Predicting FDIC Bank Failure

Mitchell Probst and Stephen Cowley

April 18, 2018

Abstract

Statistical methods have been commonly used to predict bank failure, but recent research has shown that machine learning methods can more accurately predict bank failure. We build upon previous research by applying predictive techniques to 11,977 banks and attempt to predict failure from 3 to 27 months prior to failure. We predict bank failure by using *call report* data provided by the FDIC to the public. We explain how recurrent neural networks (RNNs) work and compare them against random forests and logistic regression. Our results show that machine learning methods can accurately predict failure 15 or more months ahead of time, with an accuracy of 95.4% and a recall of 87.2%, regardless of bank size. This prediction window matches what we saw in the previous report "Analyzing Failed FDIC Banks."[1]

1 Introduction

Most major banks are insured by the Federal Deposit Insurance Corporation (FDIC). Being insured by the FDIC means that the institution pays an annual premium to the FDIC and adheres to specific guidelines to ensure that the bank is run by proper business principles. Banks want to be FDIC insured because it shows customers that their money is safe. Banks, just like any other business, can fail or go out of business. They are prone to failure due to various reasons such as poor management decisions or poor economic health of the nation.

When an FDIC insured bank fails, the FDIC assumes control of their assets and tries to find a different bank to absorb their customer accounts. Usually an acquiring institution will be offered a discount rate in order to incentivize them to do so. The FDIC pays the difference to settle the accounts and ensure depositors do not lose their money. The FDIC states that "No depositor has ever lost a penny of insured deposits since the FDIC was created in 1933".[2]

When the FDIC pays to settle accounts, most of the money comes from the premiums that banks pay. However, during a crisis (such as the 2007-2008 financial crisis), the cost of banks failing could exceed the funds available to the FDIC. When this happens, the remaining money needed is taken from the U.S. Treasury, which is money from taxpayers. Even though the FDIC tries to make transitions smooth for depositors, it can take up to 18 months for large accounts to be repaid.[3] For these reasons, it is critical to understand what causes bank failure and to be able to identify banks at risk as early as possible, allowing more time to prevent failure.

There are several research articles focused on using machine learning to predict bank failure. However, articles we found dealt with tiny datasets or were not clear how their data was organized and applied to machine learning techniques. We chose to use as much data as possible from the FDIC in an attempt to make our analysis as thorough as possible.

[1] https://github.com/Scowley4/PredictingBankFailure/blob/master/analyzing_failed_fdic.pdf

[2] https://www.fdic.gov/consumers/banking/facts/

[3] https://www.fdic.gov/bank/analytical/firstfifty/chapter5.html

One example of research includes a paper from Hong Hanh Le and Jean-Laurent Viviani[4] where information from 3,000 banks (though not necessarily FDIC insured) was used. Le and Jean-Laurent claim that they can predict failures with 75% accuracy using logistic regression, k-nearest neighbors, neural networks, and support vector machines. Unfortunately, the authors do not clearly define what time frame was used to achieve these results.

Another article written by Carmona, Climent, and Momparler[5] uses gradient boosted trees to predict failure. Though they achieve great accuracy, they limit their total dataset to about 150 banks, chosen specifically because their similar size.

We seek to expand on this research by predicting bank failure 3 to 27 months prior to failure by including banks of all sizes. We compare banks of previously incomparable sizes by engineering ratios for the data. Our research includes data from FDIC insured banks that have been around for at least 6 years, which totals to almost 12,000 banks.

Based on this previous research, we are confident that our analysis is possible and that it will prove to be useful in identifying struggling banks. Our project is focused on answering the following question: How accurately and in what time frame can we predict bank failure?

2 Data

The primary dataset that we will be using is obtained from and is publicly available on the FDIC website.[6] This dataset is a time series dataset that provides quarterly reports for financial institutions from 1992 to the present. This includes quarterly reports for the banks that end up failing. Failed banks are determined from a second smaller dataset[7] that contains each failed bank's unique identification number and a report at the time of failure.[8] As of 2017, there were 554 recorded bank failures.

There are over 900,000 available quarterly reports, spanning the course of 25 years (1992-2017). These reports "are the source of the most current statistical data available" for "monitoring the condition, performance, and risk profile of individual institutions."[9] Because the FDIC is a government run institution, all insured banks are required to submit these reports every quarter or face severe penalties.[10] Each quarterly report contains many *call reports* focusing on specific areas of a bank's performance. We focus on information included in both the "asset and liabilities" and the "incomes and expenses" *call reports*. These two reports together contain about 80 different metrics for assessing a bank's performance, including bank assets, liabilities, income, equity capital, and other financial metrics specific to banking. Because these metrics are strictly numerical values, in order to normalize our dataset to account for banks of different sizes, each metric was engineered as a ratio, such as liabilities divided by assets, or employee wages divided by total income. The above factors emphasize both the usability and reliability of this data for comparing banks of different sizes, although we still had to handle some incomplete data and throw out several metrics.

One key conclusion of the the aforementioned report "Analyzing Failed FDIC Banks" is that around 15 months prior to failure, there are noticeable trends in many key indicators of a bank's success. One such clear indicator is the ratio of net income to total income. On average, this ratio

[4] https://www.sciencedirect.com/science/article/pii/S0275531917301241

[5] https://www.sciencedirect.com/science/article/pii/S1059056017306950

[6] https://www5.fdic.gov/sdi/download_large_list_outside.asp

[7] https://www.fdic.gov/bank/individual/failed/banklist.html

[8] Only banks that failed from the year 2000 and onward are contained in the FDIC data

[9] https://www.fdic.gov/regulations/resources/call/index.html

[10] https://www.fdic.gov/regulations/laws/rules/2000-2600.html

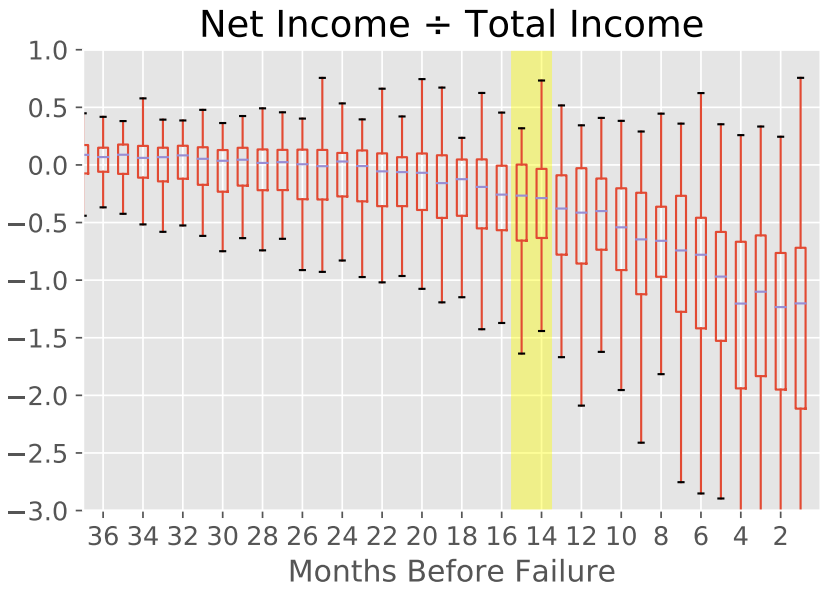

Figure 1: Box plot of net income to total income ratio for failed banks prior to their failure.

drastically drops in the months leading up to failure, as shown in Figure 1. This leads us to believe that prediction methods will be most effective between one and five quarters before failure (up to 15 months) because many other metrics also follow this trend. For more visual representations of the data and for more information about data acquisition, cleaning, and engineering, an interested reader can refer to "Analyzing Failed FDIC Banks" submitted December 2017 by the authors.[11]

2.1 Sequential Data

Traditional supervised time series data is organized where each row represents an object and the features describe observations for this object over some interval.[12] In our case, each object is a bank and each observation is a quarterly report for a specific bank. Normally this results in pairs of data of the form $(\mathbf{x}_i, y_i)_{i=i}^{N}$ and $\mathbf{x}_i \in \mathbb{R}^T$, where N is the number of banks and T describes the number of observations. In this case we could describe our input data as an an $N \times T$ matrix. However, for each quarter we were able to use over 90 metrics in analysis (those that came directly from *call reports* as well as other metrics engineered as discussed earlier). Because of this, a more natural way to represent our input data is a 3-Dimensional Matrix represented as $N \times F \times T$ matrix, where F is the number of metrics used for each report. We chose $T = 24$, meaning we use 6 years of quarterly reports. Banks that did not have 6 years of quarterly reports (either those that are too new, or those that failed before then) were excluded from the report. This leaves exactly 500 failed banks and 11,477 healthy banks.

An important note is that each bank is compared to the others strictly by their 24 most recent financial statements. This aligns all of the financials so each failed bank will have its "worst" quarterly report in the 24th period. Healthy, active banks have their latest quarterly report, December 2017, in the 24th period. It should be noted that some healthy banks in our data are inactive. Inactive banks chose to stop doing business but are not recorded as failures, so their latest quarterly

[11] https://github.com/Scowley4/PredictingBankFailure/blob/master/analyzing_failed_fdic.pdf
[12] https://link.springer.com/chapter/10.1007/3-540-70659-3_2

report might be from several years before 2017.

Setting up our data in this way enables us to use significantly more data than previous researchers have used. It also allows our classifiers to find patterns among banks over the course of several years. However, we acknowledge the limitation that by aligning our data in this way we are ignoring the effect that the overall economy may have had on banks.

3 Methods

There are numerous machine learning methods that could be utilized to try and predict bank failure. We chose to utilize three methods: random forests, logistic regression, and recurrent neural networks (RNNs). We will give a brief overview of random forest and logistic regression, although our primary focus will be on RNNs.

3.1 Random Forest Classification

A random forest hinges on the idea that several average classifiers can join together to form a good classifier. This is known as an ensemble method. A random forest is comprised of several decision trees that each use some element of randomness in constructing a decision tree to separate the data. By introducing randomness, each tree individually is sub-optimal, but it provides larger variability among the trees. This variability helps avoids overfitting. One of the greatest strengths of a random forest is its ability to highlight important features.

3.2 Logistic Regression

Logistic regression is a way to convert from a continuous regression problem to a classification problem by using the equation:

$$p(y = 1|\mathbf{x}) = \frac{1}{(1 + e^{\mathbf{w}^\top \mathbf{x}})},$$

where $\mathbf{w}$ is a weight vector learned when training data is run through the model. The resulting probability is then analyzed and classified according to a chosen threshold A natural choice is to split at $p(y = 1|\mathbf{x}) = 0.5$, but the threshold can be chosen at other values instead to adjust the precision and recall of the classifier. Another advantage of logistic regression is the ease of penalizing the model for overfitting by using L^1 or L^2 regression. These advantages come at a cost of ultimately being a "linear" model, which may fail to fully capture the complexity of our data.

3.3 Recurrent Neural Networks

Classification problems, such as image recognition, are typically seen as independent — the previous image in the dataset certainly does not influence the next image in the dataset. However, there are numerous problems where data is time-dependent. Predicting the next word in a sentence, for example, is obviously dependent (with varying importance) on each word that has been spoken up to that point. Speech recognition or text-to-speech problems are also examples of time-dependent data. The financial world is largely based on dependent events, such as using the previous price of a stock to predict future price, or in our case, using previous bank financials to predict future failure. Although the data is sequential, that does not mean there is a strong correlation for all time. Data from last quarter would certainly better predict the failure of a bank

tomorrow than a quarterly report from 5 years ago. Recurrent neural networks are made to handle this kind of data.

The following explanation assumes a base knowledge of feedforward neural networks, as well as a basic understanding of back propagation. Recurrent neural networks perform the same task on each element in the sequence, and then carry this information into the next time step. Common notation is for $\mathbf{x}_t$ to represent the input at time t, and s_t to represent the hidden information that the network has learned up to this point. The hidden information can be thought of as a sort of "memory" for the network, keeping track of what is important up to the current time step. This yields a natural way to calculate s_t:

$$s_t = \sigma\left(\mathbf{w}_x^\top \mathbf{x}_t + w_s \cdot s_{t-1}\right), \tag{1}$$

where σ is an activation function (like tanh or sigmoid) and $\mathbf{w}_x$ is the weight vector associated with $\mathbf{x}$, and w_s is the weight (scalar) associated with s. A value for s_0 must be chosen, but zero is a common choice. The weights, $\mathbf{w}_x$ and w_s, are independent of time, which is why we say that each layer of the RNN performs the same task for each element in the sequence: it uses the same weights for each step.

Many RNNs generate output at each step: $\hat{y}_t$. However, for our problem, we are actually only concerned with the prediction $\hat{y}_T$, which can be interpreted as whether or not the model thinks the bank will fail after using all information available. $\hat{y}_T$ is calculated by using the final hidden state along with its own weight, w_y:

$$\hat{y}_T = \varphi(w_y \cdot s_T),$$

where φ is another activation function. Using (1) to substitute in for s_T, and then s_{T-1} we have:

$$\begin{aligned}
\hat{y}_T &= \varphi\left(w_y \sigma\left(\mathbf{w}_x^\top \mathbf{x}_T + w_s \cdot s_{T-1}\right)\right) \\
\hat{y}_T &= \varphi\left(w_y \cdot \sigma\left(\mathbf{w}_x^\top \mathbf{x}_T + w_s \sigma\left(\mathbf{w}_x^\top \mathbf{x}_{T-1} + w_s \cdot s_{T-2}\right)\right)\right).
\end{aligned}$$

And so on, replacing each s_t using (1) until $t = 0$. A major issue that occurs with RNNs is something known as the *vanishing gradient problem*, which occurs when trying to minimize the loss function (see Appendix for details on the *vanishing gradient*). This motivated the creation of Long Short-Term Memory (LSTM) as a layer in RNNs to solve this problem.[13] An LSTM layer consists of several parts, an input activation (a_t) and three gates: input gates (i_t), forget gates (f_t), and output gates (o_t). There are variants to LSTM but in their basic form they follow:

$$\begin{aligned}
a_t &= \tanh(\mathbf{w}_a^\top \mathbf{x}_t + U_a \cdot \hat{y}_{t-1}) \\
i_t &= \sigma(\mathbf{w}_i^\top \mathbf{x}_t + U_i \cdot \hat{y}_{t-1}) \\
f_t &= \sigma(\mathbf{w}_f^\top \mathbf{x}_t + U_f \cdot \hat{y}_{t-1}) \\
o_t &= \sigma(\mathbf{w}_o^\top \mathbf{x}_t + U_o \cdot \hat{y}_{t-1}) \\
s_t &= a_t \cdot i_t + f_t \cdot s_{t-1} \\
\hat{y}_t &= o_t \cdot \tanh(s_t).
\end{aligned}$$

One can see that using this as a layer of the network avoids the *vanishing gradient problem*, as detailed in the Appendix.

3.4 Method Optimization

Setting up the data in a sequential format enables us to make predictions t quarters early by simply withholding the most recent t quarters of data from the classifiers. We trained each classifier to predict failure 1 to 9 quarters before failure.

Each of the previous methods have choices to make about the overall architecture of the data, commonly called hyperparameters. We attempted to find optimal hyperparameters for each method at each quarter by searching over a large space of possibilities for each one using 10 computers for several days. In the random forest classifier, predicting 18 months or less was best performed by trees with a depth of 4 where each split is determined by finding the optimal entropy among a random sub sample of 20% of the features. After 18 months the best forests were those that had a depth of 5 and the sub samples of 30% were taken. In logistic regression L^2 regularization with a newton solver gave the best results. However, the strength of the regularization changed at each prediction period, but in general it was between 0.1 and 100. Nueral networks have many hyperparameters describing the size of the network, and RNNs have even more hyperparameters that need to be chosen. Given the resources available, it is computationally infeasible to find the optimal hyperparameters for RNNs on this data.

4 Results

In our dataset there are about 23 times as many healthy banks as there are failed banks. This *class imbalance* means that using accuracy to judge our methods is impractical, because any method can achieve 95% accuracy by simply predicting that every bank is healthy. However, we want to be able to find the few that fail among the many healthy. Also, wrongly predicting that a bank is healthy when in fact they fail (false negatives) is considered much more costly than wrongly predicting that a bank will fail when in fact they are healthy (false positives).[14] For this reason we focus our results by judging the *accuracy*, *recall*, *precision*, *f1-score*, and *precision-recall area under the curve* for each classifier. Each is defined by numbers of True Positive, False Positive, True Negative, and False Negative.

- *Accuracy* is a general measure of how many banks were correctly identified. $\frac{TP+TN}{TP+TN+FP+FN}$
- *Recall* is the ratio of the number of bank failures correctly predicted out of the total number of banks that actually failed. $\frac{TP}{TP+FN}$
- *Precision* is the ratio of the number of bank failures correctly predicted out of the total number of banks that were predicted to fail. $\frac{TP}{TP+FP}$
- *F1-score* is the harmonic mean of precision and recall. $\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
- *Precision-Recall Area Under the Curve* (P/R AUC) is similar to a ROC curve (receiver operating characteristic curve), but instead of plotting true positives against false positives, it plots *precision* against *recall*.

The last two scores provide insight to the strength of a classifier at various thresholds, since there is a trade-off between *precision* and *recall*. Remember that in predicting bank failure *recall* receives more attention than *precision* because *recall* is maximized by minimizing false negatives, which are much more costly than false positives. Thus a good classifier is one that maximizes

[14] `https://tinyurl.com/predicting-bank-failure-1980s`

recall while still achieving high results in other categories, with 100% being perfect. The results are shown in Table 1.

Predicting failure 15 months early					
Method	Accuracy	Recall	Precision	F1-score	P/R AUC
Random Forest	95.4%	87.2%	47.6%	61.6%	74.7%
Logistic Reg.	95.4%	82.8%	47.4%	60.2%	59.0%
Recurrent NN	89.5%	71.3%	36.5%	48.3%	50.9%

Table 1: Results of methods that were obtained using 4-fold cross validation. P/R AUC is the Area under the Curve with Precision and Recall as the axes.

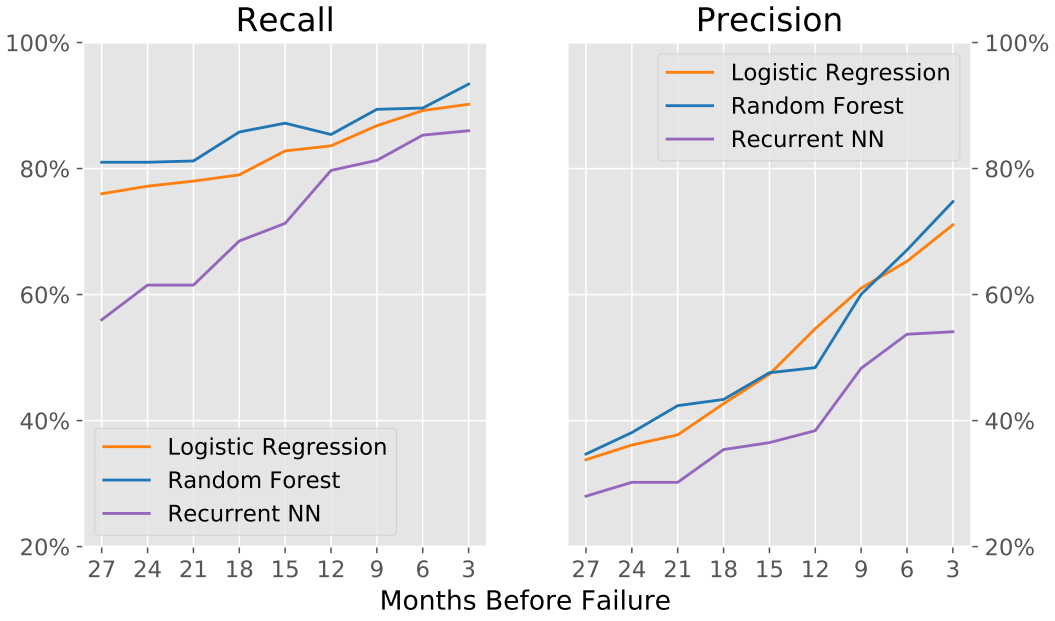

Figure 2: Both recall and precision increase as we approach the failure date.

5 Analysis

We see that our methods have significantly higher *recall* than *precision*. This corresponds to our earlier discussion that a false alarm (false positive) is preferred to a silent failure (false negative). A *precision* of 50% indicates that 1 out of every 2 banks predicted to fail will actually fail. A *recall* of 80% means that 4 out of every 5 bank failures will be found. Although it is always ideal to have higher *precision*, bank health is not actually a discrete class. Banks are not simply healthy or failing; there is a continuous spectrum that might describe the state of a bank. Thus, any bank that we predict failure for may actually be a bank that should be further inspected, because they may be operating sub-optimally.

We are unable to get classifiers that are good at both *precision* and *recall* 15 months prior to failure. This is a natural effect of the *precision*/*recall* trade-off; each method could be further tweaked to increase *recall* at the sacrifice of *precision*.[15]

[15] `https://en.wikipedia.org/wiki/Precision_and_recall`

The performance metrics in Figure 2 show (unsurprisingly) that as we approach the time of failure, the better each method is at predicting bank failure. This leads to a very interesting trade-off to consider: Accuracy vs. Actionable Time. Though we would like to be as accurate as possible, predicting the failure of a bank one quarter before it fails would probably not be helpful; the bank has likely already crossed a point of no return and will be unable to be saved. Thus, the earlier we are able to predict failure, the more actionable time a bank has to change practices and prevent failure.

Looking at the reports from 15 months prior to failure, our methods achieved 87% *recall* and 47% *precision*. This means that if our algorithm found 10 banks that are in danger of failing, in reality about 5 of them are. However, if this actually could be used to stop those 5 from failing, it is worth it to spend extra attention analyzing all ten of the banks. Our *recall* does show that over 10% of banks that fail would still go unnoticed, but this success rate is comparable to what was seen by other researchers, and in some cases even more favorable.

We report our best results as those generated by the random forest, followed closely by the logistic regression. Although we thought the RNN would outperform these other two methods given the time-series nature of the data, we encountered a computational barrier in searching hyperparameters for the network. Given more computational power, we expect all scoring metrics would increase and surpass the other classifiers. Having a strong indication that a bank is headed towards failure, 15 months prior may be early enough to engage in practices to correct negligent behavior or poor decision-making. The power of this detection is that, at subsequent quarters, the latest financial statement can be implemented in the model to see how recent changes affect predicted outcome.

6 Conclusion

We have seen that it is possible to predict bank failure several quarters before failure occurs. We are able to predict well at 15 months before failure, though predictive power continues to increase as we approach date of failure. Given that our method can be applied universally to any U.S. bank that has been operating for more than 6 years, this can be used as a powerful tool to assess a bank's health. Banks could individually assess their own health, or these methods could be applied by the FDIC as an alternative method to identify struggling banks. It would be easy to implement this method because the data we collected are already being reported by each bank; no new reporting methods need to be implemented to apply these methods.

There are many future research opportunities in this field to apply machine learning to bank failure. One suggestion would be to analyze different *call reports* in conjunction with the two main reports we used ("assets and liabilities" and "income and expenses") to see if there are other important variables not accounted for in these two *call reports*. Another option would be to examine the importance of features returned by a random forest classifier on the *call report* data and compare them to important features that are used in traditional statistical methods. Parallel to predicting bank failure, machine learning techniques could be applied to assess how much money it would cost the FDIC if a particular bank were to fail in the immediate future. Last, our results focused heavily on the fact that false negatives are more costly than false positives, but research also needs to be done to assess the actual cost of each of these errors which could then be used to alter the loss functions of our machine learning methods.

Appendix

Vanishing Gradient Problem

Because we are only concerned with predicting $\hat{y}_T$, we can use binary cross entropy, which for a single bank is given by:

$$\ell = -y_T \log \hat{y}_T - (1 - y_T) \log (1 - \hat{y}_T).$$

To illustrate the *vanishing gradient problem*, consider what happens when we try to minimize this function with respect to w_s:

$$\frac{\partial \ell}{\partial w_s} = \frac{\partial \ell}{\partial \hat{y}_T} \frac{\partial \hat{y}_T}{\partial s_T} \frac{\partial s_T}{\partial w_s}.$$

However, because $s_T = \sigma((\mathbf{w}_x^\intercal \mathbf{x}_T + w_s s_{T-1})$, we need to apply the chain rule for each s, yielding:

$$\frac{\partial \ell}{\partial w_s} = \sum_{t=0}^{T-1} \frac{\partial \ell}{\partial \hat{y}_T} \frac{\partial \hat{y}_T}{\partial s_T} \frac{\partial s_T}{\partial s_t} \frac{\partial s_t}{\partial w_s}.$$

But the partial derivative $\frac{\partial s_T}{\partial s_t}$ is actually itself a chain rule, for example:

$$\frac{\partial s_T}{\partial s_{T-2}} = \frac{\partial s_T}{\partial s_{T-1}} \frac{\partial s_{T-1}}{\partial s_{T-2}}.$$

Thus our true loss is actually

$$\frac{\partial \ell}{\partial w_s} = \sum_{t=0}^{T-1} \frac{\partial \ell}{\partial \hat{y}_T} \frac{\partial \hat{y}_T}{\partial s_T} \left(\prod_{m=t+1}^{\intercal} \frac{\partial s_m}{\partial s_{m-1}} \right) \frac{\partial s_t}{\partial w_s}. \tag{2}$$

The vanishing gradient occurs primarily because of $\left(\prod_{m=t+1}^{\intercal} \frac{\partial s_m}{\partial s_{m-1}} \right)$ in (2), because this partial derivative will include σ' (see Equation 1). Activation functions have derivatives of 0 on a large portion of their domain, which end up dominating the gradient. This results in virtually no contribution from steps that are farther than a couple of time steps back, which means that although we have defined a nice network to try and take advantage of sequential data, their contributions are minimal.

LSTM Gradient

If we let $U = [U_a, U_i, U_f, U_o]^\intercal$, (which is the LSTM equivalent of w_s) then we can compute the loss of our gradient with respect to U by using the following partials:

$$\begin{aligned}
\frac{\partial}{\partial U} a_t &= (1 - \tanh^2(\mathbf{w}_a^\intercal \mathbf{x}_T + U_a \cdot \hat{y}_{t-1})) \hat{y}_{t-1} \\
\frac{\partial}{\partial U} i_t &= \sigma'(\mathbf{w}_i^\intercal \mathbf{x}_T + U_i \cdot \hat{y}_{t-1}) \hat{y}_{t-1} \\
\frac{\partial}{\partial U} f_t &= \sigma'(\mathbf{w}_f^\intercal \mathbf{x}_T + U_f \cdot \hat{y}_{t-1}) \hat{y}_{t-1} \\
\frac{\partial}{\partial U} o_t &= \sigma'(\mathbf{w}_o^\intercal \mathbf{x}_T + U_o \cdot \hat{y}_{t-1}) \hat{y}_{t-1} \\
\frac{\partial}{\partial U} s_t &= \left[\frac{\partial}{\partial U} a_t \right] i_t + \left[\frac{\partial}{\partial U} i_t \right] a_t + \left[\frac{\partial}{\partial U} f_t \right] s_{t-1} + \left[\frac{\partial}{\partial U} s_{t-1} \right] f_t.
\end{aligned}$$

Then, as we attempt to minimize the loss function with respect to U:

$$\frac{\partial \ell}{\partial U} = \frac{\partial \ell}{\partial \hat{y}_T} \frac{\partial \hat{y}_T}{\partial U}$$

$$\frac{\partial \hat{y}_T}{\partial U} = \frac{\partial}{\partial U} \left(o_T \cdot \tanh(s_T) \right)$$

$$\frac{\partial}{\partial U} \left(o_T \cdot \tanh(s_T) \right) = \tanh(s_T) \frac{\partial}{\partial U} o_T + o_T \left(1 - \tanh^2(s_T) \right) \left[\frac{\partial}{\partial U} s_T \right]$$

$$\frac{\partial}{\partial U} \left(o_T \cdot \tanh(s_T) \right) = \tanh(s_T) \sigma'(\mathbf{w}_o^\intercal \mathbf{x}_T + U_o \cdot \hat{y}_{T-1}) \hat{y}_{T-1} + o_T (1 - \tanh^2(s_T)) \cdot \left(\left[\frac{\partial}{\partial U} a_T \right] i_T + \left[\frac{\partial}{\partial U} i_T \right] a_T + \left[\frac{\partial}{\partial U} f_T \right] s_{T-1} + \left[\frac{\partial}{\partial U} s_{T-1} \right] f_T \right)$$

The last piece of this equation (involving $\frac{\partial}{\partial U} s_{T-1}$) is still susceptible to the *vanishing gradient problem* (because s_t is defined recursively) and may collapse to 0, but the remaining components of the gradient are summed together, instead of being multiplied together like the product seen in Equation (2).