Foundations of Statistical and Machine Learning for Actuaries

Recurrent Neural Networks, Text Data

Edward (Jed) Frees, University of Wisconsin - Madison
Andrés Villegas Ramirez, University of New South Wales

July 2025

Schedule

Day and Time	Presenter	Topics	Notebooks for Participant Activity
Monday Morning	Jed	Welcome and Foundations Hello to Google Colab	Auto Liability Claims
	Jed	Classical Regression Modeling	Medical Expenditures (MEPS)
Monday Afternoon	Andrés	Regularization, Resampling, Cross-Validation	Seattle House Sales Data
	Andrés	Classification	Victoria Road Crash Data
Tuesday Morning	Andrés	Trees, Boosting, Bagging	
	Jed	Big Data, Dimension Reduction and Non-Supervised Learning	Big Data, Dimension Reduction, and Non-Supervised Learning
Tuesday Afternoon	Jed	Neural Networks	Seattle House Prices Claim Counts
	Jed	Graphic Data Neural Networks	MNIST Digits Data
Tuesday 4 pm	Fei	Fei Huang Thoughts on Ethics	
Wednesday Morning	Jed	Recurrent Neural Networks, Text Data	Insurer Stock Returns
	Jed	Artificial Intelligence, Natural Language Processing, and ChatGPT	
Wednesday After Lunch	Dani	Dani Bauer Insights	
Wednesday Afternoon	Andrés	Applications and Wrap-Up	

Wednesday Morning 5A. Recurrent Neural Networks, Text Data

This module covers:

- recurrent neural nets that deal with time-dependent sequences
- problems of vanishing and exploding gradients
- long short-term memory and gated recurrent unit solutions
- text data, and
- word embeddings

Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) deal with time-dependent and/or sequence-dependent problems.
  - This type of network is "recurrent" in the sense that they can revisit or reuse past states as inputs to predict the next or future states.
  - To put it plainly, they have memory.
- Recurrent neural networks are a family of neural networks for processing sequential data.
- Parameter sharing makes it possible to extend and apply the model to examples of different forms (different lengths, here) and generalize across them.
  - If we had separate parameters for each value of the time index, we could not generalize

Basic facts of RNNs

- A recurrent neural network (RNN) is a type of neural network that is designed to process sequences of data (e.g. time series, sentences).
  - A recurrent neural network is any network that contains a recurrent layer.
  - A recurrent layer is a layer that processes data in a sequence.
  - A RNN can have one or more recurrent layers.
- Because the output of a recurrent neuron at a time step is a function the inputs from previous time steps, they are sometimes called *memory cells*.

- Recurrent neuron are sometimes called *memory cells*.
  - A single recurrent neuron is a very basic cell, capable of learning only short patterns (typically about 10 steps long, but this varies depending on the task).

$$\begin{aligned} h^{(t)} &= f(h^{(t-1)}, x^{(t)}; \theta) \\ &= f(f(h^{(t-2)}, x^{(t-1)}; \theta), x^{(t)}; \theta) \\ &= \cdots = \end{aligned}$$

- A recurrent neural network that is "unrolled" or "unfolded"

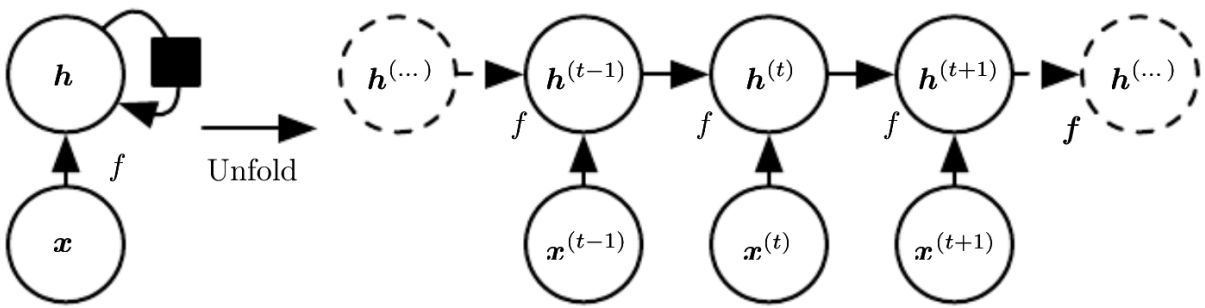


Figure 10.2: A recurrent network with no outputs. This recurrent network just processes information from the input $\boldsymbol{x}$ by incorporating it into the state $\boldsymbol{h}$ that is passed forward through time. *(Left)* Circuit diagram. The black square indicates a delay of a single time step. *(Right)* The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance.

- To train an RNN, the trick is to unroll it through time and then use regular backpropagation. This strategy is called **backpropagation through time**.
  - Just like in regular backpropagation, there is a first forward pass through the network
- Weights are shared over time; this allows the model to be used on arbitrary-length sequences.

- Here is a similar figure. Note here that the constant **W, U, B** remind us that the **same** weights are used as the sequence is processed.

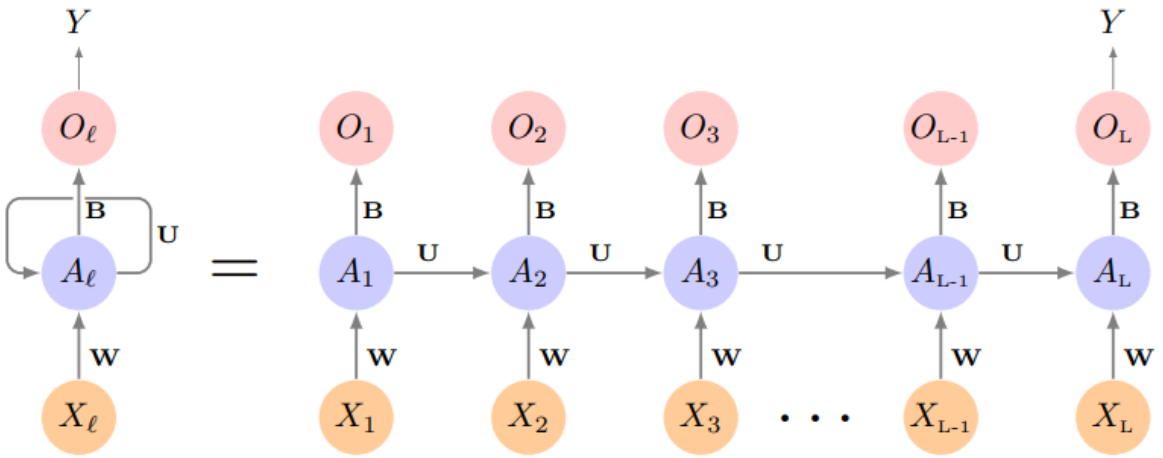


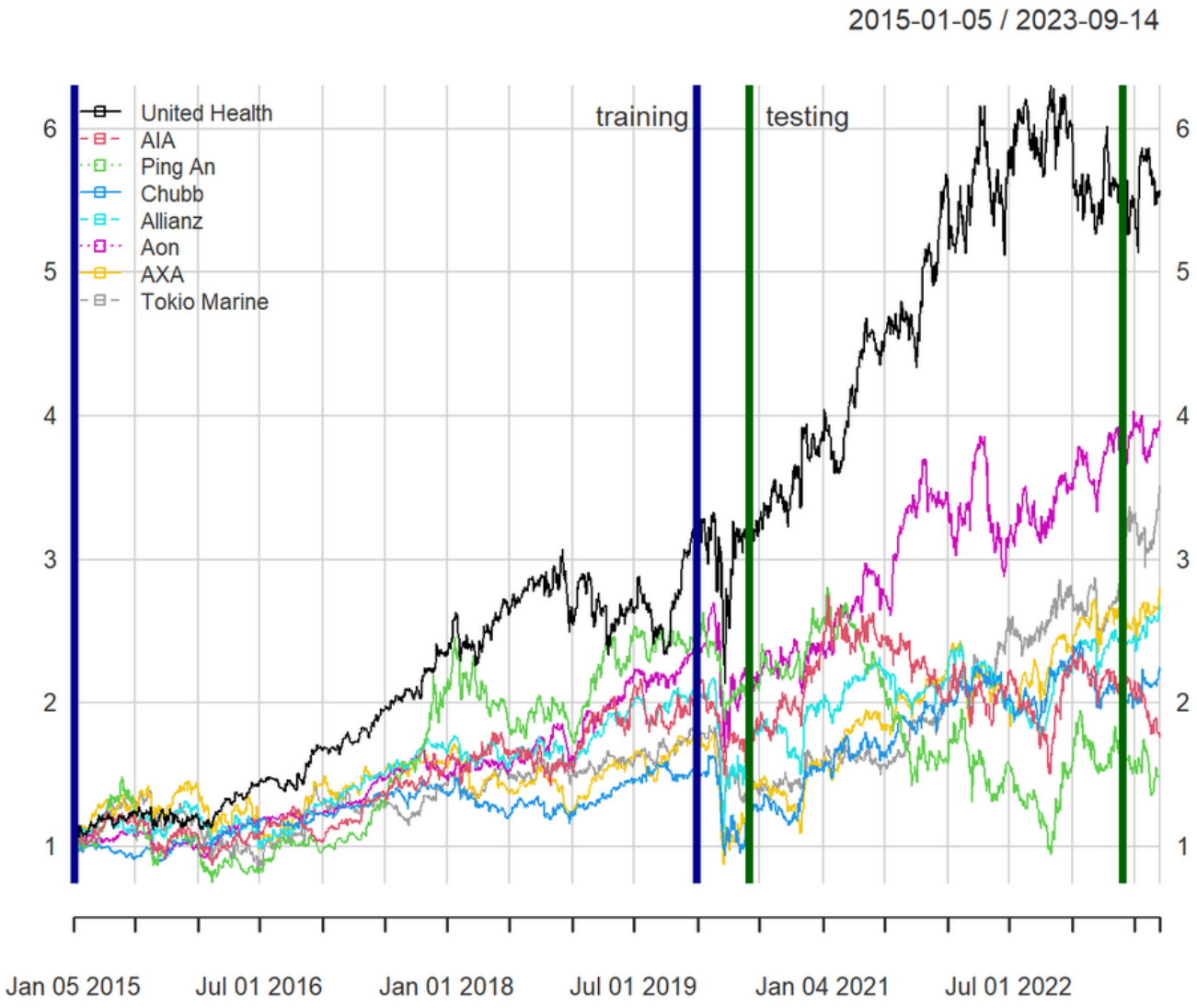
FIGURE 10.12. *Schematic of a simple recurrent neural network. The input is a sequence of vectors* $\{X_\ell\}_1^L$*, and here the target is a single response. The network processes the input sequence* X *sequentially; each* X_ℓ *feeds into the hidden layer, which also has as input the activation vector* $A_{\ell-1}$ *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*

Example. Forecasting Insurer Stock Returns

In the notebook **InsurerStockReturns.ipynb**, you will examine the largest insurance companies (by market capitalization) in each of eight countries were chosen. With ticker symbols in parens (), they are:

- United Health (UNH) - USA
- AIA Group Limited (1299.HK) - Hong Kong
- Ping An Insurance (Group) Company of China, Ltd. (2318.HK) - China
- Chubb (CB) - Switzerland
- Allianz (ALV.DE) - Germany
- Aon (AON) - UK
- AXA (CS.PA) - France
- Tokio Marine (8766.T) - Japan.

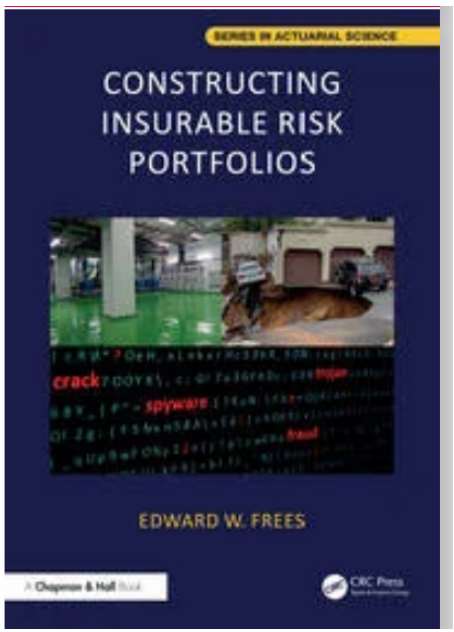
Daily stock prices were extracted from Yahoo Finance for the period 5 January 2015 to 14 September 2023.

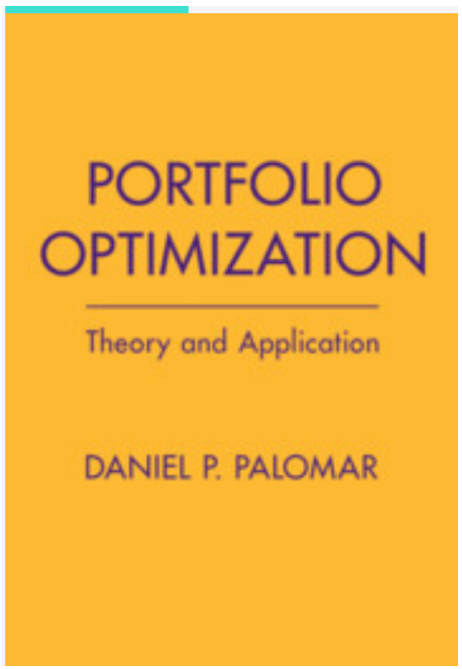


The notebook will demonstrate traditional time series forecasting such as an autoregressive model and simple recurrent NNs. Here is some illustrative code for an RNN.

```
#from keras.models import Sequential
#from keras.layers import SimpleRNN, Dense

random.seed(2025)

model = Sequential()
model.add(SimpleRNN(units=8, input_shape=(lag, 1),
                    activation='tanh'))
model.add(Dense(1))  # output layer

model.compile(optimizer='adam', loss='mse')
model.summary()
```

- These data are introduced in my recent book Constructing Insurable Risk Portfolios, Edward Frees, 2025
- Stock returns are notoriously difficult to forecast. To learn more, see the recent book by Daniel Palomar Portfolio Optimization: Theory and Applications, 2025





LSTM and Variants

Vanishing/Exploding Gradients Problems

- The backpropagation algorithm's second phase works by going from the output layer to the input layer, propagating the error gradient along the way.
  - Once the algorithm has computed the gradient of the cost function with regard to each parameter in the network, it uses these gradients to update each parameter with a gradient descent step.
  - Unfortunately, gradients often get smaller and smaller as the algorithm progresses down to the lower layers.
  - As a result, the gradient descent update leaves the lower layers' connection weights virtually unchanged, and training never converges to a good solution.
  - This is called the **vanishing gradients** problem.

- In some cases, the opposite can happen: the gradients can grow bigger and bigger until layers get insanely large weight updates and the algorithm diverges.
  - This is the **exploding gradients** problem, which surfaces most often in recurrent neural networks.
- Deep neural networks can suffer from unstable gradients; different layers may learn at widely different speeds.

{ *Source*: See the discussion of the gradient problem in Chapter 11 of Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media. }

Gradients Problems - Math

- Consider hidden states and outputs

$$h_t = f(x_t, h_{t-1}, w_h) \quad \text{and} \quad o_t = g(h_t, w_o)$$

and the loss function (measure the difference between target and the outputs)

$$L(X_1, \dots, X_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T} \sum_{t=1}^{T} l(y_t, o_t)$$

By the chain rule

$$\begin{aligned} \frac{d\ L}{d\ w_h} &= \frac{1}{T} \sum_{t=1}^{T} \frac{d\ l(y_t, o_t)}{d\ w_h} \\ &= \frac{1}{T} \sum_{t=1}^{T} \frac{d\ l(y_t, o_t)}{d\ o_t} \frac{d\ g(h_t, w_o)}{d\ h_t} \frac{d\ h_t}{d\ w_h} \end{aligned}$$

Computing $\frac{d\ h_t}{d\ w_h}$ is the difficult term.

To do so, think about a_t, b_t, c_t such that $a_t = b_t + c_t a_{t-1}$ and let $a_0 = 0$. It is easy to check that

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{k=i+1}^{t} c_k \right) b_i$$

From this, we have

$$\begin{aligned} \frac{dh_t}{dw_h} &= \frac{\partial h_t}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{k=i+1}^{t} \frac{\partial h_k}{\partial h_{k-1}} \right) \frac{\partial h_i}{\partial w_h} \\ &= \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} \\ &\quad + \sum_{i=1}^{t-1} \left(\prod_{k=i+1}^{t} \frac{\partial f(x_k, h_{k-1}, w_h)}{\partial h_{k-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h} \end{aligned}$$

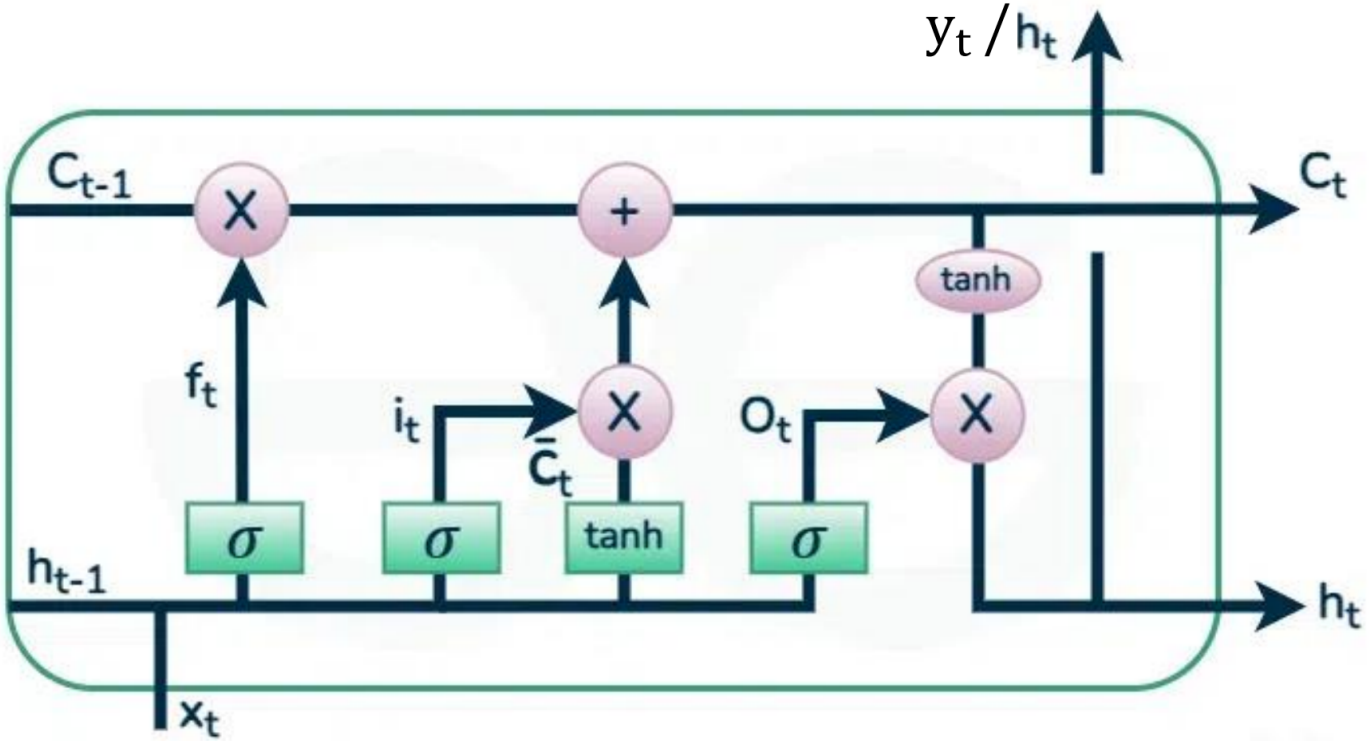
While we can use the chain rule to compute $\frac{d\ h_t}{d\ w_h}$ recursively, this chain can get very long whenever T is large. This makes RNNs prone to vanishing/exploding gradients.

{ *Credit*: Dive Into Deep Learning, Section 9.7 }

LSTM

- One solution to the gradient problem is a **Long Short-Term Memory (LSTM)** neural network.
  - This is a type of RNN specifically designed to handle sequential data and capture long-term dependencies.
- **Memory Cells and Gates**: LSTMs use memory cells and three types of gates (input, forget, and output gates) to control the flow of information, allowing them to remember and forget information as needed.
- LSTMs address the vanishing gradient problem by maintaining more stable gradients during training.
- Although powerful, LSTMs have a more complex architecture compared to simple RNNs, making them computationally expensive and slower to train.

LSTM Cell



{ *Credit*: Dani Bauer Lecture Notes }

Gates

- Each memory cell is equipped with an internal state and a number of multiplicative gates.
  - *Gating* is a dedicated mechanism for when a hidden state should be updated and when it should be reset.
  - The **input** gate determines how much of the input node's value should be added to the current memory cell internal state.
  - The **forget** gate determines whether to keep the current value of the memory or flush it.
  - The **output** gate determines whether the memory cell should influence the output at the current time step.

$$
\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{\text{xi}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{hi}} + \mathbf{b}_{\text{i}}),\\
\mathbf{F}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{\text{xf}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{hf}} + \mathbf{b}_{\text{f}}),\\
\mathbf{O}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{\text{xo}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{ho}} + \mathbf{b}_{\text{o}}),
\end{aligned}
$$

{ *Credit*: Dive Into Deep Learning, Section 10.1 }

GRU

- The gated recurrent unit (GRU) cell is a simplified version of LSTM that seems to work just as well.
- The LSTM's three gates are replaced by two: the reset gate and the update gate.

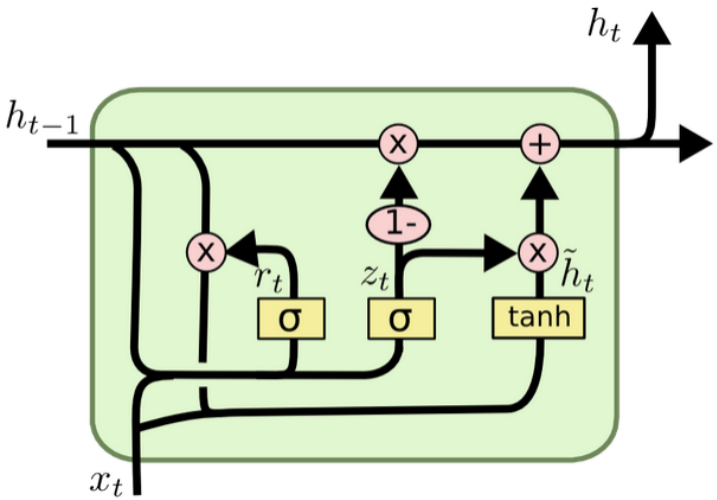


Diagram of a GRU cell.

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

{ *Credit*:UNSW: Artificial Intelligence and Deep Learning Models for Actuarial Applications
Olah (2015), Understanding LSTM Networks, Colah's Blog. }

In practice

The notebook features some sample code. Easy to implement!

```
# Model with an LSTM layer

model = Sequential()
model.add(SimpleRNN(units=8, input_shape=(lag, 1),
                    activation='tanh', return_sequences=True))
model.add(LSTM(units=8, activation='tanh'))  # No return_sequences
                                             # because it's the last RNN
model.add(Dense(1))

# Model with a GRU layer

model = Sequential()
model.add(GRU(units=16, input_shape=(lag, 1), return_sequences=True))
model.add(GRU(units=8))
model.add(Dense(1))
```

Text Data

Opportunities

- Text data naturally occurs in sequences and so we begin our study here (as RNNs).
- Machine learning provides actuaries and statisticians with terrific approaches for handling graphic and text data.
- In computer science, one refers to human languages, like English or Spanish, as "natural" languages, to distinguish them from languages that were designed for machines, like Assembly.
- Creating algorithms that can make sense of natural language is a big deal: language, and in particular text, underpins most of our communications and our cultural production.
    - The internet is mostly text.
    - Language is how we store almost all of our knowledge.

Insurance Examples of Text Data

1. Claims processing and triage
2. Underwriting support from unstructured documents
3. Customer sentiment and service feedback
4. Fraud detection from narrative discrepancies
5. Policy recommendations and **chatbots**
6. Litigation and legal document review

Tokenization

To prepare the data, one needs to:

- Standardize the text to make it easier to process, such as by converting it to lowercase or removing punctuation.
- Split the text into units (called tokens), such as characters, words, or groups of words. This is called **tokenization**.
- Convert each such token into a numerical vector.

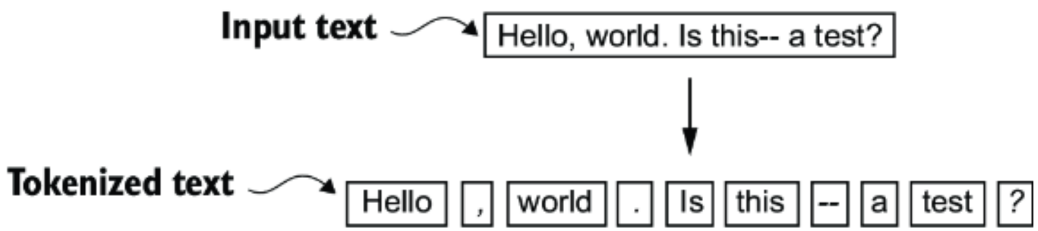


Figure 2.5 The tokenization scheme we implemented so far splits text into individual words and punctuation characters. In this specific example, the sample text gets split into 10 individual tokens.

{ *Credit*: Build a Large Language Model (From Scratch) }

Tokenization could be done as one of:

- *Word-level tokenization* — Where tokens are space-separated (or punctuation separated) substrings.
  - A variant of this is to further split words into subwords when applicable—for instance,
  - treating “staring” as “star+ing” or “called” as “call+ed.”
- *N-gram tokenization* — Where tokens are groups of N consecutive words. For instance, “the cat” or “he was” would be 2-gram tokens (also called bigrams).
- *Character-level tokenization*

Dictionary Approaches

- Just treats each all words as a categorical variable
  - Use a dummy (0-1) variable (*one-hot encoder*) to identify each word
  - It is common to restrict the vocabulary to only the top 20,000 or 30,000 most common words found in the training data.
- For analysis, one might use a *bag of word* approach.
  - Simply list all of the words in a "corpus" of documents and count how many occur.
  - Disregards grammar and word order but at least identifies word frequency
  - Yields data for word clouds
- But, no natural notion of **similarity** among words
  - Dummy variable encodings are orthogonal among different words, even words that are very similar to one another..

Dictionary Approaches 2

- Might use a dictionary approach for
  - **Sentiment Analysis of Customer Feedback.** Use a sentiment dictionary to flag negative experiences or dissatisfaction.
  - **Topic Detection in Claims**. Custom-built dictionaries (e.g., “fire”, “water”, “injury”) help classify claim types from free-text descriptions.
  - **Fraud Detection**. Track emotionally charged or over-explained descriptions. Use dictionaries for exaggeration (e.g., “severe”, “massive”) or urgency (“urgent”, “emergency”).
  - **Risk Scoring in Underwriting**. Analyze reports (e.g., inspections, physician notes) for high-risk terms.

Modern NLP approaches - Embeddings

- Recall earlier we introduced the idea of *representational learning* where a learned layer represents features such as small edges and patches of color in a CNN image data fit.
- An **embedding** is similar in that it is based on a low-dimensional (at least compared to the input space) vector that can be interpreted as representing some aspect of the inputs.
- A key difference is that embeddings are derived solely from the features and do not include information from the target variables.

Word Embeddings

- **Embeddings** convert real-world objects (like words and images) into mathematical representations (vectors of numbers)
- These representations capture inherent properties and relationships between real-world objects
  - For example, we can assess similarity or distance between objects
  - These representations can also then be used in other models as numerical inputs
- A *word embedding* is a way of representing words as vectors in a multi-dimensional space, where the distance and direction between vectors reflect the similarity and relationships among the corresponding words.
  - For example, we will argue that *positions of words* preserve semantic meaning; e.g. synonyms should appear near each other.

Similarity Among Words

- To measure similarity between two target words **v** and **w**, we need a metric that takes two vectors and gives a measure of their similarity.
  - A common similarity metric is the **cosine** of the angle between the vectors.
  - The cosine — like most measures for vector similarity used in NLP—is based on dot product (also called the inner product):

$$\textbf{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{p} v_i w_i$$

$$\textbf{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|}$$

    where $|\mathbf{w}| = \sqrt{\mathbf{w} \cdot \mathbf{w}}$.
- Distributional semantics - a word's meaning is given by the words that frequently appear close to it.
  - J.R. Firth "you shall know a word by the company it keeps."
  - You can represent a word by what context it appears in.

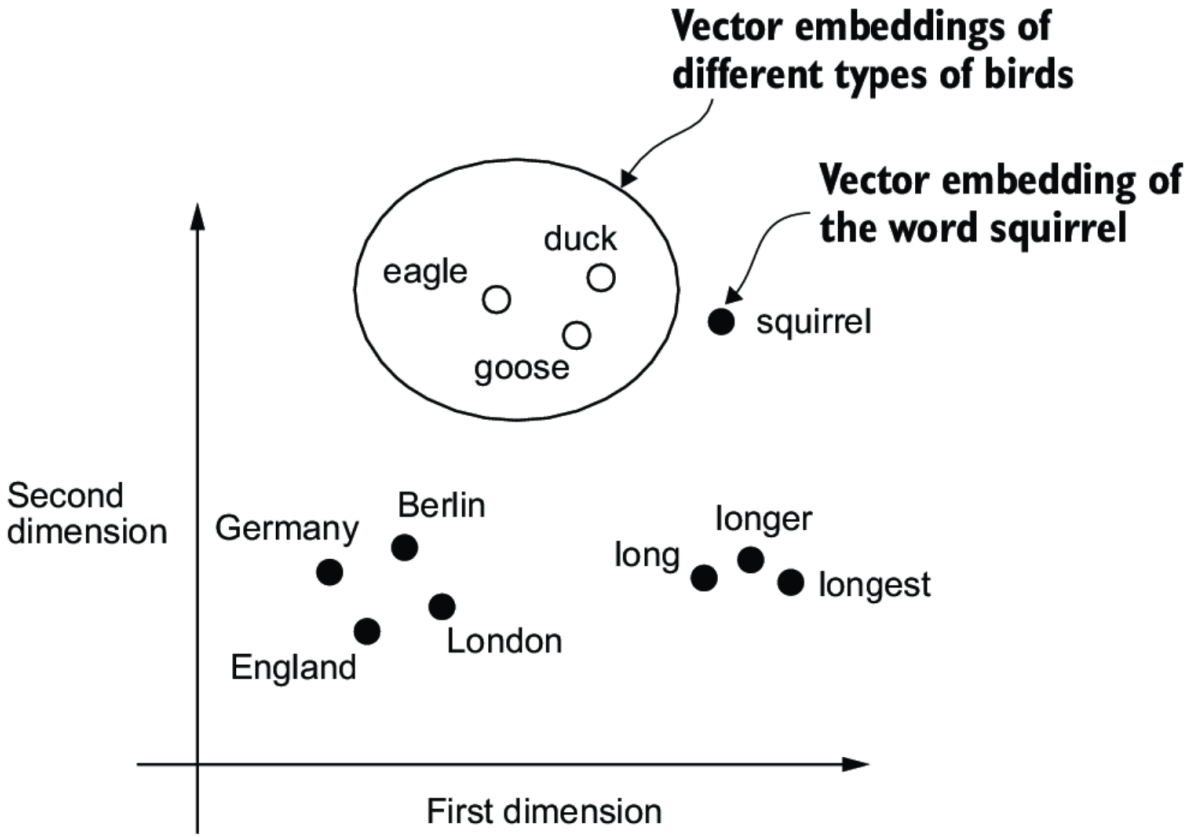


Figure 2.3 If word embeddings are two-dimensional, we can plot them in a

{ *Credit*: Build a Large Language Model (From Scratch) }

Word Embedding Actuarial Application

- Let us consider *Actuarial Applications of word embedding models* by Lee, Manski, and Maiti, in *Astin Bulletin* 50(1), 2019.
- This paper demonstrates how textual data can be easily used in insurance analytics.
  - Using the concept of word similarities, it illustrates how to extract variables from text and
  - incorporate them into claims analyses using standard generalized linear model or generalized additive regression model.
- Descriptions for the observed insurance claims are recorded in the data set.
  - These claim descriptions are human generated, and there are 2797 unique words found in the training sample and validation sample all together
  - The claim descriptions are short phrases, such as *lightning damage to building*, or *vandalism damage at recycle center*.

- The number of claims observations is 4991 in the training sample and 1039 in the validation sample, which totals to 6030 observations.

Common words include: “damage,” “vandalism”, “damaged”, “lightening”, and so forth.

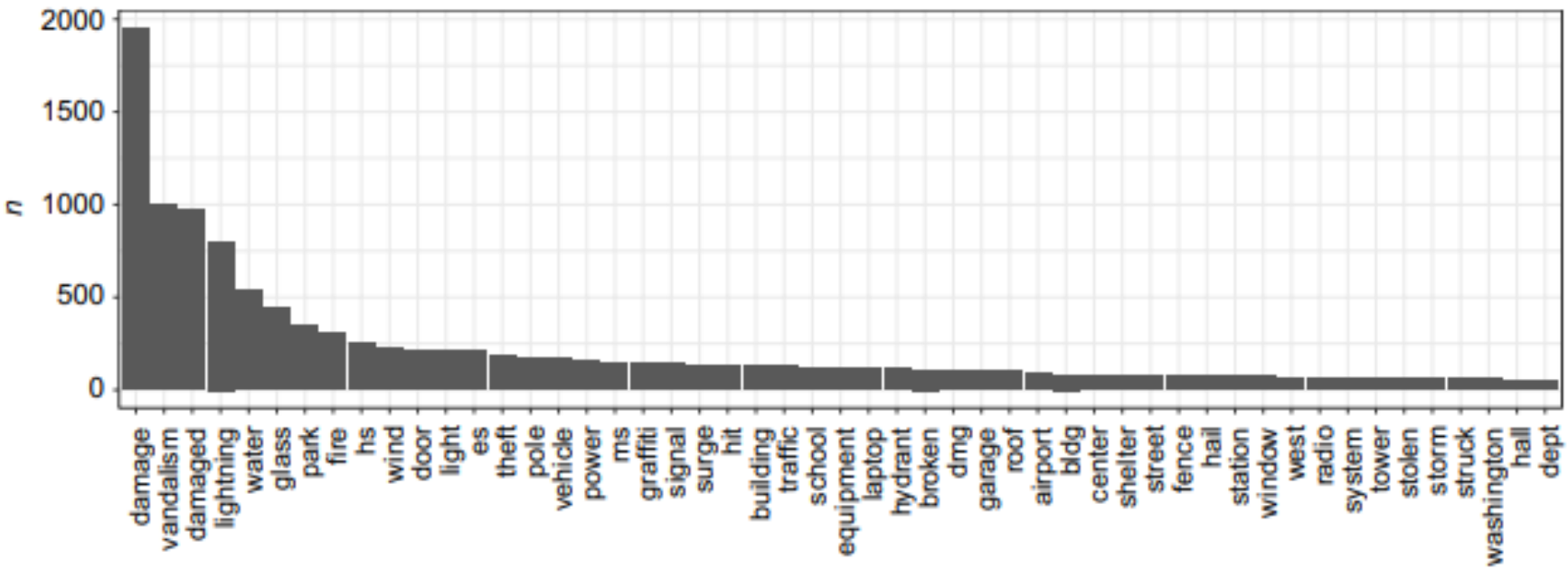


FIGURE 2: Distribution of common words.

One can project each word onto a 2 dimensional space, a "word vector". Words are said to be similar if the cosine between them is large.

Measuring cosine similarity, no similarity is expressed as a 90 degree angle, while total similarity of 1 is a 0 degree angle, complete overlap;

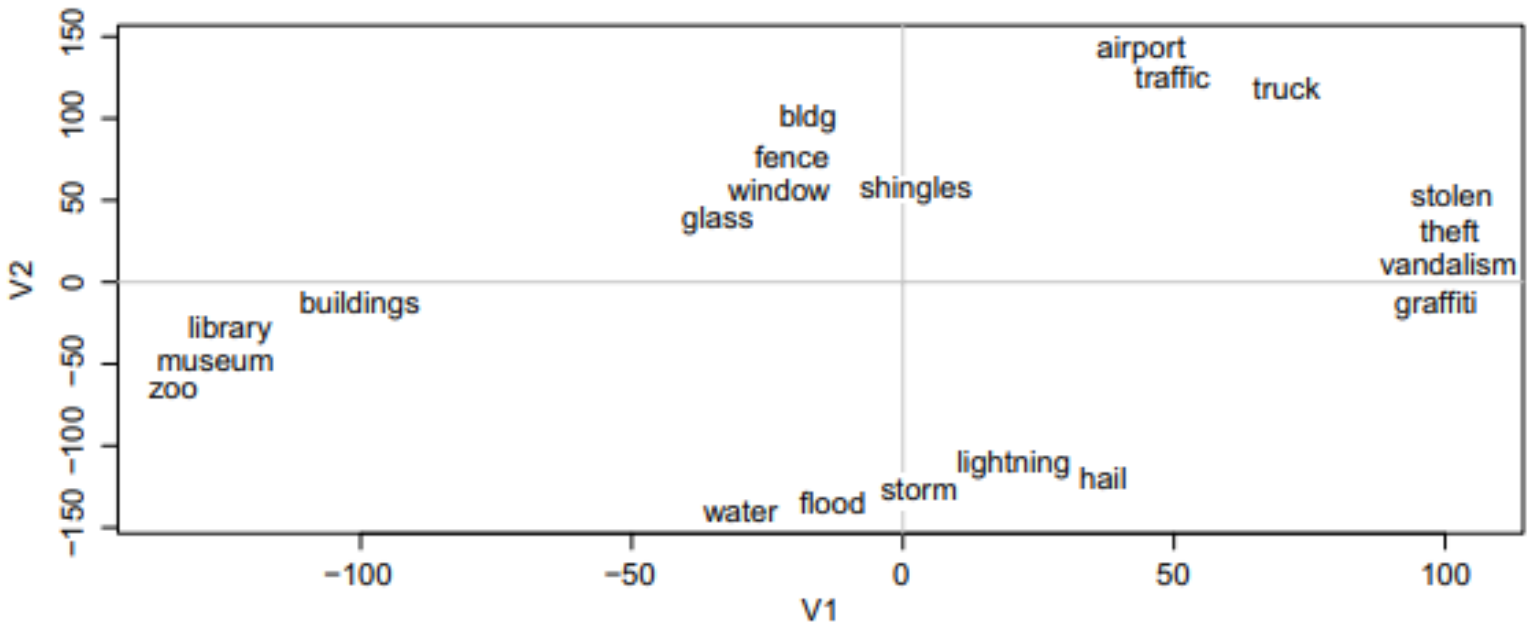


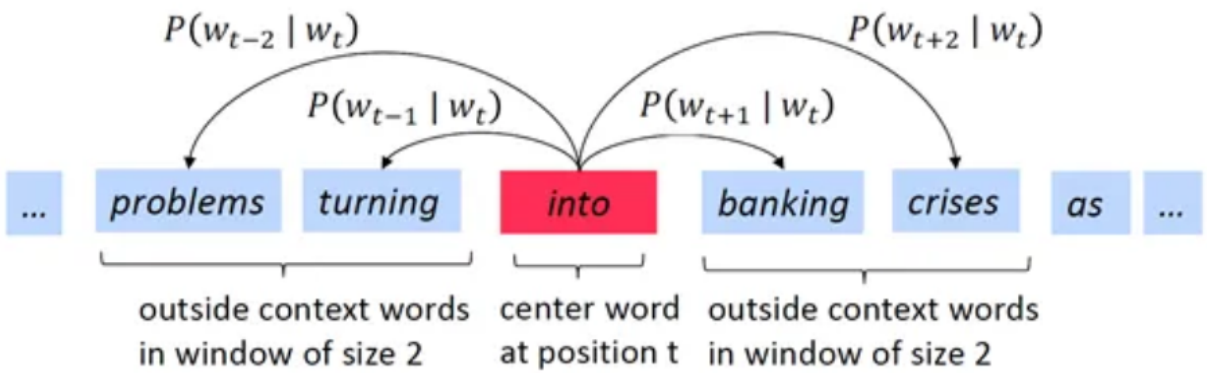
FIGURE 1: Two-dimensional projection of the word embeddings for common words.

Word Embedding Calculation

- To see how these embeddings are computed, let us examine the so-called "Word2Vec" algorithm using the "skip-gram" option.
- Word2vec Idea:
  - We have a large corpus (latin word for "body") of text
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word c and context ("outside") words o
  - Use the similarity of word vectors c and o to calculate the probability of o given c (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

- Consider the sentence: *problems turning into banking crises*
- Let us focus on the center word *into* w_t with $t = 3$, and think of the others as "context words". The context words are in a window of size $m = 2$.
- Let us first calculate the probability predicting the context words given the center word.

$$\prod_{-m \leq j \leq m, j \neq 0} \Pr(w_{t+j} | w_t; \theta)$$



Credits: Stanford CS224N

- Now, each word will be represented twice, once as a center and once as a context word.
  - To do that, we will represent each word by two sets of vectors, Uw and Vw.
  - Using these two vectors, our probability equation for center word o and context word c will look like this:

$$\Pr(O = o|C = c) = \frac{\exp(u'_o v_c)}{\sum_{w \in Vocab} \exp(u'_w v_c)}$$

- The numerator $u'_o v_c$ is a dot product and captures the **similarity** between these two vectors. **Softmax** - sort of like a max ...
- With this, the overall likelihood is of the form:

$$L(\theta) = \prod_t \prod_{-m \leq j \leq m, j \neq 0} \Pr(w_{t+j}|w_t; \theta)$$

- The algorithm does the usual things to maximize this. Think of it as a nonlinear by a variant of principal components.

{ *Credits:* https://medium.com/analytics-vidhya/maths-behind-word2vec-explained-38d74f32726b }

In practice, the algorithm is used to fit very large corpus of documents

In addition, each word is represent by a large vector (not just 2)

- Dimension is 200-300 for popular applications
- Dimension is 300 for Google's pretrained Word2Vec model (trained on Google News, about 100 billion words)

{ *Source:* https://wiki.pathmind.com/word2vec

Source: "Actuarial Applications of word embedding models" by Lee, Manski, and Maiti, in Astin Bulletin 50(1), 1-24. doi:10.1017/asb.2019.28, 2019. }

Word Embedding Applications

{ *Source:* https://www.ibm.com/think/topics/word-embeddings }
Word embeddings can be used for:

- **Text classification**. Word embeddings are often used as features in text classification tasks, such as sentiment analysis, spam detection and topic categorization.
- **Named Entity Recognition (NER)**. To accurately identify and classify entities (e.g., names of people, organizations, locations) in text, word embeddings help the model understand the context and relationships between words.
- **Machine translation**. In machine translation systems, word embeddings help represent words in a language-agnostic way, allowing the model to better understand the semantic relationships between words in the source and target languages.
- **Information retrieval**. In information retrieval systems, word embeddings can enable more accurate matching of user queries with relevant documents, which improves the effectiveness of search engines and recommendation systems.
- **Question answering**. Word embeddings contribute to the success of question answering systems by enhancing the understanding of the context in which questions are posed and answers are found.
- **Semantic similarity and clustering**. Word embeddings enable measuring semantic similarity between words or documents for tasks like clustering related articles, finding similar documents or recommending similar items based on their textual content.
- **Similarity and analogy**. Word embeddings can be used to perform word similarity tasks (e.g., finding words similar to a given word) and word analogy tasks (e.g., "king" is to "queen" as "man" is to "woman").
- **Text generation**. In text generation tasks, such as language modeling and autoencoders, word embeddings are often used to represent the input text and generate coherent and contextually relevant output sequences.

Potential Actuarial Applications of Embedding Concepts

Recall that in many actuarial applications of pricing categorical covariates are utilized.

- The traditional approach is to use dummy variables (almost the same as one-hot encoding) to represent them.
  - Can be very computationally efficient (think traditional ANOVA and AnCova where closed-form solutions are available)
  - Nonetheless, each dummy variable is orthogonal to one another, no opportunities for assessing the similarity among categories
- In the future, it may be that we represent categories through "low" dimensional continuous vectors
  - Such an approach will allow us to assess similarity.
  - This in turn will allow us to accommodate interactions.
  - It may be that these representations will easily allow for interpretability, a key actuarial concern.

See, for example, the discussion and references to this recent literature beginning in Section 2.3 of AI Tools for Actuaries (May 9, 2025)

Session 5A. Recurrent Neural Networks, Text Data Summary

This module covered:

- recurrent neural nets that deal with time-dependent sequences
- problems of vanishing and exploding gradients
- long short-term memory and gated recurrent unit solutions
- text data, and
- word embeddings

During lab, participants may follow the notebook Insurer Stock Returns.