DATA 586: Advanced Machine Learning

2023W2

Shan Du

Document Classification

- We introduce a new type of example that has important applications in industry and science: predicting attributes of documents.
- Documents can be articles in medical journals, Reuters news feeds, emails, tweets, a movie's reviews, and so on.
- The response can be the *sentiment* of a review, which will be *positive* or *negative*.

Document Classification

- Here is the beginning of a rather amusing negative review:

  This has to be one of the worst films of the 1990s. When my friends & I were watching this film (being the target audience it was aimed at) we just sat & watched the first half an hour with our jaws touching the floor at how bad it really was. The rest of the time, everyone else in the theater just started talking to each other, leaving or generally crying into their popcorn . . .

Document Classification

- Each review can be a different length, include slang or non-words, have spelling errors, etc. We need to find a way to featurize such a document.
- This is modern parlance for defining a set of predictors.

Document Classification

- The simplest and most common featurization is the *bag-of-words* model. We score each document for the presence or absence of each of the words in a language dictionary — in this case an English dictionary.
- If the dictionary contains M words, that means for each document we create a binary feature vector of length M, and score a 1 for every word present, and 0 otherwise.

Document Classification

- That can be a very wide feature vector, so we limit the dictionary — in this case to the 10,000 most frequently occurring words in the training corpus of 25,000 reviews.
- Fortunately, there are nice tools for doing this automatically.

Document Classification

- Here is the beginning of a positive review that has been redacted in this way:

  <START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came from the same scottish island as myself so i loved . . .

Document Classification

- Here we can see many words have been omitted, and some unknown words (UNK) have been marked as such.
- With this reduction, the binary feature vector has length 10,000, and consists mostly of 0's and a smattering of 1's in the positions corresponding to words that are present in the document.

Document Classification

- If we have a training set and test set, each with 25,000 examples, the resulting training feature matrix X has dimension 25,000×10,000, but only 1.3% of the binary entries are nonzero.
- We call such a matrix *sparse*, because most of the values are the same (zero in this case); it can be stored efficiently in sparse matrix format - Rather than store the whole matrix, we can store instead the location and values for the nonzero entries. In this case, since the nonzero entries are all 1, just the locations are stored.

Document Classification

- We split off a validation set of size 2,000 from the 25,000 training observations (for model tuning), and fit a two-class neural network with two hidden layers, each with 16 ReLU units.
- Note that a two-class neural network amounts to a nonlinear logistic regression model.

$$\log\left(\frac{\Pr(Y=1|X)}{\Pr(Y=0|X)}\right)$$

$$= Z_1 - Z_0 = (\beta_{10} - \beta_{00}) + \sum_{l=1}^{K_2} (\beta_{1l} - \beta_{0l}) A_l^{(2)}$$

Document Classification

- The bag-of-words model summarizes a document by the words present, and ignores their context. There are at least two popular ways to take the context into account:
  - The *bag-of-n-grams* model. For example, a bag of 2-grams records the consecutive co-occurrence of every distinct pair of words. “Blissfully long” can be seen as a positive phrase in a movie review, while “blissfully short” a negative.
  - Treat the document as a sequence, taking account of all the words in the context of those that preceded and those that follow.

Recurrent Neural Networks

- We explore models for sequences of data, which have applications in weather forecasting, speech recognition, language translation, and time-series prediction, to name a few.
- Many data sources are sequential in nature, and call for special treatment when building predictive models. Examples include:

Recurrent Neural Networks

- Documents such as book and movie reviews, newspaper articles, and weets. The sequence and relative positions of words in a document capture the narrative, theme and tone, and can be exploited in tasks such as topic classification, sentiment analysis, and language translation.
- Time series of temperature, rainfall, wind speed, air quality, and so on. We may want to forecast the weather several days ahead, or climate several decades ahead.

Recurrent Neural Networks

- Financial time series, where we track market indices, trading volumes, stock and bond prices, and exchange rates. Here prediction is often difficult, but as we will see, certain indices can be predicted with reasonable accuracy.
- Recorded speech, musical recordings, and other sound recordings. We may want to give a text transcription of a speech, or perhaps a language translation. We may want to assess the quality of a piece of music, or assign certain attributes.

Recurrent Neural Networks

- Handwriting, such as doctor's notes, and handwritten digits such as zip codes. Here we want to turn the handwriting into digital text, or read the digits (optical character recognition).

Recurrent Neural Networks

- In a *recurrent neural network* (RNN), the input object X is a *sequence*.
- Consider a corpus of documents, each document can be represented as a sequence of L words $X = \{X_1, X_2, \ldots, X_L\}$, where each X_l represents a word.
- The order of the words, and closeness of certain words in a sentence, convey semantic meaning.

Recurrent Neural Networks

- RNNs are designed to accommodate and take advantage of the sequential nature of such input objects, much like convolutional neural networks accommodate the spatial structure of image inputs.
- The output Y can also be a sequence (such as in language translation), but often is a scalar, like the binary sentiment label of a movie review document.

Recurrent Neural Networks

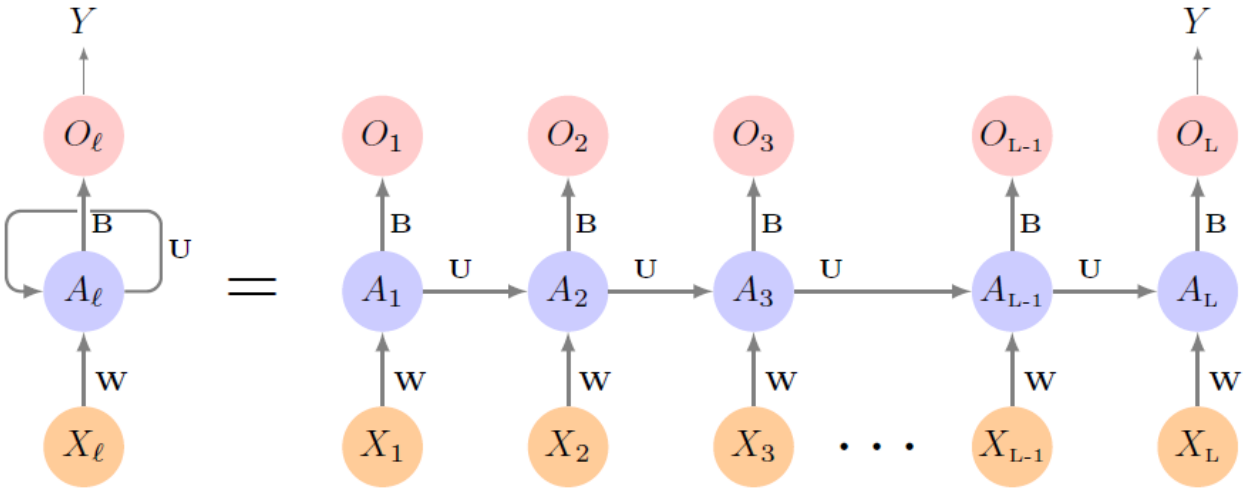

FIGURE 10.12. *Schematic of a simple recurrent neural network. The input is a sequence of vectors* $\{X_\ell\}_1^L$*, and here the target is a single response. The network processes the input sequence* X *sequentially; each* X_ℓ *feeds into the hidden layer, which also has as input the activation vector* $A_{\ell-1}$ *from the previous element in the sequence, and produces the current activation vector* A_ℓ*. The same collections of weights* $\mathbf{W}$*,* $\mathbf{U}$ *and* $\mathbf{B}$ *are used as each element of the sequence is processed. The output layer produces a sequence of predictions* O_ℓ *from the current activation* A_ℓ*, but typically only the last of these,* O_L*, is of relevance. To the left of the equal sign is a concise representation of the network, which is* unrolled *into a more explicit version on the right.*

Recurrent Neural Networks

- The structure of a very basic RNN: a sequence $X = \{X_1, X_2, \ldots, X_L\}$ as input, a simple output Y, and a hidden-layer sequence $\{A_l\}_1^L = \{A_1, A_2, \ldots, A_L\}$.
- Each X_l is a vector; could be a one-hot encoding for the lth word based on the language dictionary for the corpus.
- As the sequence is processed one vector X_l at a time, the network updates the activations A_l in the hidden layer, taking as input the vector X_l and the activation vector A_{l-1} from the previous step in the sequence.

Recurrent Neural Networks

- Each A_l feeds into the output layer and produces a prediction O_l for Y. O_L, the last of these, is the most relevant.
- Suppose each vector X_l of the input sequence has p components $X_l^T = (X_{l1}, X_{l2}, \ldots, X_{lp})$, and the hidden layer consists of K units $A_l^T = (A_{l1}, A_{l2}, \ldots, A_{lK})$, the matrix $\mathbf{W}$ of $K \times (p+1)$ shared weights w_{kj} is for the input layer, similarly $\mathbf{U}$ is a $K \times K$ matrix of the weights u_{ks} for the hidden-to-hidden layers, and $\mathbf{B}$ is a $K + 1$ vector of weights β_k for the output layer.

Recurrent Neural Networks

- Then

$$A_{lk} = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{lj} + \sum_{s=1}^{K} u_{ks} A_{l-1,s}),$$

And the output O_l is computed as

$$O_l = \beta_0 + \sum_{k=1}^{K} \beta_k A_{lk}$$

for a quantitative response, or with an additional sigmoid activation function for a binary response, for example. Here $g(\cdot)$ is an activation function such as ReLU.

Recurrent Neural Networks

- Notice that the same weights $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{B}$ are used as we process each element in the sequence, i.e., they are not functions of l.
- This is a form of *weight sharing* used by RNNs, and similar to the use of filters in convolutional neural networks.
- As we proceed from beginning to end, the activations A_l accumulate a history of what has been seen before, so that the learned context can be used for prediction.

Recurrent Neural Networks

- For regression problems the loss function for an observation (X, Y) is

$$(Y - O_L)^2$$

which only references the final output

$$O_L = \beta_0 + \sum_{k=1}^{K} \beta_k A_{Lk}.$$

Thus $O_1, O_2, \ldots, O_{L-1}$ are not used.

- When we fit the model, each element X_l of the input sequence X contributes to O_L via the chain, and hence contributes indirectly to learning the shared parameters $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{B}$ via the loss.

Recurrent Neural Networks

- With n input sequence/response pairs (x_i, y_i), the parameters are found by minimizing the sum of squares

$$\sum_{i=1}^{n}(y_i - o_{iL})^2 =$$

$$\sum_{i=1}^{n}(y_i - (\beta_0 + \sum_{k=1}^{K}\beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{iLj} + \sum_{s=1}^{K} u_{ks}a_{i,l-1,s})))^2 .$$

Here we use lowercase letters for the observed y_i and vector sequences $x_i = (x_{i1}, x_{i2}, \ldots, x_{iL})$, as well as the derived activations.

Recurrent Neural Networks

- Since the intermediate outputs O_L are not used, one may ask why they are there at all.
- First of all, they come for free, since they use the same output weights $\mathbf{B}$ needed to produce O_L, and provide an evolving prediction for the output.
- Furthermore, for some learning tasks the response is also a sequence, and so the output sequence $\{O_1, O_2, \ldots, O_L\}$ is explicitly needed.

Sequential Models for Document Classification

- We use the *sequence of words* occurring in a document to make predictions about the label for the entire document.
- We have, however, a dimensionality problem: each word in our document is represented by a one-hot-encoded vector (dummy variable) with 10,000 elements (one per word in the dictionary)!
- An approach that has become popular is to represent each word in a much *lower-dimensional embedding space*.

Sequential Models for Document Classification

- This means that rather than representing each word by a binary embedding vector with 9,999 zeros and a single one in some position, we will represent it instead by a set of m real numbers, none of which are typically zero.
- Here m is the embedding dimension, and can be in the low 100s, or even less.
- This means (in our case) that we need a matrix $\mathbf{E}$ of dimension $m \times 10{,}000$, where each column is indexed by one of the 10,000 words in our dictionary, and the values in that column give the m coordinates for that word in the embedding space.

Sequential Models for Document Classification

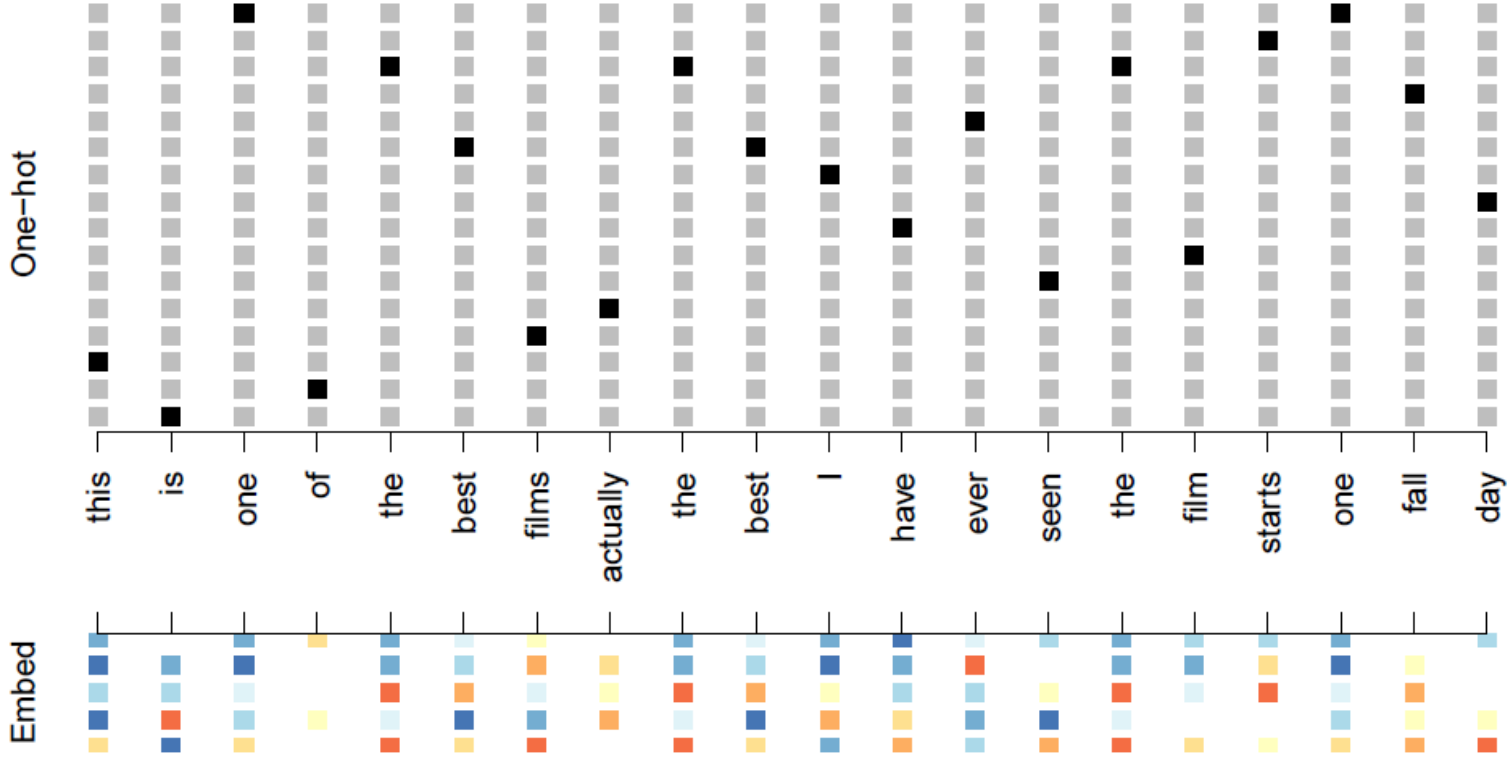

FIGURE 10.13. *Depiction of a sequence of* 20 *words representing a single document: one-hot encoded using a dictionary of* 16 *words (top panel) and embedded in an* m*-dimensional space with* $m = 5$ *(bottom panel).*

Sequential Models for Document Classification

- Where does $\mathbf{E}$ come from? If we have a large corpus of labeled documents, we can have the neural network learn $\mathbf{E}$ as part of the optimization. In this case $\mathbf{E}$ is referred to as an embedding layer, and a specialized $\mathbf{E}$ is learned for the task at hand.
- Otherwise, we can insert a precomputed matrix $\mathbf{E}$ in the embedding layer, a process known as *weight freezing*.

Sequential Models for Document Classification

- Two pretrained embeddings, *word2vec* and *GloVe*, are widely used.

  https://code.google.com/archive/p/word2vec

  https://nlp.stanford.edu/projects/glove

- These are built from a very large corpus of documents by a variant of principal components analysis. The idea is that the positions of words in the embedding space preserve semantic meaning; e.g., synonyms should appear near each other.

Sequential Models for Document Classification

- Therefore, each document is now represented as a sequence of m vectors that represents the sequence of words.
- The next step is to limit each document to L words. Documents that are shorter than L get padded with zeros upfront. So now each document is represented by a series consisting of L vectors $X = \{X_1, X_2, \ldots, X_L\}$, and each X_l in the sequence has m components.

Sequential Models for Document Classification

- The training corpus consists of n separate series (documents) of length L, each of which gets processed sequentially from left to right.
- In the process, a parallel series of hidden activation vectors $A_l, l = 1, \ldots, L$ is created for each document.
- A_l feeds into the output layer to produce the evolving prediction O_l. We use the final value O_L to predict the response: the sentiment of the review.

Sequential Models for Document Classification

- More elaborate versions of RNN use long term and short term memory (LSTM).
- Two tracks of hidden-layer activations are maintained, so that when the activation A_l is computed, it gets input from hidden units both further back in time, and closer in time — a so-called *LSTM RNN*.
- With long sequences, this overcomes the problem of early signals being washed out by the time they get propagated through the chain to the final activation vector A_L.

Time Series Forecasting

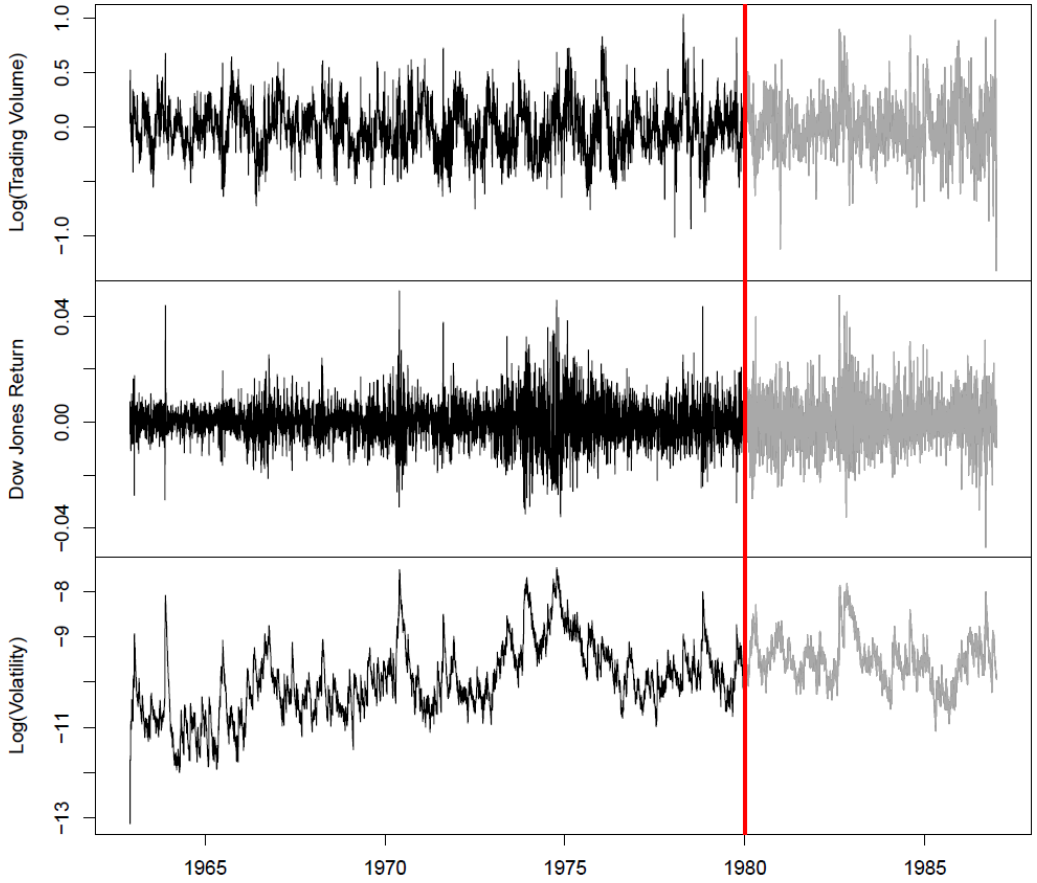

FIGURE 10.14. *Historical trading statistics from the New York Stock Exchange. Daily values of the normalized log trading volume, DJIA return, and log volatility are shown for a 24-year period from 1962–1986. We wish to predict trading volume on any day, given the history on all earlier days. To the left of the red bar (January 2, 1980) is training data, and to the right test data.*

Time Series Forecasting

- An observation here consists of the measurements (v_t, r_t, z_t) on day t, in this case the values for *log_volume*, *DJ_return* and *log_volatility*.
- There are a total of T = 6,051 such triples, each of which is plotted as a time series in Figure 10.14.
- The day-to-day observations are not independent of each other.

Time Series Forecasting

- The series exhibit *auto-correlation* — in this case values nearby in time tend to be similar to each other.

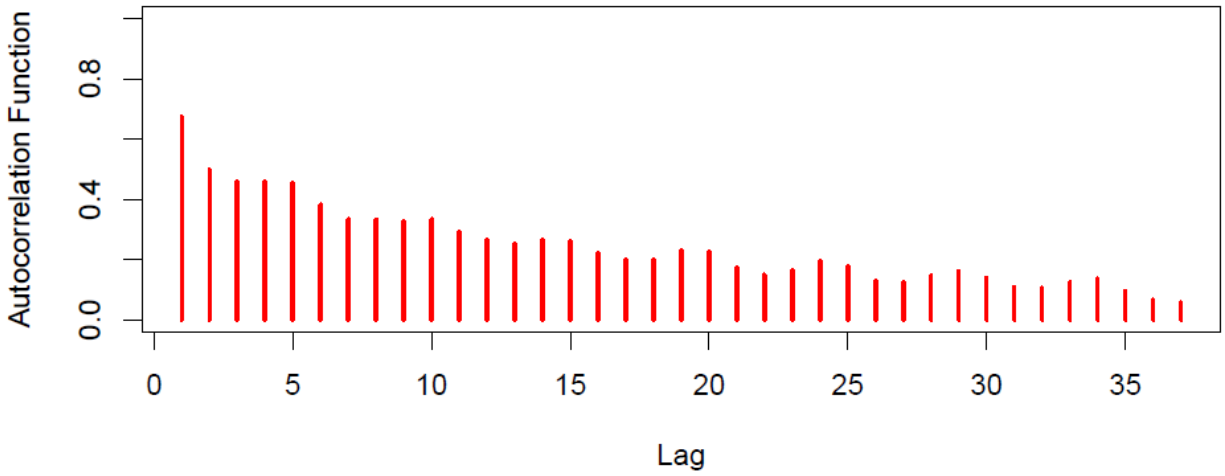

FIGURE 10.15. *The autocorrelation function for* `log_volume`*. We see that nearby values are fairly strongly correlated, with correlations above* 0.2 *as far as* 20 *days apart.*

Time Series Forecasting

- Another interesting characteristic of this forecasting problem is that the response variable v_t — *log_volume* — is also a predictor!
- In particular, we will use the past values of *log_volume* to predict the values in the future.

RNN Forecaster

- We wish to predict a value v_t from past values $v_{t-1}, v_{t-2}, \ldots$, and also to make use of past values of the other series $r_{t-1}, r_{t-2}, \ldots$ and $z_{t-1}, z_{t-2}, \ldots$.
- Although our combined data is quite a long series with 6,051 trading days, the structure of the problem is different from the previous document classification example.

RNN Forecaster

- We only have one series of data, not 25,000.
- We have an entire series of targets v_t, and the inputs include past values of this series.

- How do we represent this problem in terms of the structure of RNN?

RNN Forecaster

- The idea is to extract many short mini-series of input sequences $X = \{X_1, X_2, \ldots, X_L\}$ with a predefined length L (called the lag in this context), and a corresponding target Y.

$$X_1 = \begin{pmatrix} v_{t-L} \\ r_{t-L} \\ z_{t-L} \end{pmatrix}, X_2 = \begin{pmatrix} v_{t-L+1} \\ r_{t-L+1} \\ z_{t-L+1} \end{pmatrix}, \ldots, X_L = \begin{pmatrix} v_{t-1} \\ r_{t-1} \\ z_{t-1} \end{pmatrix}$$

and $Y = v_t$.

RNN Forecaster

- So here the target Y is the value of *log_volume* v_t at a single timepoint t, and the input sequence X is the series of 3-vectors $\{X_l\}_1^L$ each consisting of the three measurements *log_volume*, *DJ_return* and *log_volatility* from day $t-L$, $t-L+1$, up to $t-1$.
- Each value of t makes a separate (X, Y) pair, for t running from $L+1$ to T.

RNN Forecaster

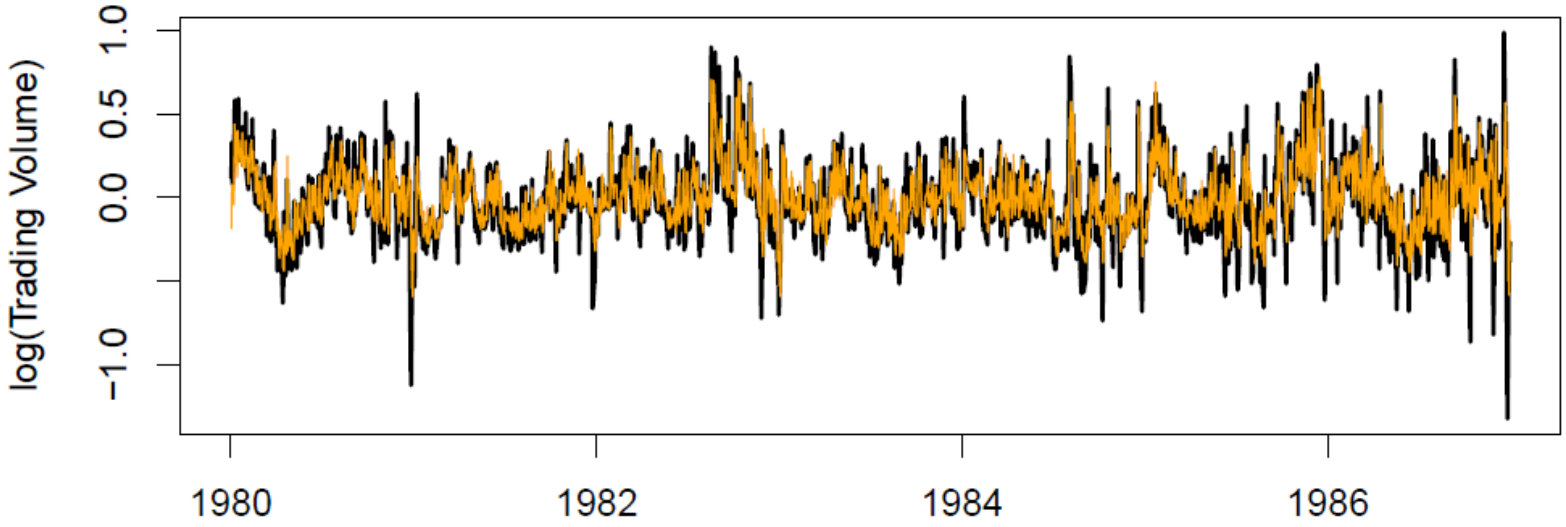

FIGURE 10.16. *RNN forecast of* `log_volume` *on the* `NYSE` *test data. The black lines are the true volumes, and the superimposed orange the forecasts. The forecasted series accounts for 42% of the variance of* `log_volume`.

Autoregression

- The RNN we fit for time series forecasting has much in common with a traditional *autoregression* (AR) linear model.
- We first consider the response sequence v_t alone, and construct a response vector $\mathbf{y}$ and a matrix $\mathbf{M}$ of predictors for least squares regression as follows:

$$\mathbf{y} = \begin{bmatrix} v_{L+1} \\ v_{L+2} \\ v_{L+3} \\ \cdots \\ v_T \end{bmatrix} \mathbf{M} = \begin{bmatrix} 1 & v_L & v_{L-1} & \cdots & v_1 \\ 1 & v_{L+1} & v_L & \cdots & v_2 \\ 1 & v_{L+2} & v_{L+1} & \cdots & v_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{T-1} & v_{T-2} & \cdots & v_{T-L} \end{bmatrix}$$

Autoregression

- $\mathbf{M}$ and $\mathbf{y}$ each have $T - L$ rows, one per observation.
- We see that the predictors for any given response v_t on day t are the previous L values of the same series. Fitting a regression of $\mathbf{y}$ on $\mathbf{M}$ amounts to fitting the model

$$\hat{v}_t = \hat{\beta}_0 + \hat{\beta}_1 \hat{v}_{t-1} + \hat{\beta}_2 \hat{v}_{t-2} + \cdots + \hat{\beta}_L \hat{v}_{t-L}$$

and is called an order-L autoregressive model, or simply AR(L).

Autoregression

- The RNN processes this sequence from left to right with the same weights $\mathbf{W}$ (for the input layer), while the AR model simply treats all L elements of the sequence equally as a vector of $L \times p$ predictors — a process called *flattening* in the neural network literature.
- Of course the RNN also includes the hidden layer activations flattening A_l which transfer information along the sequence, and introduces additional nonlinearity.
- RNN has more parameters than the AR model.

Summary of RNNs

- We can have additional hidden layers in an RNN. The sequence A_l is treated as an input sequence to the next hidden layer in an obvious fashion.
- The RNN we used scanned the document from beginning to end; alternative *bidirectional* RNNs scan the sequences in both directions.
- In language translation the target is also a sequence of words, in a language different from that of the input sequence. Both the input sequence and the target sequence are represented by a structure similar, and they share the hidden units. In this so-called *Seq2Seq* learning, the hidden units are thought to capture the semantic meaning of the sentences.