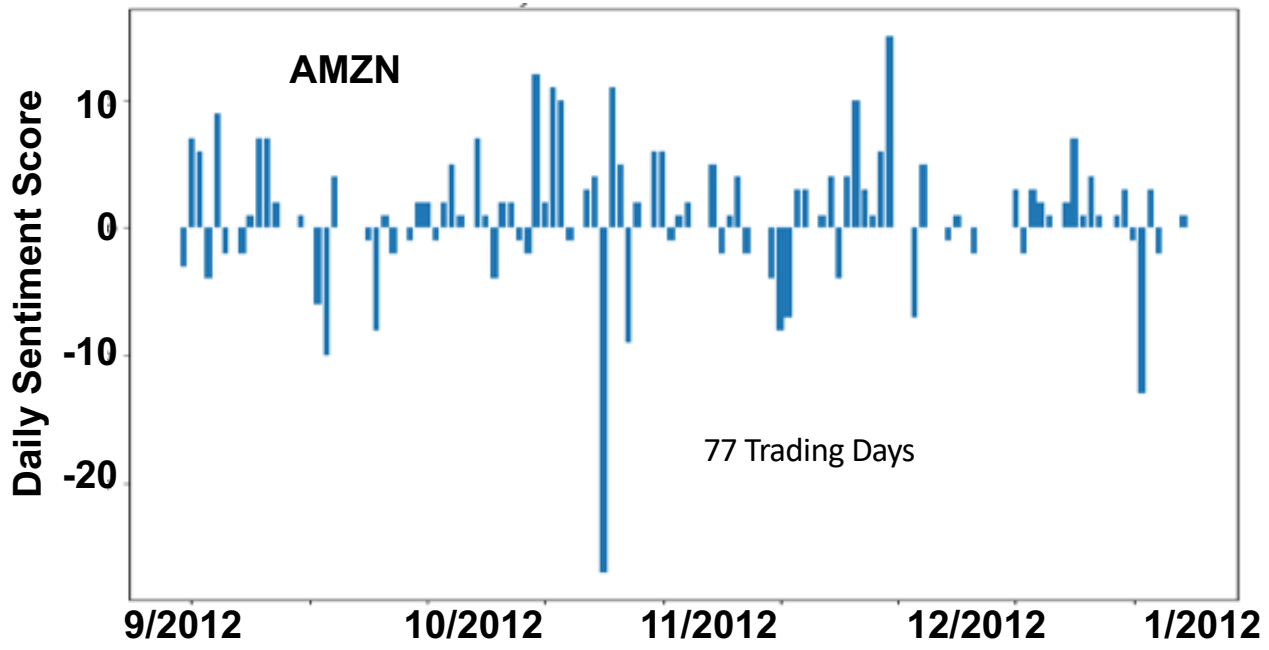


Predicting Returns from Individual Firms

In our first set of studies we summed the sentiment scores from the 2 sigma data on individual firms and used it to go long or short their bonds.

- **Trading strategy for Individual firms from sentiment data**
  - **For each target company, aggregate all the news before 4 pm on each trading day and sum their daily sentiment scores**
  - **Go long their bonds if the sum of daily scores is positive; go short if negative; and do nothing if zero**
  - **Trade with that day's closing price and close position on the next trading day's closing price**
  - **Test on different time lag of the signals**
- **An example of the summed daily sentiment scores for Amazon appears on the right**

Daily Sentiment Scores for Amazon



Sentiment Strategy for Individual Firms

We calculate the daily P/L of our long/short trades based on sentiment data by looking at the change in spread on the bond over the period in question.

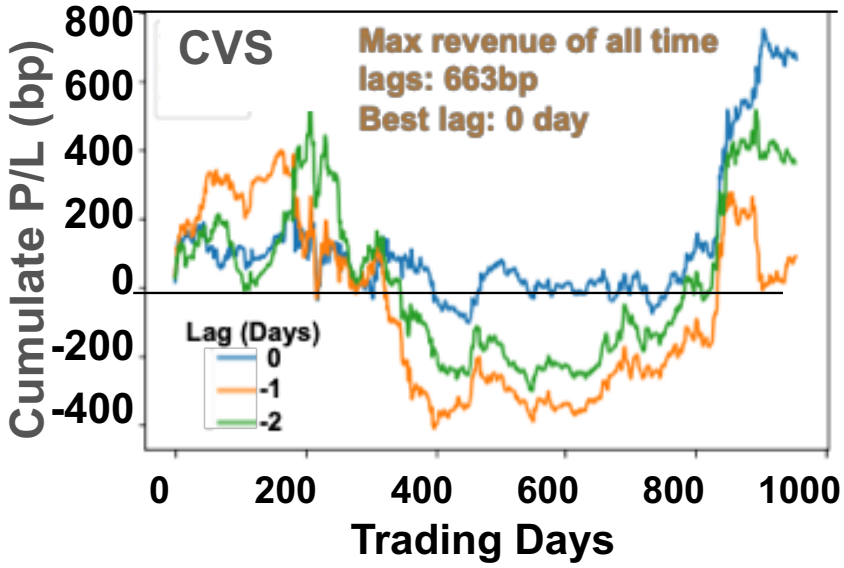
- For a long position in a given bond, we calculate the daily P/L as: $P\&L_{t_1} = -Dur_{adjusted} * (Spead_{t1} - Spead_{t0})$ in bp

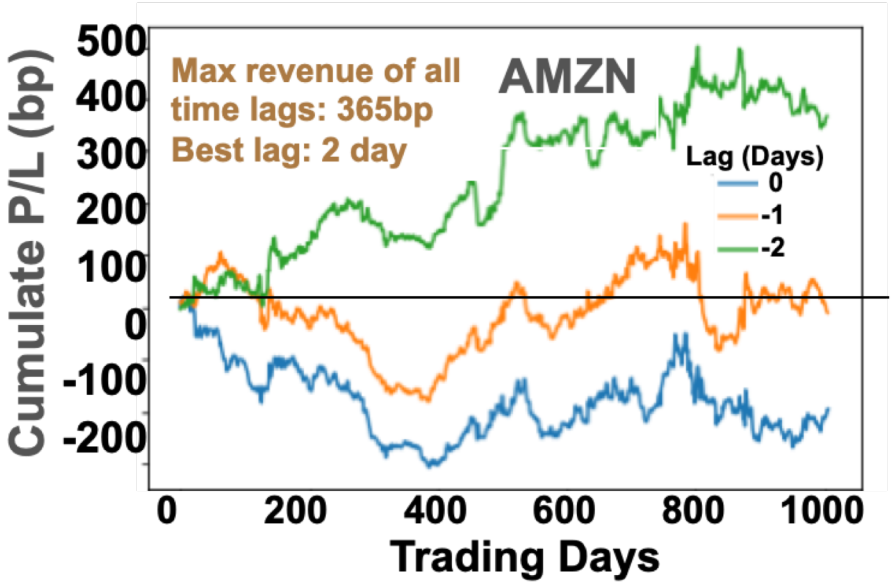
  which is the change is spread times the duration of the bond.

- For a short position in a given bond, the P/L is the negative of the long position P/L above
- To eliminate the effect of changes in market spreads, we hedge the single name companies with the LQD index
- That is, we take the P/L of a single firm and subtract the appropriate P&L of the index for each trading day
- Use the adjusted duration and spread change of LQD index to get the daily P/L of the firm in question

Some Examples of Results for Individual Firms

We provide some examples of cumulative uncompounded returns from bonds of individual firms based on sentiment long/short trades (ignore transactions costs)





Ticker	Best revenue	Best lag
AAPL	184.0172	2
ABT	263.8626	0
AMZN	365.4643	2
BAC	534.7491	2
CHTR	211.3387	1
CVS	663.8265	0
GE	347.3975	0
HD	755.7168	1
MDT	787.6135	1
MSFT	-176.5507	2
V	361.9183	1
VZ	-26.3855	1
WFC	1356.4677	1

- **The performance differences across different time lags vary, which may indicate the limited power of the strategy**

Firm Level Sentiment Prediction

We will not have access to 2 Sigma data going forward, we decided to build a model to mimic the 2 Sigma sentiment scores and use the to predict market moves.

- We will not have access to the 2 Sigma data going forward
- Need to “back-calculate” firm-level sentiment model from given information so as to be useful in trading
- Disadvantages of neural nets for sentiment prediction
  - Too complex
  - ~100k parameters
  - Computationally difficult to train
- Past approaches have shown benefits of simpler models like Naive Bayes in sentiment classification, after appropriate text clean-up
- We decided to build a boosted tree model trained on the 2 Sigma data to generate sentiment scores
- We then would use those sentiment scores to try to predict market moves for individual firms.

Firm Level Sentiment Prediction - Methodology

We trained an XGBoost tree model to predict sentiments scores from the 9 million Thompson-Reuters 2 Sigma news items.

- **For training, we used only those headlines with a minimum of a 70% relevance score for a given firm as per 2 Sigma**
- **Split data into 80% training and 20% test (non-random sampling)**
- **To process the news headlines we do the following:**
  - **Convert to lower case and remove numbers**
  - **Remove short words (with length < 3)**
  - **Remove stop words - *the, and, of, ...***
  - **Remove words not recognized as part of an "English dictionary."**
  - **Lemmatize words**
  - **Choose words that are relatively common across all news headlines for a specific firm**
- **Run a boosted trees model (bagged trees)**
- **Check sentiment prediction accuracy, number of trades annually based on different probability thresholds**

Input to XGBoost Model

The inputs to the XGBoost model are "term count" matrices for each firm: each row is a news headline, and each value is the number of times a word occurred.

- **The input data is a "term count" matrix**
  - **Each row corresponds to a news headline**
  - **Each value in the matrix corresponds to the number of times a certain word occurred in the sentence**
    - **For instance, in the 2nd sentence, the word "update" occurred once**
- **The matrix has 291 columns for this firm, where each word corresponds to a unique word. The list of words is unique for each firm and is selected algorithmically, depending on certain criteria.**
- **The XGBoost model is built on the foundation of decision trees**
  - **It creates weak classifier trees (AKA small trees) and sequentially keeps adding such trees to the model, focusing more on rows that were mis-classified in previous trees**

Sample of Input to XGBoost Model

	access	acquisition	add	adherence	adjusted	aet	affiliation	affirms	agreement	aid	...	update	upgrade	urge	value	view	vote	wag	walk
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

Out-of-Sample Performance

The out-of-sample performance of the XGBoost sentiment model appeared to be very good.

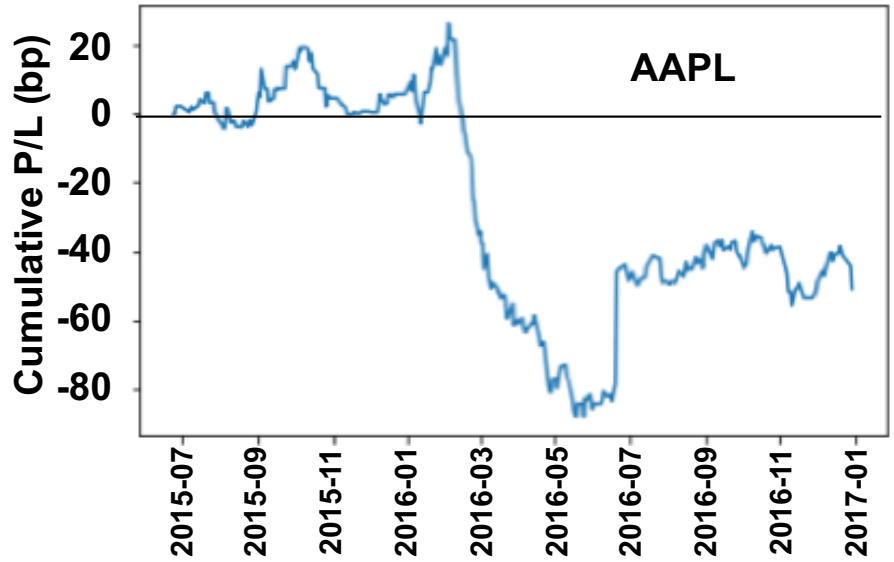
- The model outputs a signal of percent positive, neutral or negative
  - The Softmax function normalized their sum to 1.0
  - The category with the largest value is the one with the highest value
- To illustrate the performance of the model, the table shows the model accuracy for various threshold values
  - The table shows that the model performs above chance for almost all thresholds for both positive and negative sentiment
    Also, performance increases as threshold increases, but number of cases decreases

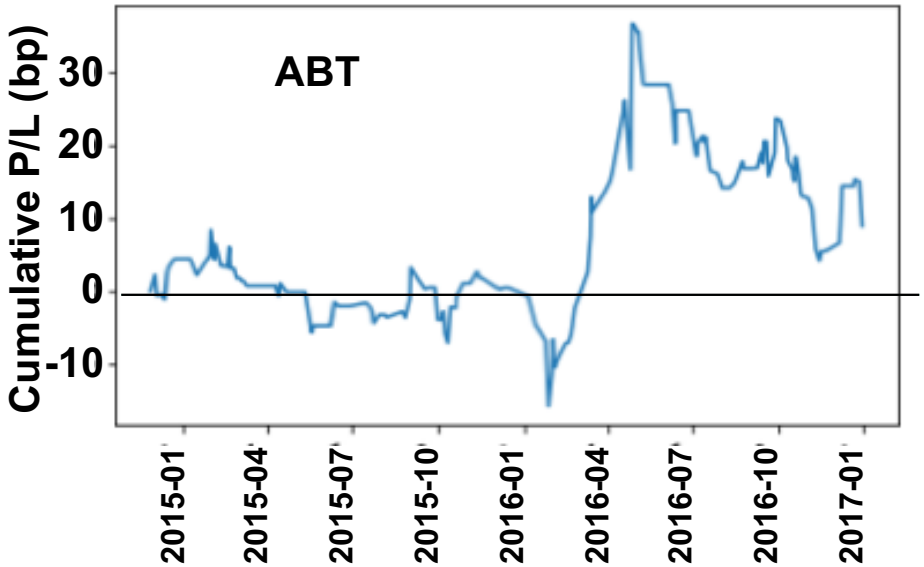
Out-of-Sample Accuracy for Selected Firms

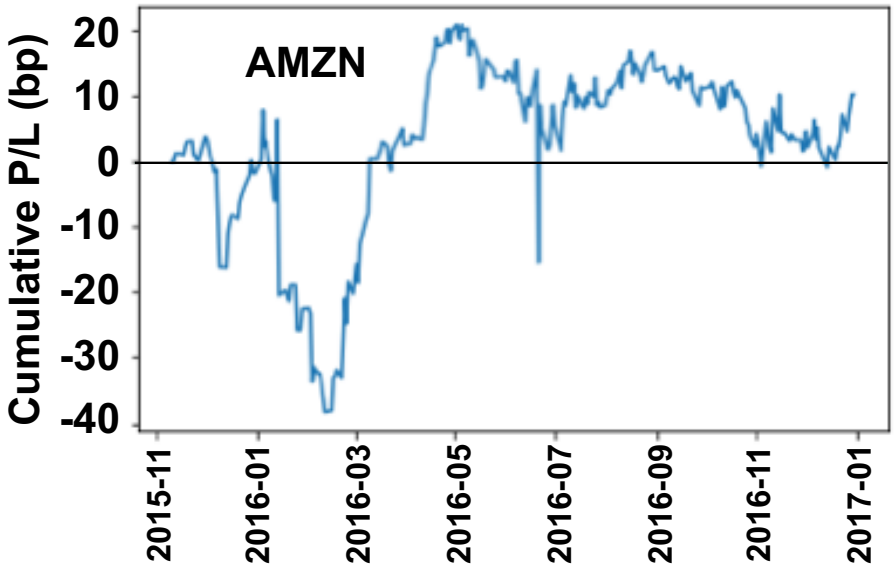
Tickr	Threshold	Precision Positives	Precision Negatives	Annual Signals
AAPL	0.33	48%	79%	245
AAPL	0.5	71%	88%	199
AAPL	0.6	70%	90%	156
AAPL	0.7	78%	92%	121
ABT	0.33	67%	79%	74
ABT	0.5	77%	87%	62
ABT	0.6	83%	91%	50
ABT	0.7	86%	97%	37
AMZN	0.33	60%	71%	233
AMZN	0.5	64%	82%	227
AMZN	0.6	74%	89%	204

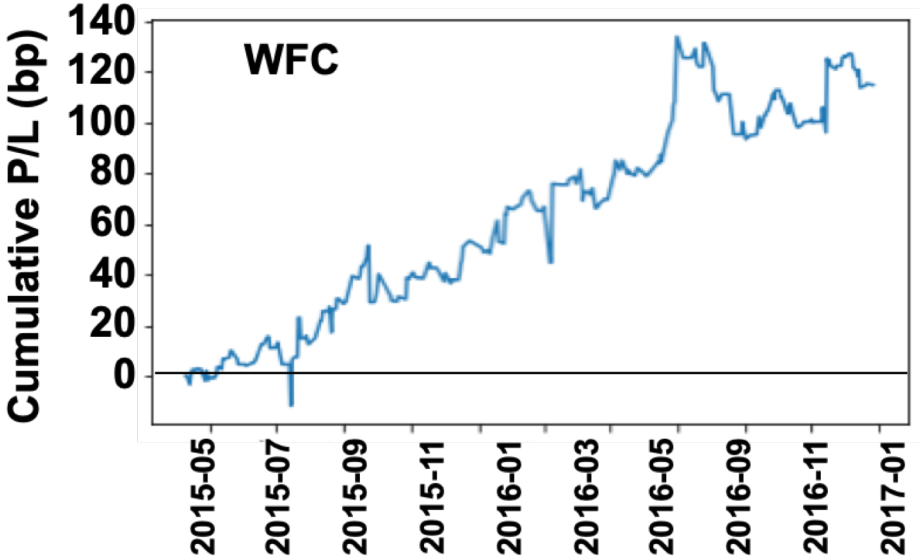
Predicting Returns from XGBoost

Although the XGBoost model appears good at predicting sentiment, it does not perform as well at predicting returns from individual firms.









Predicting Index Returns from Sentiment Data

We decided to predict index returns because of the greater number of news headlines and lower transactions costs

Objective:

- Directly use the sentiment scores to forecast index movements (LQD / HYG)
- These are ETFs for the investment-grade and high yield corporate bond indexes, respectively
- They are extremely liquid and have small bid/ask spreads

Challenges:

- Removing daily price impact on the ETF from movements in Treasury yields
- How to handle features, as we have multiple news headlines per day
- How to determine the effects of the time horizon for prediction and testing

We Use Bond Indexes as a Proxy for the ETFs

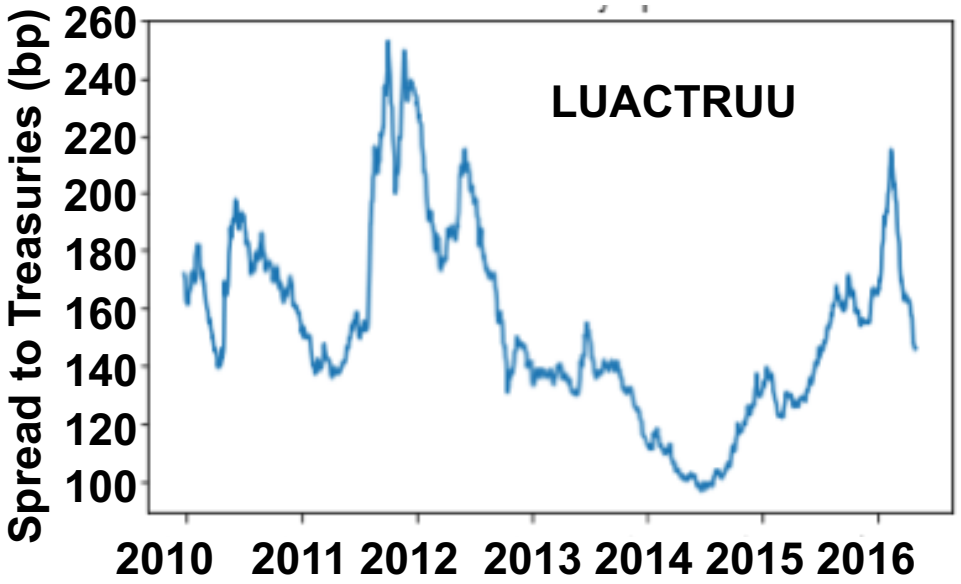
Because we can get daily spread changes on investment-grade and high yield bond indexes, we use those as a proxy for their ETFs.

- **We used spread changes in Bloomberg/Barclays corporate bond indexes as proxies for the ETF moves**
  - **LUACTRUU is the investment-grade index**
  - **LF90TRUU is high yield indexes**
  - **Their daily spread moves are available with changes in US Treasury yields removed**
- **Metrics**
  - **Duration (OPTION_ADJ_DURATION_SOV_CRV)**
  - **Spread (OAS_SOVEREIGN_CURVE)**
  - **Close Price (PX_LAST)**
- **Dependent Variable for Training: Spread move up or down**
  - **Up (Label "1"): Spread(t+1) – Spread(t) >= 0**
  - **Down (Label "0"): Spread(t+1) – Spread(t) < 0**
- **Methodology**
  - **Use news before 4pm to predict the spread's movements**
  - **News after 4pm will be used to predict next day's movements**
  - **Do feature engineering (flatten, extraction, …) for news per day**

LUACTRUU – Investment-Grade Bond Index

- Our sample for the LUACTRUU consisted of daily spread changes from 01-Jan-2010 to 05-31-2016
- The training sample consisted of OAS changes from 01-Jan-2010 to 12-31-2014
  - This was 1249 trading days
  - 45% of the trading days the OAS change was positive (market bond yields rose)
- The out-of-sample test periods ranged from 01-01-2015 to 05-31-2016
  - This was 354 trading days
  - 51% of the trading days the OAS change was positive

Daily Bond Spreads



Sample Statistics

Training data size	1249
Test data size	354
Training data positive ratio	45%
Test data positive ratio	51%

Feature Extraction

We summed sentiment scores over the course of a given day in order to make predictions for a trade (long / short / no trade) just before the close.

- **Daily sentiment data from Thompson-Reuters 2 Sigma were divided into positive, negative, and neutral sentiment**
- **Each class was then "flattened" before input to the regression**
  - **Flattened data were: mean, max, min, 25th and 75th percentiles**
- **How to handle features, as we have multiple news headlines per day**

Example of Flattened Data for Positive Sentiment

Date	Positive_ mean	Positive_ max	Positive_ min	Positive_ q0.25	Positive_ q0.75	...	Is_Monday	...	Is_Jan	...
1/4/10	0.42765462	0.856822	0.0259025	0.198021	0.580439	...	1	...	1	...
1/5/10	0.39551849	0.856924	0.0242462	0.1914795	0.56168425	...	0	...	1	...
1/6/10	0.37966194	0.856859	0.020735	0.177211	0.5601955	...	0	...	1	...
1/7/10	0.36742522	0.856851	0.0229957	0.17956425	0.55421	...	0	...	1	...
1/8/10	0.35409329	0.856712	0.0212103	0.175775	0.543231	...	0	...	1	...

Elastic Net Regression Model

The elastic net is a regularized regression method that linearly combines the L1 and L2 penalties of the lasso and ridge methods.

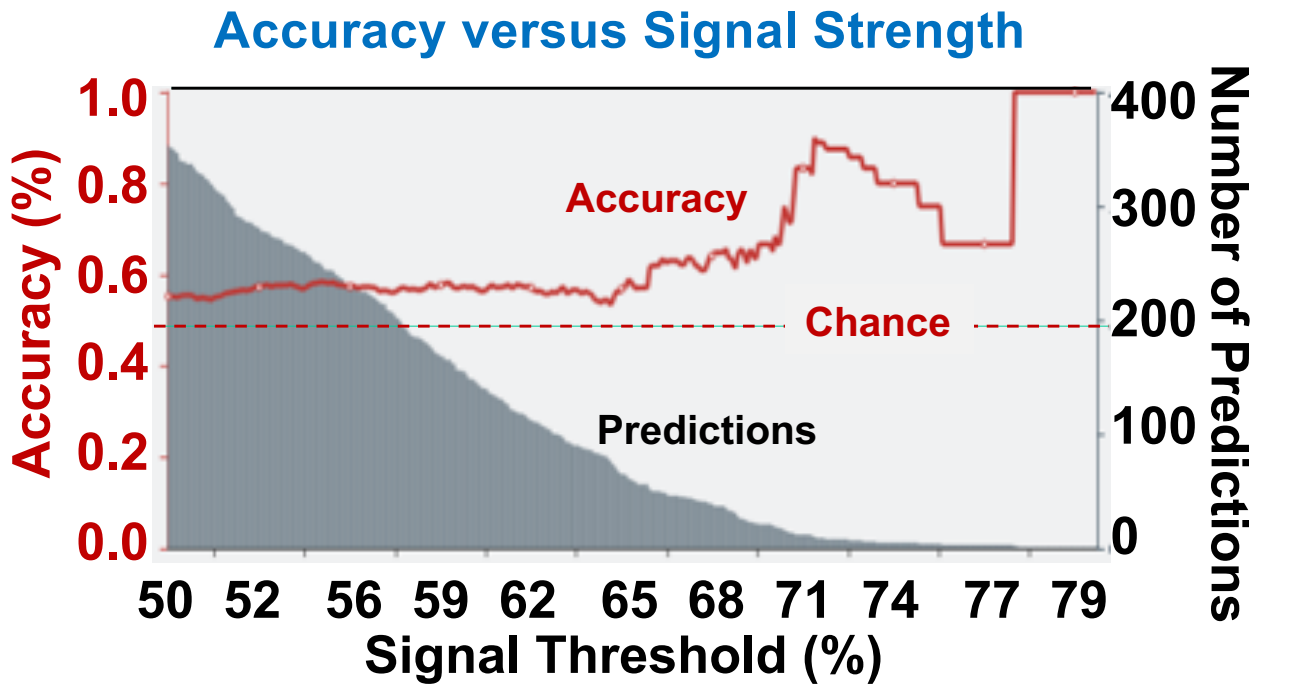
- **Elastic Net is a linear regression model trained with L1 and L2 prior as regularizer**
  - **Regularization methods are designed to avoid overfitting**
  - **Our features are not complicated**
- **The Elastic-net is useful when there are multiple features which are correlated with one another**
  - **The Lasso method will pick only one of them**
- **Elastic-Net also inherits some of Ridge's stability under rotation**

Reference: https://app.datarobot.com/model-docs/tasks/LENETCD-Elastic-Net-Classifier-mixing-alpha-auto-Binomial-Deviance-.html

Elastic Net Regression – Out-of-Sample Performance

With a binary 50% threshold, the model is only marginally predictive, but as signal strength increases, accuracy also increases.

- **For a binary 50% threshold, out-of-sample probability correct is 55% for long and short trades combined**
  - This corresponds to an area under the ROC curve of 0.571



- **Performance increases as thresholds increase, but number of opportunities decrease**
  - For 60% accuracy, one gets only 52 predictions of 354 days (14% trading days)
  - For 70% accuracy, one gets only 17 (5% trading days) predictions
  - For 80% accuracy one gets only 12 (3% trading days) predictions

Predicting Market Moves Directly from News Headlines

We decided to predict market moves directly from news headlines, rather than going through the intermediate step of sentiment scores.

Methodology

- **Word to vector technology transforms words in natural language into dense vectors, and semantically similar words have similar vector representations**
  - **The methodology of generating word vectors is based on statistics (co-occurrence matrix, SVD decomposition) to the neural network-based language model**
- **Before discussing the model, we discuss briefly the classical language models: from word2vec, ELMo to most recent and innovative model, BERT**
  - **BERT stands for (Bidirectional Encoder Representations from Transformers)**

Methods for Standardized Embeddings

Word and sentence embeddings have become an essential part of any Deep-Learning-based natural language processing systems.

- **A huge trend is the quest for Universal Embeddings: word embeddings that are pre-trained on a large corpus and can be plugged in a variety of downstream task models (sentimental analysis, classification, translation…)**
  - **A word embedding represents a word with numbers**
  - **By doing so it makes natural language computer-readable**
  - **These universal embeddings incorporate some general word/sentence representations learned on the large dataset**
- **Word2vec and ELMo are two versions of universal embeddings we consider here**
  - **Word2vec was created by a team of researchers led by Tomáš Mikolov at Google and patented**
  - **ELMo was developed by. Peters, Neumann, Iyyer, Gardner, Clark, Lee and Zettlemoyer at the Allen Institute for Artificial Intelligence**
- **The Word2vec and ELMo models are described in the next couple slides**

Word2Vec

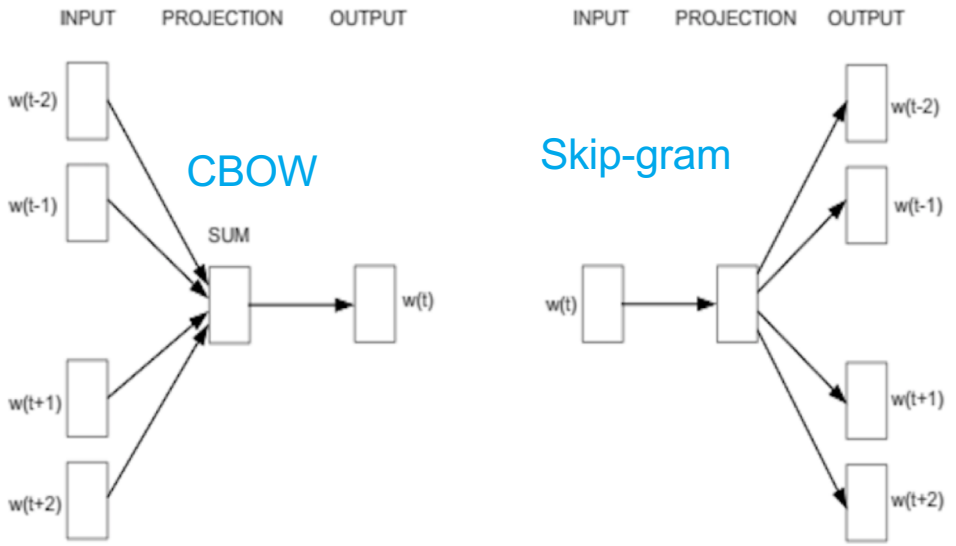
Word2vec is a group of related models that are used to produce word embeddings.

- **Word2Vec models are two-layer neural networks that are trained to reconstruct linguistic contexts of words.**
- **Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space**
- **Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.**
  - **It doesn't distinguish the different meaning of a word with the same tokens**
  - **For example, the word "bank" can relate to the financial institution or a river bank. The traditional word2vec is not able to capture this granularity**

Word2Vec (cont.)

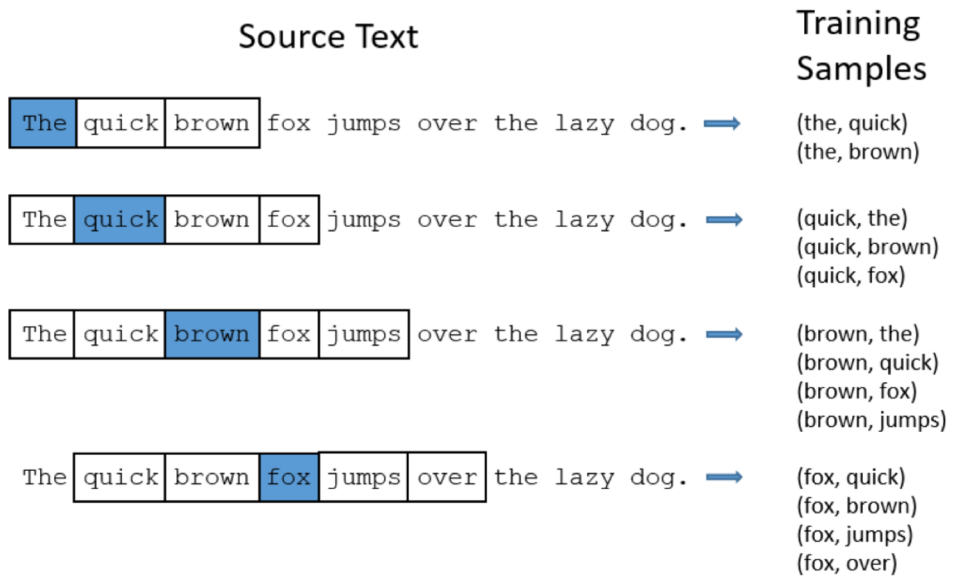
- **Word2vec trains words against other words that neighbor them in the input corpus**
  - **It does so using context to predict a target word (continuous bag of words - CBOW) or using a word to predict a target context, which is called skip-gram**

- **Train the network by feeding it word pairs found in training documents**
  - **The network is learns the statistics from the number of times each pairing shows up**
  - **For example, the network is probably going to get many more training samples of (“Soviet”, “Union”) than it is of (“Soviet”, “Sasquatch”)**
  - **After training, if you give the network “Soviet” as input, the it will output a much higher probability for “Union” or “Russia” than it will for “Sasquatch”**



Training the Word2Vec Network

(Target word is in blue)

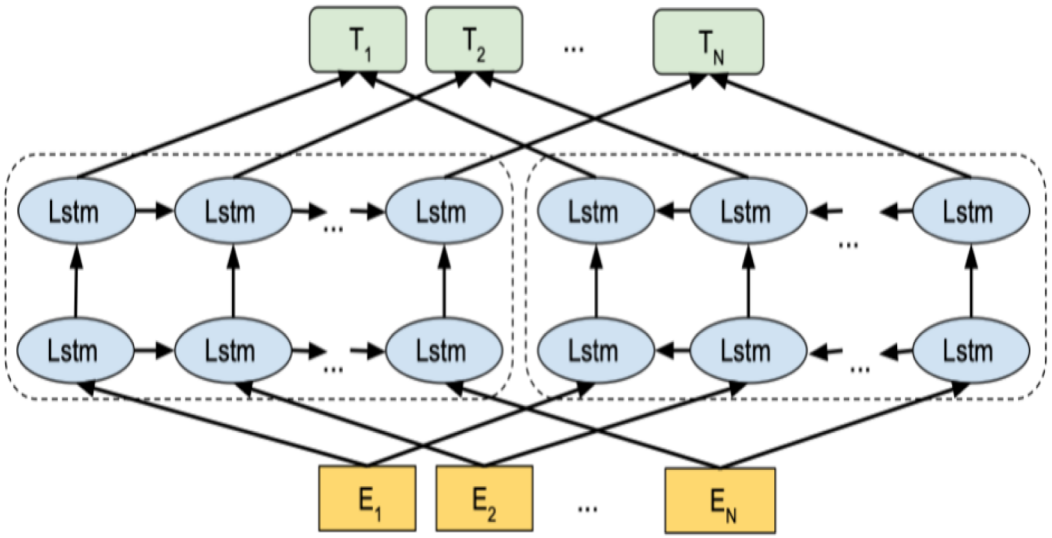


ELMo (Embeddings from Language) Model

The ELMo model solves the failure of Word2vec to distinguish the different meaning of a word with the same tokens.

- **ELMo uses bi-directional LSTMs to generate features for downstream tasks, which bring two advantages:**
  1. **ELMo representations are purely character based and can learn the complex characteristic of word usage**
  2. **Learn the change of word usage according to the different context in which it is used**

The ELMo Model



- **The bi-directional LSTM consists of 2 parts: a forward LM and a backward LM**
  - **The forward LM tries to predict the next word given all the previous words from left to right:**

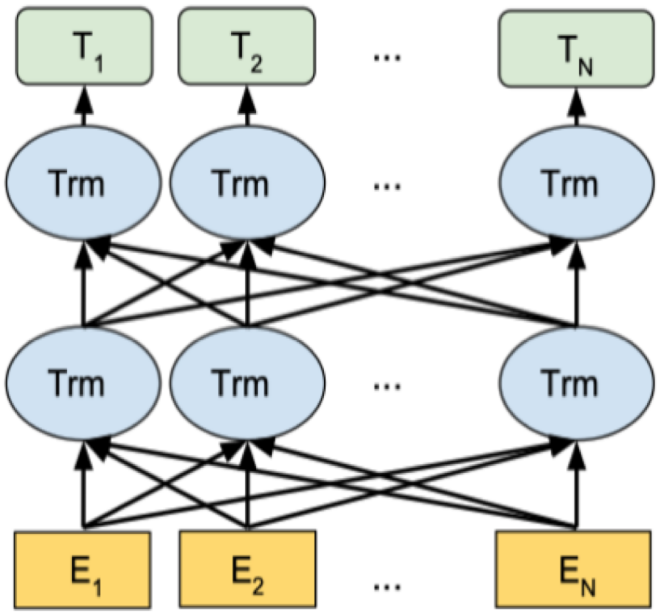
$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$$

  - For each position *k*, the LSTM outputs a context-dependent representation $\overrightarrow{h_{k,j}^{LM}}$, where where *j=1,...,L* and the top layer $\overrightarrow{h_{k,L}^{LM}}$ is applied on a Softmax function to predict the next word t_{k+1}

The BERT Model Architecture

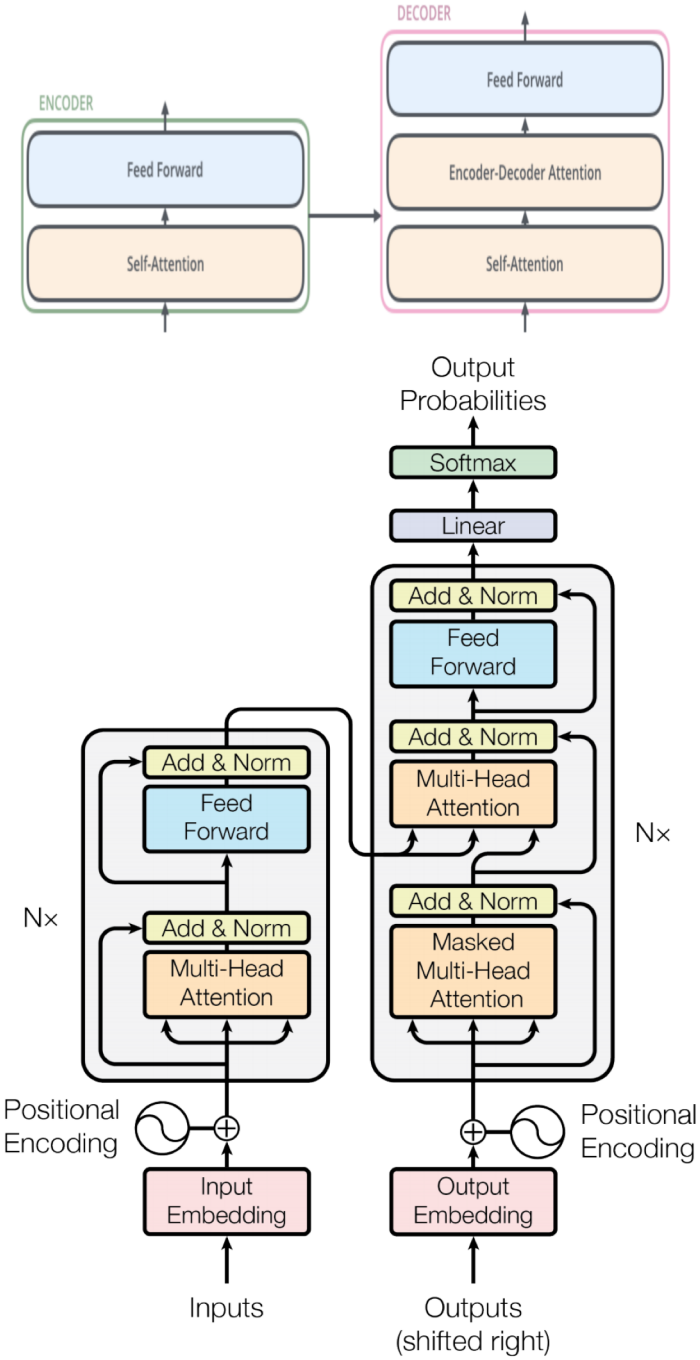
BERT Model

- Bidirectional Encoder Representation from Transformers
- Unsupervised Pre-training
- Pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers



Transformer Architecture & Benefits

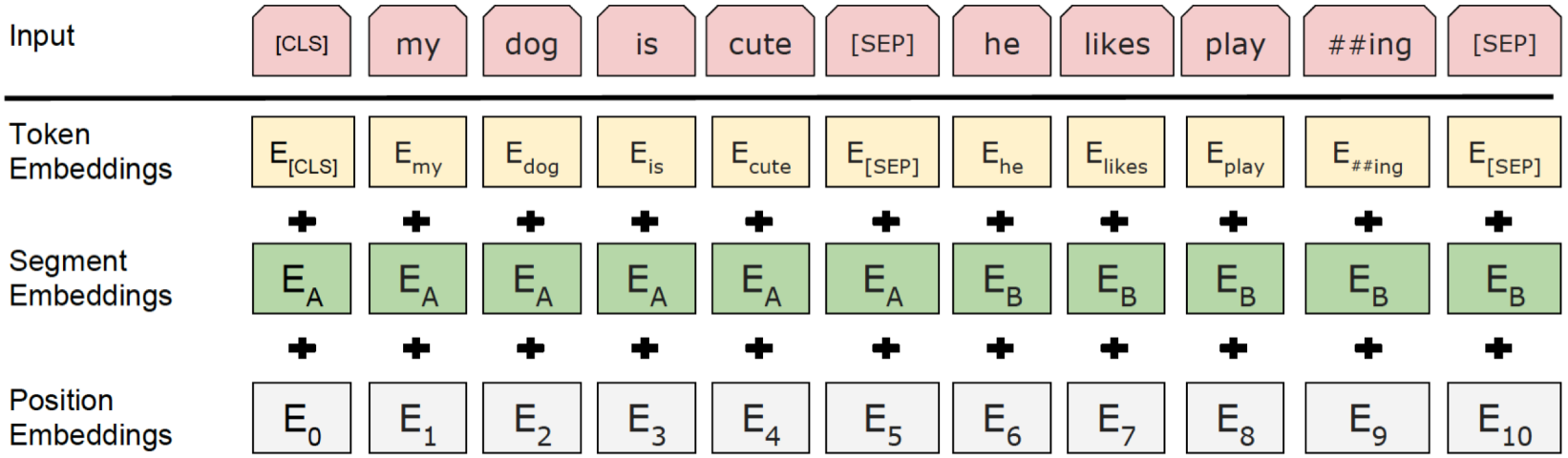
- Instead of the recurrent neural network, it uses attention to boost the speed with which these models can be trained, lends itself to parallelization
- Can be extended to an intense layer and improve accuracy

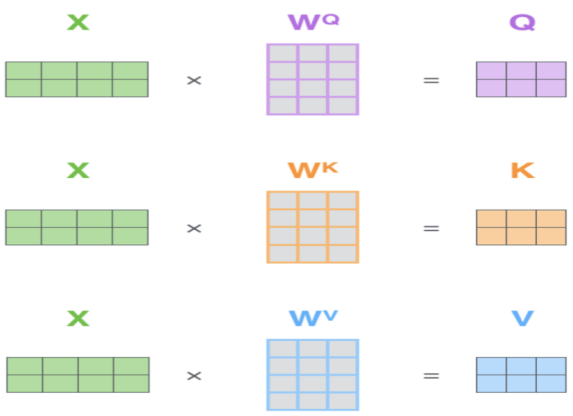


Sentence-Level Embedding and Encoder Detail

BERT Input Representation

The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings

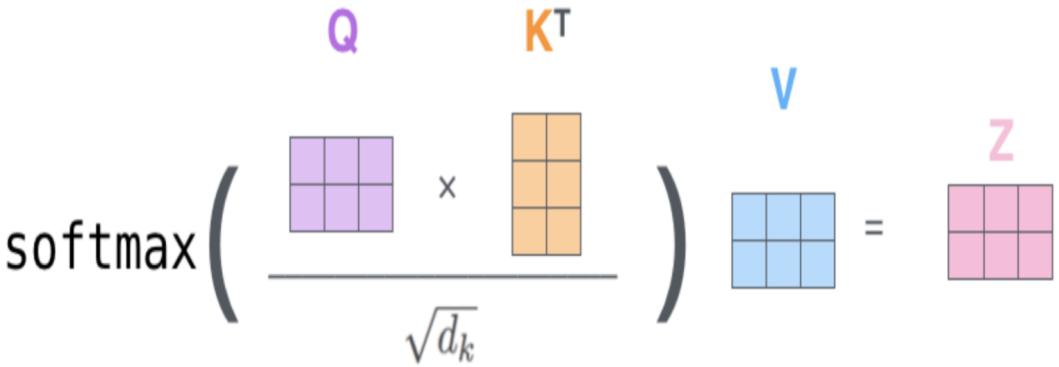




First step:

Calculate the Query, Key, and Value matrices for each word
The dimension of embedding changed to 512 to 64

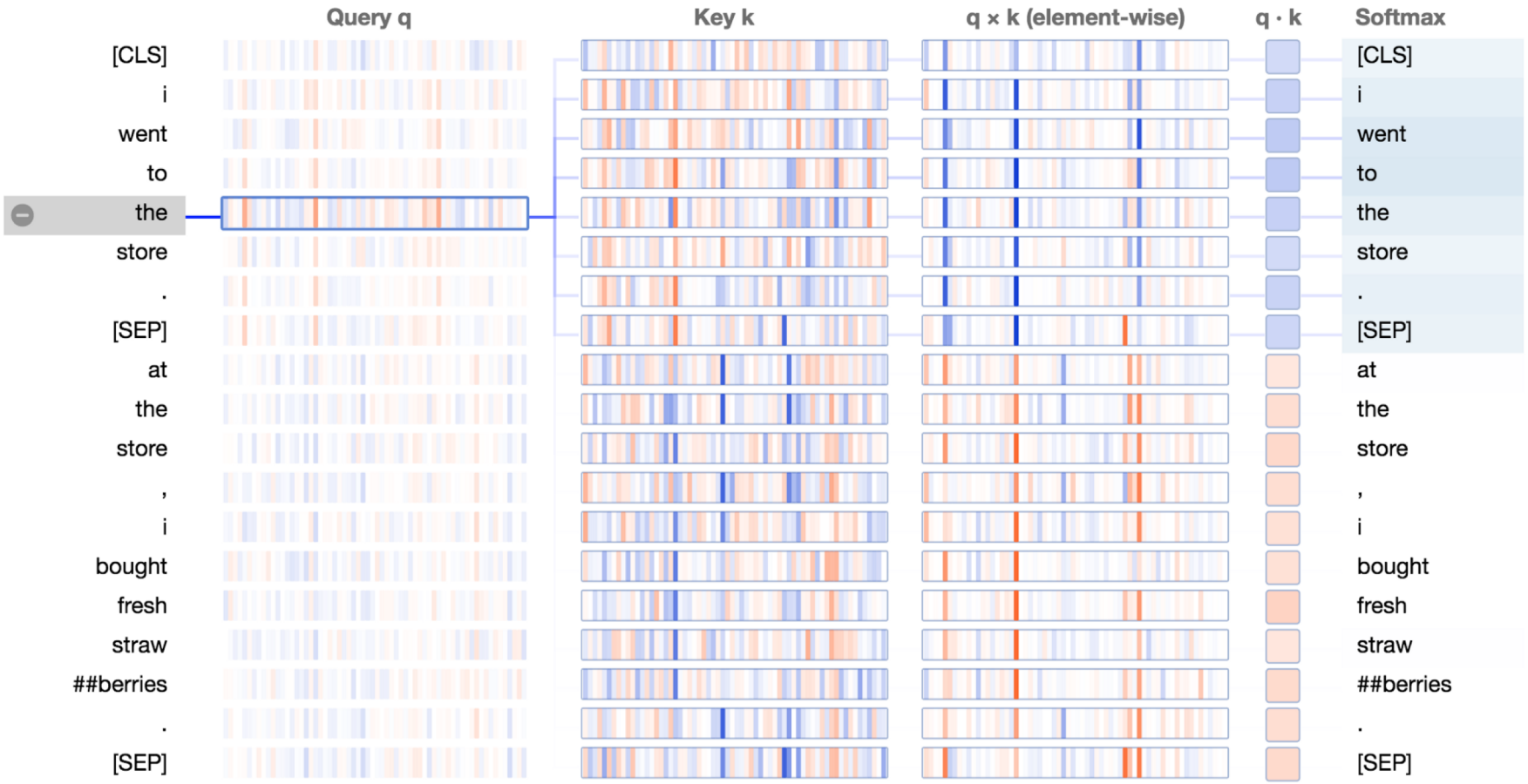
Self-Attention Calculation



Following steps:

Q*K(score) determines how much focus to place on other parts of the input sentence as we encode a word at a certain position. dk=64 here

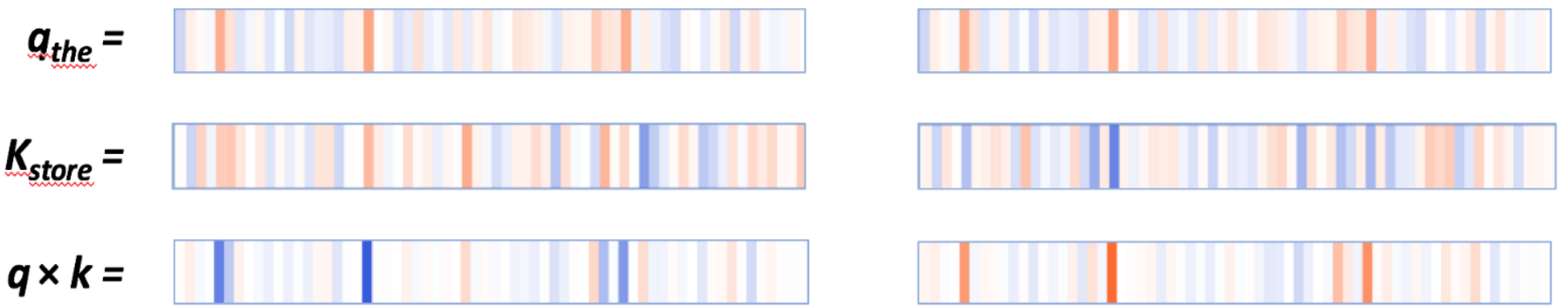
Relation to Attention and Bag or Words Pattern



- The query-key product is high when query and key are in the same sentence (left), and low when they are in different sentences (right)

*"the" and "store" in **sentence 1***

*"the" in **sentence 1**, "store" in **sentence 2***

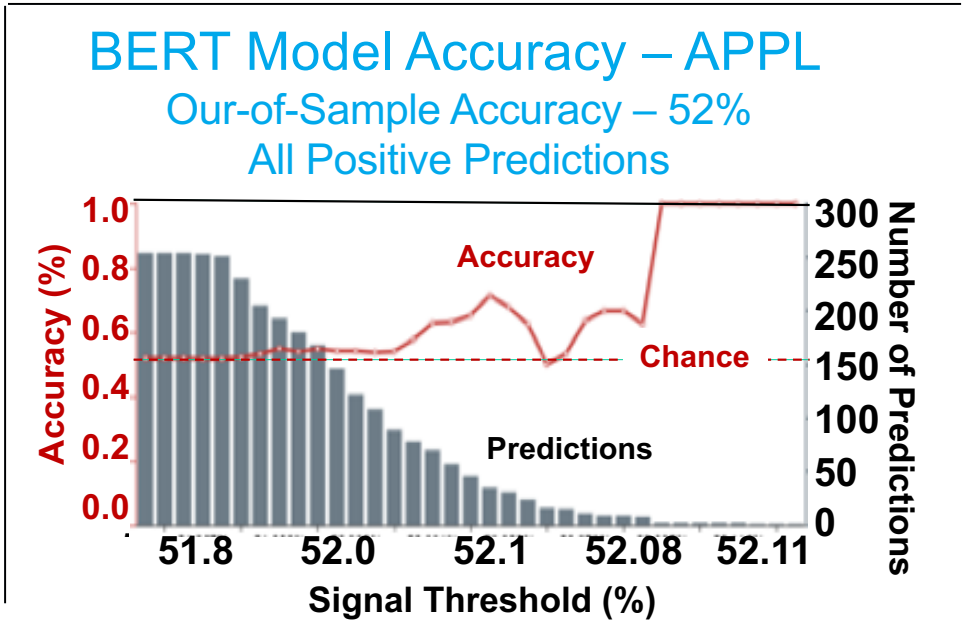


Predicting Index Moves with the BERT Model

Google has released pre-trained models from and we applied this architecture and fine-tune with our own data, sentence level news headlines from Two Sigma.

- We tested this standard workflow on a single firm, AAPL, and tried to predict its daily spread changes based relevant headlines
  - The direct fine-tuned BERT model made almost one-side predictions (all positive) within a very narrow forecasting range (51 to 52%)
  - This means the model cannot capture any signals at all and simply takes the slight imbalance from the training set to boost its accuracy

- Predicting AAPL spread changes
  - Less data, no large GPU required
  - 16K headlines from 2013-2016
    - Train Set: 2013 - 2015
      - 12.7K headlines; 50.4% Up
    - Test Set: 2016
      - 4.1K headlines; 52.7% Up
- Also tried predicting 2Sigma's sentiment scores
  - labels: -1, 0, 1



Predicting Sentiment with the BERT Model

We used the BERT model to predict 2 Sigma sentiment data with greater success.

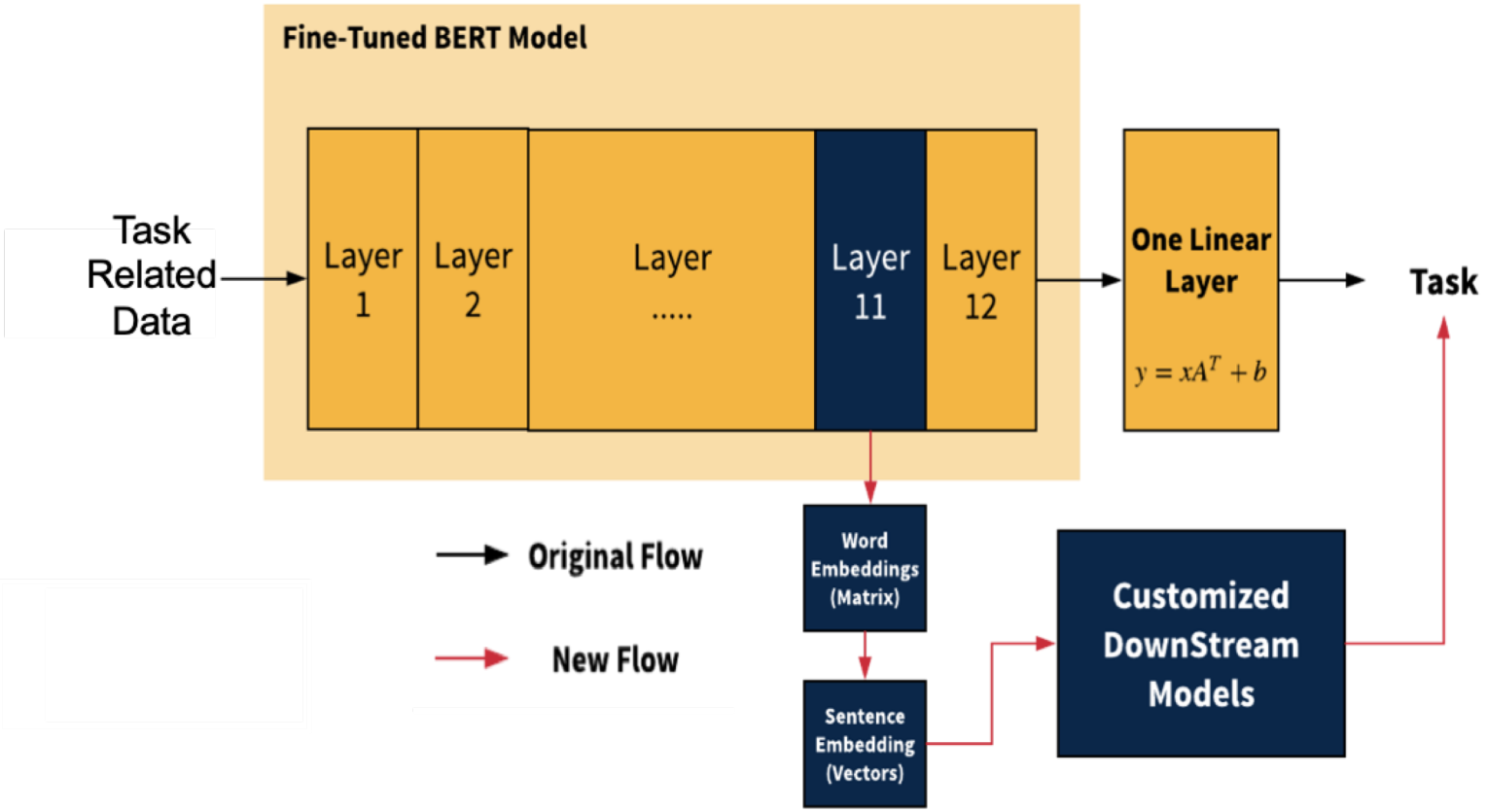
- **Why does the fine-tuned BERT model fail on the spread change task?**
  - **To answer this question, we switched the task from predicting spread changes to predicting the sentiment labels**
- **Despite sentiment prediction being a three classification problem, the accuracy increased to 66%**
- **The word-embedding trained from BERT is for a general-language purpose by a set of standard NLP techniques such as work masking and contextual predictions**
  - **We looked to modify BERT to work better in the context of spread changes**
  - **Almost impossible to modify pre-trained based model**
    - **Need to redo the pre-training process, large finance specific corpus, large GPU computing resources, time**
  - **Is possible to change last layers / downstream models**
    - **Only use a fine-tuned BERT as an embedding tool to extract features**
    - **Still takes advantage of Google's large pre-trained model but allows more flexible downstream models**

BERT: Sentence Embedding

We suspected that using a more complicated downstream model to replace the original soft-max layer might generate better predictions

- **After fine-tuning the Base (12 layers) BERT model, we pick the word embedding from the 11th layer and use average pooling to get a fixed-dimension vector that represents each headline**
  - **Those fixed-dimension vectors then act as the input features for a downstream machine learning model to forecast spread changes**

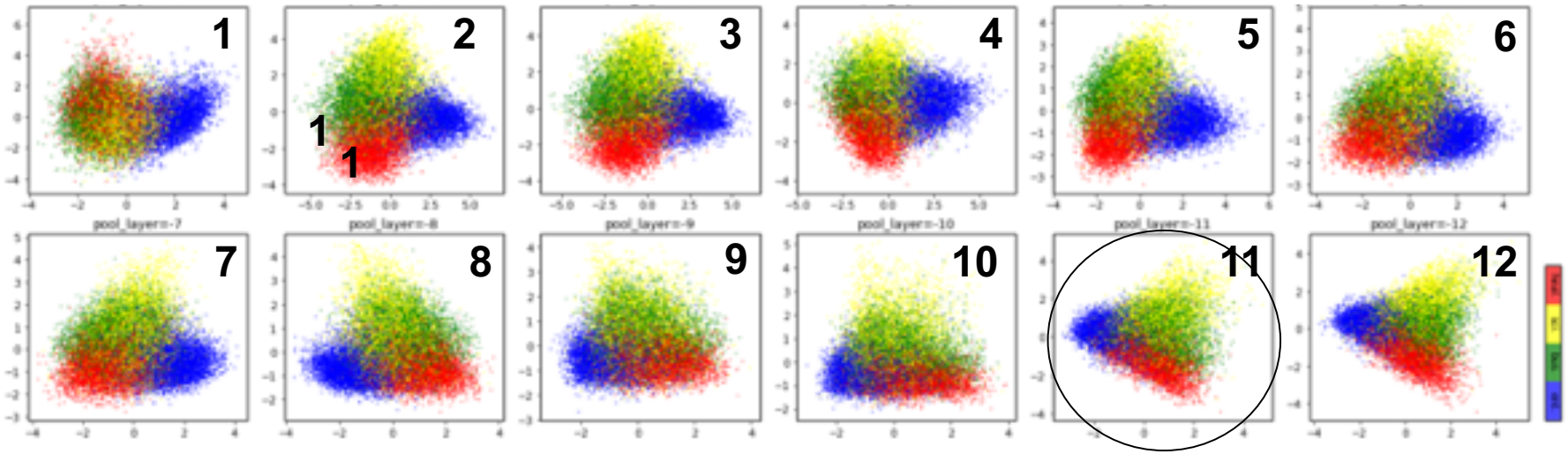
The Revised BERT Workflow



BERT: Why We Pick the 11th Layer

- The BERT model is pretrained with a bi-partite target (masked language model and next sentence prediction), which makes the last layer too biased to those two targets
  - Using the last layer is as same as in stacked LSTM/CNN
  - Taking a layer in front, the transformed embedding still carries the original word information without BERT's self-attention benefits
- Xiao Han applied BERT model on 20K news titles and used PCA to flatten each layer's output into a 2D plot
  - Each color represents a topic of those news, we can easily observe that the classification effects are most obvious in the last two layers

PCA Evaluation of BERT Layers using "Best-as-Service"



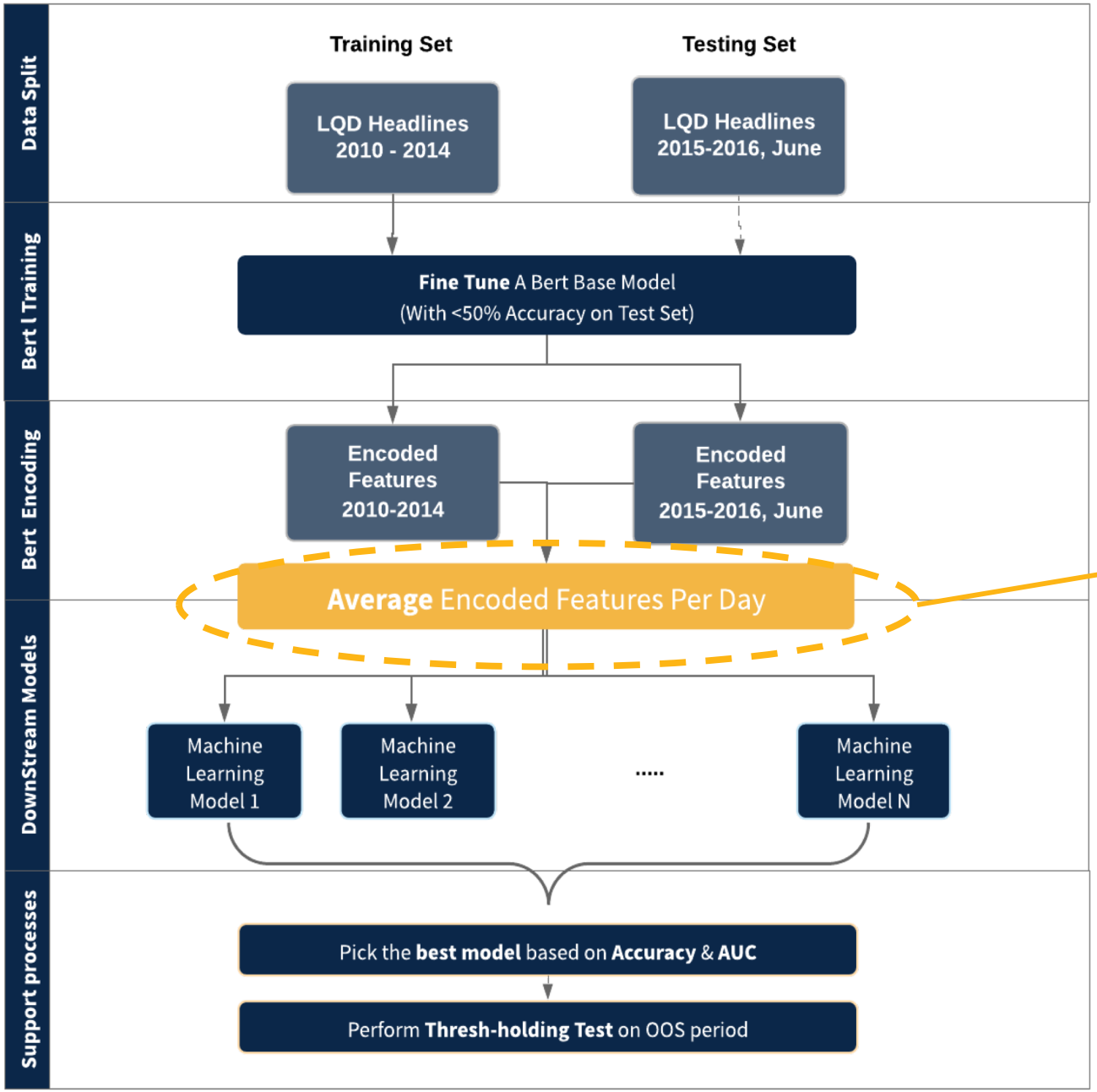
Reference: https://hanxiao.github.io/2019/01/02/Serving-Google-BERT-in-Production-using-Tensorflow-and-ZeroMQ/

BERT Model: Encoded Features

- **Each headline was converted into a vector (1 x 768)**
- **Also used (one-hot) categorical features from date for days of week and months of year**

Sub-Sample of Input Vector

Date	headline	Is_Monday	...	Is_Jan	...	0	1	2	...
5/1/13	garmin profit misses estimates	0	...	0	...	-0.21515	0.66142	0.068004	...
5/1/13	reuters insider - u.s. ...	0	...	0	...	-0.30309	0.81203	0.199902	...



AAPL:

Per news data directly

Index:

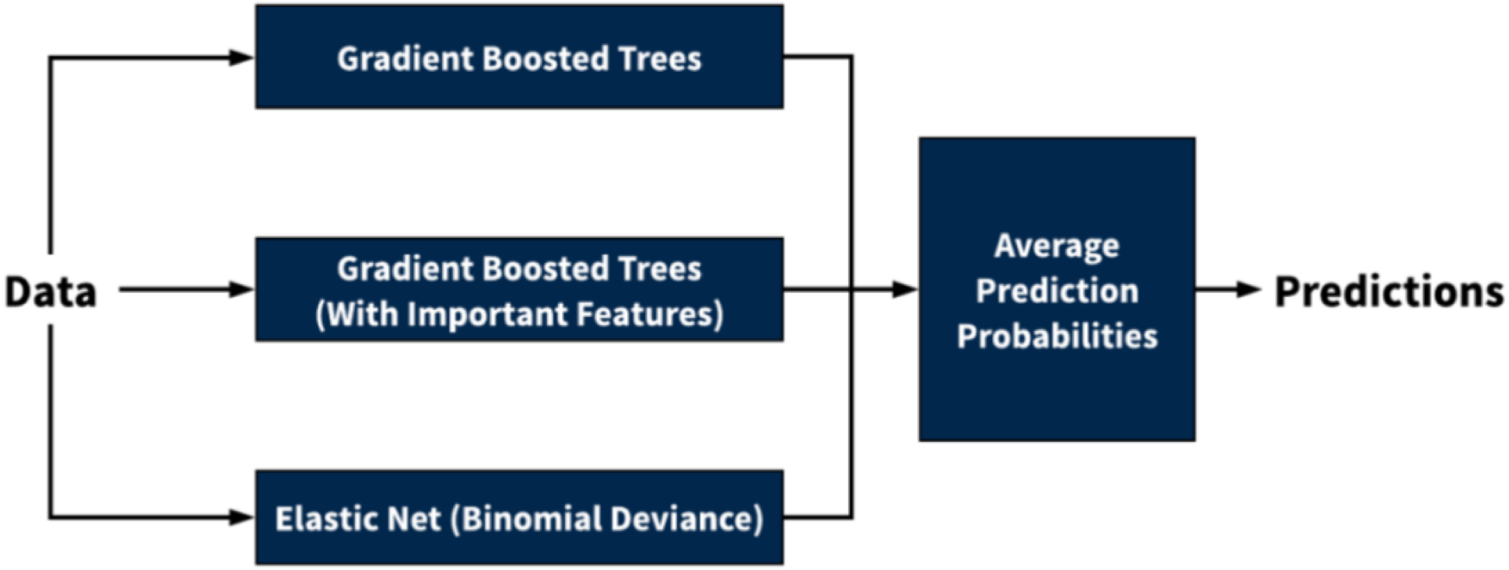
Cannot do it that way as data too large (13GB after encoding)

Empirical Results: Predicting LUACTRUU

- We use headlines from 2010-01-01 tot 2014-12-31 as the training set and headlines from 2015-01-01 to 2016-05-31 as the test set
  - More than a million LQD relevant headlines are selected
  - The fine-tuning process took more than 24 hours with a 20GB RAM GPU and then 4 hours to convert into numerical sentence embedding, where each embedding has a fixed length 768
- Data Pre-Processing
  - Use list of tickers (Bloomberg) to filter relevant news from 2010 to 2016
- Predicting LQD's spread changes
  - Haas GPU Server (20GB GPU RAM)
  - 1M+ headlines from 2010 to 2016
- Training Time
  - 1M+ News took more than 24hrs with a 20GB RAM GPU to fine tune a BERT base model (12 layers)
  - After Embedding, the embedded data (numerical features) is around 13GB
  - Average encoded vectors per day instead of training at news level (How we did for AAPL)

BERT Predicting Corporate Bond Index Changes

Downstream Models using BERT Embeddings



Downstream Model Performance

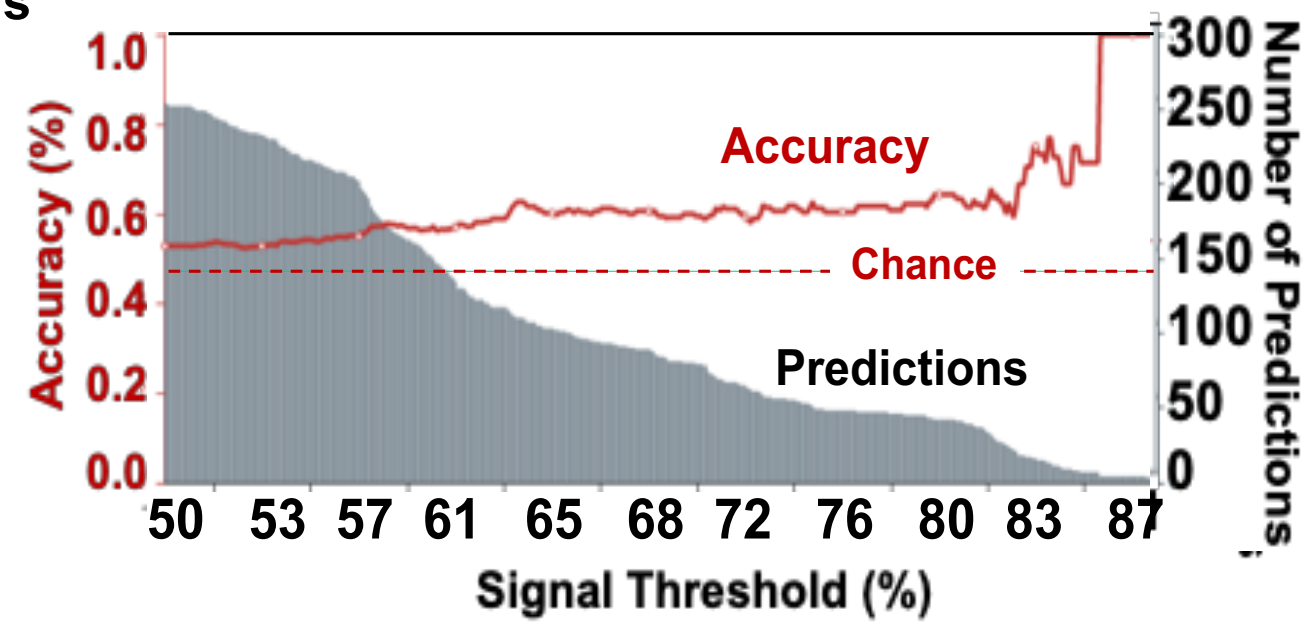
Model Name	Accuracy	AUC	True Positive Rate	True Negative Rate
Gradient Boosted Greedy Trees Classifier with Early Stopping	**0.564**	**0.6237**	**0.5633**	**0.5648**
RuleFit Classifier	**0.5724**	**0.6157**	**0.5829**	**0.5607**
AVG Blender	**0.5721**	**0.613**	**0.5924**	**0.5495**
Generalized Additive Model	0.5728	0.6037	0.6056	0.5363
Elastic-Net Classifier	0.5623	0.5997	0.5829	0.5394
Extra Trees Classifier (Gini)	0.564	0.5953	0.5943	0.5302
Decision Tree	0.5585	0.5782	0.5856	0.5282
Tensorflow Neural Network (1 layer, 128K hidden units)	0.5541	0.5756	0.6826	0.4109
Logistic Regression	0.5402	0.5752	0.5428	0.5373
Random Forest	0.5256	0.5346	0.5965	0.4464

Results: Sentiment Model versus BERT

- **The BERT model outperforms the original sentiment model (with 2 sigma data) with 1.2% more accuracy without thresholding**
  - **1% less accurate on positive predictions and 3% more accurate on negative predictions.**
- **Thresholds**
  - **60% Accuracy with 114 predictions; 70% with 17 predictions; 100% with 4 predictions**

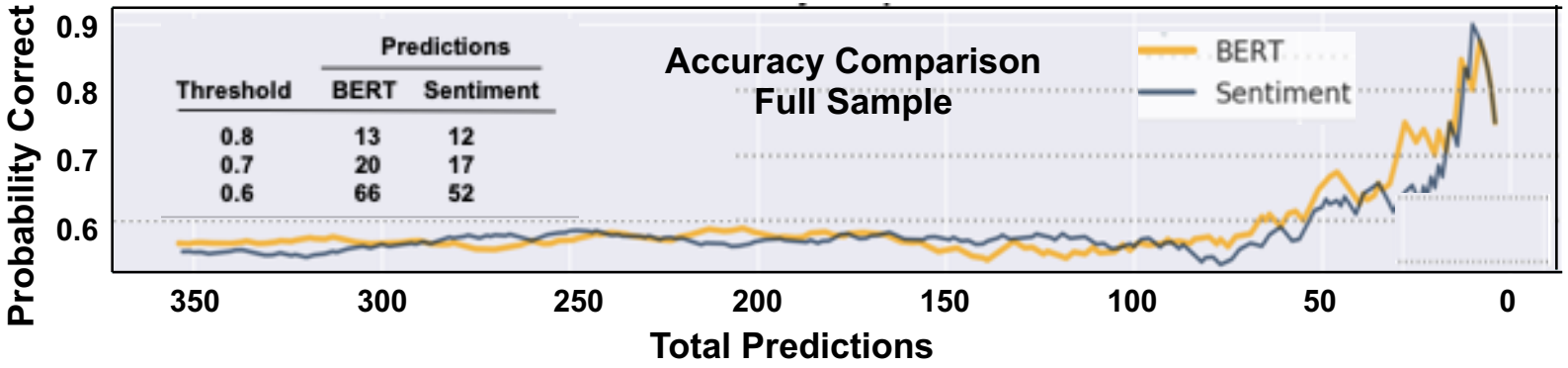
Performance : BERT vs. Sentiment Benchmark

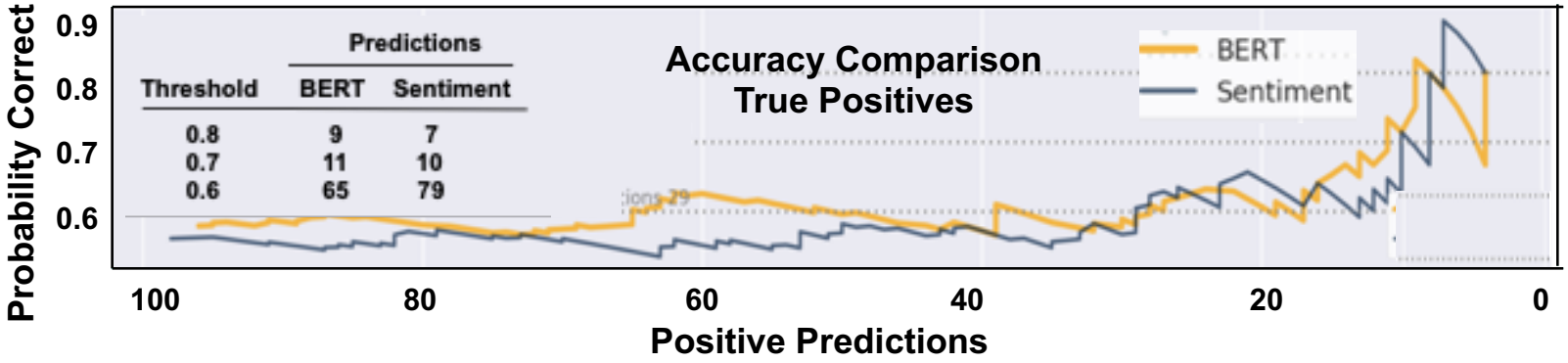
OOS Metrics	Sentiment	BERT
Accuracy	55.24%	56.50%
AUC	0.5717	0.5716
F1	0.5537	0.5575
Precision	56.00%	57.75%
True Pos Rate	54.75%	53.89%
True Neg Rate	55.75%	59.20%

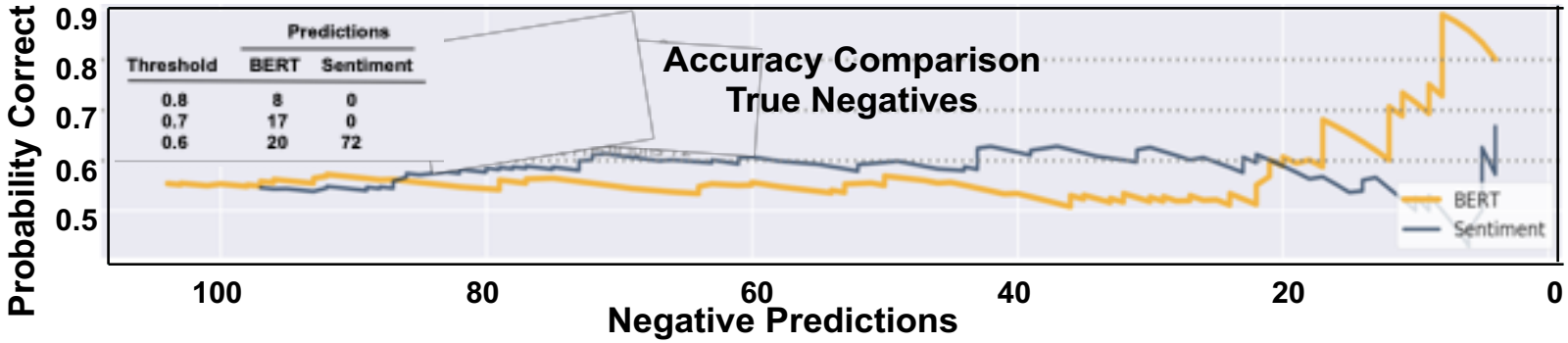


Results: Sentiment versus BERT (cont.)

- **BERT results are hitting thresholds faster in terms of Accuracy and TPR. In terms of TNR, the original model is much faster at 60% threshold but break afterward.**







Summary of BERT Model Performance

- **It is possible to predict one-day changes in corporate bond spreads at better-than-chance levels**
  - **Models based on 2 Sigma's sentiment data as well as the modified BERT model perform better than chance**
- **The BERT model outperforms the original sentiment model with 1.2% greater accuracy without thresholding**
  - **BERT is 1% less accurate on positive predictions and 3% more accurate on negative predictions**
- **By examining performance at probability correct thresholds of 60, 70, 80% Accuracy/TPR/TNR:**
  - **In Sample: BERT results are much smoother and always beats the original model**
  - **Out of Sample:**
    - **BERT results are hitting thresholds faster in terms of Accuracy and True Positive Rate**
    - **In terms of True Negative Rate, the original model the 60% threshold much faster, but underperforms the BERT model at higher thresholds.**

I thank Yiming Yu, Juntao Fang, Nathan Johnson and Teddy Legros from the University of California at Berkeley's MFE program for their important contributions to this project.

Machine Learning and Neural Networks in Finance

How AI/ML is Transforming Bond Markets

How AI/ML is Transforming Credit Markets

One of the last projects I worked on in the credit trading business was optimizing credit trading, inventory management, and building a credit trading robot.

- **Recently, we have made progress at developing a corporate bond trading robot (i.e. an algorithm that makes markets in corporate bonds**
  - **This was done using a deep learning neural network**
- **Typically, traders are concerned with making money on the bid/ask spread of the bonds that they trade, but this ignores other important considerations, many of which have only begun to be studied (by us).**
- **For example, these include:**
  1. **Client Accuracy by Holding Period: The previous study showed how we can predict bond price moves from client activity (we have other evidence as well). Should we bid aggressively for bonds that are going to decrease in price, and vice versa? In general, the answer is "no", but not always as it depends on other factors.**

How AI/ML is Transforming Credit Markets

2. **Bond Relative Value:** For example, if a client sells a bond and the trading desk holds it in inventory for one day, over 52% (of 200,000 trades) the bond will decrease in price. Furthermore, we find that, on average, clients buy the "cheap" bonds and sell the "rich" ones (which we know from the CaR strategy that cheap bonds richen and vice versa). Thus, bond relative value is an important consideration in what should be the bid/offer for a bond.
3. **Bond Liquidity:** We have analyzed how long on average bonds with given characteristics will remain in the trading desk's inventory (i.e., their liquidity). If a bond is hard to sell (or buy) and its price will go against us, we ought not bid aggressively for that trade.
4. **Net Inventory Position:** Trading desks get charged for the price volatility of their inventory as it uses the firm's capital. Thus, if the desk is net long and a customer wants to sell a bond, one should, all else equal, be eager to buy it as it makes the desk more neutral. Thus, it is important to consider how a bond will affect one's net holdings.

How AI/ML is Transforming Credit Markets

5. **Other Factors: There are other factors as well (bond volatility and suitability for inclusion in an ETF), but we can ignore those for now**

- **It is difficult, if not impossible, for a trader to take into account all these factors when faced with an RFQ (request for quote). However, a machine can do this.**
- **In fact, we have begun to take these factors into account to present to traders what we think is the optimal bid/ask for any given bond.**
- **Right now, we are only able to generate a "red" (do not bid aggressively) or "green" (bid aggressively) signal to the traders when an RFQ comes in. However, the ultimate goal is to actually set the bid/ask spread for the trader.**

How AI/ML is Transforming Credit Markets

AI and ML methods are becoming critical in bond trading activities. This has important implications for quants, traders, salespeople and fundamental analysts.

Quants

- The rise of machine learning is a watershed event for quants
  - However, new sets of skills are required and old ones are less critical
- Although it remains important for quants to know some things about quantitative finance, including derivatives pricing
  - These include term structure modelling of interest rates, options pricing, stochastic calculus and credit models,
- However, these will likely not be central to many jobs
- Successful quants will require expertise on machine learning methods of all types, along with an increasing reliance on statistics and inference methods
  - Experience and confidence in handline unstructured problems and related data will be in demand
- Strong computing science skills that focus on data storage and management will be required
- Finally, there will be no substitute for knowledge of the trading business.

How AI/ML is Transforming Credit Markets

Traders

- The role of the trader of the future will also change
- Traders will need to be cognizant of the many factors that affect the optimality of their trades and. For example, if a trader can not
  - They will be evaluated with respect to those factors
  - These include term structure modelling of interest rates, options pricing, stochastic calculus and credit models,
- However, these will likely not be central to many jobs
- Successful quants will require expertise on machine learning methods of all types, along with an increasing reliance on statistics and inference methods
  - Experience and confidence in handline unstructured problems and related data will be in demand
- Strong computing science skills that focus on data storage and management will be required
- Finally, there will be no substitute for knowledge of the trading business.

How AI/ML is Transforming Credit Markets

Salespeople

- **Salespeople will now be required to know their client in even greater detail**
  - **They will need to understand the client's profitability over time along with the characteristics of the bonds they tend to buy**
  - **Clients will reward them for keeping them out of trades that are unprofitable and showing them unskillful patterns of trading behavior**
- **Salespeople will need to know the characteristics of bonds they are selling**
  - **Is the bond trading "rich", "cheap" or "fair"**
  - **What is the probability that the trade will be profitable for the client? Over what horizon?**
- **Salespeople will be more proactive**
  - **If a bond becomes available that the salesperson thinks would be profitable for the client, they should contact the client**
  - **If the client bids for a bond not in inventory but there is a bond with similar characteristics in inventory, the salesperson should let the client know**

How AI/ML is Transforming Credit Markets

Fundamental Analysts

- **Machines will take over a portion of what fundamental analysts do**
  - **Systematic computer-based trading strategies have already become ubiquitous in the foreign exchange and equity markets**
  - **This is also moving to the bond market**
  - **The combination of advances in natural language processing and its relationship to market moves will provide competition to analysts from machines**
- **The ability to analyze sentiment data independent of a human will pose fundamental changes for analysts**
  - **There will be a greater demand for good research to feed those models**
- **Analysts will be more accountable**
  - **It will now become easier to track analysts forecasts for accuracy**
  - **Analysts will be evaluated as to whether they add useful information over what comes out on average**
  - **Analysts who are correct will be highly sought after for their opinions, but those that are poor will now be exposed and eliminated.**

References

Benzschawel, T. and Guth, S. *ATDN: Toward a Uniform Color Space*, Color Research and Applications 9 (3), pp. 133-141, 1984

Breiman, L. *Bagging Predictors,* Machine Learning 24 (2), pp. 123-140, 1996

De Ono, J. and Garrido, C. *Extracting the Contribution of Independent Variables in Neural Network Models: a New Approach to Handle Instability*, Neural Computing and Applications 25 (3-4):859-869 · September 2014

Freund, Y. and Schapire, R. E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". Journal of Computer and System Sciences. 55: 119, 1997

Garson, D. *Interpreting Neural-Network Connection Weights*. Artificial Intelligence Expert 6, pp. 47–51, 1991

Hahnloser, R, Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit". Nature. 405: 947–951, 2000

Hastie, T., Tibshirani, R. and Friedman, J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). New York: Springer, 2009

Klimasauskas, C. C. *Neural Networks: A New Technology for Information Processing,* Data Base 20, pp. 21-23, 1989

MacAdam, D. *Visual Sensitivities to Color Differences in Daylight,* Journal of the Optical Society of America 32 (5), pp. 247-274, 1942

References (cont.)

Mikolov, T., Sutskever, I,. Chen, K., Corrado, G., and Dean, J. Distributed Representations of Words and Phrases and their Compositionality. In: Advances in Neural Information Processing Systems, 2013

McCulloch, W. and Pitts, W. *A Logical Calculus of the Ideas Immanent in Nervous Activity,* Bulletin of Mathematical Biophysics 5, p. 155-133, 1943

Minsky M. L. and Papert, S. A. Perceptrons. Cambridge, MA: MIT Press, 1969

Moore, A. and Rayson, P. *Evaluation Metrics Matter: Predicting Sentiment from Financial News Headlines*, In: arXiv preprint arXiv:1705.00571, 2017

Olden, J. D. and Jackson, D. A. *Illuminating the "Black Box": A Randomization Approach for Understanding Variable Contributions in Artificial Neural Networks*, Ecological Modelling 154, pp. 135-150, 2002

Robbins, H. and Monro, S. *A Stochastic Approximation Method,* The Annals of Mathematical Statistics, 22, 3, pp. 400-407, 1951

Rosenblatt, Frank (1958), *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*, Cornell Aeronautical Laboratory, Psychological Review, 65, No. 6, pp. 386–408, 1958

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. *Learning Representations by Back-Propagating Errors,* Nature, 323, 533--536, 1986

Rumelhart, D., McClelland, J. and the PDP Research Group (Eds.): Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge, 1986

References (cont.)

Velay, M. and Daniel, F.. *Using NLP on News Headlines to Predict Index Trends*. In: arXiv:1806.09533v1, June 2018

Schumaker, R. and Chen, H. *A Quantitative Stock Prediction System Based on Financial News,* In: Information Processing and Management, pp. 571–583, 2009

Thompson-Reuters, *Thomson Reuters StarMine Quantitative Analytics,* 2016

Velay, M. and Daniel, F.. *Using NLP on News Headlines to Predict Index Trends*. In: arXiv:1806.09533v1, June 2018