Recurrent neural network

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Sequence data

- 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

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 …

25ms
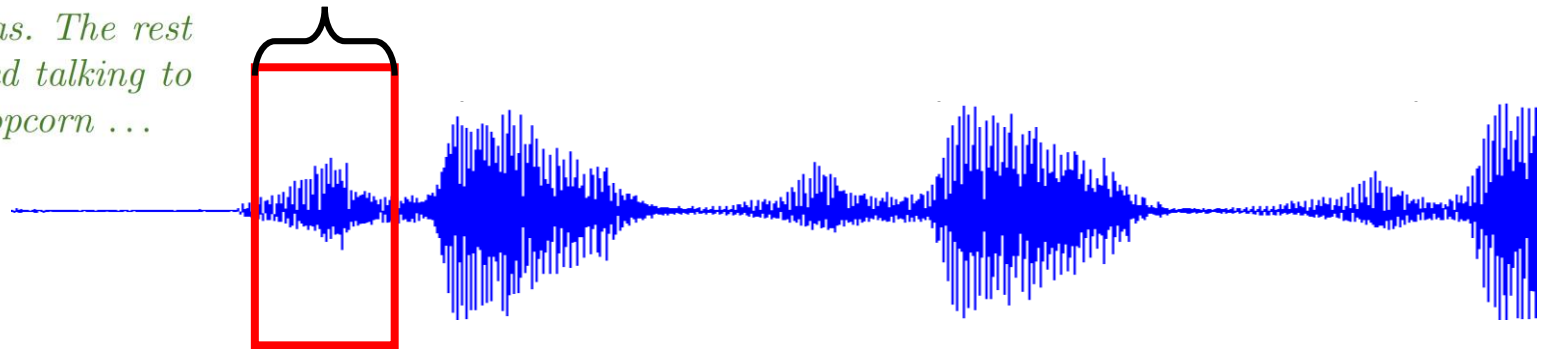
Time series

▸ A *time series* can be any data obtained via measurements at regular intervals, like the daily price of a stock, the hourly electricity consumption of a city, or the weekly sales of a store. Common tasks for time series including

1. **Forecast**: Predicting what will happen next in a series
2. **Classification**: Assign one or more categorical labels to a time series. For instance, given the time series of the activity of a visitor on a website, classify whether the visitor is a bot or a human
3. **Event detection**: Identify the occurrence of a specific expected event within a continuous data stream. For instance, "hot word detection," where a model monitors an audio stream and detects utterances like "Ok Google" or "Hey Alexa."
4. **Anomaly detection**: is typically done via unsupervised learning, because you often don't know what kind of anomaly you're looking for, so you can't train on specific anomaly examples.

Time series

▸ When working with time series, you'll encounter a wide range of domain-specific data representation techniques

- ▸ For instance, Fourier transform, which consists of expressing a series of values in terms of a superposition of waves of different frequencies. The Fourier transform can be highly valuable when preprocessing any data that is primarily characterized by its cycles and oscillations

- ▸ In the context of deep learning, Fourier analysis and other domain-specific representations can be useful as a form of *feature engineering*, a way to prepare data before training a model on it, to make the job of the model easier

- ▸ There are various statistical models and methods for time series

Natural Language Processing (NLP)

▸ In computer science, we refer to human languages, like English or Mandarin, as "natural" languages, to distinguish them from languages that were designed for machines, like C++, R, or Python

  ▸ Every machine language was *designed*: its starting point was a human engineer writing down a set of formal rules to describe what statements you could make in that language and what they meant

  ▸ With human language, it's the reverse: usage comes first, rules arise later. Natural language was shaped by an evolution process, much like biological organisms— that's what makes it "natural." Its "rules," like the grammar of English, were formalized after the fact and are often ignored or broken by its users

  ▸ As a result, while machine-readable language is highly structured and rigorous, natural language is messy—ambiguous, chaotic, sprawling, and constantly in flux

Natural Language Processing (NLP)

▸ That's what modern NLP is about: using machine learning and large datasets to give computers the ability to understand language and to ingest a piece of language as input and return something useful

   ▸ "What's the topic of this text?" (text classification)

   ▸ "Does this text contain abuse?" (content filtering)

   ▸ "Does this text sound positive or negative?" (sentiment analysis)

   ▸ "What should be the next word in this incomplete sentence?" (language modeling)

   ▸ "How would you say this in German?" (translation)

   ▸ "How would you summarize this article in one paragraph?" (summarization)

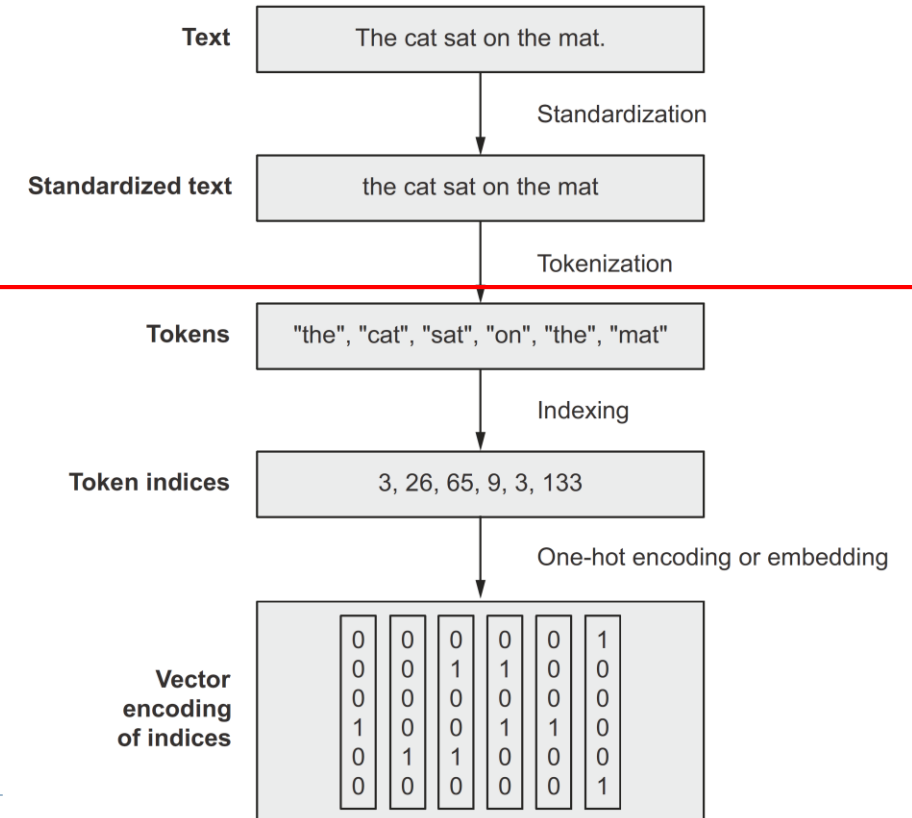# Featurization or vectorization

▸ Text standardization

  ▸ Text standardization is a basic form of feature engineering that aims to erase encoding differences that you don't want your model to have to deal with

▸ Text splitting (tokenization)

  ▸ Word-level tokenization, N-gram tokenization or Character-level tokenization

▸ Vocabulary indexing

  ▸ Encode each token into a numerical representation. Keep a small dictionary (restrict the number of tokens) and replace unknown word with 1 (Out Of Vocabulary, OOV token)

  ▸ Masking is usually represented by 0

Text | The cat sat on the mat.

Standardization

Standardized text | the cat sat on the mat

Tokenization

Tokens | "the", "cat", "sat", "on", "the", "mat"

Indexing

Token indices | 3, 26, 65, 9, 3, 133

One-hot encoding or embedding

Vector encoding of indices

Featurization or vectorization - Word Embedding

‣ The sequence model

- ‣ In order to preserve the order, you'd start by representing input samples as sequences of integer indices Then, you map each integer to a vector to obtain vector sequences
- ‣ To condense our representation and preserve the semantic relationship between these words, we may use *Word embeddings* instead of one-hot encoding

‣ Transformer

- ‣ Order-agnostic, yet it injects word-position information, which enables it to simultaneously look at different parts of a sentence while still being order-aware
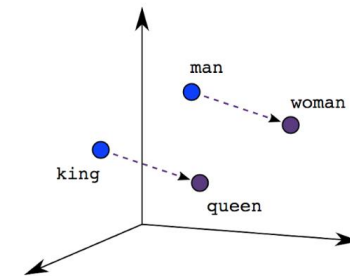
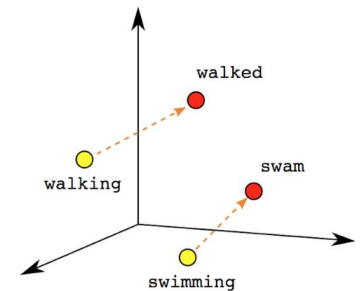# Featurization or vectorization - Word Embedding

▸ Embeddings are pretrained on very large corpora of documents, using methods similar to principal components. *word2vec* and *GloVe* are popular
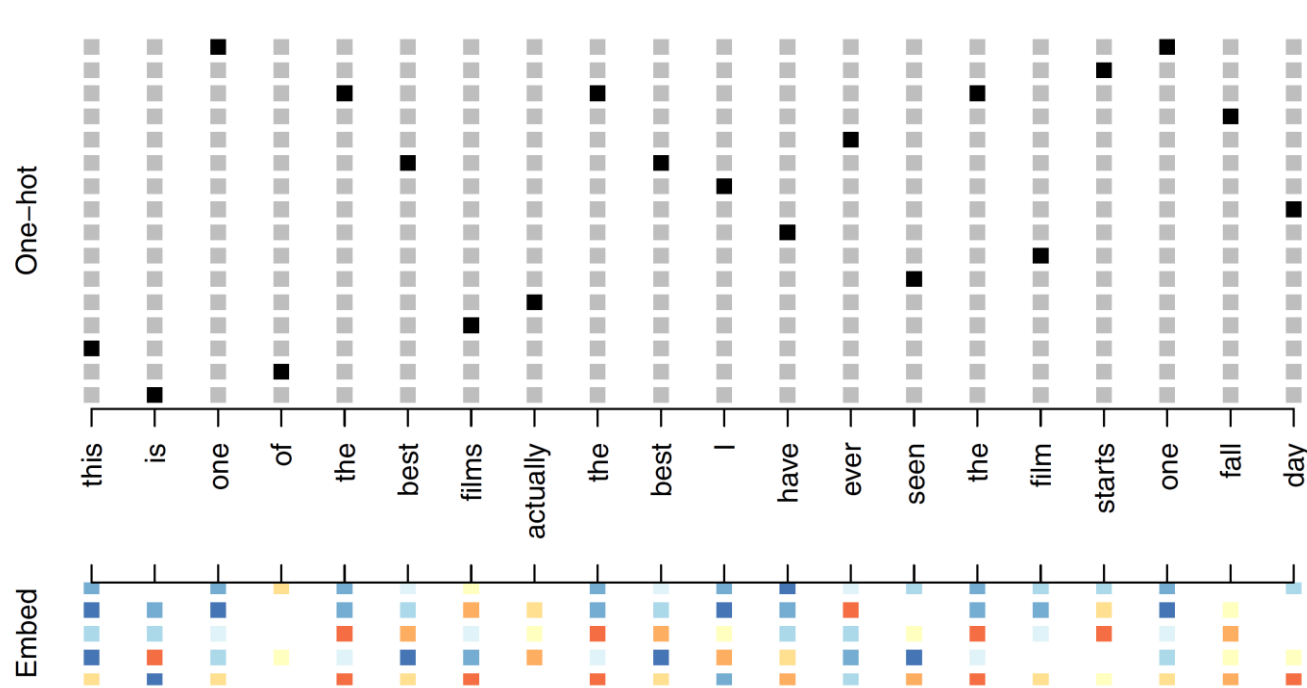
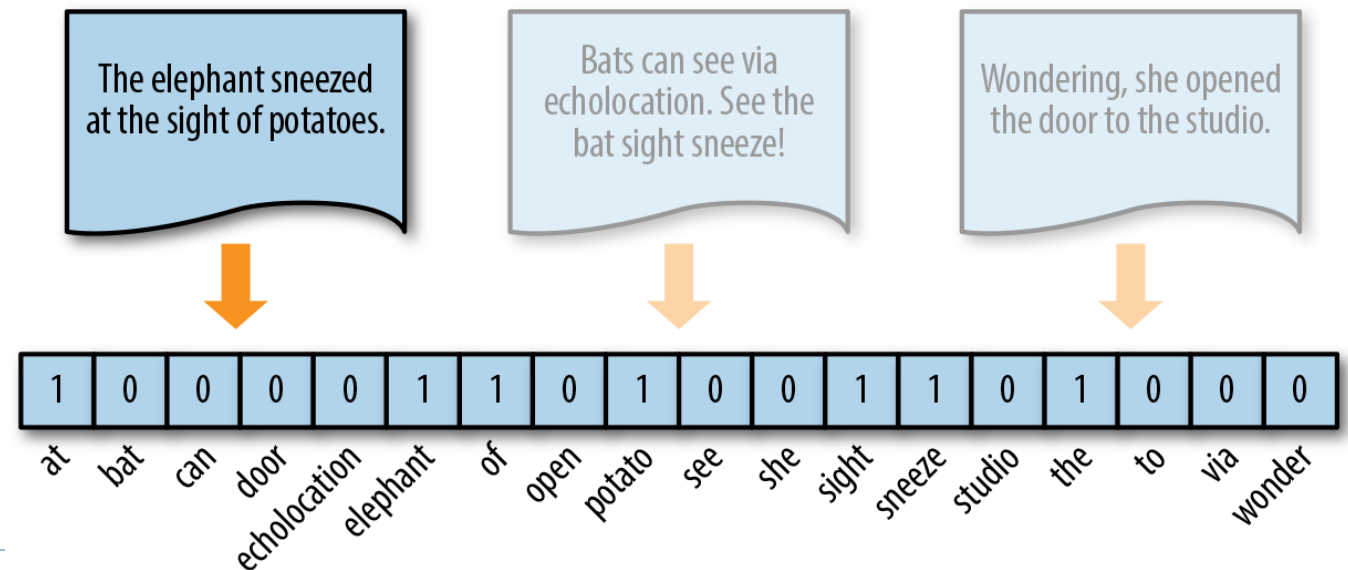- Sparse
- High-dimensional
- Hardcoded

Male-Female

Verb Tense

- Dense
- Low-dimensional
- Learned from data
- Structured

Featuralizion or vectorization - Representing groups of words

▸ **Bag-of-words models: discard order and treat text as an unordered set of words**

  ▸ 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

  ▸ The main advantage of this encoding is that you can represent an <u>entire text as a single vector</u> using multi-hot, count encoding or TF-IDF

▸ **Bag-of-words are unigrams. To take context into consideration, we can use** *bag-of-n-gram*

 ▸ the cat sat on the mat using Bag of 2-grams {"the cat", "cat sat", "sat on", "on the", "the mat"}

https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html

1. Recurrent Neural Networks (RNN)

▸ We will discuss recurrent neural networks (RNNs), a class of nets that can predict the future
  - ▸ They can analyze time series data
  - ▸ They can work on sequences of arbitrary lengths
  - ▸ They can take sentences, documents, or audio samples as input, making them extremely useful for natural language processing or speech recognition

▸ RNNs are not the only types of NN capable of handling sequential data: for small sequences, a regular dense network can do the trick; and for very long sequences, such as audio samples or text, convolutional neural networks or transformer can actually work quite well
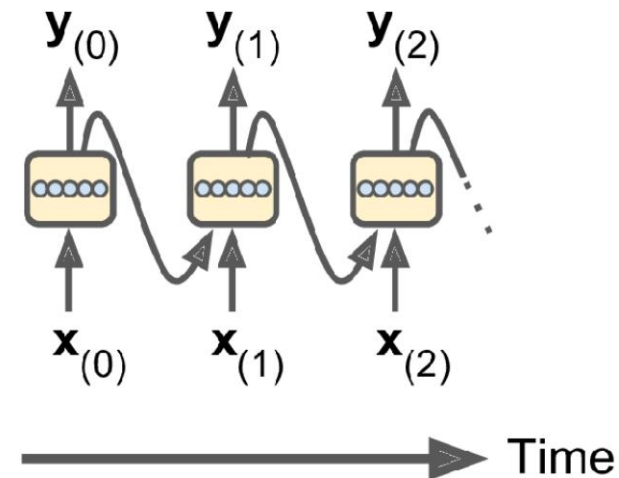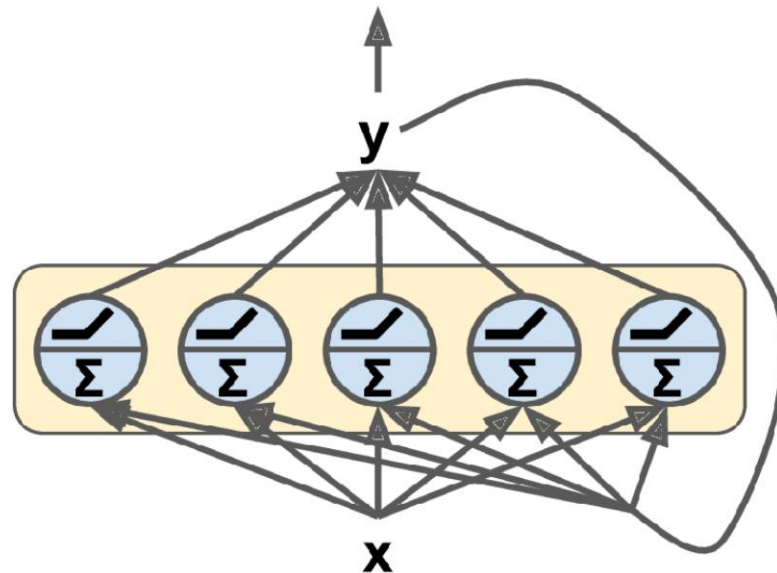
Recurrent Neural Networks (RNN)

▶ RNNs build models that take into account sequential nature of the data

  ▶ The feature for each observation is a sequence of vectors $X = \{X_1, X_2, ..., X_L\}$

  ▶ The target Y is often a single variable such as Sentiment, or a one-hot vector for multiclass. It can also be a sequence, such as the same document in a different language

  ▶ The same weights W, U and B are used at each step - hence the term recurrent

  ▶ We can represent this tiny network against the time axis
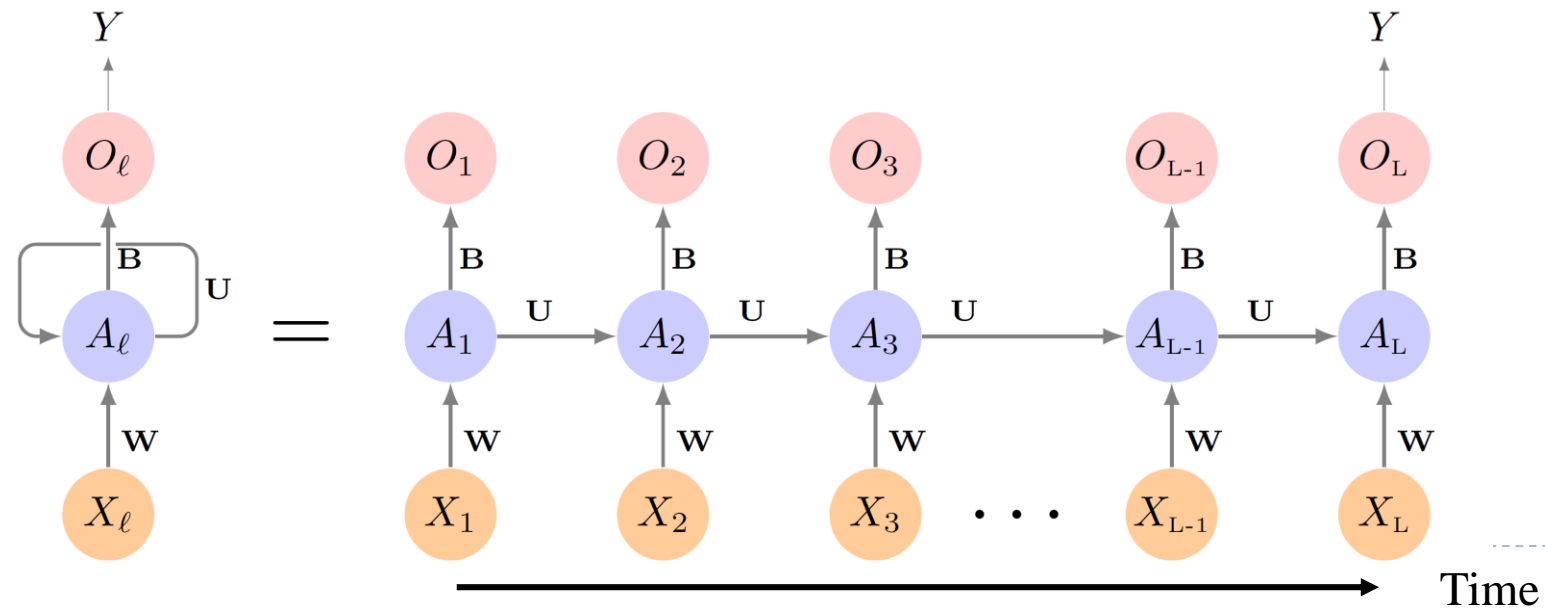
Recurrent Neural Networks (RNN)

▸ The hidden layer is a sequence of vectors (hidden states) A_l, receiving as input X_l as well as A_{l-1}. A_l produces an output O_l

 ▸ The A_l sequence represents an evolving model for the response that is updated as each element X_l is processed and is thus known as *memory cells*

  ▸ Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, you could say it has a form of memory

 ▸ You can create a layer of recurrent neurons. Both the inputs and outputs are vectors now

Recurrent Neural Networks (RNN)

▸ Suppose $X_l = (X_{l1}, X_{l2}, \dots, X_{lp})$ has p components, and $A_l = (A_{l1}, A_{l2}, \dots, A_{lK})$ has K components. Then the computation at the kth components of hidden unit A_l is

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

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

▸ Often we are concerned only with the prediction O_L at the last unit. For squared error loss, and $\underline{n \text{ sequence/response pairs}}$, we would minimize. See appendix for training with BPTT

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

Types of RNN

‣ An RNN can be divided into several configuration

    ‣ *Sequence-to-sequence network* is useful for predicting time series such as stock prices

    ‣ A sequence-to-vector network. For example, you could feed the network a sequence of words corresponding to a movie review, and the network would output a sentiment score

    ‣ A vector-to-sequence network. For example, the input could be an image (or the output of a CNN), and the output could be a caption for that image

    ‣ You could have a sequence-to-vector network, called an encoder, followed by a vector-to-sequence network, called a decoder. For example, this could be used for translating a sentence from one language to another
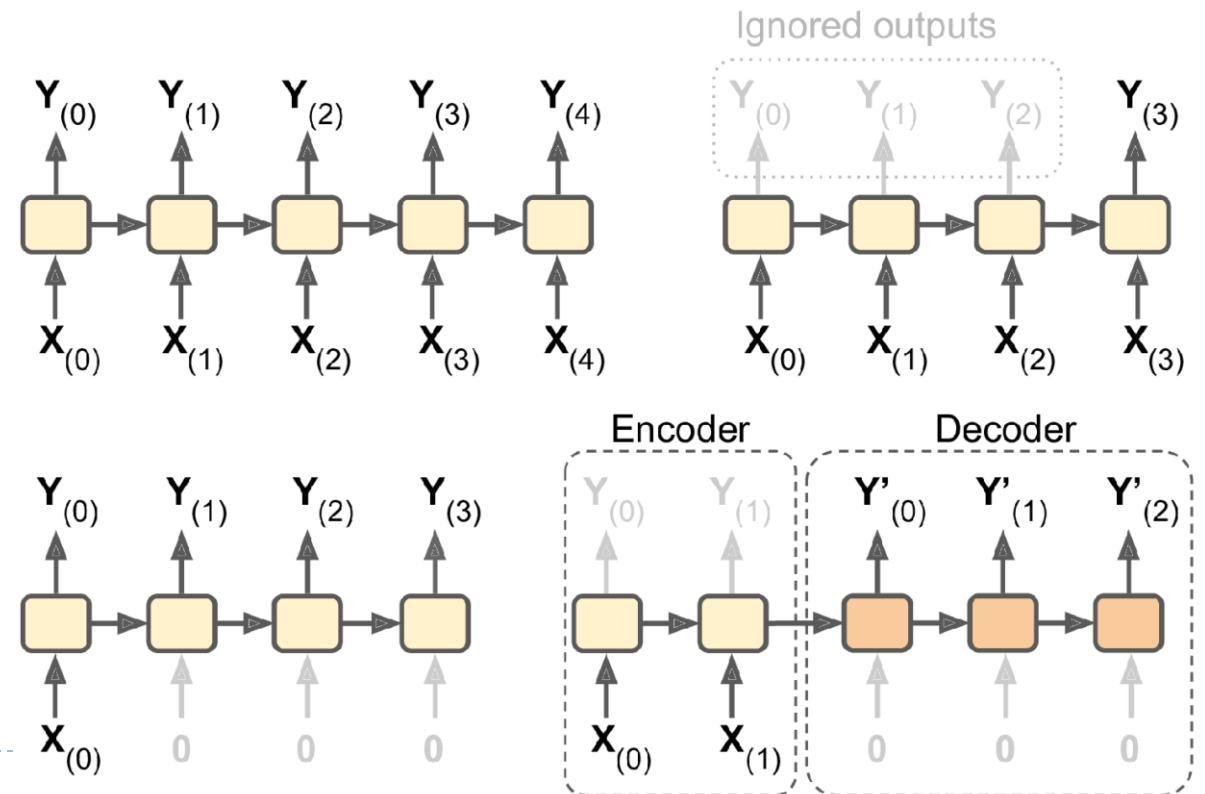
Deep RNN

‣ It is quite common to stack multiple layers of cells. This means it can have additional hidden layers, where each hidden layer is a sequence, and treats the <u>previous hidden layer as an input sequence</u>
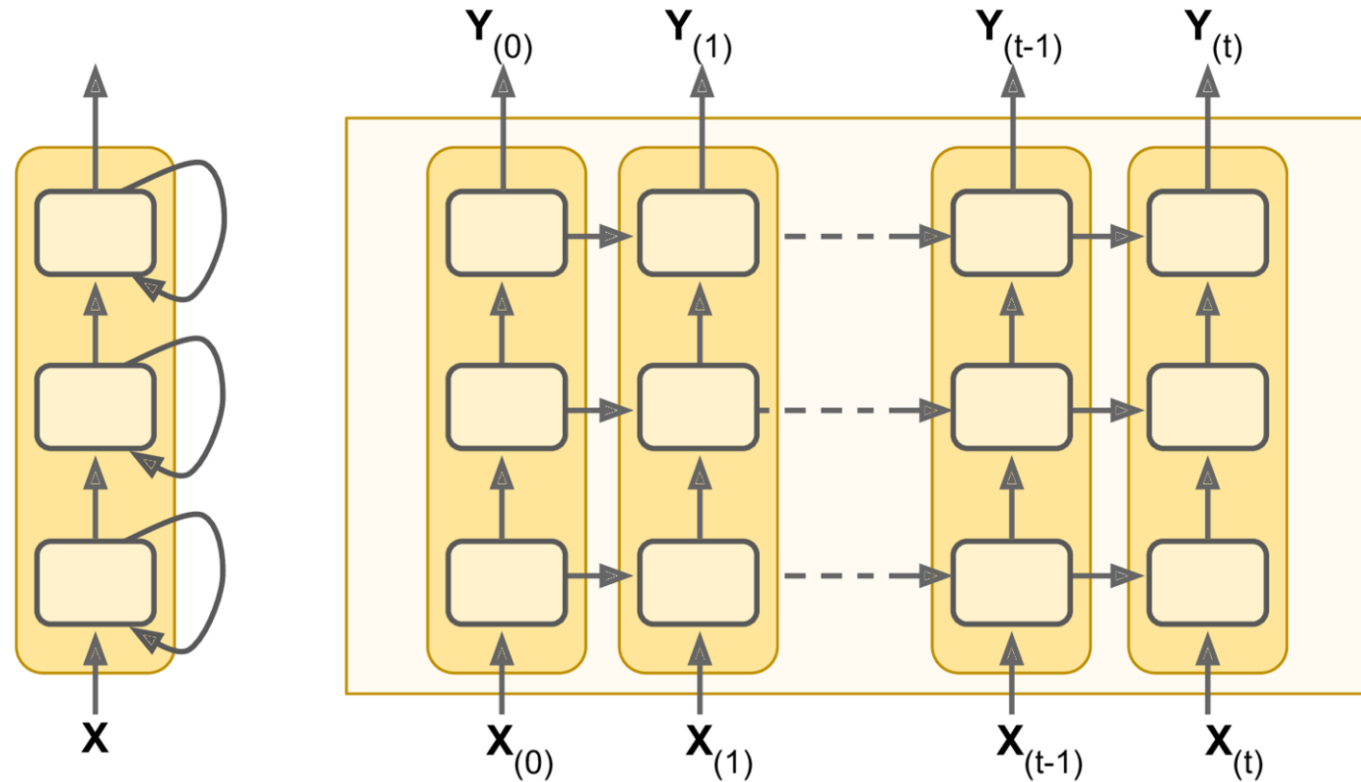
2. Fighting the Unstable Gradients Problem

▶ Nonsaturating activation functions (e.g., ReLU) may not help here; in fact, they may actually lead the RNN to be even more unstable during training

  ▶ If SGD updates the weights in a way that increases the outputs slightly at the first time step. Because the same weights are used at every time step, the outputs at the second time step may also be increased, and so on until the outputs explode. This is the same for gradient, try to monitor using Tensorboard and apply gradient clipping

  ▶ We will usually use hyperbolic tangent instead of ReLU

  ▶ Batch Normalization cannot be used as efficiently with RNNs as with deep feedforward nets. In practice, this does not yield good results

    ▶ Another form of normalization often works better with RNNs: *Layer Normalization*. This idea was introduced in a 2016 paper: it is very similar to Batch Normalization, but instead of normalizing across the batch dimension, it normalizes across the features dimension

Fighting the Unstable Gradients Problem

▶ How to correctly apply dropout in recurrent networks isn't a trivial question

- ▶ In 2016, Yarin Gal determined the proper way to use dropout with a recurrent network: the same dropout mask (the same pattern of dropped units) should be applied at every timestep, instead of using a dropout mask that varies randomly from timestep to timestep
- ▶ What's more, in order to regularize the representations formed by the recurrent gates of layers such as GRU and LSTM, a temporally constant dropout mask should be applied to the inner recurrent activations (hidden states) of the layer (a recurrent dropout mask). Using the same dropout mask at every timestep allows the network to properly propagate its learning error through time
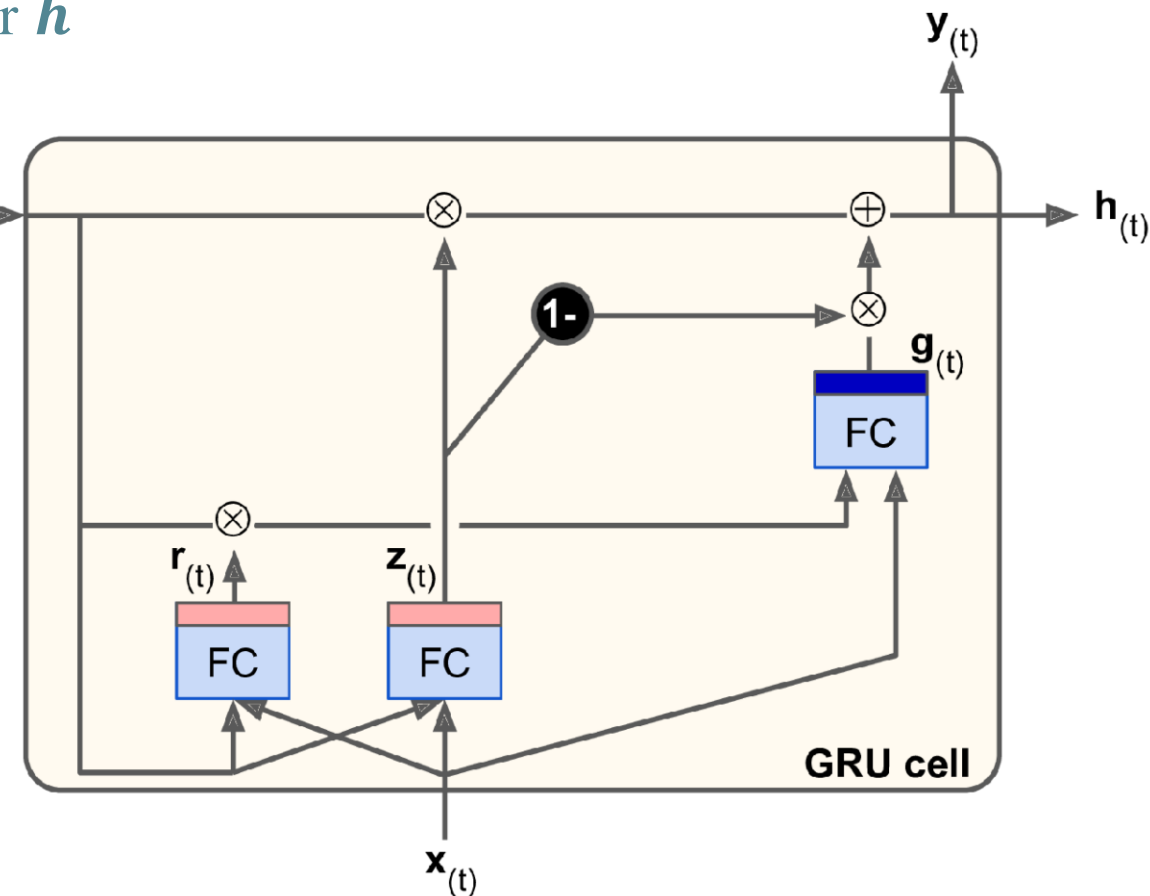
3. Tackling the Short-Term Memory Problem

▸ Due to the data goes through when traversing an RNN. After a while, the RNN's state contains virtually no trace of the first inputs

  ▸ Various types of cells with long-term memory have been introduced

▸ The Long Short-Term Memory (LSTM) cell

  ▸ You can think of h as the short-term and c is new dataflow as the long-term state

  ▸ LSTM cell can learn to recognize an important input (that's the role of the input gate), store it in the long-term state, preserve it for as long as it is needed (that's the role of the forget gate), and extract it whenever it is needed (output gate)
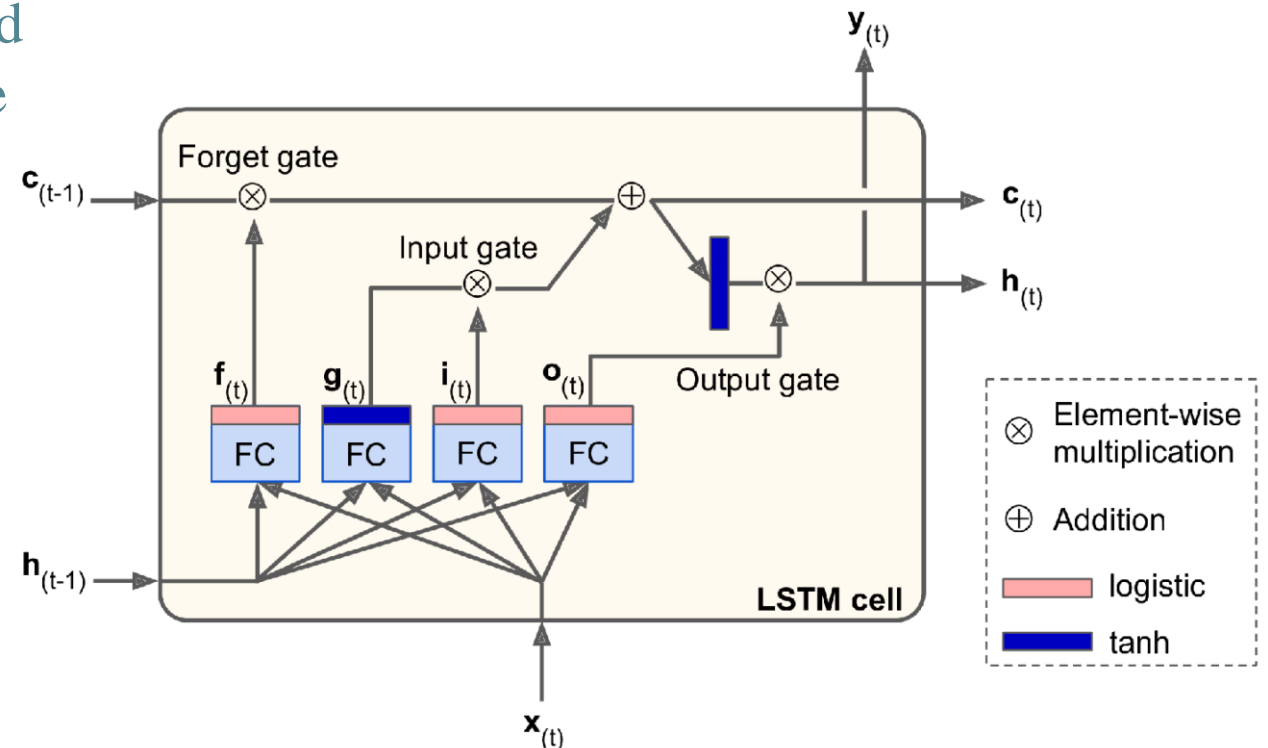
Tackling the Short-Term Memory Problem

▸ The Gated Recurrent Unit (GRU) cell was proposed by Kyunghyun Cho et al. in a 2014 paper that also introduced the Encoder–Decoder network

▸ Both state vectors are merged into a single vector h

▸ A single gate controller z controls both the forget gate and the input gate. If the gate controller outputs a 1, the forget gate is open and the input gate is closed ($1 - 1 = 0$). If it outputs a 0, the opposite happens

▸ There is no output gate; the full state vector is output at every time step. However, there is a new gate controller r that controls which part of the previous state will be shown to the main layer (g)

Tackling the Short-Term Memory Problem

▸ A 1D convolutional layer slides several kernels across a sequence, producing a 1D feature map per kernel. Each kernel will learn to detect a single very short sequential pattern (no longer than the kernel size)

- ▸ If you use 10 kernels, then the layer's output will be composed of 10 1-dimensional sequences (all of the same length)

- ▸ Equivalently you can view this output as a single 10-dimensional sequence. This means that you can build a neural network composed of a mix of recurrent layers and 1D convolutional layers (or even 1D pooling layers)

- ▸ Model can learn to preserve the useful information, dropping only the unimportant details. By shortening the sequences, the convolutional layer may help the GRU layers detect longer patterns
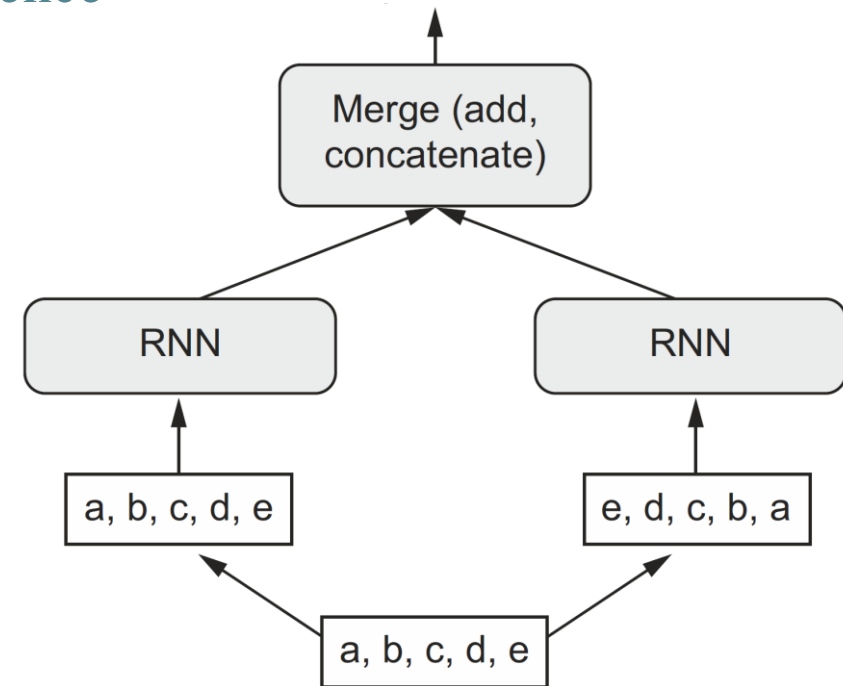
Sequence-to-sequence learning

▸ A sequence-to-sequence model takes a sequence as input and translates it into a different sequence and is compose of

   ▸ An encoder model turns the source sequence into an intermediate representation

   ▸ A decoder is trained to predict the next token i in the target sequence by looking at both previous tokens and the encoded source sequence
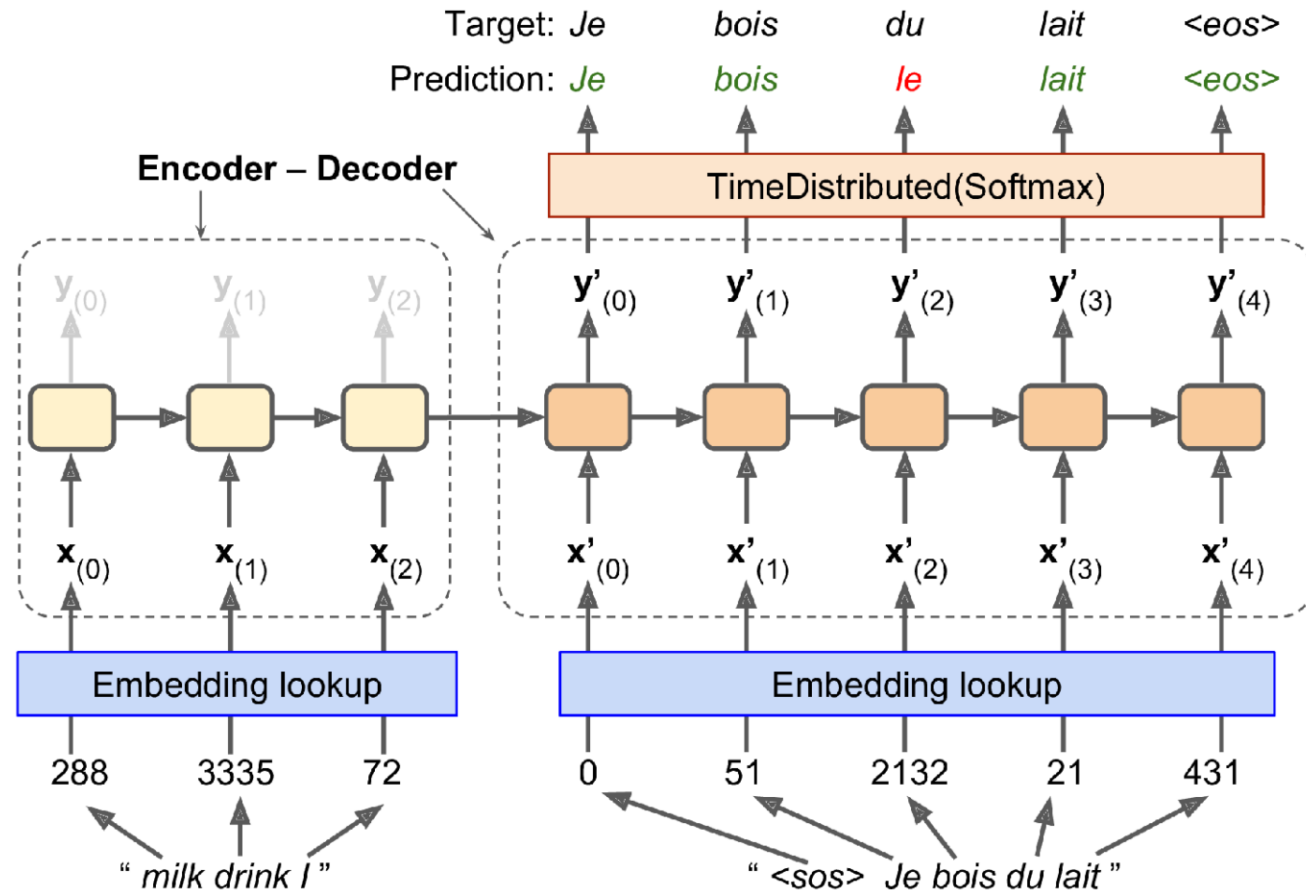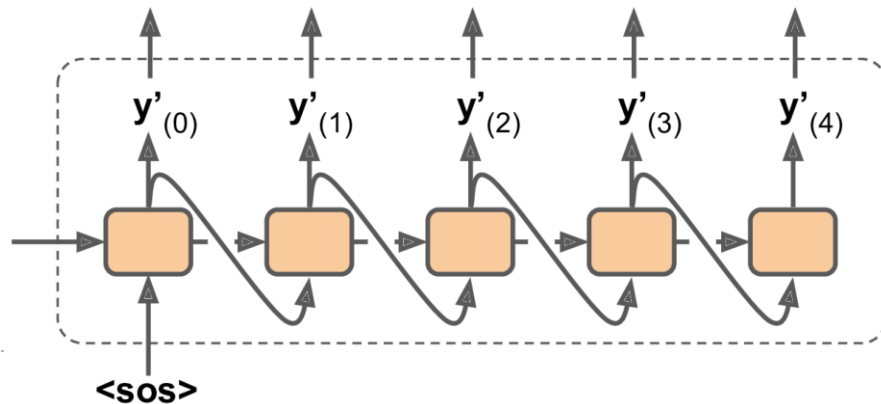
Bidirectional RNN

▸ A bidirectional RNN is a common RNN variant that can offer greater performance than a regular RNN on NLP tasks

    ▸ RNNs are notably order-dependent for time series data: they process the timesteps of their input sequences in order, and shuffling or reversing the timesteps can completely change the representations the RNN extracts from the sequence

    ▸ But in NLP, it is often preferable to look ahead at the next words and the order are not that important as in time series

        ▸ A bidirectional RNN exploits the order sensitivity of RNNs: it uses two regular RNNs each of which processes the input sequence in one direction, and then merges their representations

Merge (add, concatenate)

RNN

RNN

a, b, c, d, e

e, d, c, b, a

a, b, c, d, e

Attention

- Attention has fast become one of the most influential ideas in deep learning
  - Input features get assigned "attention scores," which can be used to inform the next representation of the input
  - Smart embedding space would provide a different vector representation for a word depending on the other words surrounding it. That's where *self-attention* comes in

```
outputs = sum(inputs * pairwise_scores(inputs, inputs))
```

Attention

- ## Crucially, there's nothing that requires A, B, and C to refer to the same input sequence

  - ### This concept is widely used in search engine

    - outputs = sum(values* pairwise_scores (query, keys))

  - ### We may need to use projection to see specifc characteristic

   - outputs = sum(W_v values* pairwise_scores (W_k query, W_q keys))
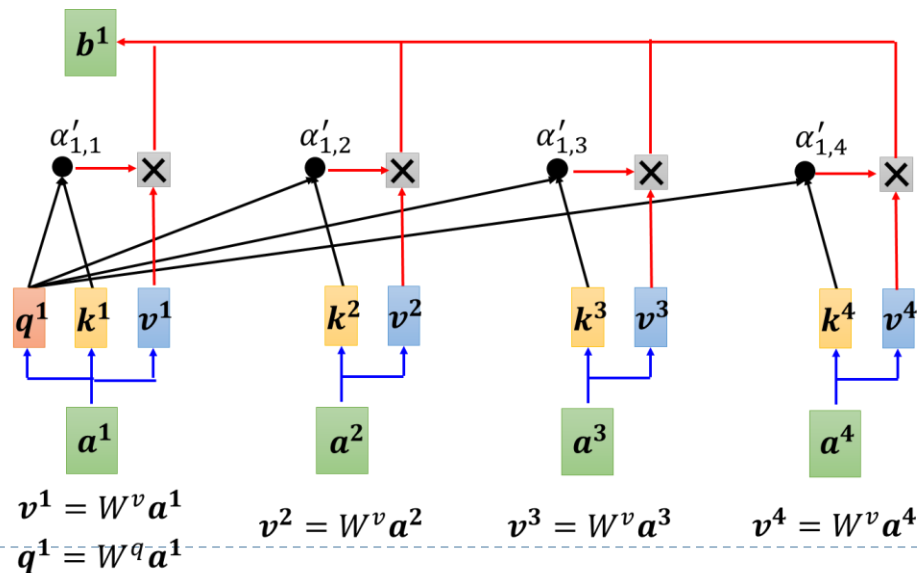
$$v^1 = W^v a^1$$
$$q^1 = W^q a^1$$
$$v^2 = W^v a^2$$
$$v^3 = W^v a^3$$
$$v^4 = W^v a^4$$

Attention

▶ Multi-head attention refers to the fact that the output space of the self-attention layer gets factored into a set of independent subspaces, learned separately

  ▶ The presence of the learnable dense projections enables the layer to learn some projection

  ▶ Independent heads helps the layer learn different groups of features for each token, where features within one group are correlated with each other but are mostly independent from features in a different group

  ▶ Notice outputs are computed in parallel for the sequence in contrast to RNN
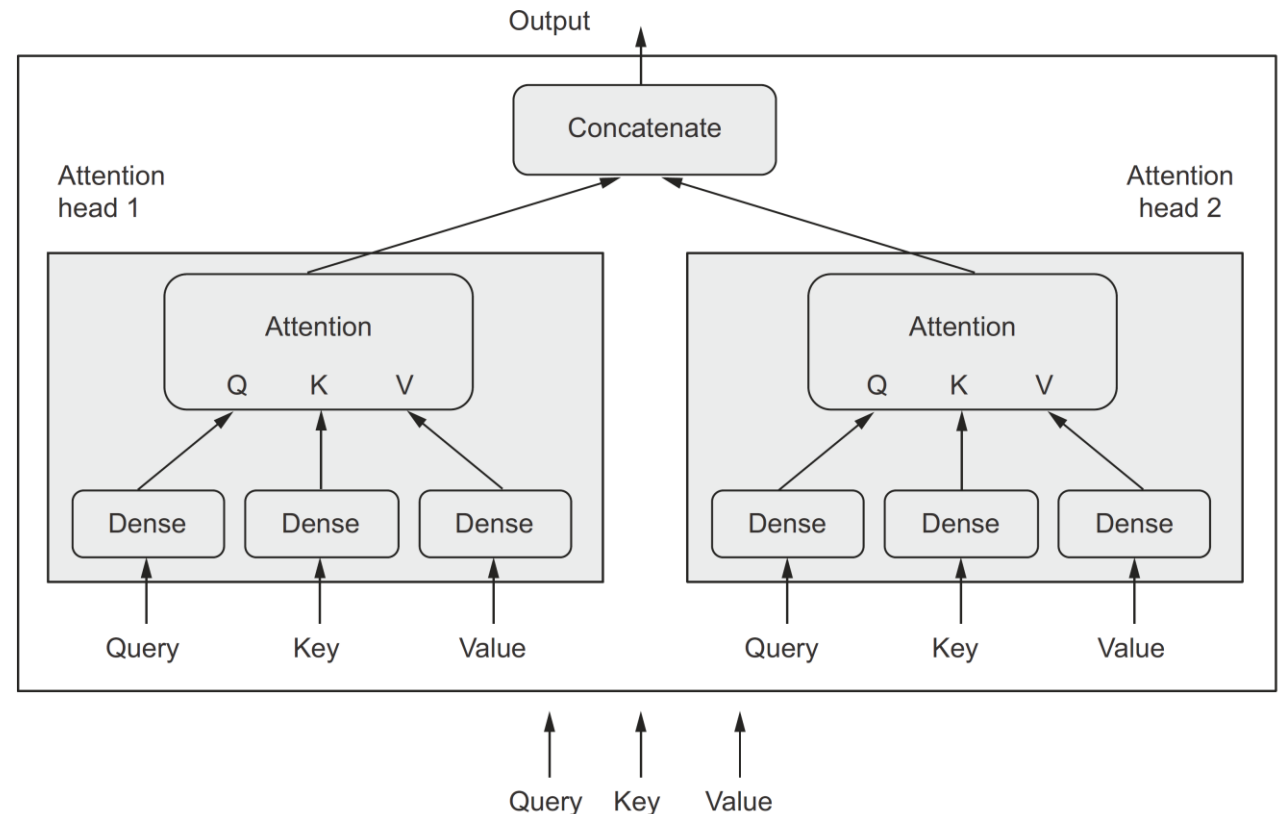
Transformer

▸ **Add additional dense layer, residual connection and layer normalization**

   ▸ Together, these bells and whistles form the Transformer encoder

   ▸ The decoder is very similar to the Transformer encoder, except that an extra attention block is inserted between the self-attention block applied to the target sequence and the dense layers of the exit block

   ▸ Manually injects order information in the representations (Positional encoding that can be learned or manually created)

   ▸ More variants can be found here
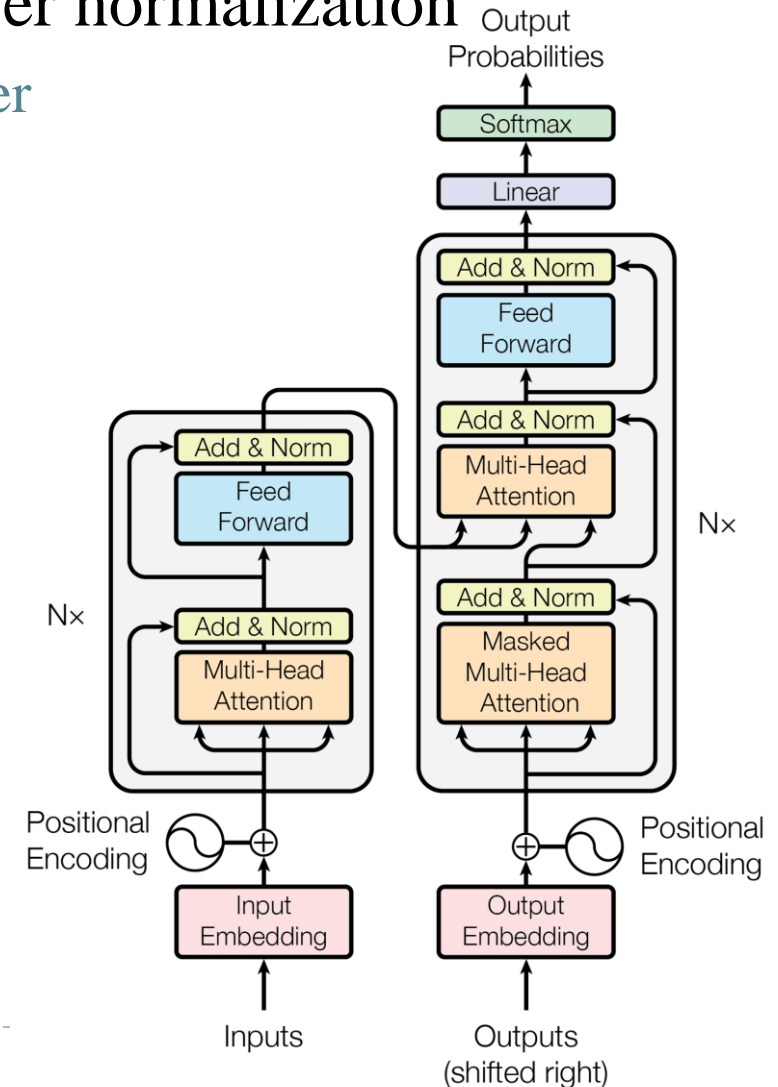
Document Classification: IMDB Movie Reviews

▶ The IMDB corpus consists of user-supplied movie ratings for a large collection of movies. Each has been labeled for sentiment as <u>positive or negative</u>.

▶ 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 classier to predict the sentiment of a review

Lasso versus Neural Network on IMDB Reviews

▸ Bag-of-words with simpler lasso logistic regression model works as well as neural network

▸ Python was used to fit the lasso model, and is very effective because it can exploit sparsity in the X matrix
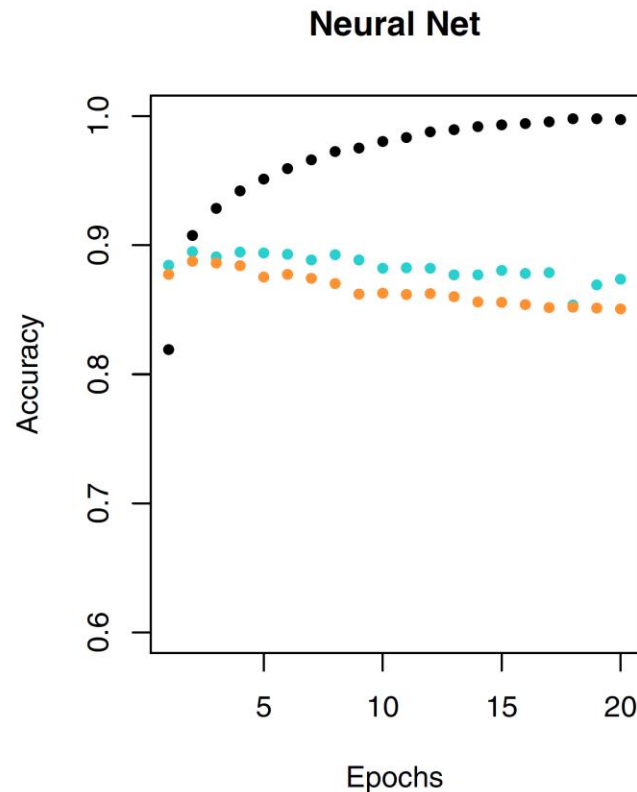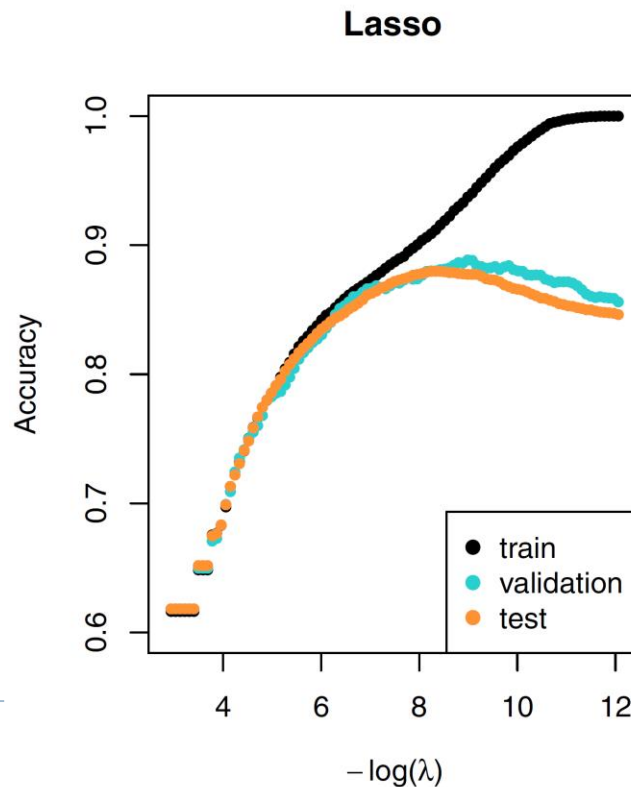
RNN and IMDB Reviews

▸ The document feature is a sequence of words $\{W_l\}_1^L$. We typically truncate/pad the documents to the same number L of words (we use $L = 500$)

  ▸ Each word 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 E $(m \times 10K)$ for mapping

  ▸ 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.)

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_{l-1} (short term memory) as well as from a version that reaches further back in time (long term memory). Now we get 87% accuracy

▸ These data have been used as a benchmark for new RNN architectures. The best reported result found was around 97%

▸ For more leader boards on common benchmark dataset, see https://paperswithcode.com/sota

When to use sequence models over bag-of-words models?

▸ It turns out that when approaching a new text-classification task, you should pay close attention to the ratio between the number of samples in your training data and the mean number of words per sample

  ▸ This is the rule-of-thumb for test classification task
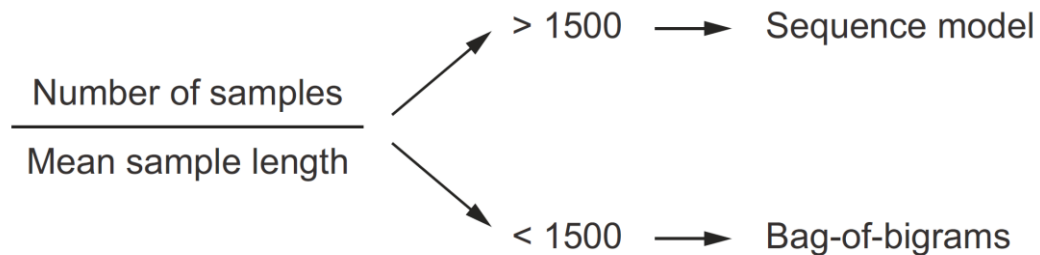
	Word order awareness	Context awareness (cross-words interactions)
Bag-of-unigrams	No	No
Bag-of-bigrams	Very limited	No
RNN	Yes	No
Self-attention	No	Yes
Transformer	Yes	Yes

$$\frac{\text{Number of samples}}{\text{Mean sample length}}$$

> 1500 → Sequence model

< 1500 → Bag-of-bigrams

Time Series Forecasting - New-York Stock Exchange Data

▸ 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 v_t

  ▸ **Dow Jones return**. This is the difference between the log of the Dow Jones Industrial Index on consecutive trading days r_t

  ▸ **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 z_t
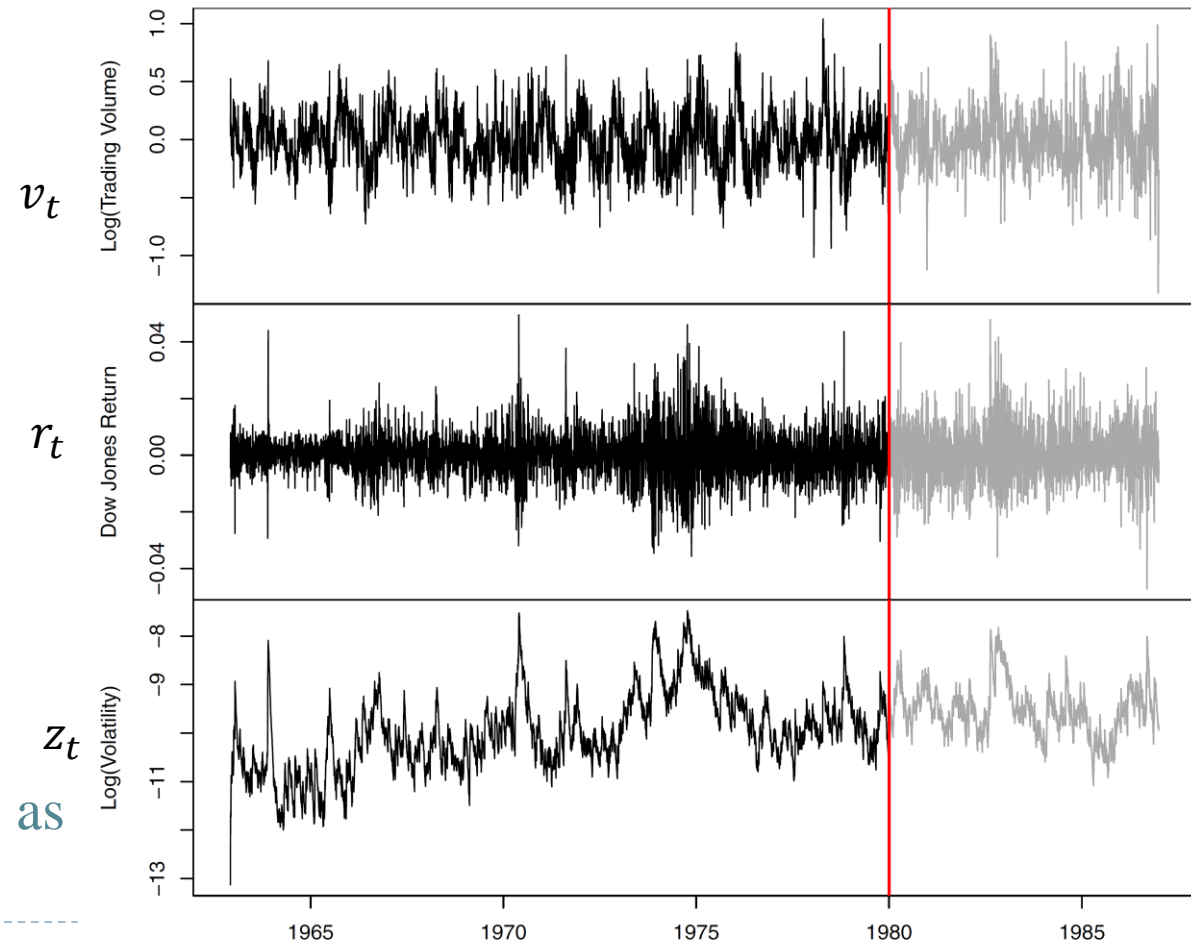
Autocorrelation

▶ The autocorrelation at lag l is the correlation of all pairs (v_t, v_{t-l}) 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-l}!

Log(Trading Volume)
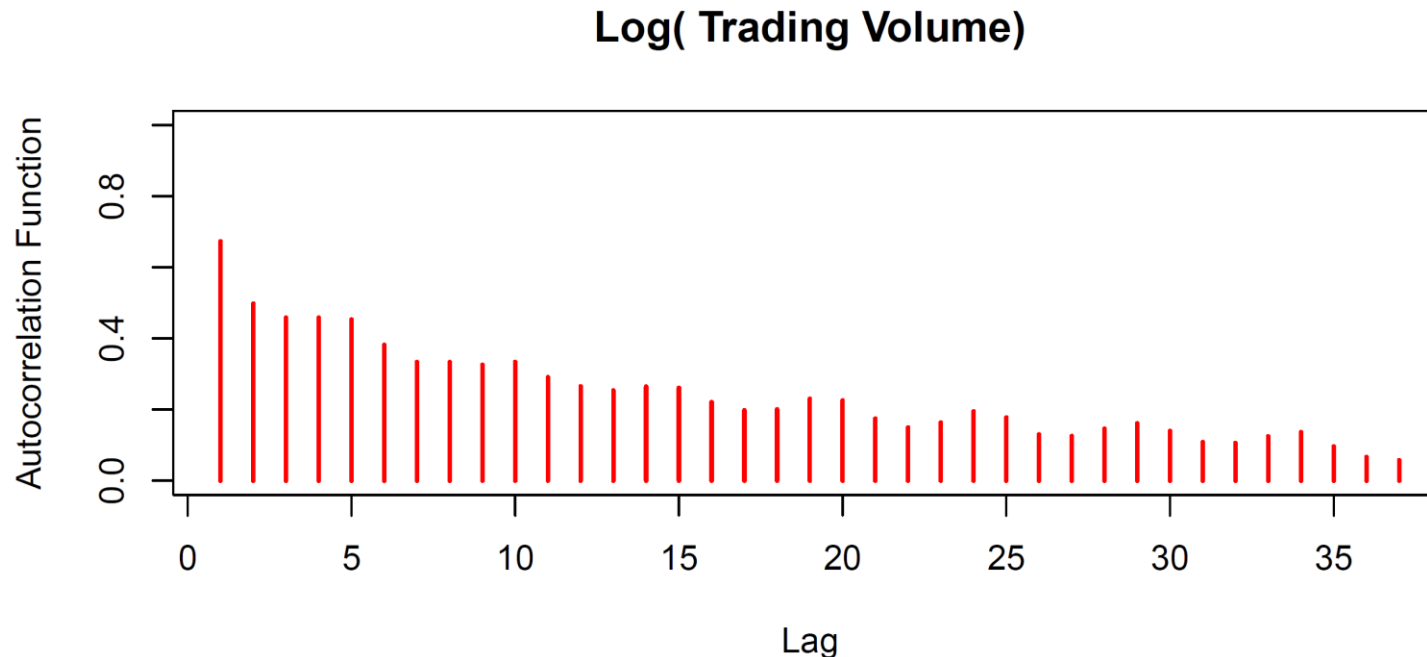
RNN Forecaster

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

▸ We extract many short mini-series of input sequences $X_l = (X_1, X_2, \ldots, X_L)$ with a predefined length L known as the *lag*

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

▸ 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 remaining as test data. We fit an RNN with 12 hidden units per lag step (i.e. per A_l)

RNN Results for NYSE Data

▸ Predictions and truth for test period (black: truth, orange: predicted)

▸ $R^2 = 0.42$ for RNN

▸ $R^2 = 0.18$ for straw man - <u>use yesterday's value of Log trading volume to predict that of today</u>

Test Period: Observed and Predicted
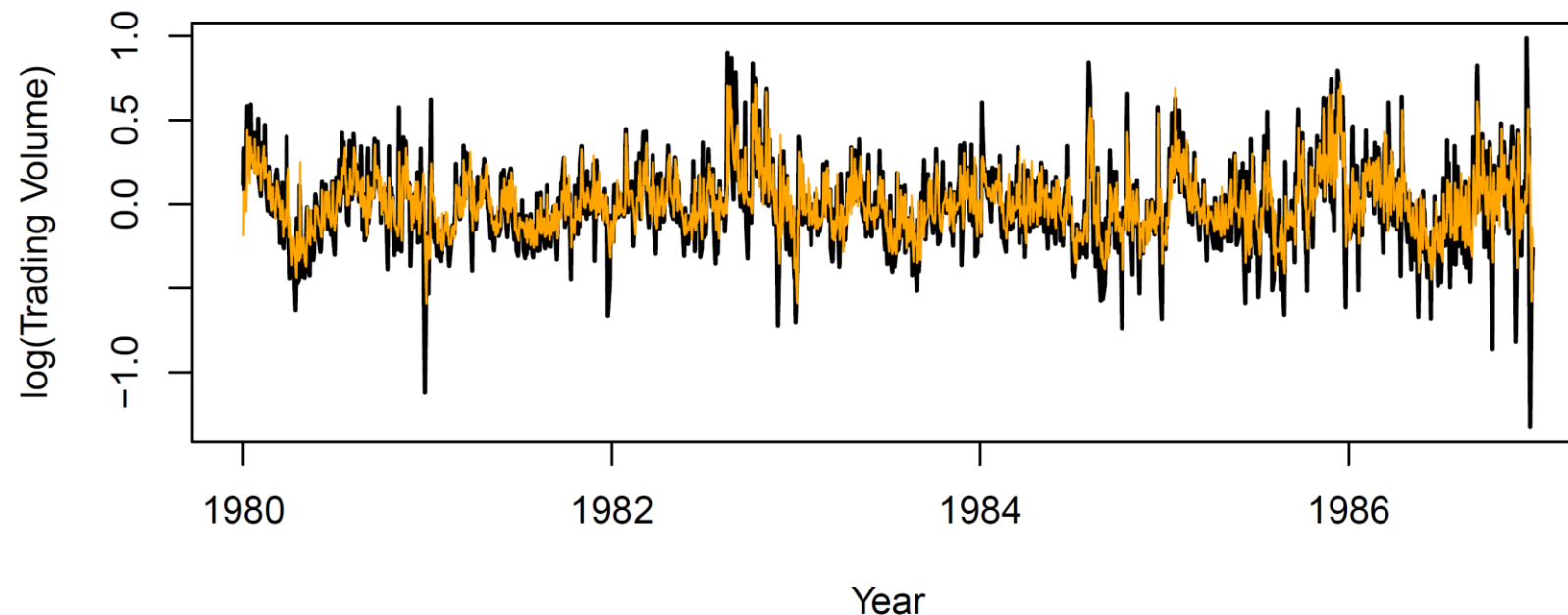
Conclusion

- When you have data where ordering matters, and in particular for time series data, recurrent networks are a great fit and easily outperform models that first flatten the temporal data. The two essential RNN layers are the LSTM layer and the GRU layer

- There are two kinds of NLP models: bag-of-words models that process sets of words or N-grams without taking into account their order, and sequence models that process word order. A bag-of-words model is made of Dense layers, while a sequence model could be an RNN, a 1D convnet, or a Transformer

- Transformer are powerful model that relates to CNN and RNN, but the computational complexity is high

- We have presented some variants of RNNs. Many more complex variations exist. Especially with the development of self-supervised training

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition Chapter 15~16

[2] An Introduction to Statistical Learning with Applications in R. Second Edition Chapter 10

[3] Deep learning with Python, 2nd Edition Chapter 10~11

Appendix

Resources

- NLP libraries
  - https://github.com/explosion/spaCy
  - https://github.com/RaRe-Technologies/gensim
  - https://github.com/sloria/textblob
- Time series libraries
  - https://github.com/blue-yonder/tsfresh
  - https://github.com/alan-turing-institute/sktime
- Visualization
  - https://github.com/HendrikStrobelt/LSTMVis
- Tutorials
  - https://d2l.ai/chapter_recurrent-neural-networks/index.html#
  - https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php

Word Embedding

‣ We will discuss more in Lecture 7
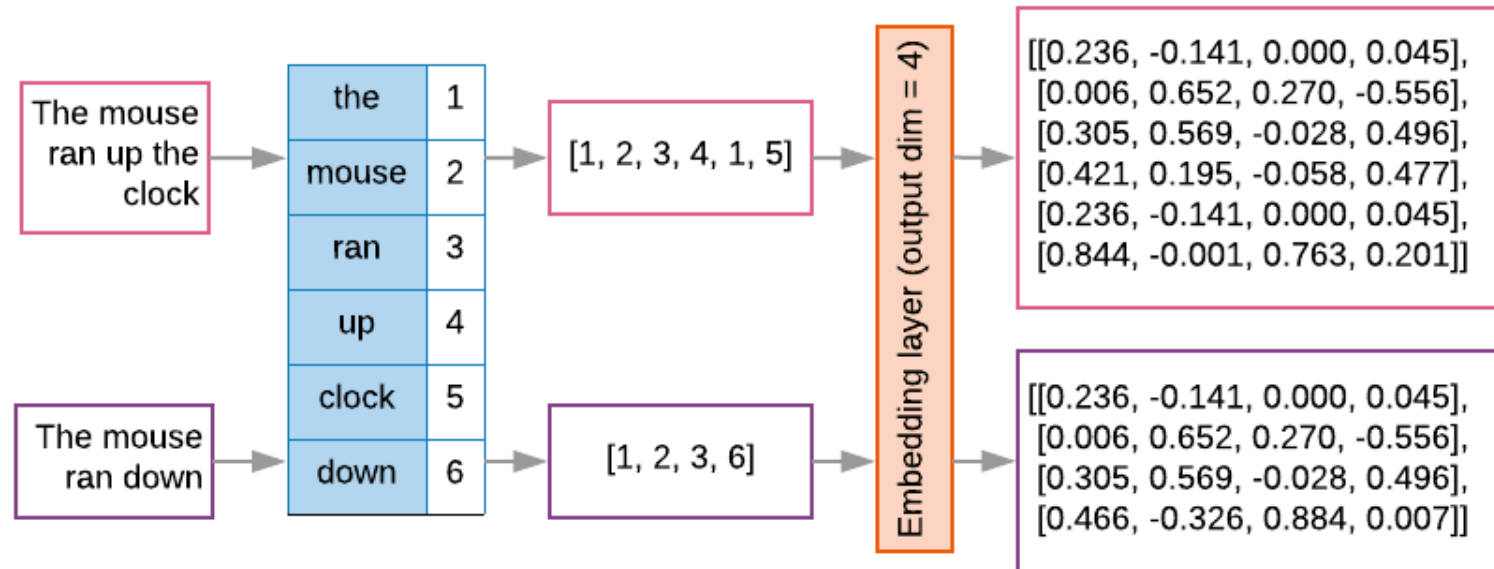
Training RNNs

▸ To train an RNN, the trick is to unroll it through time and then simply use regular backpropagation. This strategy is called backpropagation through time (BPTT)

  ▸ Like in regular backpropagation, there is a first forward pass through the unrolled network. Then the output sequence is evaluated using a cost function $C(Y_{(0)}, Y_{(1)}, \ldots Y_{(T)})$ (where T is the max time step). Note that this cost function may ignore some outputs

  ▸ The gradients of that cost function are then propagated backward through the unrolled network (represented by the solid arrows in the next slides). Finally the model parameters are updated using the gradients computed during BPTT. Note that the gradients flow backward through all the outputs used by the cost function, not just through the final output Moreover, since the same parameters W and b are used at each time step, backpropagation will do the right thing and sum over all time steps
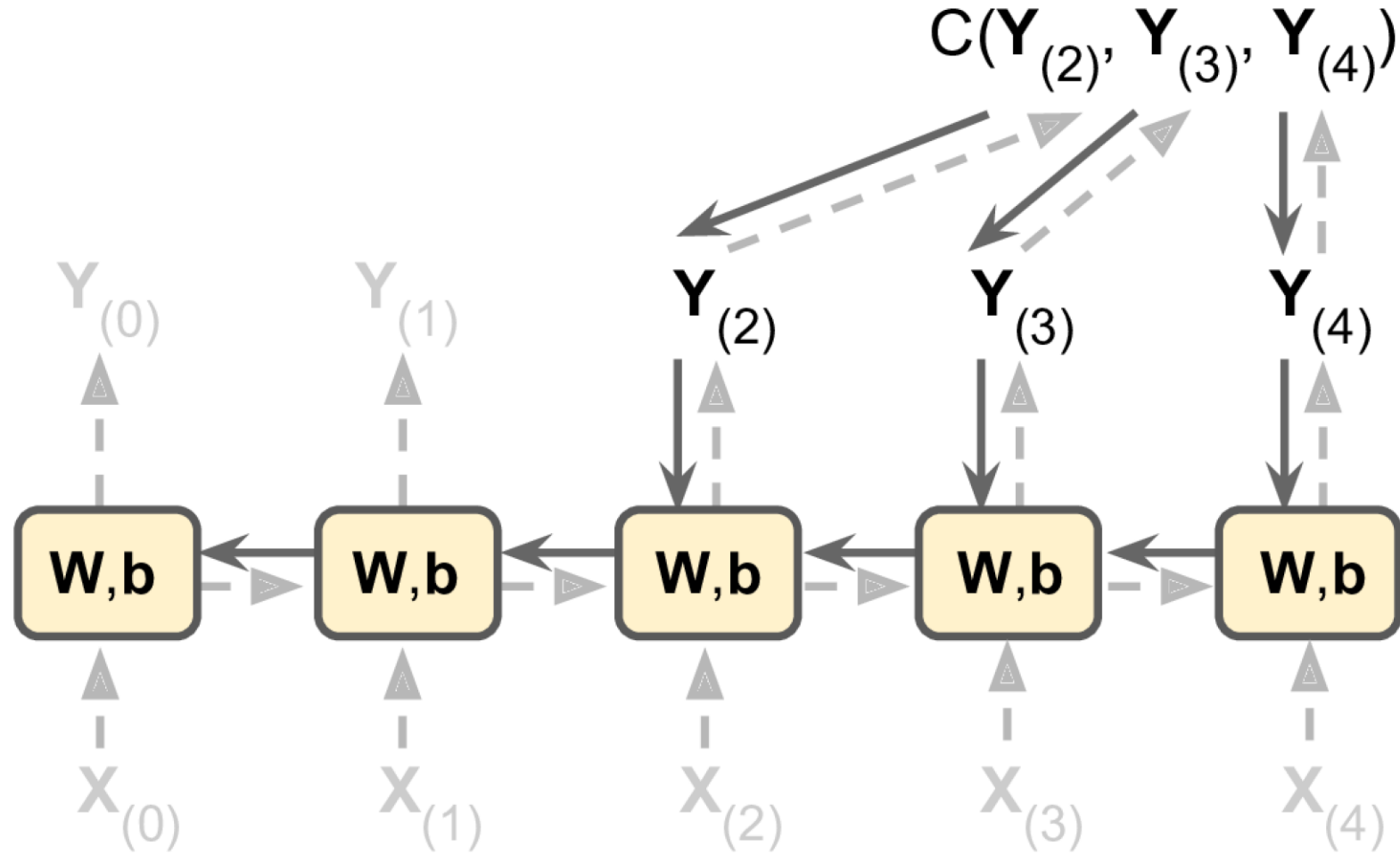
WaveNet

▶ In a 2016 paper, DeepMind introduced an architecture called WaveNet

  ▶ They stacked 1D convolutional layers, doubling the dilation rate (how spread apart each neuron's inputs are) at every layer: the first convolutional layer gets a glimpse of just two time steps at a time, while the next one sees four time steps (its receptive field is four time steps long), the next one sees eight time steps, and so on

  ▶ This way, the lower layers learn short-term patterns, while the higher layers learn long term patterns.

Conv1D dilation 8

Conv1D dilation 4

Conv1D dilation 2

Conv1D dilation 1

Input
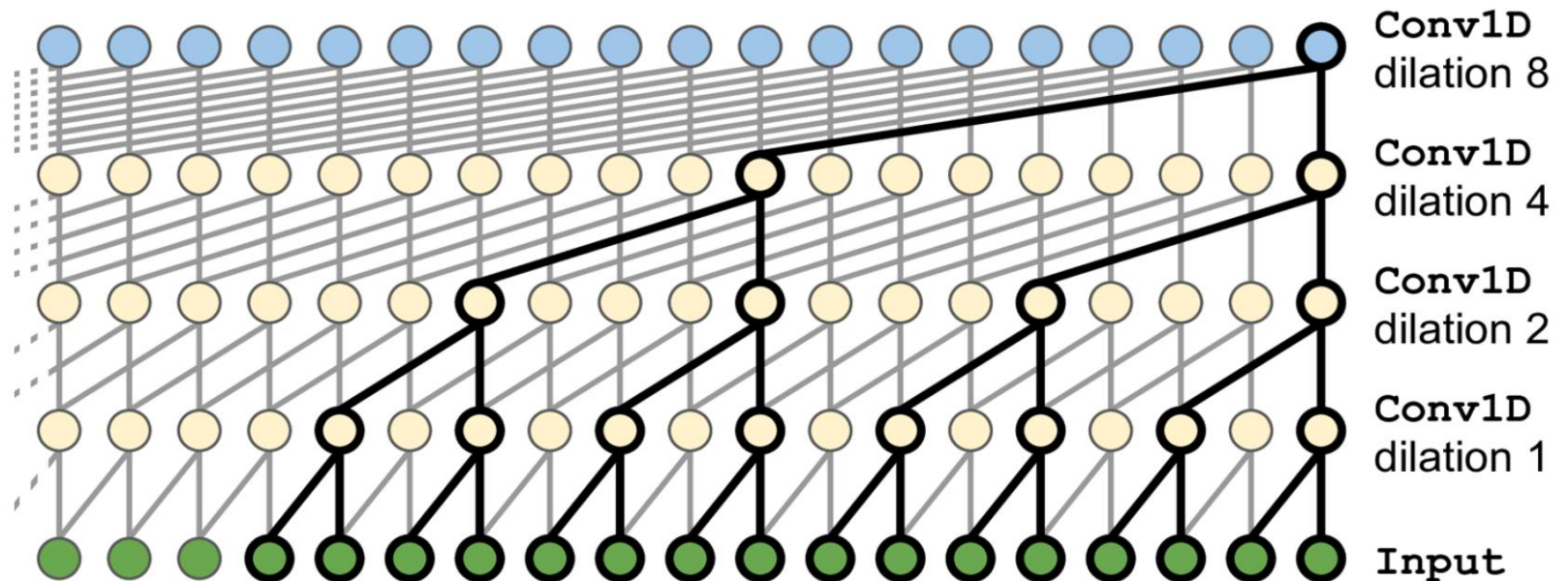
Beam search

▸ Suppose you train an Encoder–Decoder model, and use to translate the French sentence "Comment vas-tu?" to English. You hope that it will output the proper translation ("How are you?"), but unfortunately it outputs "How will you?"

   ▸ Looking at the training set, you notice many sentences such as "Comment vas-tu jouer?" which translates to "How will you play?"

   ▸ Unfortunately, in this case it was a mistake, and the model could not go back and fix it, so it tried to complete the sentence as best it could. By greedily outputting the most likely word at every step, it ended up with a suboptimal translation

   ▸ Beam search keeps track of a short list of the k most promising sentences and at each decoder step it tries to extend them by one word, keeping only the k most likely sentences by replicate the model

Autoregression Forecaster

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

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

▶ Fit an OLS regression of y on 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 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 <u>day of week</u> of day being predicted
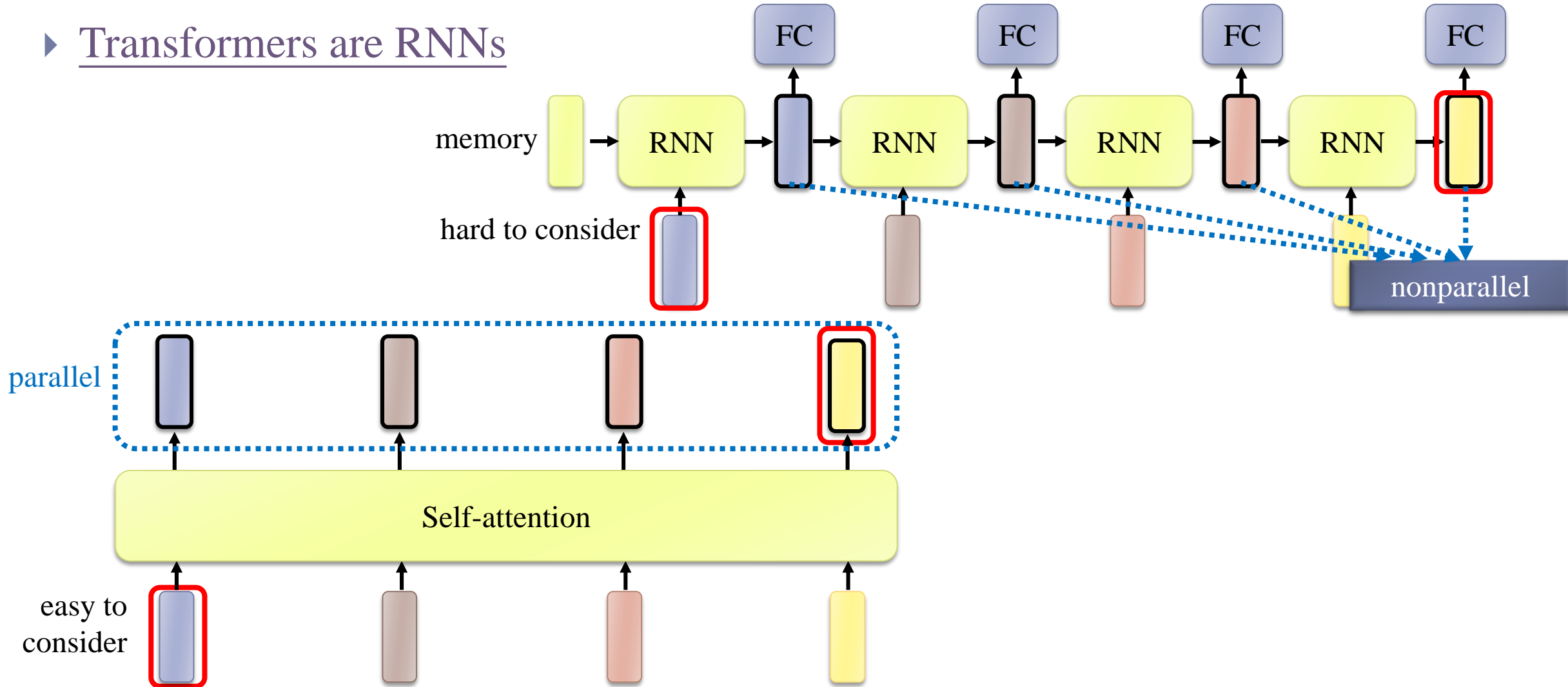
Self-attention vs CNN

▸ CNN: self-attention that can only attends in a receptive field

  ▸ Consider images as a vector set

  ▸ Self-attention is a CNN with learnable receptive field

  ▸ CNN is good for less data while attention is good for more data
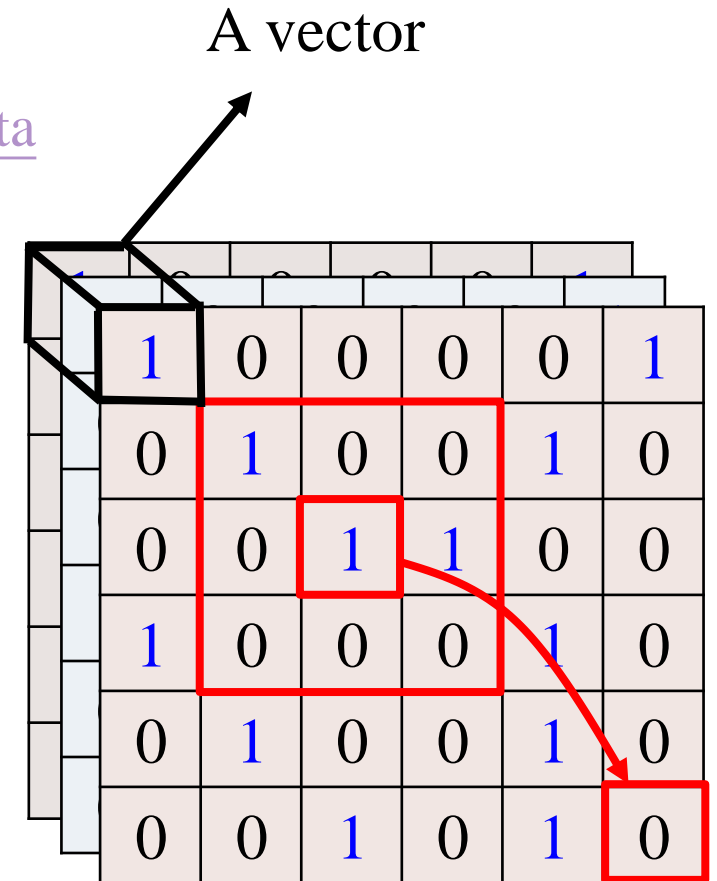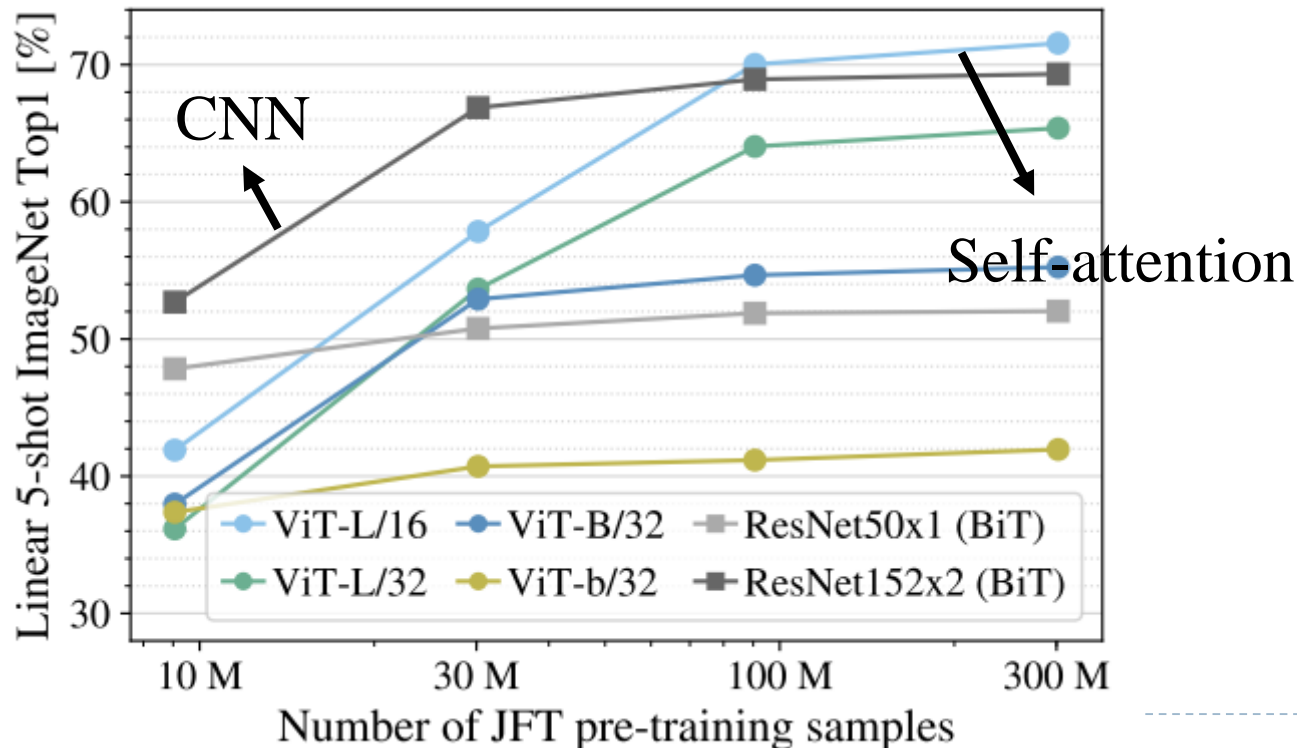
Self-attention vs RNN

▸ Transformers are RNNs

https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/self_v7.pptx

Complexity comparison

- https://d2l.ai/chapter_attention-mechanisms/self-attention-and-positional-encoding.html#subsec-cnn-rnn-self-attention

- https://stackoverflow.com/questions/65703260/computational-complexity-of-self-attention-in-the-transformer-model

- https://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/slides/lec16.pdf