Natural Language Processing

Class 4: Neural Networks & Deep Learning

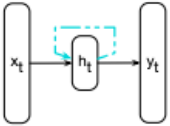
Adam Faulkner

Sept 16, 2025

Recurrent Neural Networks (RNNs)

- A recurrent neural network (RNN) is any network that contains a cycle within its network connections, meaning that the value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input.

Recurrent Neural Networks (RNNs)



- Key difference from a FFN: the recurrent link shown in the figure with the dashed line. This link augments the input to the computation at the hidden layer with the value of the hidden layer from the preceding point in time.
- The hidden layer from the previous time step provides a form of memory that encodes earlier processing and informs the decisions to be made at later points in time.

Recurrent Neural Networks (RNNs)

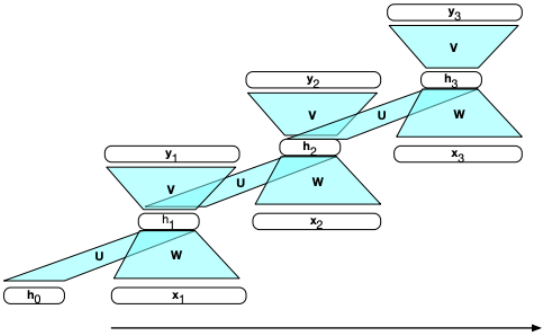
- The fact that the computation at time *t* requires the value of the hidden layer from time *t - 1* mandates an incremental inference algorithm that proceeds from the start of the sequence to the end:

function FORWARDRNN(**x**, *network*) **returns** output sequence **y**

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(**x**) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return *y*

Recurrent Neural Networks (RNNs)

- It's also standard to represent RNNs in their "unrolled"state.
- The various layers of units are copied for each time step to illustrate that they will have differing values over time. However, the various weight matrices are shared across time.



RNNs for Language Modelling

- Earlier, we motivated RNNs by noting their ability to model sequential data such as language
- Learning a probability distribution over text is the task of *language models*.
- This involves predicting the next word given some n previous words or *context*
- If the preceding context is “Thanks for all the“ and we want to know how likely the next word “fish“ is we would compute:

$$P(fish|Thanks\ for\ all\ the)$$

RNNs for Language Modelling

- Language models give us the ability to assign such a conditional probability to every possible next word, giving us a distribution over the entire vocabulary.
- We can also assign probabilities to entire sequences by combining these conditional probabilities using the chain rule of probability:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i | w < i)$$

Training an RNN-based Language Model

- All language modelling (include more sophisticated LMs such as ChatGPT) relies on the concept of *self-training*
- Given a massive *corpus* (collection of text) as training material, the model is asked to predict the next word at each time-step.
- “Self-trained“, or “self-supervised“ because we dont have to add any special gold labels to the data; the natural sequence of words is its own supervision

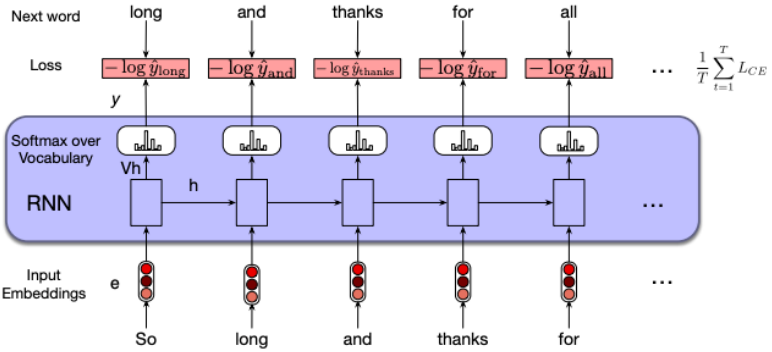
Training an RNN-based Language Model

- During self-training, the task is to minimize the error in predicting the true next word in the training sequence, using cross-entropy as the loss function
- Cross-entropy loss can be used to measure the difference between a predicted probability distribution and the correct distribution

$$L_{CE} = -\sum_{w \in V} y_t[w] \log \hat{y}_t[w]$$

- In the case of language modeling, the correct distribution y_t comes from knowing the next word.

Training an RNN-based Language Model

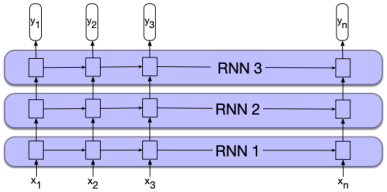


Teacher Forcing

- At each word position t of the input, the model takes as input the the correct word w_t together with $h_t - 1$, encoding information from the preceding $w_1 : t-1$, and uses them to compute a probability distribution over possible next words so as to compute the models loss for the next token $w_t + 1$.
- *But*, when predicting the next word after that, rather than using the model's prediction for the next word, we use the correct word $w_t + 1$ along with the prior history encoded to estimate the probability of token $w_t + 2$
- This is called *teacher forcing*

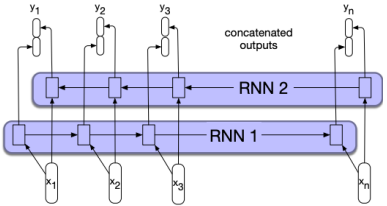
Stacked RNNs

- Stacked RNNs consist of multiple networks where the output of one layer serves as the input to a subsequent layer
- Stacked RNNs tend to outperform single RNNs. This is often explained as the result of the network learning different abstractions regarding language (parts-of-speech, syntax, etc.) at earlier layers which are then utilized by later layers.
- We'll revisit the benefits of stacked layers when we discuss Transformers



Bidirectional RNNs

- In bidirectional RNNs two independent bidirectional RNNs are combined, one where the input is processed from the start to the end, and the other from the end to the start
- The two representations computed by the networks are then concatenated into a single vector that captures both the left and right contexts of an input
- We'll revisit this use of bidirectionality in our discuss of *BERT*, one of the first and most powerful Transformer-based LMs



Long-distance dependencies

- A key feature of English is the so-called *long-distance* dependency. This can take many forms but a basic example is

  The man with the hat that I saw yesterday after lunch went fishing

  where *The man* takes *went fishing* as a verb. In this case *went fishing* is a long-distance dependency of *The man*
- Despite having access to the entire preceding sequence, the information encoded in the hidden states of RNNs tends to be fairly local, more relevant to the most recent parts of the input sequence
- Thus, if the RNN just sees the context *after lunch*, it might predict a comma and a subject pronoun as the next tokens, as in *after lunch, I*—it has lost the information contained in the initial tokens *The man*, etc.

Vanishing Gradients in RNNs

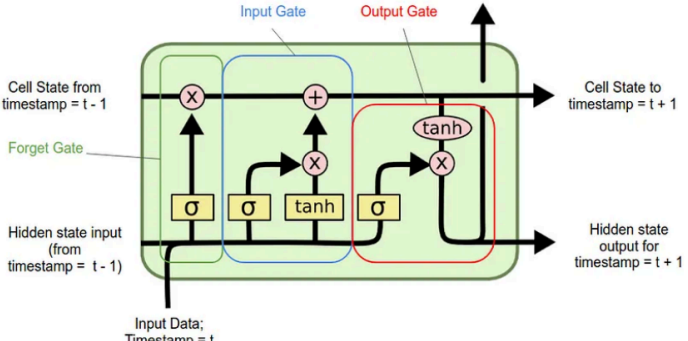
- Also, when RNNs attempt to model lengthy contexts, the hidden layers are subject to repeated multiplications, as determined by the length of the sequence
- A frequent result of this process is that the gradients are eventually driven to zero —the vanishing gradients problem

LSTMs

- These issues prompted the creation of the Long Short-Term Memory (LSTM) network
- LSTMs manage the task of maintaining relevant context over time, by enabling the network to learn to forget information that is no longer needed and to remember information required for decisions still to come
- LSTMs make use of specialized neural units that employ of *gates* to control the flow of information into and out of the units that comprise the network layers

LSTM Cells

- The LSTM architecture contains four neural networks chained together and different memory blocks called *cells*.
- Information in the cell is managed by three gates: The *forget, input* (sometimes called *add*), and *output* gates



The *forget* gate

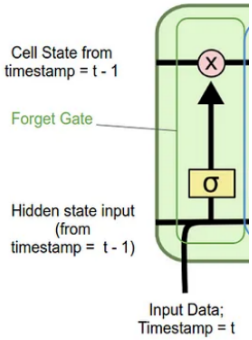
- Two inputs x_t (input at time t) and $h_t - 1$ (previous cell output) are fed to the forget gate and multiplied with weight matrices
- The result is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use
- The forget gate values are calculated using

$$f_t = \sigma(W_f \cdot [h_t, x_t + b_f])$$

where:
W_f represents the weight matrix associated with the forget gate
h_t, x_t denotes the concatenation of the current input and the previous hidden state
b_f is the bias with the forget gate
σ is the sigmoid activation function

The *forget* gate

$$f_t = \sigma(W_f \cdot [h_t, x_t + b_f])$$



The *input* gate

- The addition of useful information in the current context to the cell state is managed by the *input* gate.
- A sigmoid layer first decides which values we'll update.
- Next, a vector is created using the tanh function that gives an output from -1 to +1 —this contains all the possible values from $h_t - 1$ and x_t.
- The values of the vector and the results of the sigmoid are multiplied to obtain the new context vector

The *input* gate

- The first two equations needed for the input gate are

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_t, x_t + b_f]) \\
\hat{C}_t &= tanh(W_c \cdot [h_t, x_t + b_c])
\end{aligned}
\tag{1}
$$

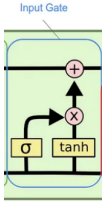
- We start by multiplying the previous state by f_t, disregarding the information we had previously chosen to ignore.
- Next, we include our two equations i_t and $\hat{C}_t$ This represents the updated candidate values, adjusted for the amount that we chose to update each state value

$$C_t = f_t \circ C_{t-1} + i_t \circ \hat{C}_t$$

where $\circ$ denotes element-wise multiplication

The *input* gate

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_t, x_t + b_f]) \\
\hat{C}_t &= tanh(W_c \cdot [h_t, x_t + b_c]) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \hat{C}_t
\end{aligned}
\tag{2}
$$

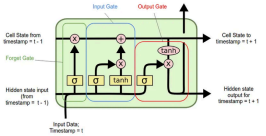


The *output* gate

- The final gate, the *output* gate, is used to decide what information is required for the current hidden state (as opposed to what information needs to be preserved for future decisions).
- The first equation for this gate is essentially the same used in the forget and input gates

$$o_t = \sigma(W_o \cdot [h_t, x_t + b_o])$$

- o_t filters the values to be remembered using inputs h_t and x_t



LSTMs Out-performed all Traditional ML-based Approaches to NLP Tasks

- The use of LSTMs peaked in the 2010s
- Outperformed all traditional ML-based approaches in both Speech and NLP, leading to abandonment of traditional, feature-engineering-based ML

The Transformer

- LLMs are built out of transformers
- Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

The Transformer

A very approximate timeline

- 2003 Neural Language Model
- 2008 Multi-Task Learning
- 2013 Static Word Embeddings
- 2015 Attention
- 2017 Transformer
- 2018 Contextual Word Embeddings and Pretraining

Contextual Embeddings

Problem with static embeddings (word2vec) They are static! The embedding for a word doesn't reflect how its meaning changes in context.

The chicken didn't cross the road because **it** was too tired

What is the meaning represented in the static embedding for "it"?

Contextual Embeddings

- Intuition: a representation of meaning of a word should be different in different contexts!
- Contextual Embedding: each word has a different vector that expresses different meanings depending on the surrounding words
- How to compute contextual embeddings?
- Attention

Contextual Embeddings

The chicken didn't cross the road because it

What should be the properties of "it"?

The chicken didn't cross the road because it was too **tired**

The chicken didn't cross the road because it was too **wide**

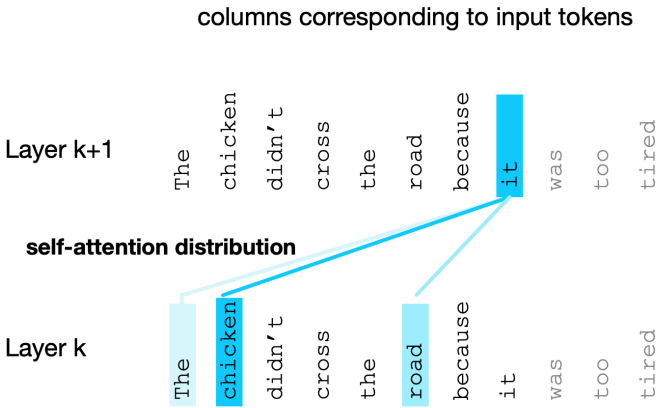
At this point in the sentence, **it**'s probably referring to either the chicken or the street

Attention

- Intuition of attention
- Build up the contextual embedding from a word by selectively integrating information from all the neighboring words
- We say that a word "attends to" some neighboring words more than others

Attention

Intuition of Attention:



Attention

- **Attention definition:** A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens (at the previous layer). More formally: a method for doing a weighted sum of vectors.

Attention

- Simplified version of attention: a sum of prior words weighted by their similarity with the current word
- Given a sequence of token embeddings:

$$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_i$$

- Produce: a_i = a weighted sum of x_1 through x_7 (and x_i)

$$score(x_i, x_j) = x_i \mathring{u} x_j$$

$$\alpha_{ij} = softmax(score(x_i, x_j)) \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij}, x_j$$

Next class: 2/17

The Transformer

- Attention
- Self-attention
- Multi-Head Attention
- Sparse Attention
- Encoder-Decoder architectures

Reading

- Jurafsky & Martin Chapter 8: Transformers
- Attention is all you need