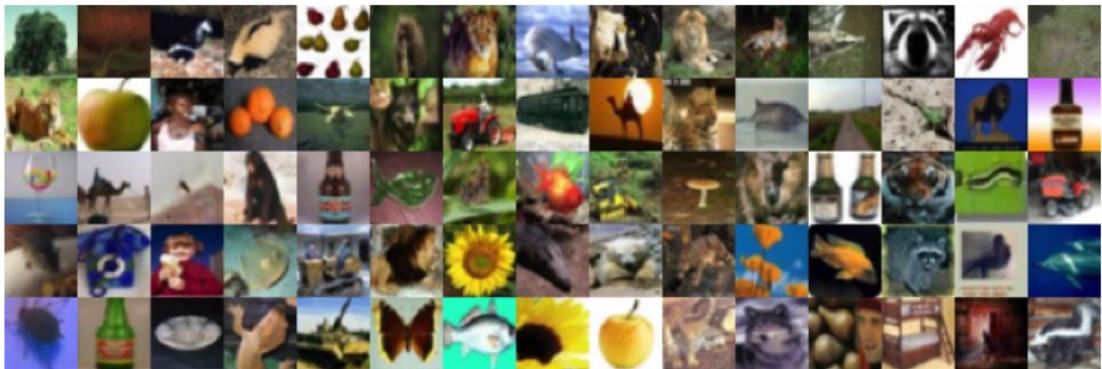


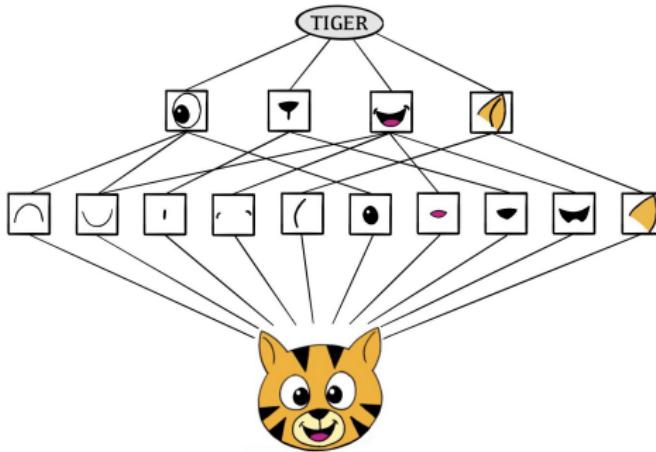
Convolutional Neural Network — CNN



- Major success story for classifying images.
- Shown are samples from **CIFAR100** database. 32×32 color natural images, with 100 classes.
- $50K$ training images, $10K$ test images.

Each image is a three-dimensional array or *feature map*:
 $32 \times 32 \times 3$ array of 8-bit numbers. The last dimension represents the three color channels for red, green and blue.

How CNNs Work



- The CNN builds up an image in a hierarchical fashion.
- Edges and shapes are recognized and pieced together to form more complex shapes, eventually assembling the target image.
- This hierarchical construction is achieved using *convolution* and *pooling* layers.

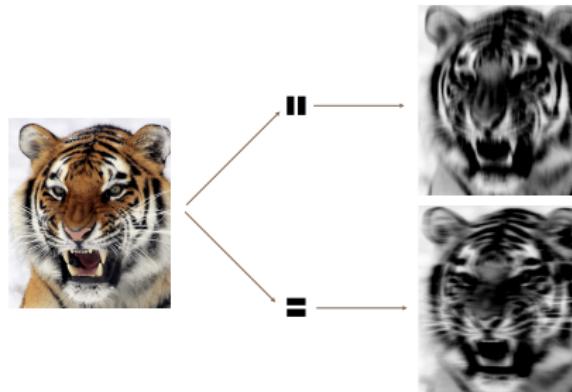
Convolution Filter

$$\text{Input Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \quad \text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

- The filter is itself an image, and represents a small shape, edge etc.
- We slide it around the input image, scoring for matches.
- The scoring is done via *dot-products*, illustrated above.
- If the subimage of the input image is similar to the filter, the score is high, otherwise low.
- The filters are *learned* during training.

Convolution Example



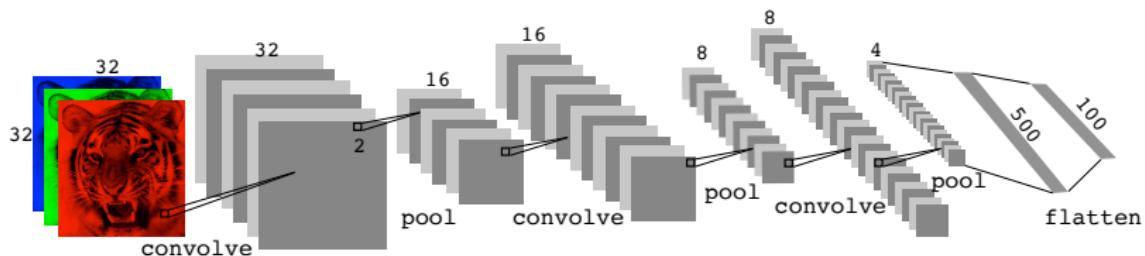
- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.
- Since images have three colors channels, the filter does as well: one filter per channel, and dot-products are summed.
- The weights in the filters are *learned* by the network.

Pooling

$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

- Each non-overlapping 2×2 block is replaced by its maximum.
- This sharpens the feature identification.
- Allows for locational invariance.
- Reduces the dimension by a factor of 4 — i.e. factor of 2 in each dimension.

Architecture of a CNN



- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3×3 .
- Each filter creates a new channel in convolution layer.
- As pooling reduces size, the number of filters/channels is typically increased.
- Number of layers can be very large. E.g. **resnet50** trained on **imagenet** 1000-class image data base has 50 layers!

Using Pretrained Networks to Classify Images



Using Pretrained Networks to Classify Images



flamingo

Cooper's hawk

Cooper's hawk

flamingo	0.83	kite (raptor)	0.60	fountain	0.35
spoonbill	0.17	great grey owl	0.09	nail	0.12
white stork	0.00	robin	0.06	hook	0.07

Lhasa Apso

cat

Cape weaver

Tibetan terrier	0.56	Old English sheepdog	0.82	jacamar	0.28
Lhasa	0.32	Shih-Tzu	0.04	macaw	0.12
cocker spaniel	0.03	Persian cat	0.04	robin	0.12

Here we use the 50-layer **resnet50** network trained on the 1000-class **imagenet** corpus to classify some photographs.

Document Classification: IMDB Movie Reviews

The **IMDB** corpus consists of user-supplied movie ratings for a large collection of movies. Each has been labeled for **sentiment** as **positive** or **negative**. Here is the beginning of a 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 ...

We have labeled training and test sets, each consisting of 25,000 reviews, and each balanced with regard to sentiment.

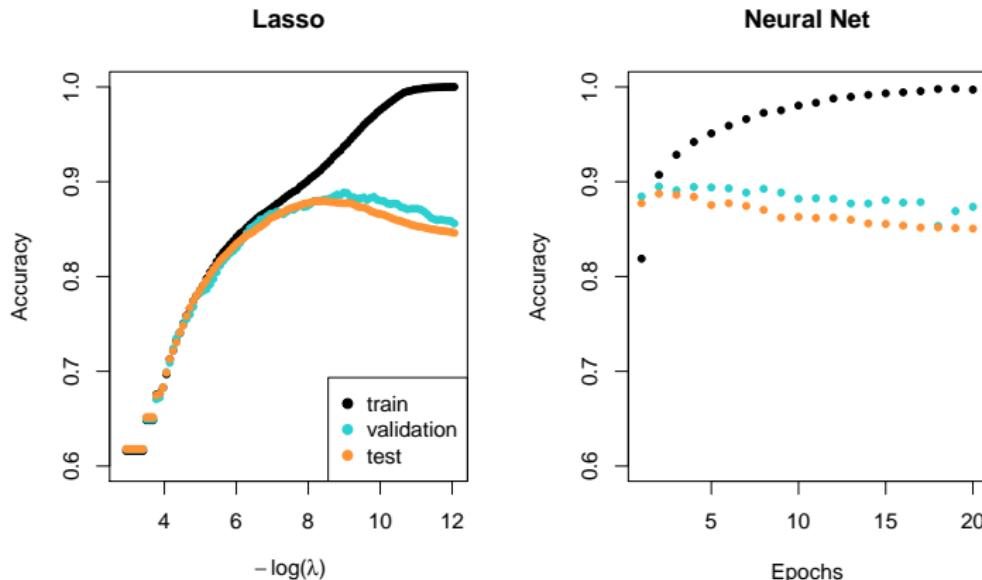
We wish to build a classifier to predict the sentiment of a review.

Featurization: Bag-of-Words

Documents have different lengths, and consist of sequences of words. How do we create features X to characterize a document?

- From a dictionary, identify the $10K$ most frequently occurring words.
- Create a binary vector of length $p = 10K$ for each document, and score a 1 in every position that the corresponding word occurred.
- With n documents, we now have a $n \times p$ *sparse* feature matrix \mathbf{X} .
- We compare a lasso logistic regression model to a two-hidden-layer neural network on the next slide. (No convolutions here!)
- Bag-of-words are *unigrams*. We can instead use *bigrams* (occurrences of adjacent word pairs), and in general *m-grams*.

Lasso versus Neural Network — IMDB Reviews



- Simpler lasso logistic regression model works as well as neural network in this case.
- **glmnet** was used to fit the lasso model, and is very effective because it can exploit sparsity in the \mathbf{X} matrix.

Recurrent Neural Networks

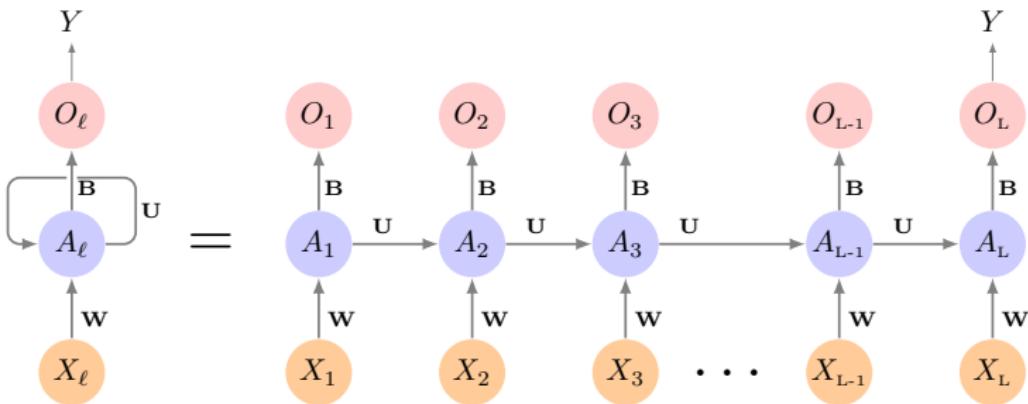
Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

- The feature for each observation is a *sequence* of vectors $X = \{X_1, X_2, \dots, X_L\}$.
- The target Y is often of the usual kind — e.g. a single variable such as **Sentiment**, or a one-hot vector for multiclass.
- However, Y can also be a sequence, such as the same document in a different language.

Simple Recurrent Neural Network Architecture



- The hidden layer is a sequence of vectors A_ℓ , receiving as input X_ℓ as well as $A_{\ell-1}$. A_ℓ produces an output O_ℓ .
- The *same* weights \mathbf{W} , \mathbf{U} and \mathbf{B} are used at each step in the sequence — hence the term *recurrent*.
- The A_ℓ sequence represents an evolving model for the response that is updated as each element X_ℓ is processed.

RNN in Detail

Suppose $X_\ell = (X_{\ell 1}, X_{\ell 2}, \dots, X_{\ell p})$ has p components, and $A_\ell = (A_{\ell 1}, A_{\ell 2}, \dots, A_{\ell K})$ has K components. Then the computation at the k th components of hidden unit A_ℓ is

$$\begin{aligned} A_{\ell k} &= g\left(w_{k0} + \sum_{j=1}^p w_{kj} X_{\ell j} + \sum_{s=1}^K u_{ks} A_{\ell-1,s}\right) \\ O_\ell &= \beta_0 + \sum_{k=1}^K \beta_k A_{\ell k} \end{aligned}$$

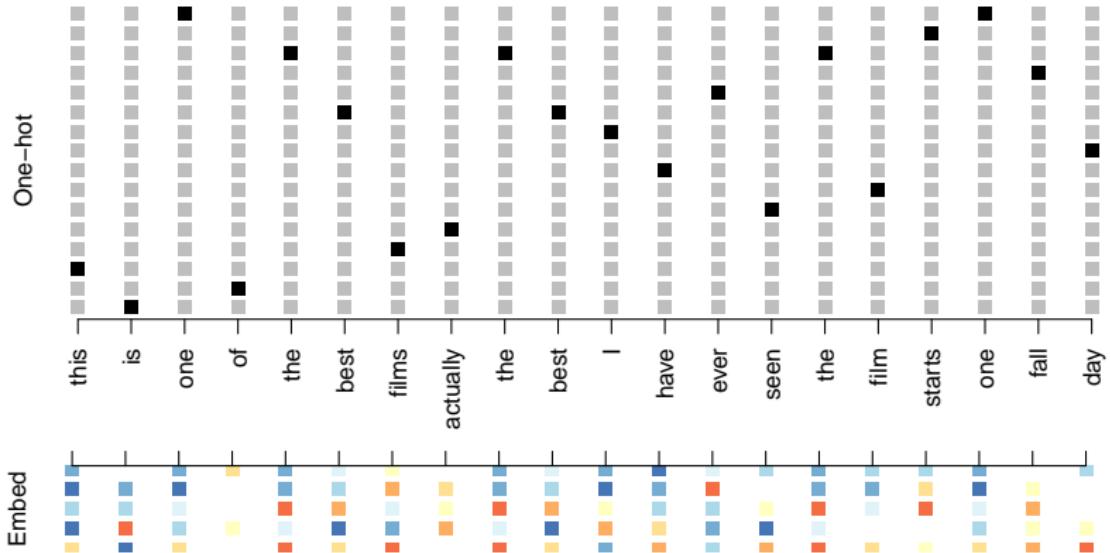
Often we are concerned only with the prediction O_L at the last unit. For squared error loss, and n sequence/response pairs, we would minimize

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

RNN and IMDB Reviews

- The document feature is a sequence of words $\{\mathcal{W}_\ell\}_1^L$. We typically truncate/pad the documents to the same number L of words (we use $L = 500$).
- Each word \mathcal{W}_ℓ is represented as a *one-hot encoded* binary vector X_ℓ (dummy variable) of length $10K$, with all zeros and a single one in the position for that word in the dictionary.
- This results in an extremely sparse feature representation, and would not work well.
- Instead we use a lower-dimensional pretrained *word embedding* matrix \mathbf{E} ($m \times 10K$, next slide).
- This reduces the binary feature vector of length $10K$ to a real feature vector of dimension $m \ll 10K$ (e.g. m in the low hundreds.)

Word Embedding



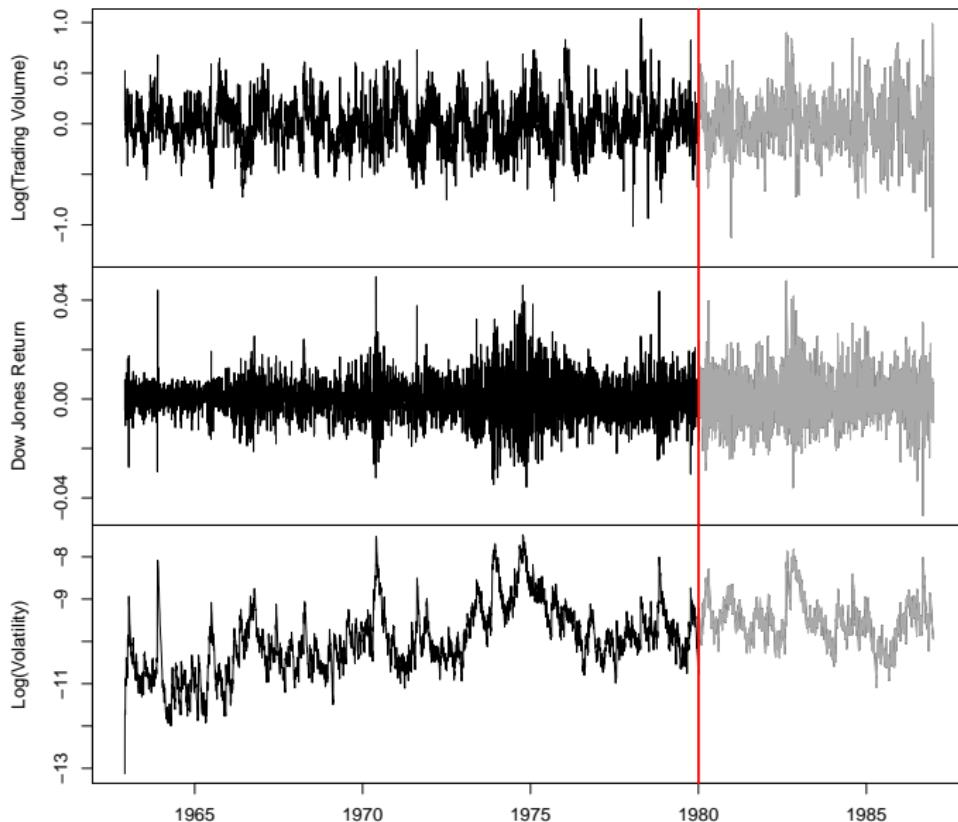
this is one of the best films actually the best I have ever seen the film
starts one fall day

Embeddings are pretrained on very large corpora of documents, using methods similar to principal components. **word2vec** and **GloVe** are popular.

RNN on IMDB Reviews

- After a lot of work, the results are a disappointing 76% accuracy.
- We then fit a more exotic RNN than the one displayed — a *LSTM* with *long and short term memory*. Here A_ℓ receives input from $A_{\ell-1}$ (short term memory) as well as from a version that reaches further back in time (long term memory). Now we get 87% accuracy, slightly less than the 88% achieved by *glmnet*.
- These data have been used as a benchmark for new RNN architectures. The best reported result found at the time of writing (2020) was around 95%. We point to a *leaderboard* in Section 10.5.1.

Time Series Forecasting



New-York Stock Exchange Data

Shown in previous slide are three daily time series for the period December 3, 1962 to December 31, 1986 (6,051 trading days):

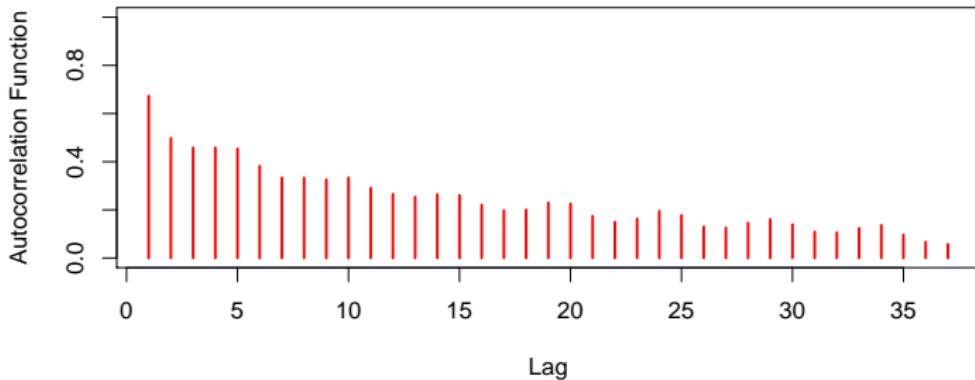
- **Log trading volume.** This is the fraction of all outstanding shares that are traded on that day, relative to a 100-day moving average of past turnover, on the log scale.
- **Dow Jones return.** This is the difference between the log of the Dow Jones Industrial Index on consecutive trading days.
- **Log volatility.** This is based on the absolute values of daily price movements.

Goal: predict **Log trading volume** tomorrow, given its observed values up to today, as well as those of **Dow Jones return** and **Log volatility**.

These data were assembled by LeBaron and Weigend (1998) *IEEE Transactions on Neural Networks*, 9(1): 213–220.

Autocorrelation

Log(Trading Volume)



- The *autocorrelation* at lag ℓ is the correlation of all pairs $(v_t, v_{t-\ell})$ that are ℓ trading days apart.
- These sizable correlations give us confidence that past values will be helpful in predicting the future.
- This is a curious prediction problem: the response v_t is also a feature $v_{t-\ell}$!

RNN Forecaster

We only have one series of data! How do we set up for an RNN?

We extract many short mini-series of input sequences

$X = \{X_1, X_2, \dots, X_L\}$ with a predefined length L known as the *lag*:

$$X_1 = \begin{pmatrix} v_{t-L} \\ r_{t-L} \\ z_{t-L} \end{pmatrix}, \quad X_2 = \begin{pmatrix} v_{t-L+1} \\ r_{t-L+1} \\ z_{t-L+1} \end{pmatrix}, \dots, \quad X_L = \begin{pmatrix} v_{t-1} \\ r_{t-1} \\ z_{t-1} \end{pmatrix}, \quad \text{and } Y = v_t.$$

Since $T = 6,051$, with $L = 5$ we can create 6,046 such (X, Y) pairs.

We use the first 4,281 as training data, and the following 1,770 as test data. We fit an RNN with 12 hidden units per lag step (i.e. per A_ℓ .)

RNN Results for NYSE Data

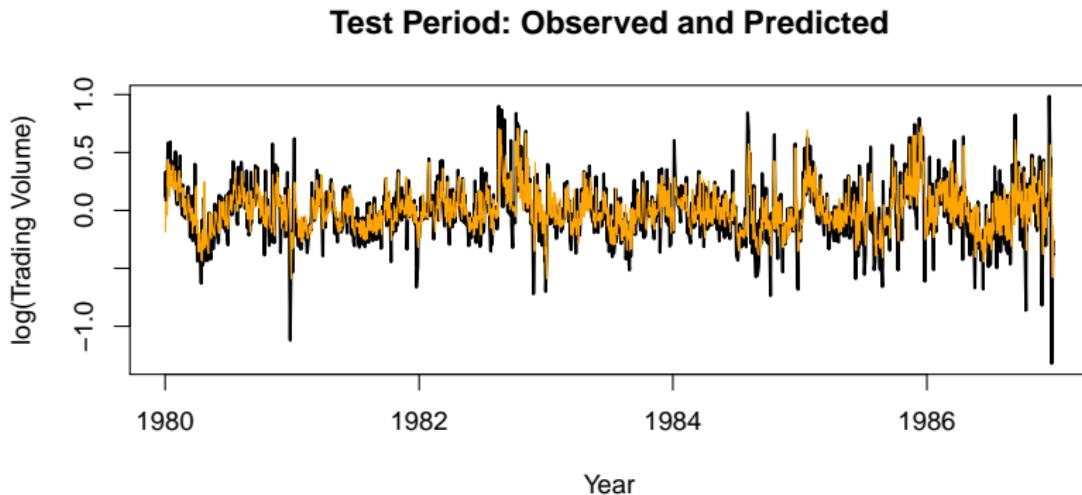


Figure shows predictions and truth for test period.

$R^2 = 0.42$ for RNN

$R^2 = 0.18$ for *straw man* — use yesterday's value of **Log trading volume** to predict that of today.

Autoregression Forecaster

The RNN forecaster is similar in structure to a traditional *autoregression* procedure.

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

Fit an OLS regression of \mathbf{y} on \mathbf{M} , giving

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

Known as an *order- L autoregression* model or AR(L).

For the **NYSE** data we can include lagged versions of **DJ_return** and **log_volatility** in matrix \mathbf{M} , resulting in $3L + 1$ columns.

Autoregression Results for NYSE Data

$R^2 = 0.41$ for AR(5) model (16 parameters)

$R^2 = 0.42$ for RNN model (205 parameters)

$R^2 = 0.42$ for AR(5) model fit by neural network.

$R^2 = 0.46$ for all models if we include **day_of_week** of day being predicted.

Summary of RNNs

- We have presented the simplest of RNNs. Many more complex variations exist.
- One variation treats the sequence as a one-dimensional image, and uses CNNs for fitting. For example, a sequence of words using an embedding representation can be viewed as an image, and the CNN convolves by sliding a convolutional filter along the sequence.
- Can have additional hidden layers, where each hidden layer is a sequence, and treats the previous hidden layer as an input sequence.
- Can have output also be a sequence, and input and output share the hidden units. So called **seq2seq** learning are used for language translation.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

Should we always use deep learning models?

- Often the big successes occur when the *signal to noise ratio* is high — e.g. image recognition and language translation. Datasets are large, and overfitting is not a big problem.
- For noisier data, simpler models can often work better.
 - On the **NYSE** data, the AR(5) model is much simpler than a RNN, and performed as well.
 - On the **IMDB** review data, the linear model fit by **glmnet** did as well as the neural network, and better than the RNN.
- We endorse the *Occam's razor* principle — we prefer simpler models if they work as well. More interpretable!