Macroeconomic Indicator Forecasting with Deep Neural Networks

Thomas R. Cook*† Aaron Smalter Hall†‡

Abstract

Economic policymaking relies upon accurate forecasts of economic conditions. Current methods for unconditional forecasting are dominated by inherently linear models that exhibit model dependence and have high data demands. We explore deep neural networks as an opportunity to improve upon forecast accurac y with limited data and while remaining agnostic as to functional form. We focus on predicting civilian unemployment using models based on four different neural network architectures. Each of these models outperforms benchmark models at short time horizons. One model, based on an Encoder Decoder architecture outperforms benchmark models at every forecast horizon (up to four quarters).

JEL Classifications: C45 C53 C14

1 Introduction

Macroeconomic forecasting methods have improved considerably over the last fifty years[1]. Current methods are capable of generating relatively accurate forecasts of

*Federal Reserve Bank of Kansas City Email: thomas.cook@kc.frb.org

†The views expressed in this article are those of the authors and do not necessarily reflect the views of the Federal Reserve Bank of Kansas City or the Federal Reserve System.

‡Federal Reserve Bank of Kansas City Email: aaron.smalterhall@kc.frb.org

[1]Diebold, 1997.

macroeconomic indicators. But these approaches bring a variety of undesirable properties, ranging from high sensitivity to model specification to high data requirements. There is considerable room to advance the state of macroeconomic forecasting by leveraging recent advancements in machine learning and neural networks.

Accordingly, we explore the use of neural networks in forecasting macroeconomic indicators, focusing specifically on unemployment. The models we present improve on the widely-cited SPF forecast in near-term prediction. Our best-performing model, based on an encoder-decoder architecture, outperforms the Survey of Professional Forecasters (SPF) at every forecast horizon. Additionally, our approach provides a number of advantages that lay the groundwork for developing new and increasingly accurate forecasts of unemployment and other economic indicators. The approach provides good single-series performance and can incorporate novel data if desired. Further, the approach is robust to network architecture (i.e. model specification).

1.1 Background: Extant Forecasting Approaches

The potential contribution of neural networks to forecasting is not entirely obvious without a better understanding of the dominant forecasting paradigms and their attendant challenges. Generally, approaches to macroeconomic forecasting can be divided into two paradigms: structural and non-structural[2].

Structural approaches to forecasting are those in which a theoretical model of the economy is written down and used as the basis for forming an empirical forecast. Put differently, it is an approach to forecasting in which economic theory guides the specification of the forecasting model. This approach to forecasting is particularly useful since it can be used to produce conditional forecasts – forecasts of macroeconomic indicators in response to a particular change in policy or institutional structure. Dynamic stochastic general equilibrium (DSGE) models dominate the contemporary structural approach.

The neural network models we present in this paper are not suitable as a replacement for structural models. It is worth noting, however, that neural networks are universal function approximations. Though beyond the scope of this paper, we would

[2]Diebold, 1997.

expect neural networks to prove useful in approximating DSGE model solutions[3].

In contrast to structural models, non-structural approaches to forecasting are those in which models or methods are chosen without explicit reliance on an underlying economic model. Instead, non-structural approaches tend to focus on particular properties of the data itself and the statistical models or methods that provide the best forecast of the data. Non-structural approaches tend to forego causal inference in exchange for predictive accuracy. Consequently, non-structural approaches are suitable for unconditional forecasting but are not suitable for conditional forecasting[4]. The models we present in this paper are suitable as a substitute for extant non-structural forecasts.

1.1.1 Vector Autoregression

Vector autoregressive (VAR) type models are one of the most common types of models used in non-structural forecasting[5]. These types of models are particularly appealing since they require the researcher to impose relatively few assumptions about the underlying data-generating process. Further, they allow for multiple series to be incorporated into analysis (unlike more restrictive ARMA models) and can be straightforwardly estimated using OLS/GLS procedures[6].

Vector autoregressive models are limited in a few crucial ways. First, VAR models are linear models. The estimation of a VAR model will not find or account for non-linearities when using normal estimation techniques. It is possible, of course, to explicitly model non-linearities but this requires that the researcher know precisely which adjustments to make to the model. Detection and correction for non-linearities increases in difficulty with the complexity of the model.

Second, and relatedly, VAR-type models are sensitive to proper specification. In addition to the appropriate handling of non-linearities, researchers must decide how many lags to include for each series, which series to include, how/whether to incorporate moving-average components, and how to accommodate cointegration, non-

[3]See Creel (2016) for some initial work in this area
[4]Lucas, 1976.
[5]Diebold, 1997; Pescatori and Zaman, 2011.
[6]see Sims, 1980

stationarity, and so on. These decisions force the researcher to take strong positions on the nature of the underlying data generating process. This is a problem because researchers choose non-structural forecasting approaches explicitly to avoid taking these types of positions.

1.1.2 Consensus Forecasting

Another popular type of non-structural forecast is the consensus forecast. Due to the various ways that a researcher might construct a forecast, there is variation from one researcher/company/agency's forecast to another; some forecasts are overly optimistic while others are overly pessimistic. For a given macroeconomic indicator, however, there is a tendency for the forecasts to cluster near the realized value. Consensus forecasts take advantage of this by aggregating many forecasts and producing a singular, aggregate, prediction. The central limitation of the consensus forecast, of course, is its reliance upon the accuracy of the individual forecasts that inform it.

For our purposes, the consensus forecast is meaningful for two reasons. First, consensus forecasts are useful as benchmarks for testing new forecasting methods. This is obvious when one considers the consensus forecast as representative of the general expectation among experts. But consensus forecasts are also useful benchmarks because they indicate a threshold for improving the accuracy of the consensus forecast itself. Put more simply, the consensus forecast improves in accuracy when incorporating new forecasts that are more accurate than the current consensus forecast.

Second, the SPF in particular, is meaningful because it reveals the extent of the variance in forecasts of macroeconomic indicators. The extent of the variation is indicative of uncertainty in the consensus forecast or, alternatively, imprecision in the measure of the consensus forecast[7]. The variation in individual forecasts thus serves as an additional, albeit informal, benchmark for evaluating new models.

[7]This follows if we view each individual forecast as a measure of a (latent) consensus-among experts forecast

2 Methods

To overcome the limitations of extant approaches to non-structural forecasting, we present a new forecasting approach, based on neural networks. Neural networks were originally developed in the 1950's, but have experienced several cycles of development and dormancy. Until the 2000's, neural networks were primarily the focus of computer science researchers.

Recent developments have led to the widespread application of neural networks in many fields. A combination several factors is responsible for this. Biological advancements in our understanding of neurons, and particularly the functioning of the visual cortex, inspired innovations in the modeling and structure of contemporary neural networks. Additional innovations have helped to improve the efficiency of training, restrict overfitting, escape local minima and otherwise make model training[8] a feasible task in applied settings. Advancements in computing technology such as the increased availability of massive data stores, increases in access to massive compute power via cloud service providers, and general advancements in the speed of computer hardware have made training complex neural networks computationally feasible.

2.0.1 Neural Networks: Key Concepts

A comprehensive overview of neural networks is well beyond the scope of this paper[9]. We are well served, however, by introducing some of the basic concepts of neural networks before proceeding to the advanced architectures we test in this paper.

Neural networks were developed in a tradition similar to graphical models. That is, instead of representing a model in an equation, or system of equations, models are represented as directed graphs. Typically, information flows from inputs to a target output through the structure of a graph. Graph nodes represent operations on the data as it moves from input to output. Any graphical model can be represented as a set of mathematical equations but the number and length of these equations increases dramatically as the complexity of the model increases.

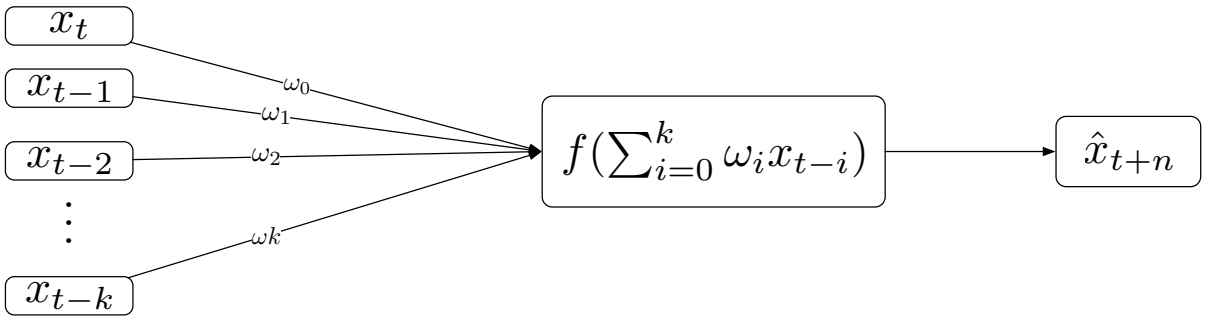
[8] In the literature on machine learning, model training is essentially synonymous with model fitting or model estimation as used in the econometric literature

[9] Interested readers should consult Gurney (1997), Kriesel (2007) and the vast array of online resources on this topic.

We tend to talk about neural network models in a way that is slightly different than the way we would ordinarily talk about an economic or forecasting model. Instead of focusing on the specification of model equations, neural network researchers tend to discuss model architectures. Architectures refer, generally, to the configuration of nodes on the graph, the interconnections among those nodes, and the nature of the operations preformed at each node.

The basic architecture upon which deep learning models are built is the perceptron[10]. A perceptron has three distinct sets of nodes: (1) a set of nodes representing model inputs, (2) a set of computational nodes, and (3) a set of nodes representing model outputs. We refer to each set as a layer. The defining feature of the perceptron is that only one layer of computational neurons is used to transform inputs to target outputs.

Figure 1: Precepteron Model Architecture



This figure provides a visual representation of a perceptron. As should be apparent, the model takes a linear combination of weighted inputs and applies a transformation function to produce an estimated output. In the case of this diagram, the inputs are k distributed lags of x and the estimated output is the value of x at n-periods after time t

A simple perceptron (suitable for forecasting) is provided in graphical form in Figure 1. It is notable that this model can also be represented in the familiar form:

$$\hat{x}_{t+n} = f(\mathbf{x}_{(k)}\boldsymbol{\omega}_{(k)}) \tag{1}$$

[10]See Minsky and Papert (1988) and Rosenblatt (1958) for foundational works in this area and Rojas (2013) for a more introduction to the mechanics of perceptrons

Where $\mathbf{x}$ is a vector of distributed lags (up to k-periods) of some macroeconomic indicator of interest. And where $\boldsymbol{\omega}$ is a corresponding vector of weights, which are adjusted through the training process. The product of these two vectors yields a linear combination of inputs, which is further transformed through some function f to forecast the indicator at some prediction horizon n periods in the future.

Readers should note that the structure of this model essentially follows the structure of a GLM model. The neural network approach does not, however, employ MLE (maximum likelihood estimation) for estimation. Instead, we rely on the back-propagation training algorithm[11], which is essentially nonparametric. Through the training process, the model will identify which features and parameters(i.e. computational nodes) are relevant for prediction[12]. This enables us to be less selective about what data we supply to a model and less concerned with how we should pre-process that data[13].

Perceptrons give rise to powerful modeling architectures when they are layered, stacked or otherwise connected into massive network-model architectures. We explore these below.

2.1 Architecture

We present four neural network models, each built on a different architecture. We chose these architectures because they represent some of the most commonly used architectures in the deep learning research. Further, we chose these specific architectures because of their relationships to key aspects of time-series forecasting.

[11] Rumelhart, Hinton, and Williams, 1986.

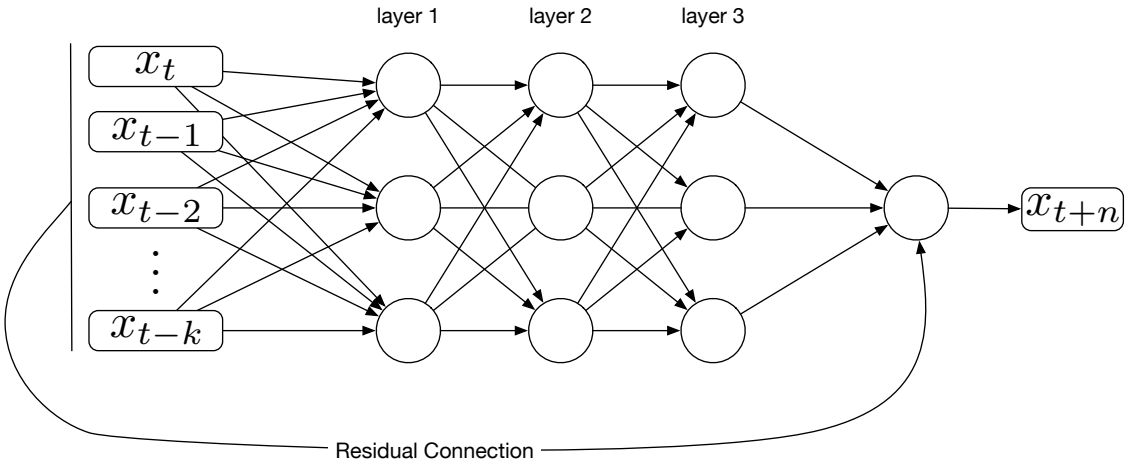
[12] As identified by recent work, this amounts to implicit regularization. See Gao and Jojic (2016), and relatedly, Ye (1998)

[13] Alternatively stated, we can provide the model with high complexity and many variables and the model will, to some extent, self-regularize. Some variables will be ignored by the model and some computational nodes will 'die', leaving only the relevant data for use by the trained model.

2.1.1 Fully Connected Architecture

The first network architecture that we apply to forecasting unemployment is a feed forward[14], fully connected (FC) network. This is a type of neural network architecture which consists, simply, of several stacked layers of computational nodes. Each layer contains one or more perceptron-like nodes. The inputs to each node are the outputs from each of the nodes in the preceding layer[15]. The essence of this architecture is provided in Figure 2.

Figure 2: Fully Connected Architecture



This is a stylized portrayal of a basic feed-forward, fully connected network. Each circle represents one computational node and each computational layer consists of three computational nodes. A residual connection indicates the combination of the original input with the output of layer 3.

Beyond the notable inclusion of additional computational layers, the fully-connected architecture we test here includes a residual connection between the inputs and the outputs from the final computational layer. In Figure 2 this is represented by the line labeled "Residual Connection". The node connecting the inputs to the outputs from the last computational layer performs element-wise addition between the layer outputs and the inputs. The addition of the residual connection is consistent with

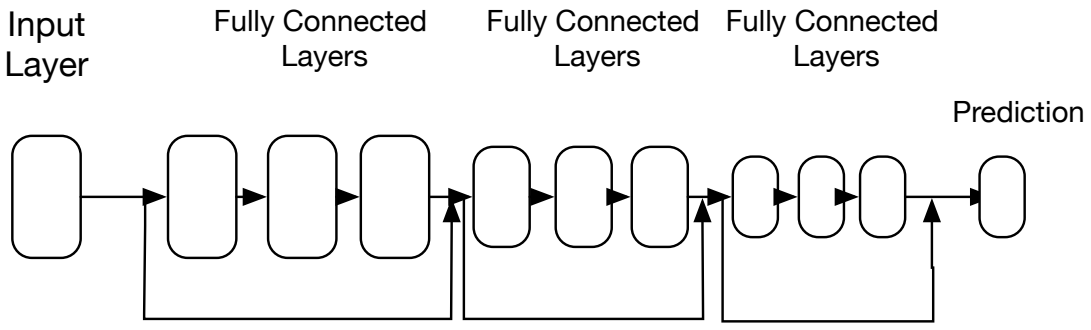
[14]The model is feed-forward in the sense that data only moves in one direction through the model. This is to be contrasted with the recurrent-based architectures discussed later (LSTM and encoder-decoder architectures)

[15]It is precisely because every node in one layer is connected to every node in the subsequent layer that we call this type of network *fully connected*

recent developments in model architecture[16].

Figure 2 portrays a stylized version of the model. In application, the model contains far more layers. Further, in the model we test in this paper, we stack the basic structure portrayed in Figure 2 (several fully connected layers and a residual connection). In each stack, the number of number of nodes per fully-connected layer is reduced. A more accurate illustration of the model, as tested is provided in Figure 3.

Figure 3: Feed Forward (fully-connected) model architecture



This figure portrays the model as tested. Each rectangle consists of a layer of neurons. Layers between the input layer and the prediction layer consist of computational nodes. The size of the rectangles are scaled to suggest the number of nodes in each layer. Moving from left to right, each layer of computational nodes is smaller or equal in size to the layer that precedes it.

[16]He et al., 2015.

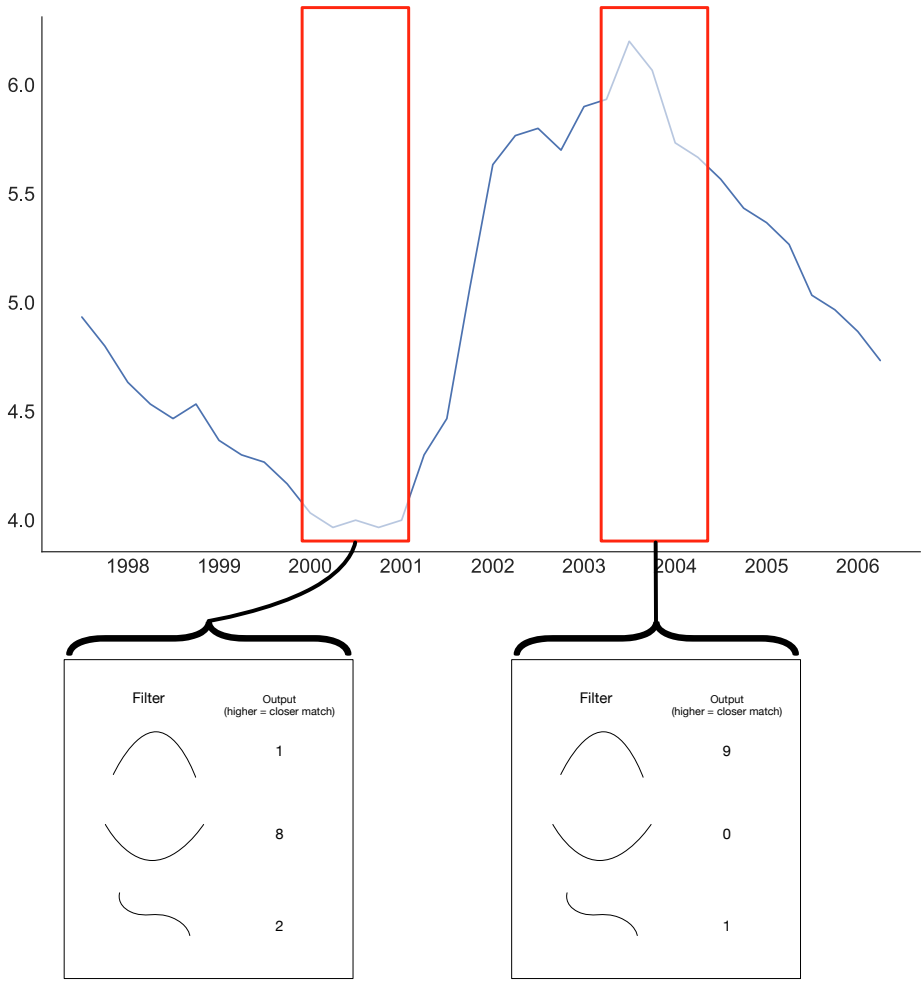
2.1.2 Convolutional Neural Network

The second architecture we present is a convolutional neural network. Convolutional networks are feed-forward in that data only moves in one direction through the network. Their development arose with research on the use of neural networks for image recognition. In a convolutional layer, each input to the layer is connected to only a few computational nodes (unlike the fully-connected setting, where each input is connected to every computational node). Conversely, each computational node only receives inputs from a small, clustered set of the input nodes. Further, the weights on inputs to the computational nodes are shared across all computational nodes in the layer. If we conceive of the input data as an image, then the convolutional layer is a sliding window across the data that filters for a specific pattern in the data. The convolutional architecture identifies which patterns maximize predictive accuracy and searches for those patterns in the data[17]. A convolutional layer can apply several filters concurrently. A stylized illustration of a convolutional layer in operation is illustrated in Figure 4.

The convolutional model we build contains a few notable components. As with the previous model, we stack several convolutional layers together. Between each stack, we include residual connections. The layers in each stack are smaller than the layers in the stack that precede it. The reduction in nodes between convolutional stacks is accomplished by applying a max pool filter. After several stacks of convolutional layers, we append stacks of fully connected layers. These stacks also contain residual connections. Also, the layers in each fully connected stack have fewer nodes than the preceding stack. The output from the final stack produces a prediction of the target variable. A diagram of the convolutional model is provided in Figure 5.

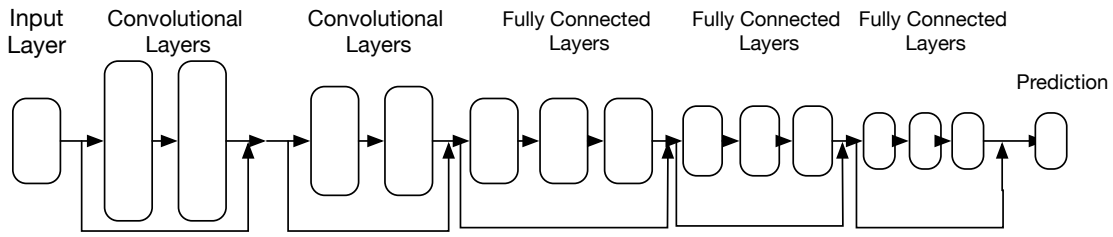
[17] Cui, Chen, and Chen, 2016; Nouri, 2014.

Figure 4: Stylized depiction of application of convolutional filters



The chart depicts the unemployment rate over the period 1998 to 2006. The red rectangles overlaid on the chart at different points in time depict the movement of the convolutional kernel (i.e. the sliding window) over the data. Callouts attached to each red rectangle depict several filters applied by the convolutional layer. The output for each filter is a score indicating the extent to which the data under the kernel matches the shape of a that particular filter. Under the first rectangle, the data most closely matches U-shaped filter, while the data under the second rectangle most closely matches the inverted U-shaped filter.

Figure 5: Convolutional Model Architecture



2.1.3 Recurrant Network Architecture

The third architecture we present here is a recurrent neural network (RNN) architecture. RNN models developed along with research on the application of neural networks to language parsing and translation. Similar to the convolutional architecture, the RNN architecture draws information from the temporal structure of the input data. Specifically, these types of networks draw upon the sequence in which the input data is presented to the model. Recurrent neural networks do this by accepting input not only from the current input in a sequence but from the state of the network that arose when considering previous inputs in that sequence. To describe this more simply, RNNs have *memory*.

There are many variants of RNN architectures. In this paper, we focus on a particular type called a long short term memory (LSTM) networks. This type of RNN architecture is notable because it has the capacity to store a long-running memory about the sequence along with short-run memory of the most recent network outputs. Consequently, this allows the network to draw upon broad contextual features in the data (long-run memory) as well as information provided by only the most recent elements in a sequence (short-run memory).

A verbose illustration of the LSTM architecture is provided in Figure 6. As illustrated, each input x is fed into a computational node (we might think of this as a 1-node layer), which also accepts inputs from the output of the preceding layer[18] (denoted s_i, h_i). The term s_i represents the state of the network at the ith member of the sequence. The state is the long-running memory (or understanding) of the sequence, as informed by the elements of the sequence to which the network has been exposed. The term h_i is the prediction (or output) of the layer that corresponds to a given element (i) in the sequence. Notably then, we can understand this architecture as consisting of many layers and having the following properties: each layer corresponds to a particular element in the sequence; each layer receives the network's long-run understanding of the sequence *so far*; and each layer receives the output generated from the previous element in the sequence.

A more concise representation of the recurrent architecture is provided in Figure 7.

[18]The exception to this is the first layer, for which values s_0 and h_0 are initialized to random values and zero, respectively.

This "rolled" version of a recurrent (LSTM) architecture represents the recurrent architecture as a "cell". The arrow from the cell to itself indicates the feedback loop created as the output from one element of the sequence is taken as input along with the next element in the sequence.

Figure 6: Diagram of an unrolled recurrant network

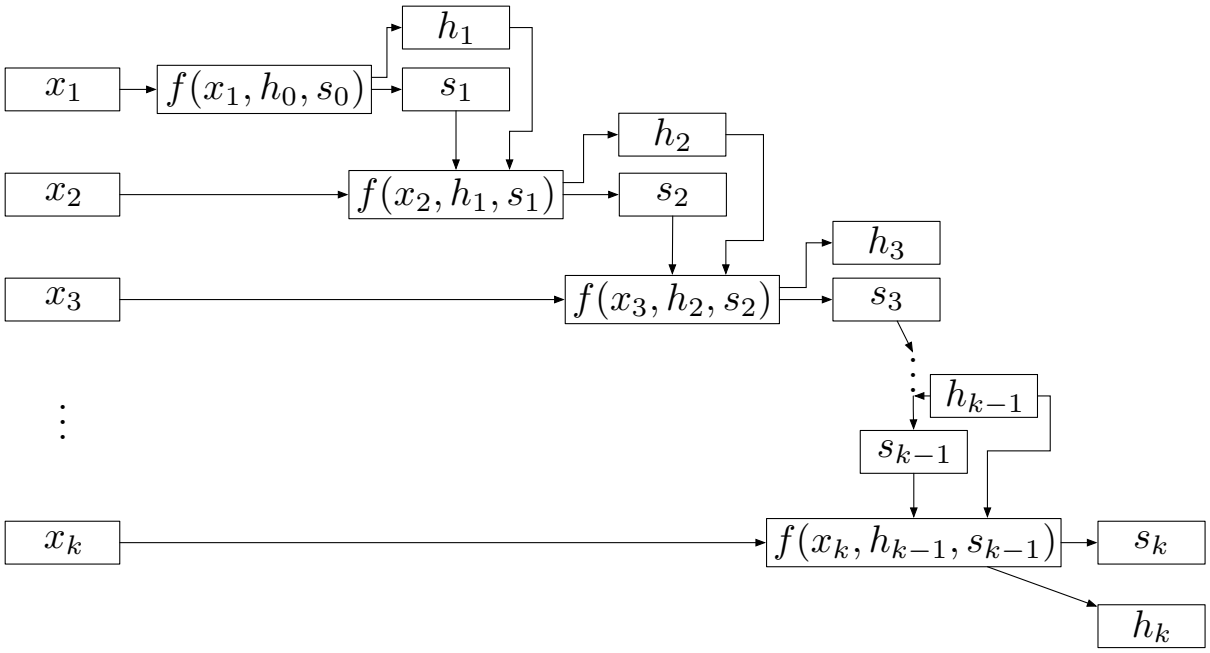
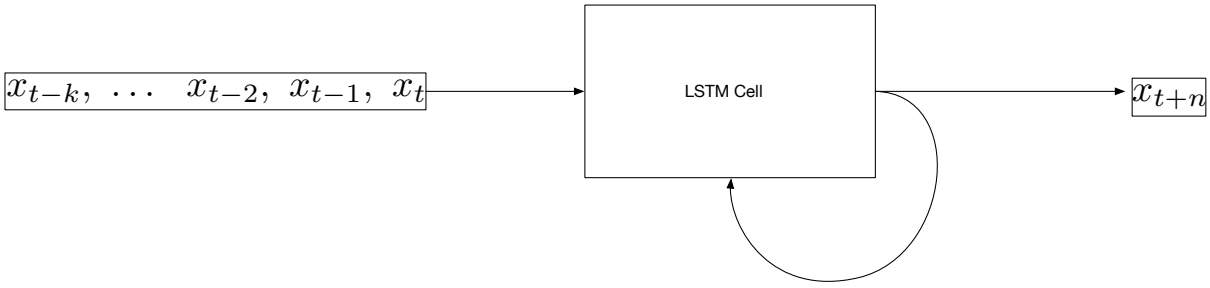
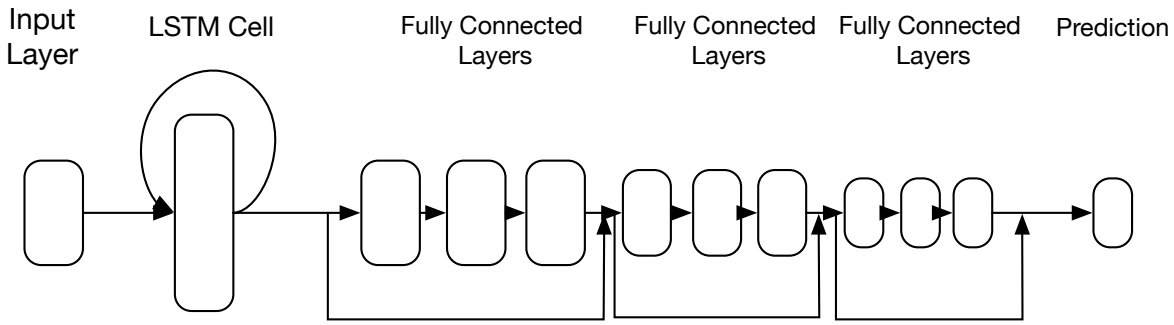


Figure 7: Diagram of rolled recurrant network



We build a model around the LSTM architecture and test it in this paper. A diagram of the model is presented in Figure 8. In addition to the LSTM cell, we add several stacks of fully-connected layers (with residual connections). These stacks are

Figure 8: Complete of LSTM model as tested



identical in structure to the stacks that appear at the end of the convolutional model. Their central purpose is to fine-tune the predictions from the LSTM cell and convert the output into a shape that is appropriate for prediction.

2.1.4 Encoder-Decoder Network

The last type of architecture that we employ here is an extension of the LSTM architecture discussed above. It is called an encoder-decoder architecture[19]. It is a member of a broader class of networks called *sequence to sequence* models. The encoder-decoder architecture was initially developed to facilitate language modeling. Specifically, it was developed to allow a model to predict words in the output while considering the context of individual words in the input along with the context of the words that have already been predicted in the output.

The architecture is comprised of two components, aptly named the encoder and the decoder. The encoder is a LSTM architecture (or similar RNN architecture). As discussed in section 2.1.3 this portion of the model will consider each portion of a sequence of data as well as the response of the encoder to the preceding portions of the sequence. The decoder module is also an LSTM architecture (or similar). It will consider each portion of the sequence output by the encoder. The decoder also considers its own responses to earlier portions of the encoder-produced sequence. In other words, the decoder allows for the model to make predictions that fit with the

[19]Cho et al., 2014; Sutskever, Vinyals, and Le, 2014.

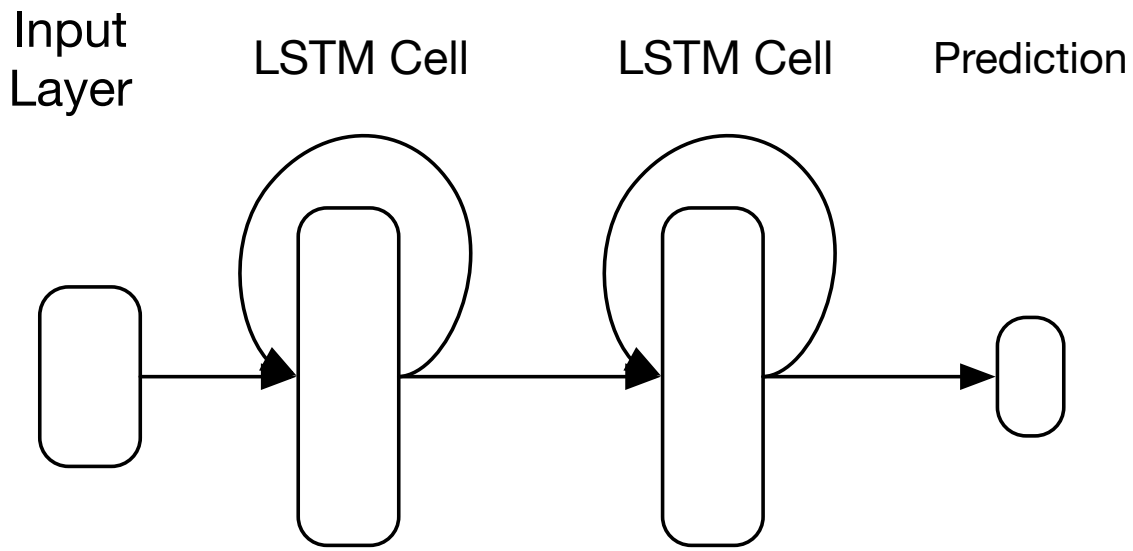
context established its earlier predictions. In its use in language models, this allows for the model to, for example, ensure that the model's response is grammatically appropriate (e.g. that subjects and verbs agree).

When used in a forecasting model, the decoder module assists with predicting long forecast horizons. The model discussed in the previous section forces an LSTM module to produce predictions of a desired horizon in a single step. When the prediction horizon is one, the LSTM module is tasked with predicting the next element in the sequence. When it is two, the module skips any consideration of the next element in the sequence and attempts to predict the second-to-next element in the sequence, etc. We expect the LSTM module to be generally accurate at predicting the next, or nearly next element in a sequence and less accurate when forced to skip several elements in the sequence. By adding a decoder module, the encoder module can, essentially, produce a one step ahead prediction, and the decoder module can iteratively extrapolate that one step prediction out to the desired prediction horizon.

The encoder-decoder architecture that we use for testing is diagrammed in Figure 9. As discussed above, it is essentially two LSTM networks stacked on top of one another. The first network takes a sequence of input data and produces a representation of it. The output representation is fed into the second network, which is initialized with the same *state* as the the final *state* from the encoder module (i.e. the first LSTM cell)[20]. Note that, unlike the convolutional model and LSTM model presented above, this model does not contain a stack of fully connected layers at the end. While we certainly could add the layers, we expect the decoder module (the second LSTM layer) to actually perform the function of fine-tuning that the fully connected layers would otherwise provide.

[20]Initializing the decoder with this state provides additional summary information about the encoded representation to the decoder module and is consistent with extant literature on encoder-decoder modules.

Figure 9: Encoder-Decoder Diagram



2.2 Performance goals/constraints

In section 1.1, we refer to a number of limitations, or challenges to extant forecasting models. These included constraints on functional form and sensitivity to proper specification.

The neural network approach is, by construction, not subject to the limitations of VAR models; we thereby establish the usefulness of this approach by demonstrating the competitive performance of these models. Further, as noted above, the SPF provides us with two benchmarks for model performance: mean absolute error across individual forecasts, and the variance across individual forecasts.

We will test our model's performance by forecasting unemployment. Our performance goals are for the models to provide forecasts with lower mean absolute error than the SPF forecasts and for models to exhibit lower variance in repeated runs than the variance across individual forecasts in the SPF. Achieving these goals for each of the models presented above suggests an additional benefit to the neural network approach: robustness of the approach to model architecture.

3 Data

To test the performance of the neural network approach, we trained each of the models presented above to predict the civilian unemployment rate ($UNRATE$). This measure is collected monthly by the US Bureau of Labor and Statistics[21]. It measures percentage of the labor force that is currently unemployed. The civilian unemployment rate only measures unemployment in the US. At the time of this writing, data for the $UNRATE$ is available as far back as 1948, and as recently as last month.

We chose to test our model on unemployment forecasting (as opposed to another macroeconomic indicator) for a few reasons. First, unemployment is a substantively meaningful indicator to forecast: the Federal Reserve promotes maximum sustainable employment as part of its monetary policy mandate[22], and it is closely monitored by

[21] *Civilian Unemployment Rate [UNRATE].*

[22] `https://www.federalreserve.gov/aboutthefed/files/pf_3.pdf`

economic actors and scholars across a variety of sectors.

Second, unemployment usually undergoes limited revision after its initial release. This is an important consideration since it allows us to generally sidestep the problems of collecting and assembling appropriate 'vintages' of the data. We use the last release of $UNRATE$ for all training and testing. To be clear, the largest discrepancy between the original vintage of the data and final release of the data is about 23 basis points with the average discrepancy being 9 basis points. We assume the impact of these discrepancies on the predictive accuracy of our forecasts to be negligible.

We target 0, 3, 6, 9 and 12 forecast horizons for $UNRATE$, consistent with the forecast horizons for SPF. For each forecast horizon, we train each of the three models presented above, yielding 20 total model variants for training.

The civilian unemployment rate is the sole series used as input for each of the models. For each observation, the model inputs are the previous 36 monthly values of $UNRATE$, along with first and second order differences in $UNRATE$. In theory, the model could identify and extract the first and second order differences of the input data, but we supply them directly because we are fairly certain that they will supply the model with useful information and because it allows us to reduce the training time and simplify the model structure.

It is possible and relatively easy to add additional series to these models and we would expect performance gains from doing so. We refrain from adding additional series in this paper to allow us to simplify our discussion of the model and to present models that are easily transferable to different contexts (e.g. regional/state level prediction).

3.1 Model Training

We construct a training dataset from $UNRATE$ data from 1963 to 1996. Every tenth observation in this period is sequestered into a validation dataset. We use the validation dataset to evaluate the performance and accuracy of the model over the course of the training process. The remainder of the data, from 1997 to 2014 onwards, is sequestered into a testing dataset. We use this dataset to evaluate the performance of the trained model. We chose this time-period for the testing dataset because it is

the time period that the SPF considers when reporting their error statistics.

The training process is subject to stochasticity. The initial weights for each model network are randomly distributed. Additionally, we implement a regularization technique called 'dropout' in which, for each step in the training process, the output of a randomly selected subset of nodes are ignored. This limits the over-dependence of the model on any one node and thereby reduces the potential for overfitting. Beyond this, there are a few other sources of stochasticity in the training process including the optimization routine itself (a variant of mini-batch stochastic gradient descent).

As a consequence of the stochasticity inherent to the model training process, repeated runs of the same model will yield trained networks that vary in their weights and, consequently, in forecasts. To accommodate this variance, we train 30 instances of each model. This allows us to assess expected model performance as well as assess the variance in performance across repeated runs of the same model.

4 Results

Model performance is provided in Table 1. Each of the first three columns describe the performance of a model in terms of mean absolute error (MAE), aggregated across repeated iterations. The mean MAE indicates the average model performance. The standard deviation of the MAE gives some sense of the distribution in model performance across repeated trainings of a model. The final two columns provide performance metrics on two benchmark models.

The first benchmark model is a directed[23] autoregressive model (DARM) that uses monthly data. The model is specified as follows:

$$\widehat{UNRATE}_{t+n} = \sum_{i=1}^{k} \beta_i \, UNRATE_{t-i} \tag{2}$$

Where t indexes the time of forecast, k is the number of lags, and n indicates the forecast horizon. In this paper, we use the DARM model estimates published by the

[23]This is to be contrasted with an iterative model, in which the next-step-ahead is forecast and then iterative extrapolation is used to generate a prediction for the desired forecast horizon.

SPF[24].

The DARM is the best-performing benchmark used by SPF in their model assessments[25]. Generally speaking, each model performs favorably compared to the DARM, with the exception of the Convolutional and LSTM models, which perform worse than the DARM model at the third and fourth quarter prediction horizons.

The last column characterizes the performance of SPF participants. For each participant (i.e. each company/agency/forecaster that submits forecasts to the SPF), we calculated the MAE of their forecasts on the years 1997-2015 for 0-4 quarter prediction horizons. Accordingly, the minimum and maximum MAE reported in the fourth column of Table 1 indicates the MAE of the best and worst *individual respondent* at a given time horizon. The mean indicates the average individual performance, and the standard deviation indicates the variation across individual respondents to the SPF. A visualization of these results is provided in Figure 10[26].

Table 1: Performance metrics for SPF, DARM and neural network models at 0-4 quarter prediction horizons

Horizon		Fully Connected	CONV	LSTM	Encoder Decoder	DARM	SPF
0 Months	Mean MAE	7.6	13.4	10.4	4.4	11.7	13.5
	St. Dev.	2.7	5.4	05.8	0.2		6.3
3 Months	Mean MAE	25.3	25.7	27.3	018.4	32.8	27.1
	St. Dev.	2.9	3.5	2.8	0.1		9.3
6 Months	Mean MAE	44.1	48.3	47.3	30.5	49.3	41.2
	St. Dev.	4.1	3.1	6.6	0.5		15.7
9 Months	Mean MAE	63.8	80.4	74.8	46.1	65.8	56.8
	St. Dev.	4.8	4.7	11.0	0.7		22.8
12 Months	Mean MAE	87.0	111.6	101.7	62.0	90.7	72.0
	St. Dev.	4.9	5.7	20.7	0.6		29.1

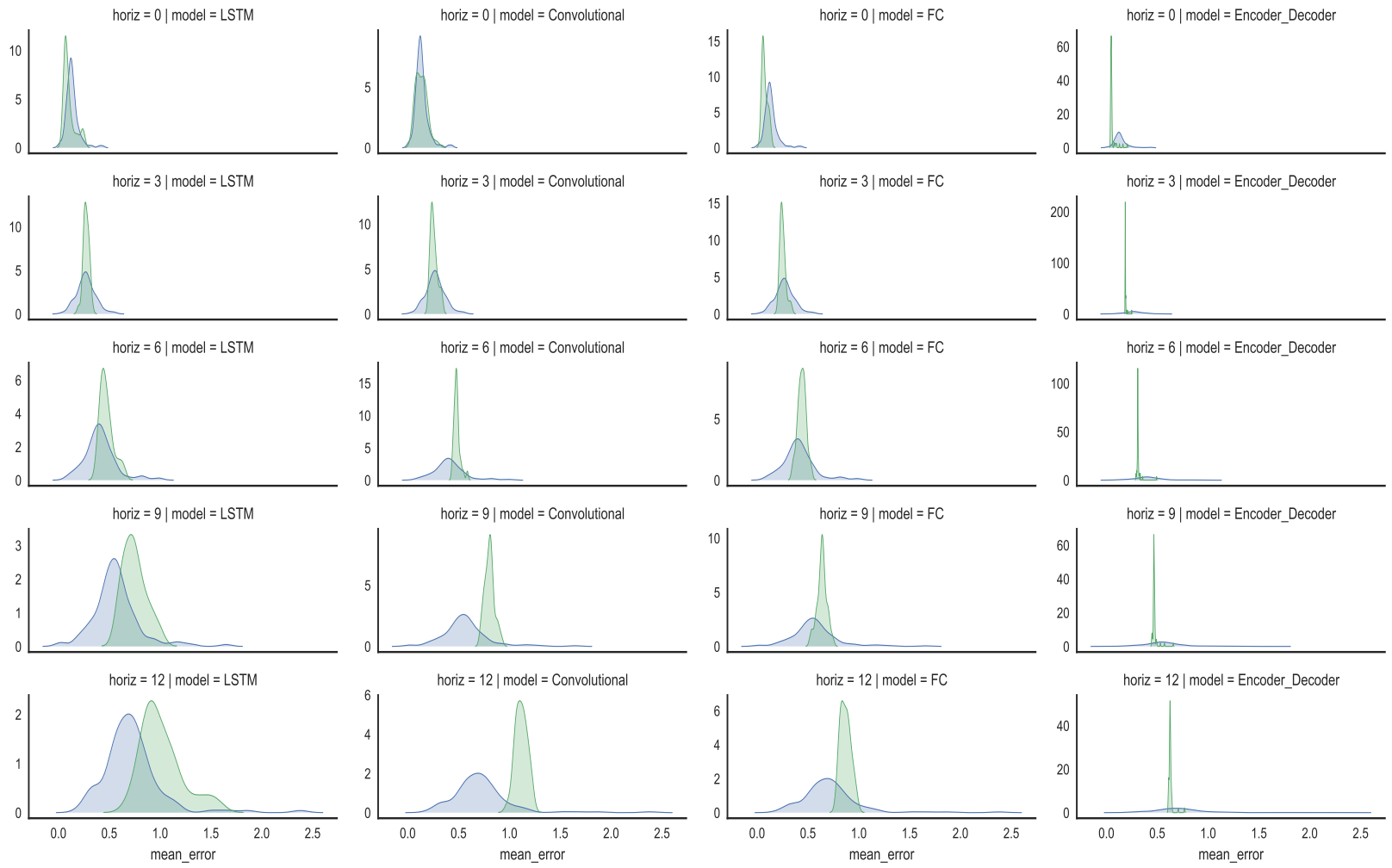
All metrics presented in basis points.

[24]Stark, 2017.

[25]Ibid.

[26]Additional visualizations of the distribution of errors for both SPF participant responses and the neural network models tested are provided in the Appendix, in Figure 13 and Figure 14.

Figure 10: Distribution of mean absolute error across network architectures at various horizons.



Each facet in this figure compares the distribution of mean absolute error across SPF participants (green) with the distribution of mean absolute error across repeated runs of a network architecture (blue), at a given forecast horizon. Each column of facets corresponds to a different network architecture, each row of facets corresponds to a specific forecast horizon. Across all columns, there is a tendency for the MAE of network models to cluster at or below the average MAE for the SPF participants for the first three forecast horizons (first three rows). Beyond this, the MAE of network models clearly starts to cluster above the average SPF participant MAE, indicating performance inferior to the SPF at longer forecast horizons.

Broadly speaking, the encoder-decoder model outperforms the SPF and all other models tested. At every time horizon, the encoder-decoder model produces predictions that improve on the predictions of any of the other three models. The encoder-decoder model predictions exhibit as much as an 89% reduction in error when compared to the SPF participant's mean absolute error.

All four models yield predictions that exceed SPF participant MAE at least one prediction horizon and bear further discussion.

At the current-quarter horizon, all four neural network models exceed the SPF participants in terms of average performance. As mentioned above, this is a meaningful benchmark for models since it establishes a threshold whereby the inclusion of the model into SPF forecasts would yield an improvement in the accuracy of the SPF forecast. The fully connected (FC) and LSTM models perform similarly, while the convolutional (CONV) model lags slightly behind (but still remains competitive with the mean SPF participant performance). The encoder-decoder is the best performing model.

At a one-quarter prediction horizon, fully connected and convolutional model averages continue to outperform SPF respondent averages. The LSTM model average falls very slightly behind the average SPF respondent performance. Convolutional, fully connected, and LSTM models perform similarly. The encoder-decoder model continues to demonstrate a substantial reduction in mean absolute error, compared to SPF participant MAE.

All three neural network models remain competitive with the SPF respondents at the two-quarter prediction horizon. It is at this point, however, that the average SPF respondent models outperform the fully connected, convolutional and LSTM neural-network models (albeit by only a small margin). The encoder-decoder network continues to outperform the mean SPF participant predictions. Beyond the two quarter prediction horizon, only the encoder-decoder model continues to outperform the mean SPF participant predictions.

Generally, the variation across repeated trainings of a model is smaller than the overall variation across individual responses in the SPF. Intuitively, this should be expected – the participants in the SPF are expected to use state of the art forecasting methods, but no participant is constrained to use a particular method. It follows

that the individual responses to the SPF reflect the variation in outcomes from a set of similar but likely heterogeneous models. The repeated trainings of neural network models reflect the outcomes of a comparatively less heterogeneous set of models (varying only with respect to the sources of stochasticity as described above). It is therefore not very surprising that the variance across repeated trainings of the neural network models is smaller than across SPF responses but the comparison is nevertheless useful, if only to provide a sense of scale for the precision of model estimates.

4.1 Ensemble Models

The reported SPF forecast for unemployment is a singular, point-estimate of unemployment at zero to four quarter forecast horizons. Technically, this forecast is an ensemble forecast; it is it is the median forecast across all survey participants.

Since we have repeatedly trained the neural network models discussed above, we can similarly produce ensemble estimates of unemployment. Unlike SPF, however, we will produce ensemble estimates by employing a neural network. The design of the network is deliberately simple, perceptron model (functionally, a linear model).

We create several ensemble predictions of each architecture individually as well as an ensemble of all the combined predictions of all four architectures. Performance metrics for these models are provided in Table 2. The first five columns present performance for our various model ensembles, across each forecast horizon. In addition to the ensemble MAE, we also report the ratio of the ensemble MAE to benchmark model MAE, which indicates the extent of the performance improvement compared to each benchmark model.

Table 2: Mean Absolute Error for Ensembled models and aggregate SPF forecast

Horizon	Metric	Fully Connected	Convolutional	LSTM	Encoder Decoder	Combined	DARM	SPF
0 month	MAE	4.3	7.2	4.8	4.1	9.4	11.7	10.1
	MAE/DARM	36.3	61.3	41.1	35.2	80.1		
	MAE/SPF	42.3	71.6	48.0	41.1	93.6		
3 month	MAE	21.5	22.4	24.2	18.4	21.5	32.8	23.1
	MAE/DARM	65.6	68.4	73.6	56.0	65.6		
	MAE/SPF	93.4	97.3	104.8	079.8	093.3		
6 month	MAE	41.4	43.8	40.1	30.1	41.6	49.3	35.7
	MAE/DARM	83.9	88.9	81.3	61.1	84.3		
	MAE/SPF	116.1	123.0	112.4	84.5	116.6		
9 month	MAE	54.8	73.1	66.4	45.9	65.0	65.8	50.3
	MAE/DARM	83.3	111.1	101.0	69.8	98.8		
	MAE/SPF	108.9	145.2	132.0	91.2	129.1		
12 month	MAE	77.9	108.3	90.0	61.8	82.5	90.7	63.0
	MAE/DARM	85.8	119.3	99.1	68.1	91.0		
	MAE/SPF	123.6	171.8	142.7	098.0	131.0		

All metrics presented in basis points.

The results presented in Table 2 are similar to the the comparison of individual model estimates and SPF participant estimates, presented above. The encoder-decoder ensemble out performs the SPF model at every horizon. At the current-quarter horizon, each of the individual ensembles out-performs the SPF model (in-

dicated in Table 2 by an MAE/SPF less than one). The ensembles outperform or remain competitive with the SPF at the first-quarter horizon. Neural network ensembles remain competitive at the two quarter horizon. With the exception of the encoder-decoder ensemble, the ensembles fall behind the SPF at the third and fourth quarter horizons. The models perform slightly better compared to the DARM benchmark, but the convolutional and LSTM ensembles still perform worse than the DARM at some forecast horizons.

4.2 Encoder-Decoder Performance

The encoder-decoder model is, in every regard, the best model tested in this analysis. The best performing iteration of the encoder-decoder model outperforms not only the mean SPF participant, but it also outperforms the overall SPF forecasts.

Figure 11 provides us with a different perspective of model performance[27]. It shows the encoder-decoder model forecasts for every quarter from Q1 1997 through Q4 2015. The actual observed values of $UNRATE$ are also provided, as are the SPF forecasts. As expected, the line representing the encoder-decoder forecast is typically closer to the line representing the observed value than the line representing the SPF forecast.

More revealing, however, is the location of inflection points in the encoder- decoder forecast and SPF forecast around the time of the 2007-2008 financial crisis. These are more narrowly illustrated in Figure 12. Note that the encoder-decoder forecast reverses course more quickly than the SPF forecast as the recession begins. We might speculate that this means that the encoder-decoder model is more responsive, or adapts more quickly to major shifts in the data. This is also suggested by the quickness with which the encoder-decoder responds to the subsequent recovery in 2010.

Table 3 provides more precise information about the responsiveness of the encoder-decoder model. This table shows the location of the inflection points corresponding to the observed unemployment minimum and unemployment maximum for the 2007

[27]Similar figures for the other model architectures are provided in the appendix.

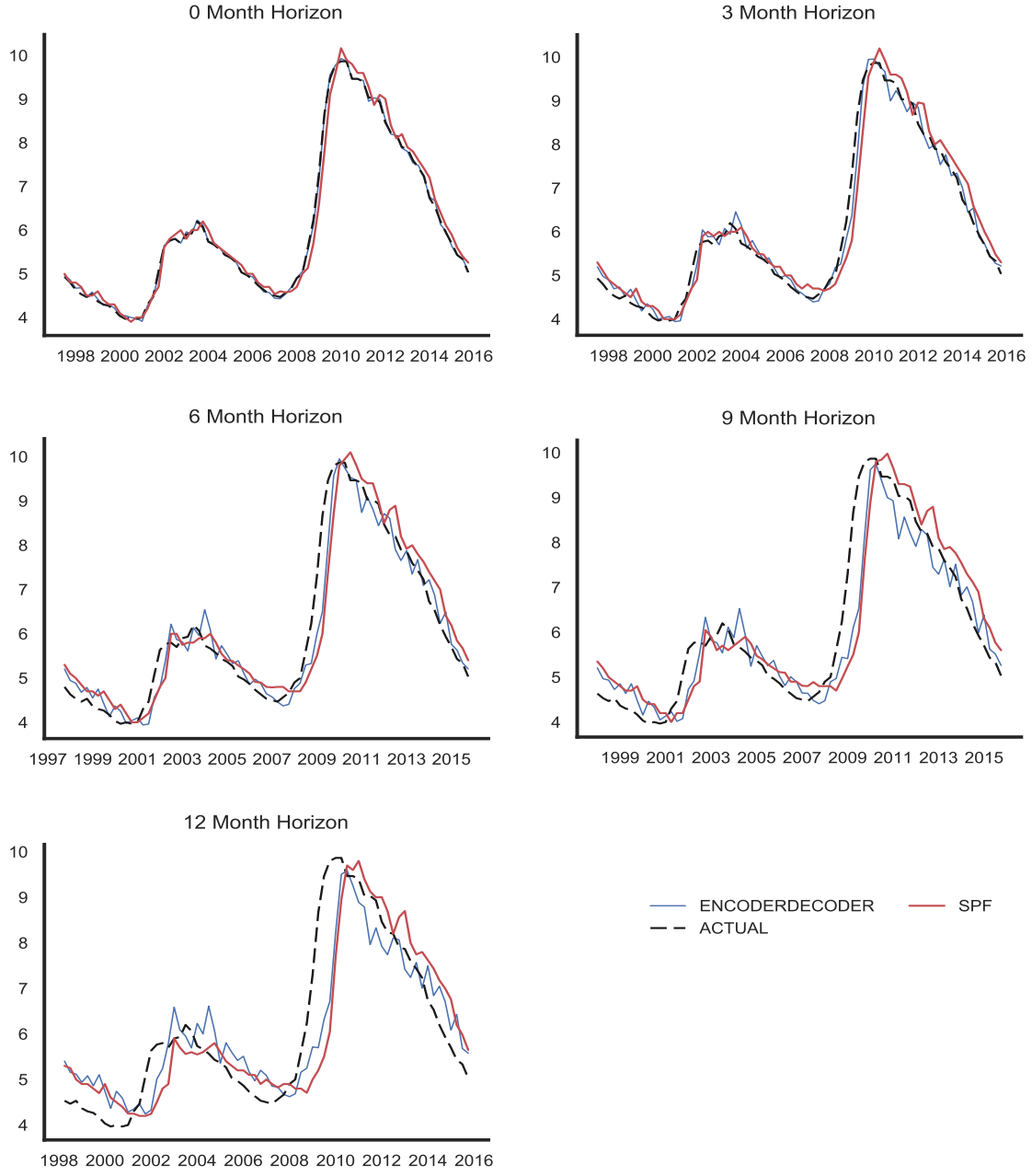
recession and recovery. Locations shown correspond to smoothed[28] SPF and encoder-decoder forecasts for one to four quarter horizons. Broadly, Table 3 suggests that the encoder-decoder responded to the 2007 recession between one and three quarters sooner than the SPF forecasts. The table also indicates that the encoder-decoder model responded to the onset of recovery between one and two quarters sooner than the SPF forecasts.

Table 3: Inflection points for SPF and encoder-decoder forecasts near the 2007 recession

	SPF	Encoder Decoder
Unemployment Nadir (Q1 2007)		
3 Month Horizon Model	Q3 2007	Q1 2007
6 Month Horizon Model	Q3 2007	Q2 2007
9 Month Horizon Model	Q2 2008	Q3 2007
12 Month Horizon Model	Q3 2008	Q1 2008
Unemployment Apex: Q1 2010		
3 Month Horizon Model	Q1 2010	Q4 2009
6 Month Horizon Model	Q2 2010	Q4 2009
9 Month Horizon Model	Q3 2010	Q1 2010
12 Month Horizon Model	Q4 2010	Q2 2010

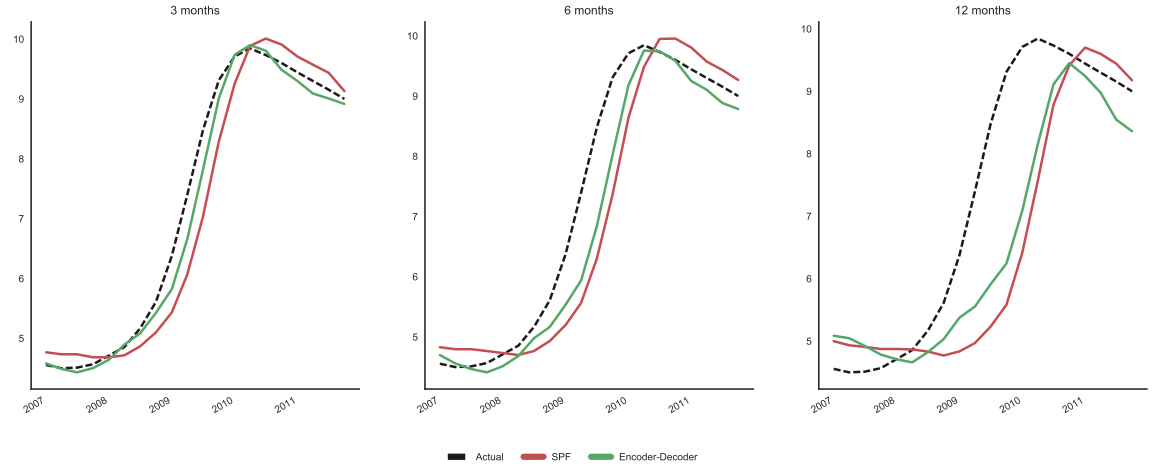
[28]Using a 2-period Gaussian window smoother. Generally, other window sizes/smoothers alter the location of the inflection point slightly, but do not substantially alter the difference in location between SPF and encoder-decoder forecasts.

Figure 11: Encoder-Decoder model forecasts of $UNRATE$ at varying horizons



Each facet represented here shows the predictions of the SPF (red line) and best-performing encoder-decoder model (blue line). The black line indicates the actual, observed level of unemployment.

Figure 12: Smoothed forecasts of SPF and encoder-decoder model at three, six and twelve month horizons



This is a representation of forecasts of unemployment over 2007-2008 recession/recovery. The vertical axis represents the level of unemployment. The horizontal axis represents time (in quarters). The lines in the figure portray (smoothed) unemployment forecasts of the encoder-decoder model (green) along with the SPF model predictions (red). The remaining line (black) indicates the observed level of unemployment.

5 Discussion

The models presented here use a novel technique for forecasting unemployment. Each model architecture provided competitive near-term forecasting performance. This suggests a general robustness to model architecture – at least with respect to near-term forecasting. The encoder-decoder model provides an overall improvement in performance over the SPF.

The implementation of neural network models, however, are not without practical challenges. We have encountered several of these when developing the models for this paper. First, networks can require a lot of computational power and take a long time to converge if they are very complicated. Second, network performance will be far more sensitive to architecture/design (depth, types of layers, regularization parameters, etc.) when training data is in short supply. Third, and relatedly, it is easier to train networks on data that is scaled to (-1,1) or (0,1). This is easy to accomplish with metrics that are taken as a rate (such as the unemployment rate), but it is more difficult with non-stationary metrics such as nominal gross domestic product.

5.1 Avenues for further development

There are a few ways that we expect to improve model performance as we continue to develop this project. First, we anticipate that the inclusion of additional information as model inputs will improve model considerably. In this paper we have deliberately restricted ourselves to the use of a single series as the basis for model inputs. The addition of labor flows, temporal markers (e.g. month and season information), and other macro-indicators as model inputs should provide more information for model forecasts.

Second, we expect that employing more advanced architectures will provide increased performance. The architectures that we explore in this paper are deliberately simple and intended to apply across a variety of forecasting problems. We expect the use of more advanced architectures to produce better performance. Specifically, we may be able to improve performance over the encoder decoder model by augmenting the model with an attention module, which has become a common component of

sequence to sequence models. Moreover, deep learning models are undergoing rapid development, with new techniques published every week. As new techniques emerge, we expect that we will be able to integrate them into the networks presented above to further improve performance.

References

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". eprint: 1406.1078. URL: https://arxiv.org/abs/1406.1078.

Creel, Michael (2016). "Neural Nets for Indirect Inference". *Econometrics and Statistics*.

Cui, Zhicheng, Wenlin Chen, and Yixin Chen (2016). "Multi-Scale Convolutional Neural Networks for Time Series Classification". *CoRR* abs/1603.06995. URL: http://arxiv.org/abs/1603.06995.

Diebold, Francis X (1997). *The past, present, and future of macroeconomic forecasting*. Tech. rep. National Bureau of Economic Research. URL: http://www.nber.org/papers/w6290.pdf.

Gao, Tianxiang and Vladimir Jojic (2016). "Degrees of Freedom in Deep Neural Networks". eprint: 1603.09260. URL: https://arxiv.org/abs/1603.09260.

Gurney, Kevin (1997). *An introduction to neural networks*. CRC press. URL: http://www.inf.ed.ac.uk/teaching/courses/nlu/reading/Gurney_et_al.pdf.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". eprint: 1512.03385. URL: https://arxiv.org/abs/1512.03385.

Kriesel, David (2007). *A brief Introduction on Neural Networks*. Zeta2. URL: http://www.dkriesel.com.

Lucas, Robert E (1976). "Econometric policy evaluation: A critique". In: *Carnegie-Rochester conference series on public policy*. Vol. 1. Elsevier, pp. 19–46.

Minsky, Marvin and Seymour Papert (1988). *Perceptrons: an introduction to computational geometry (expanded edition)*. MIT Press, Cambridge, Ma.

Nouri, Daniel (2014). *Using deep learning to listen for whales*. URL: http://danielnouri.org/notes/2014/01/10/using-deep-learning-to-listen-for-whales/.

Pescatori, Andrea, Saeed Zaman, et al. (2011). "Macroeconomic models, forecasting, and policymaking". *Economic Commentary* 19.20, p. 1.

Rojas, Raúl (2013). *Neural networks: a systematic introduction*. Springer Science & Business Media.

Rosenblatt, Frank (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review* 65.6, p. 386.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". *Nature* 323.6088, pp. 533–536. URL: http://dx.doi.org/10.1038/323533a0.

Sims, Christopher A (1980). "Macroeconomics and reality". *Econometrica: Journal of the Econometric Society*, pp. 1–48.

Stark, Tom (2017). *Error Statistics for the Survey of Professional Forecasters for Unemployment Rate*. Tech. rep. Federal Reserve Bank of Philadelphia. URL: https://www.philadelphiafed.org/-/media/research-and-data/real-time-center/survey-of-professional-forecasters/data-files/unemp/spf_error_statistics_unemp_1_aic.pdf?la=en.

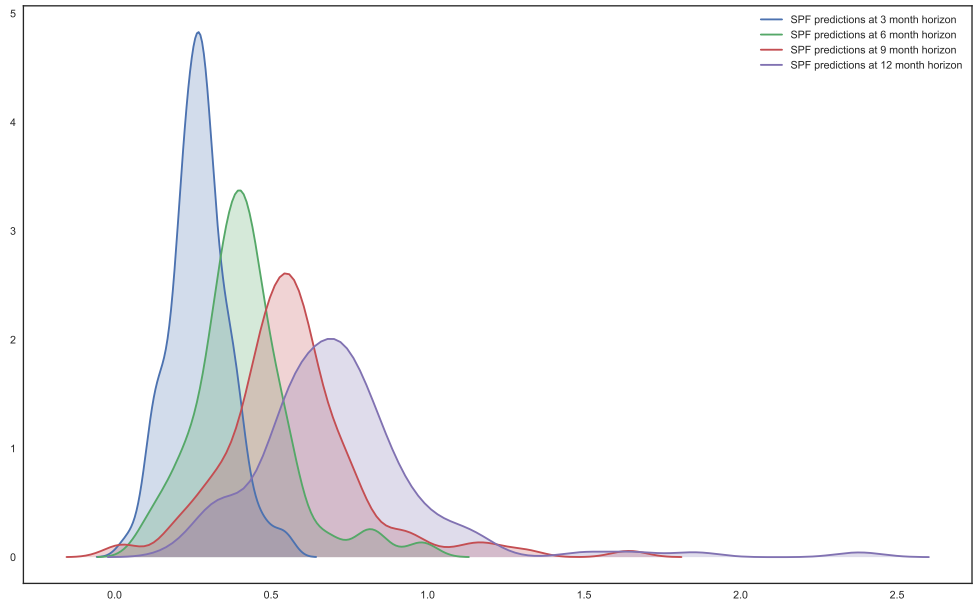
Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks". eprint: 1409.3215. URL: https://arxiv.org/abs/1409.3215.

U.S. Bureau of Labor Statistics. *Civilian Unemployment Rate [UNRATE]*. Retrieved from FRED, Federal Reserve Bank of St. Louis. URL: https://fred.stlouisfed.org/series/UNRATE.

Ye, Jianming (1998). "On measuring and correcting the effects of data mining and model selection". *Journal of the American Statistical Association* 93.441, pp. 120–131.

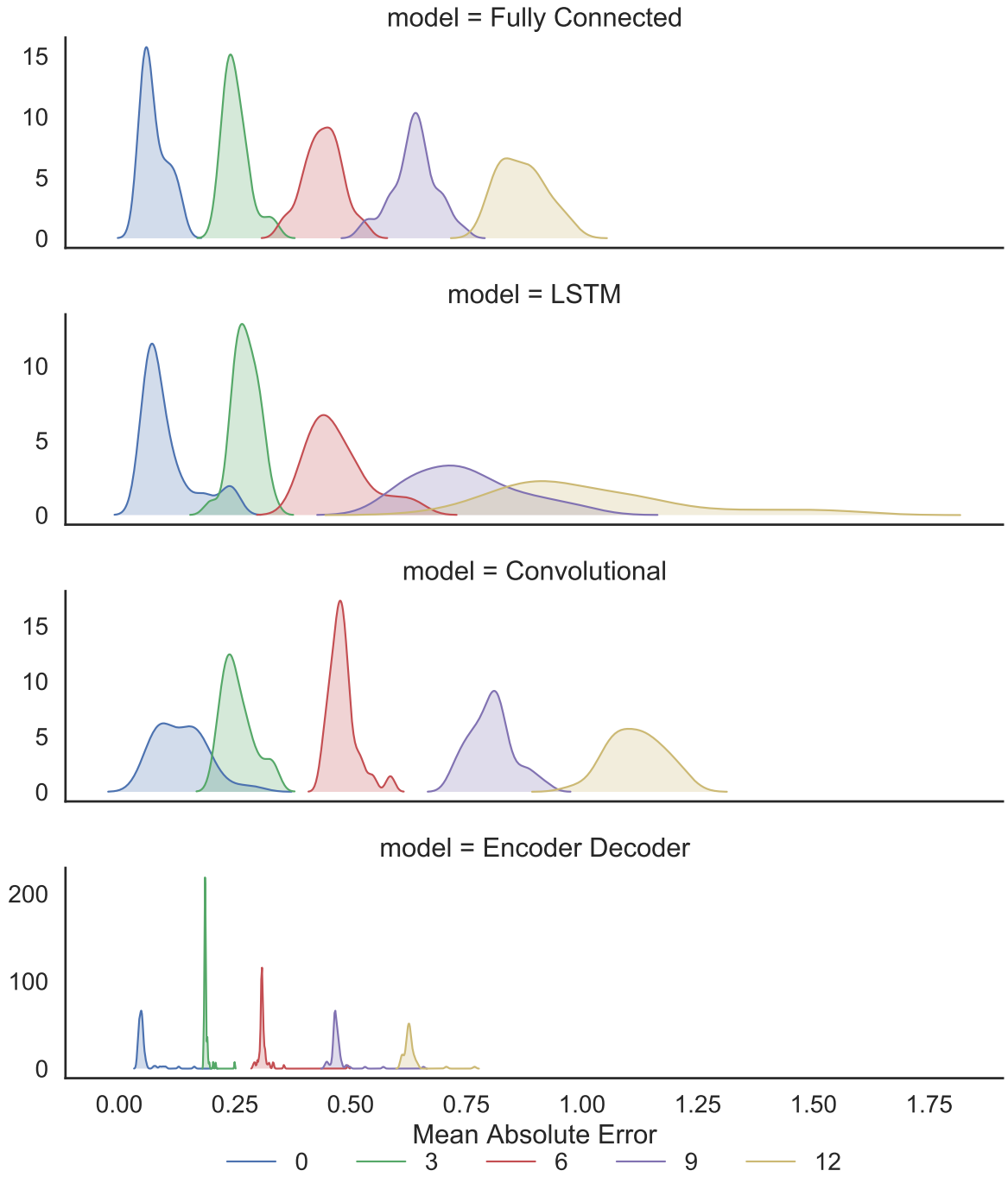
A Additional Figures

Figure 13: Distribution of mean absolute error across SPF participants at various forecast horizons



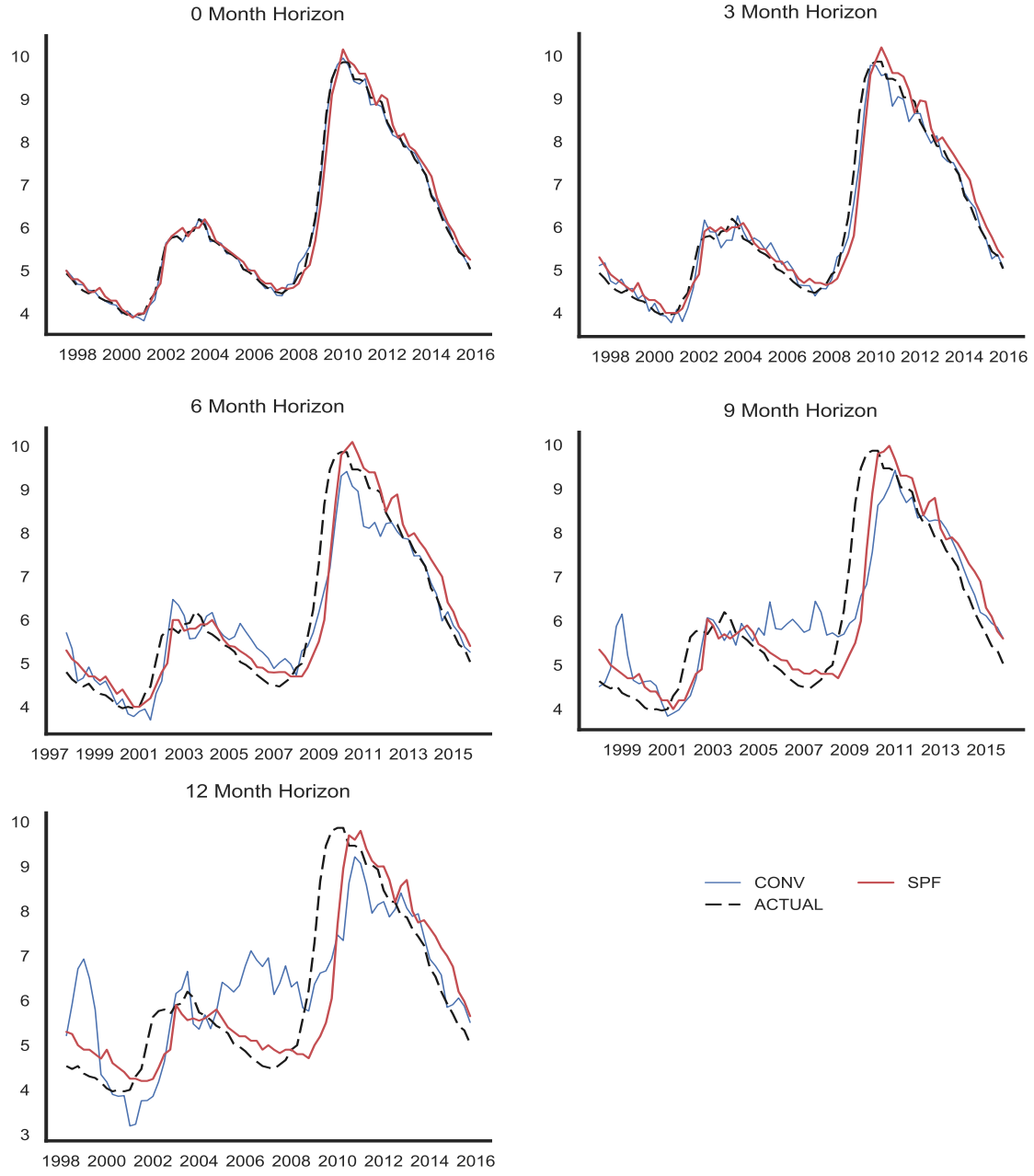
This figure illustrates the distribution in performance in SPF participants at each forecast horizon. Errors cluster at progressively higher levels of MAE as the forecast horizon lengthens, indicating poorer performance at longer horizons. Additionally, participant MAE grows more diffuse as forecast horizons lengthen, indicating less consensus in predictions at longer forecast horizons.

Figure 14: Distribution of mean absolute error at various forecast horizons across all architectures tested



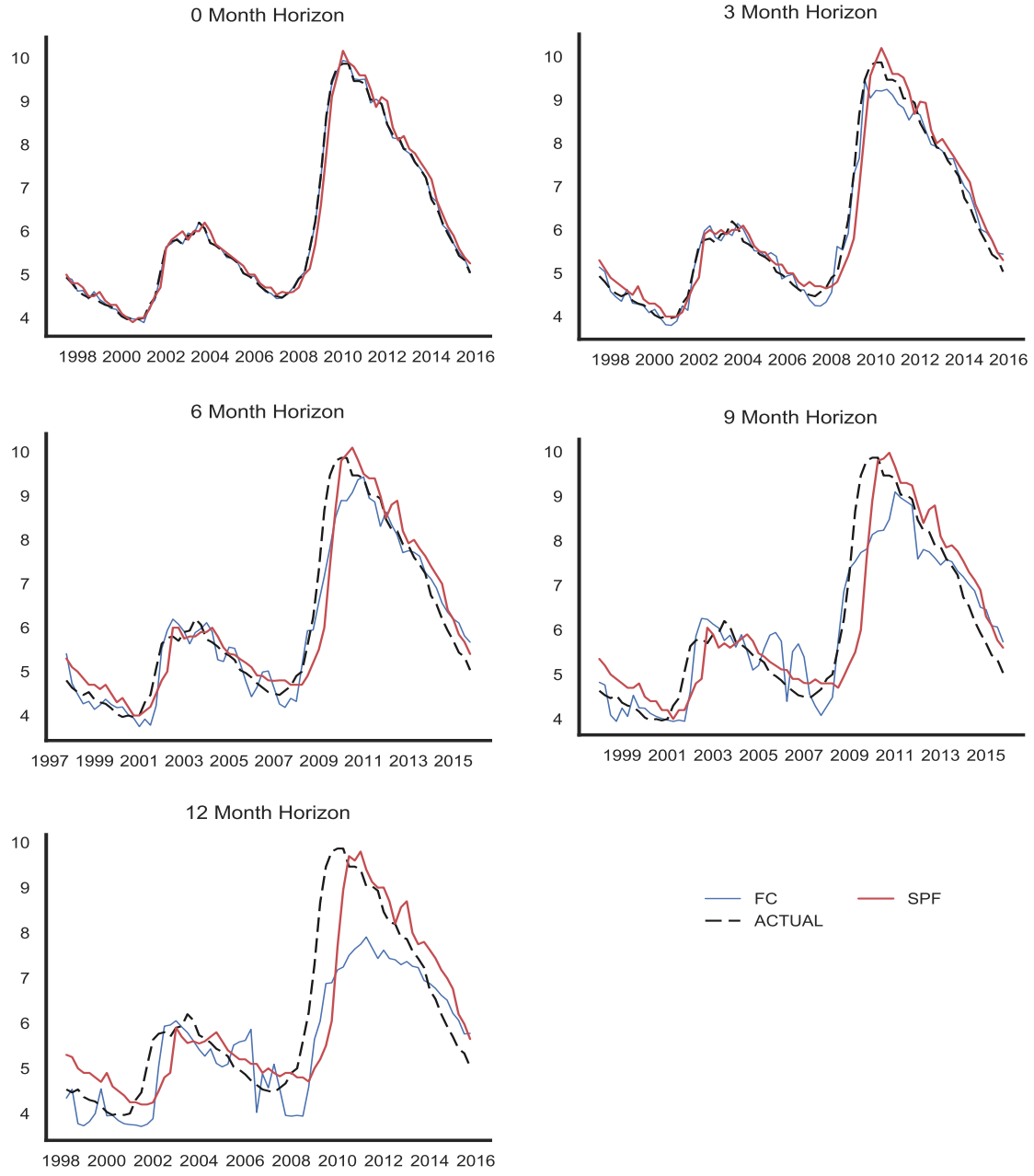
This figure illustrates the distribution in performance across repeated trainings of each of the architectures tested. Each facet displays the distribution of mean absolute error over repeated trainings of a specific architecture, for each of the 0-4 quarter forecast horizons targeted in this analysis. While the fully connected, LSTM, and convolutional models all exhibit roughly comparable error distributions, the error distribution of the encoder decoder model exhibits far less variance in performance.

Figure 15: Convolutional model forecasts of $UNRATE$ at varying horizons



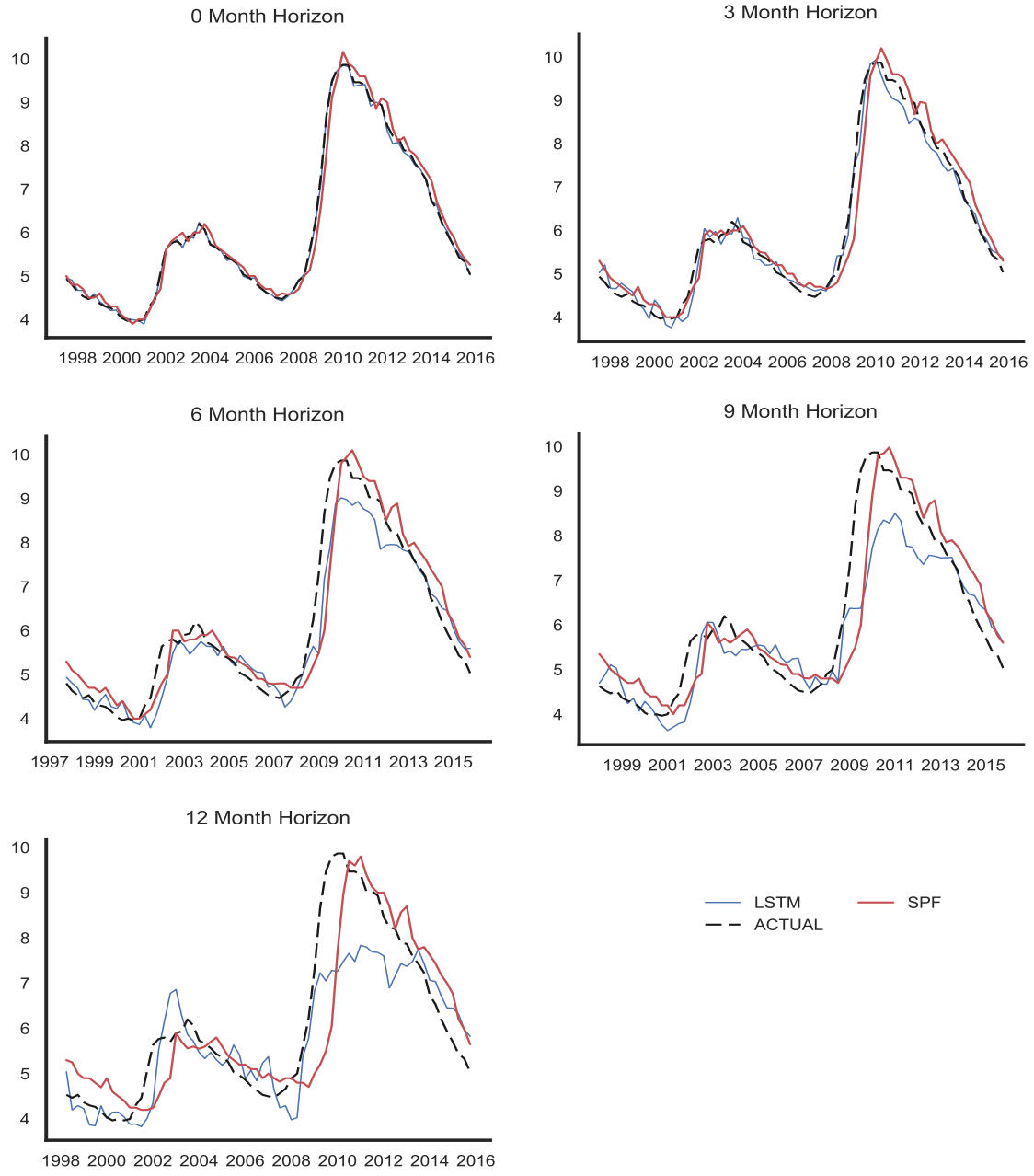
Each facet represented here shows the predictions of the SPF (red line) and best-performing convolutional decoder model (blue line). The black line indicates the actual, observed level of unemployment.

Figure 16: Fully Connected model forecasts of $UNRATE$ at varying horizons



Each facet represented here shows the predictions of the SPF (red line) and best-performing Fully Connected model (blue line). The black line indicates the actual, observed level of unemployment.

Figure 17: LSTM model forecasts of $UNRATE$ at varying horizons



Each facet represented here shows the predictions of the SPF (red line) and best-performing LSTM model (blue line). The black line indicates the actual, observed level of unemployment.