Intro to Natural Language Processing and Transformers

10/03/2023

Agenda

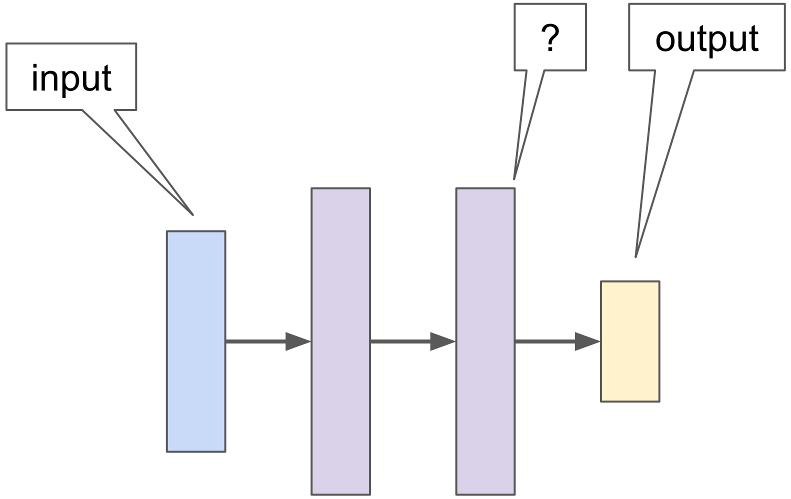
- Announcements:
  - Midterm quiz this Thursday; check the previous lecture slide for more information
  - Small update: the first a few slides today will be covered in quiz
- Today’s topics:
  - Network feature
  - Simple Natural Language Processing concepts and techniques
  - Attention mechanism and Transformer

More about Neural Networks (Again)

- Shallow network vs. deep network
  - Recall that in the homework, for each epoch, you pass all training data before updating the gradient.
  - Imagine doing so on a deep network, what problems may arise?
    - Insufficient memory (dependent on implementation)
    - Extremely long training time
    - Overfitting
  - Therefore, usually we use batch method: put training samples into small batches, and then for each batch, randomly choose one sample for training

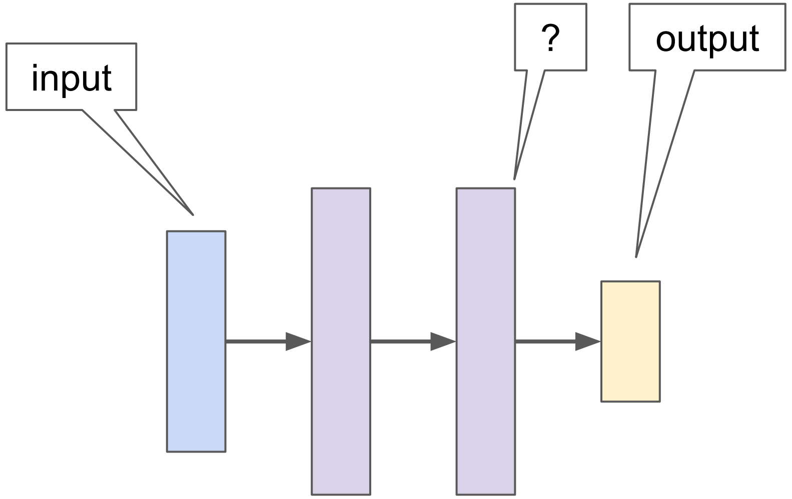
Even More about Neural Networks

- We know that the first layer in a network is the input while the last layer is the output.
- But what about each hidden layer?
  - More specifically, what do their outputs mean?



Even More about Neural Networks

- The input layer encodes raw information.
- The output layer encodes label information.
- Each hidden layer encodes some unreadable information.
- The last hidden layer encodes the aggregated information.
  - Its outputs often called **features**. They are like the network's encryptions of data.
  - Other hidden layers encode information too, but are much less commonly used.
- The output layer classifies based on these **features**, hence it's also called the "classifier layer".



Even More about Neural Networks

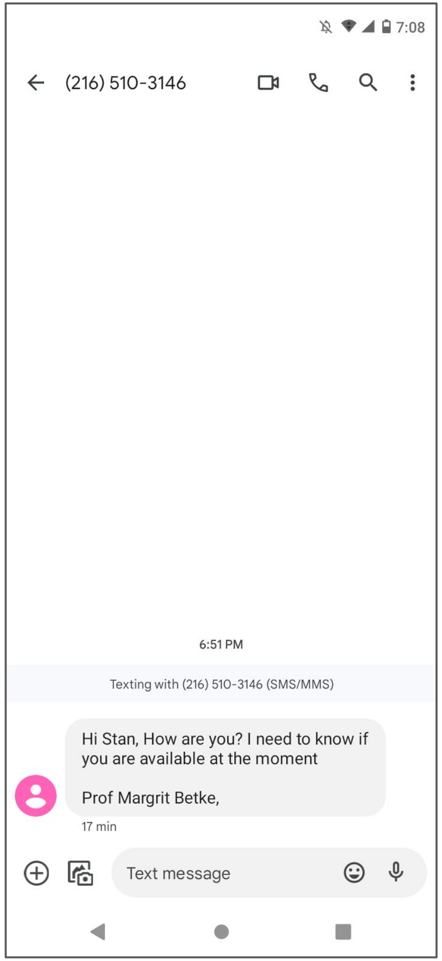
- A network model without the classifier layer outputs features. People often refer to is as “feature extractor” or “featurizer”.
- When using a model that has been pre-trained on some other dataset, we usually
  - remove the classifier layer and replace it with a fresh one (with random weights) that matches our task labels, and then
  - fine-tune (mostly the new classifier layer) on our dataset that is usually much smaller.
- This procedure is called **transfer learning**.
- A “good” model should produce “good” features.

Disclaimer

The rest of today’s contents won’t appear in your midterm, and **probably** won’t appear in your final.

Spam vs Not Spam

- This is a real message I received last year
- Is it a spam?
- Without verifying the number and without knowing Margrit's texting style, how can you tell?



Hi Stan, How are you? I need to know if you are available at the moment

Prof Margrit Betke,

Spam vs Not Spam

- Given a collection of spam messages and a collection of non-spam ones, how do we build a model to classify them?
- The most straightforward method is to find words that commonly appear in the spam messages.
- Then, for each message, we can accumulate the “bad word” frequencies, compute the ratio between the sum of bad frequencies and the sum of all words, and apply a threshold.
- Problems with this approach?
    - How to determine “bad words?”
- The idea makes sense, so let’s make it work, via probabilities.

Probability Review

- P(A): probability of event A happening
- P(A, B): probability of event A and event B happening
- P(A | B): probability of event A happening, given that B happens
- P(A, B) = P(A | B) P(B)
- If A and B are independent events, then P(A, B) = P(A) P(B)
  - So, in this case, P(A | B) = P(A)
- If A and B are **conditionally independent** given event C, then P(A, B | C) = P(A | C) P(B | C)
- P(A) = P(A | B) P(B) + P(A | C) P(C) + …
  - As long the sum of the second terms is one
- P(A | B) P(B) = P(B | A) P(A)

S(pam) vs H(am)

- Trivia: in network security, “ham” = “non-spam”
- We use **S** to denote the event that a given message is spam and **H** if not
- We use $\mathbf{w_i}$ to denote the event that some word “$\mathbf{w_i}$” is found in the message
- Then, given all words in a message $\mathbf{W} = (\mathbf{w_1}, \mathbf{w_2}, \ldots)$, we would like to know the quantity $P(S \mid W)$

S(pam) vs H(am)

- By Bayes rule and marginal probability

$$P(S|W) = \frac{P(W|S)P(S)}{P(W)}$$
$$= \frac{P(W|S)P(S)}{P(W|S)P(S) + P(W|H)P(H)}$$

- If we assume that the words **W** = ($\mathbf{w_1}$, $\mathbf{w_2}$, …) are conditionally independent

$$P(W|S) = \prod_i P(w_i|S)$$

$$P(W|H) = \prod_i P(w_i|H)$$

S(pam) vs H(am)

- Putting them together

$$P(S|W) = \frac{P(S)\prod_i P(w_i|S)}{P(S)\prod_i P(w_i|S) + P(H)\prod_i P(w_i|H)}$$

- Given a (training) dataset, all quantities on the RHS can be obtained by counting
- This is the Naive Bayes method
  - It is naive due to the conditional independence assumption
- Some improvements can be made
  - Add 1s to avoid zeros in the products
  - Take log to avoid floating point errors in long products

Text Representation

- We just saw one idea to represent text in numbers: counting
  - There are other methods that do not depend on task (unlike in NB).
  - For example, one can build a count vector for each piece of input text such that entry represents one word in a large vocabulary.
- The modern way is called **tokenization**, which includes three steps:
  - Break text into words (originally what tokenization is only about)
  - Change words into their base forms in necessary (stemming)
  - Find the ID of each word according to some vocabulary and use the IDs to represent the words
- Tokenization outputs a vector for each piece of input text, where each entry is an ID
  - Since IDs are categorical, it is common to apply one-hot encoding before using them.

Count Vector vs. ID Vector

- Which is better? Count vector or ID vector?
    - What information does a count vector loses while an ID vector preserves?
- If we simply want to classify the type of text (spam vs ham), using counts may be sufficient.
- But for more complex tasks, the order of words is essential to understand the text meanings.
    - In particular, words in different positions of a sentence can be related and is many cases the relation is key for understanding.

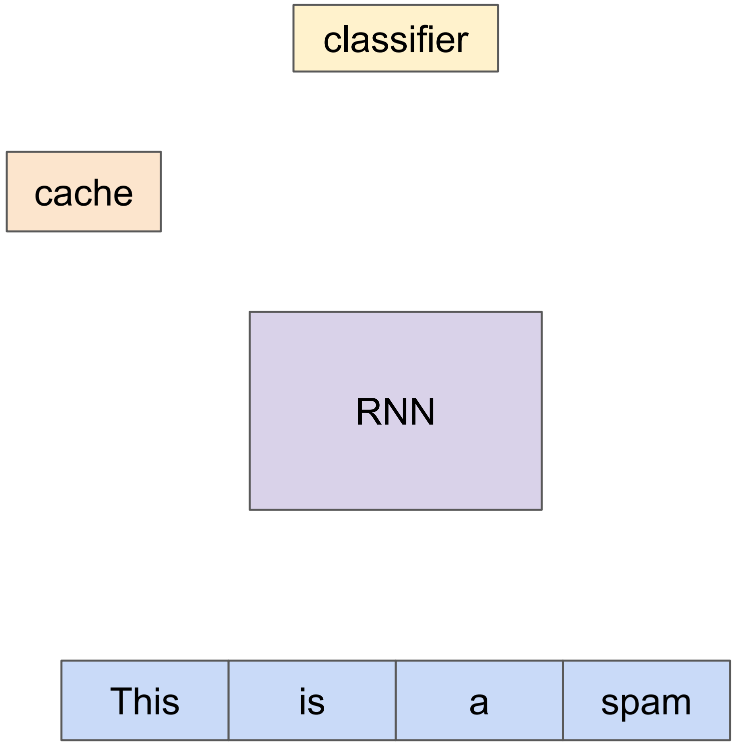
Time Series Input

- The order of the words matters
- The simplest way to reflect the order is to handle the the words one at at time under the order
- This turns a sentence into time series data
- To handle such data, researchers invented a special network called **Recurrent Neural Network (RNN)**

RNN Overview

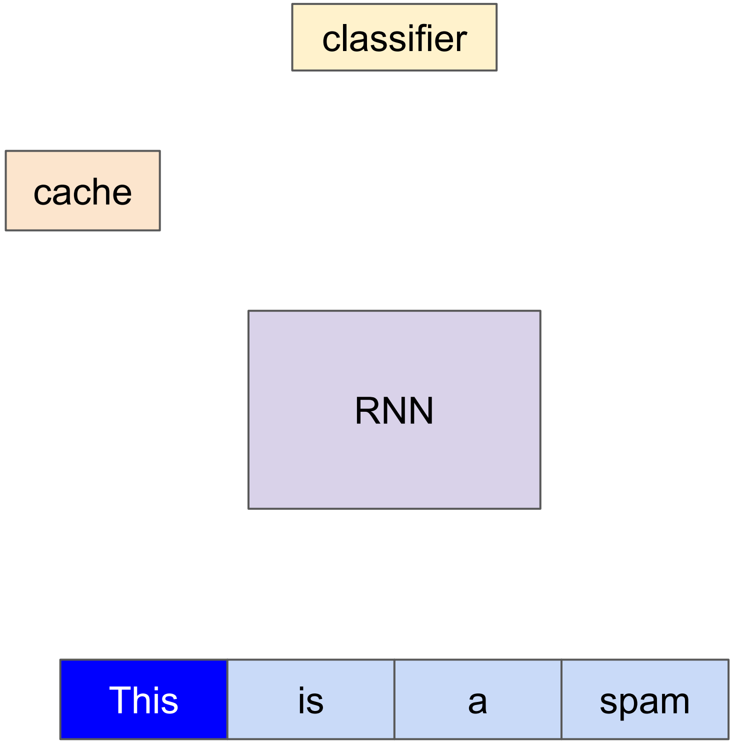
At a high level, RNN takes one input word (or rather, the token vector) at a time.

Classification is performed when all words in a sentence are processed.



RNN Forward Propagation

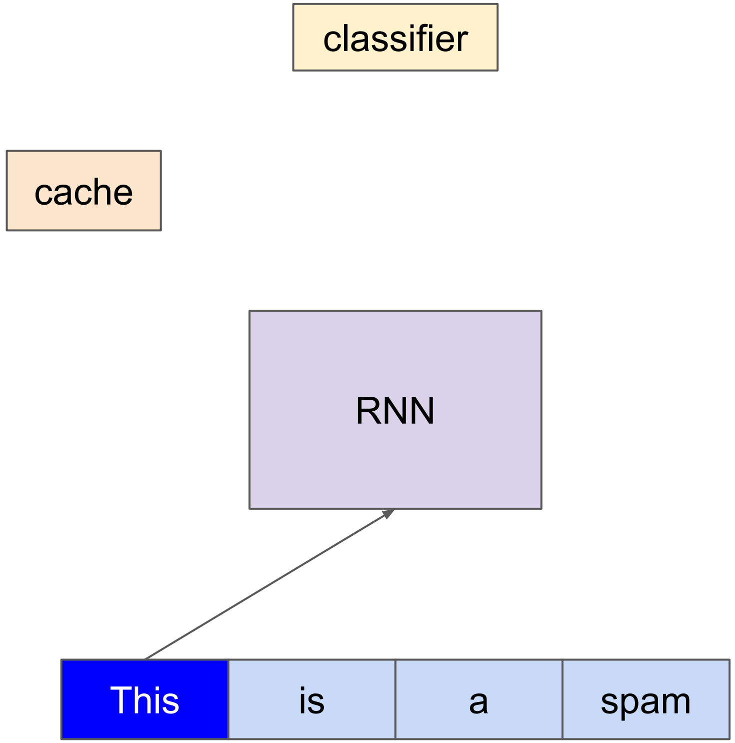
If the current word is the first in a sentence, the model



RNN Forward Propagation

If the current word is the first in a sentence, the model

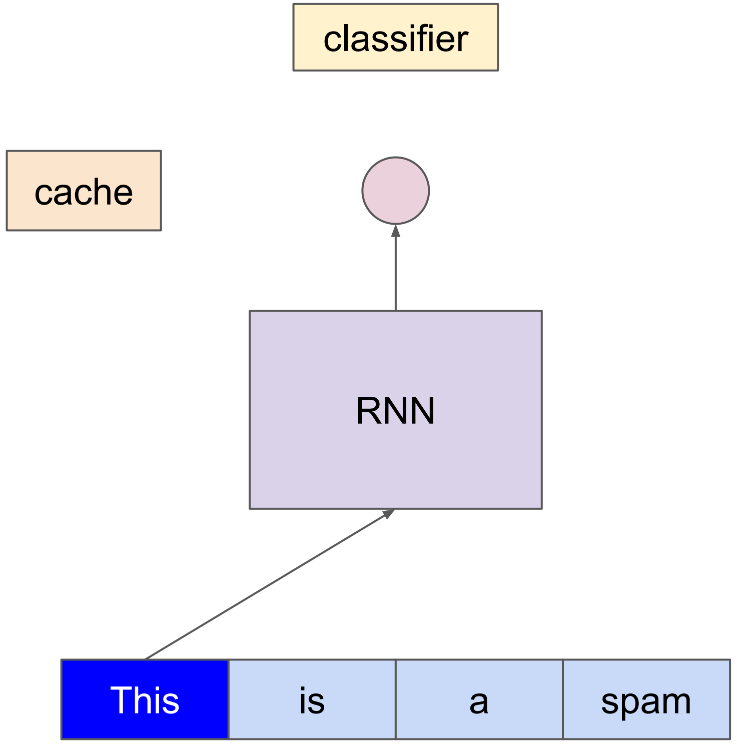
1. takes the token as input,



RNN Forward Propagation

If the current word is the first in a sentence, the model

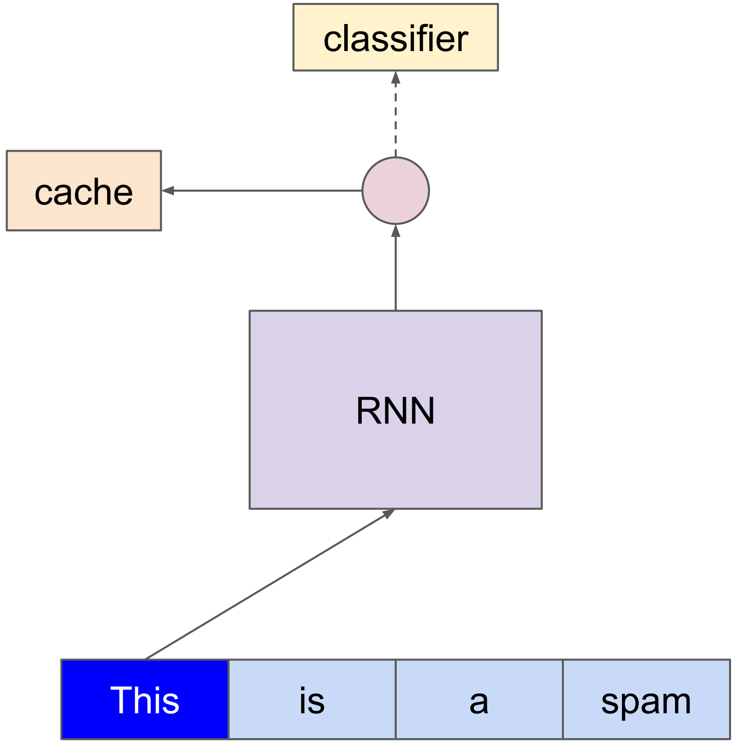
1. takes the token as input,
2. obtains the “feature vector” from the last hidden layer, and



RNN Forward Propagation

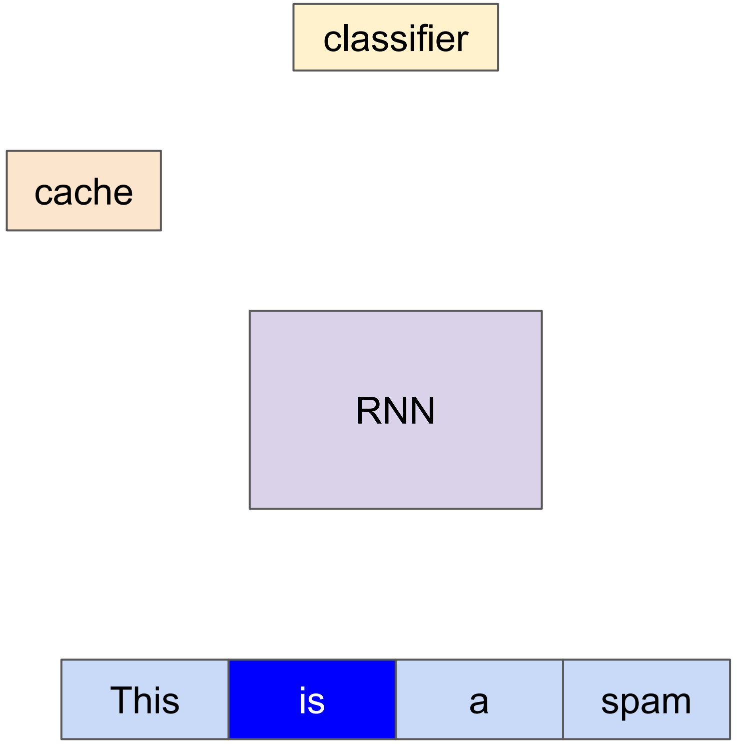
If the current word is the first in a sentence, the model

1. takes the token as input,
2. obtains the “feature vector” from the last hidden layer, and
3. stores it (without necessarily sending it to the classifier layer).



RNN Forward Propagation

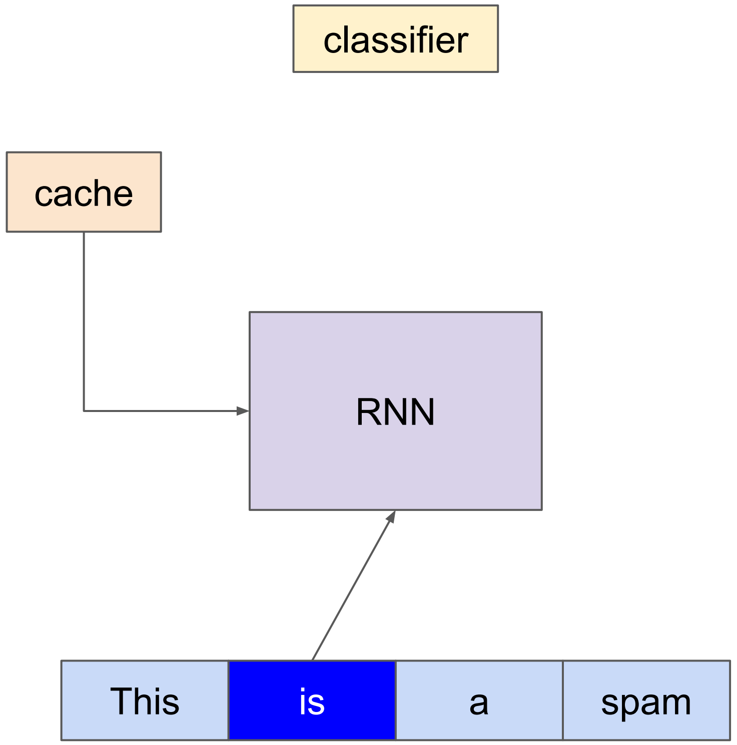
If the current word is between the first and the last, the model



RNN Forward Propagation

If the current word is between the first and the last, the model

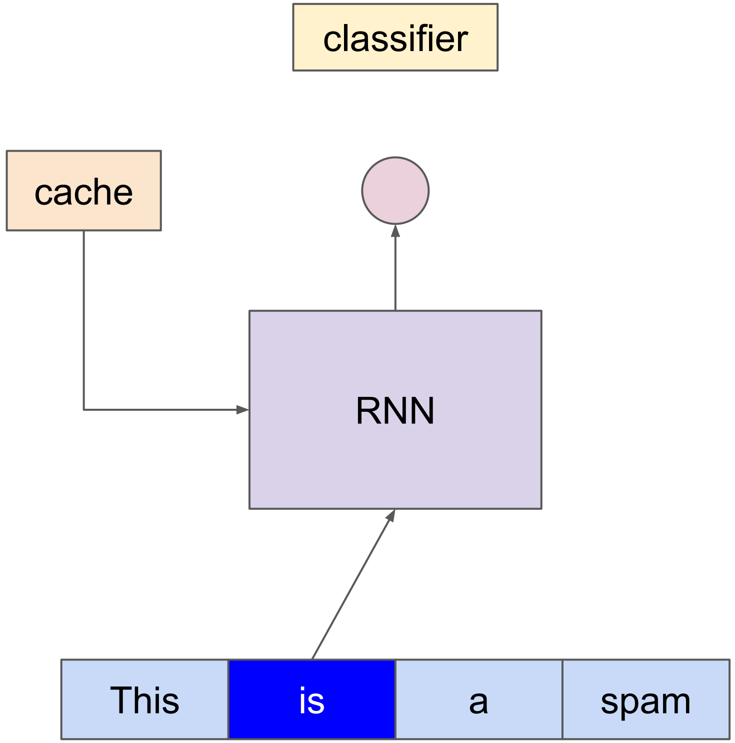
1. takes the token as input and combines it with the most recently stored feature vector,



RNN Forward Propagation

If the current word is between the first and the last, the model

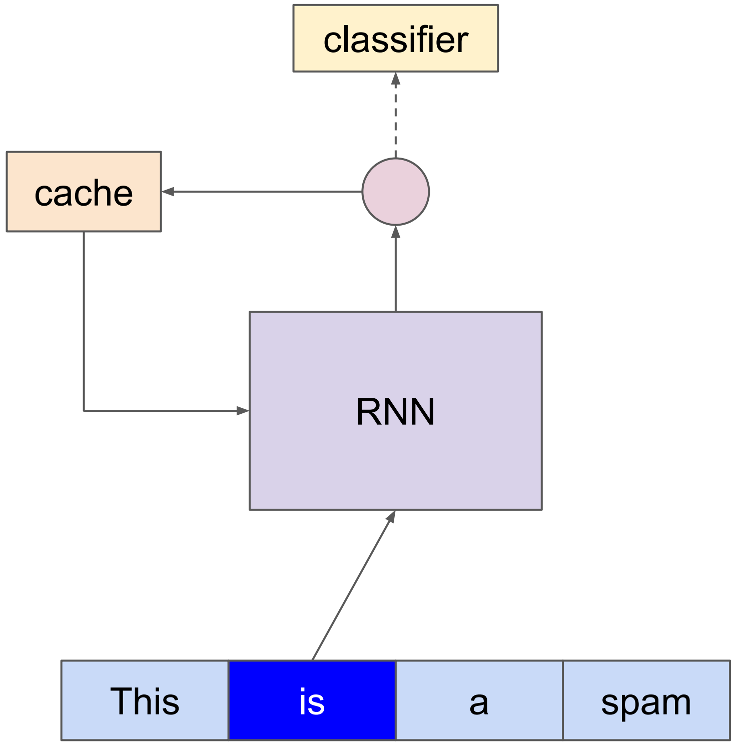
1. takes the token as input and combines it with the most recently stored feature vector,
2. obtains the new “feature vector” from the last hidden layer,



RNN Forward Propagation

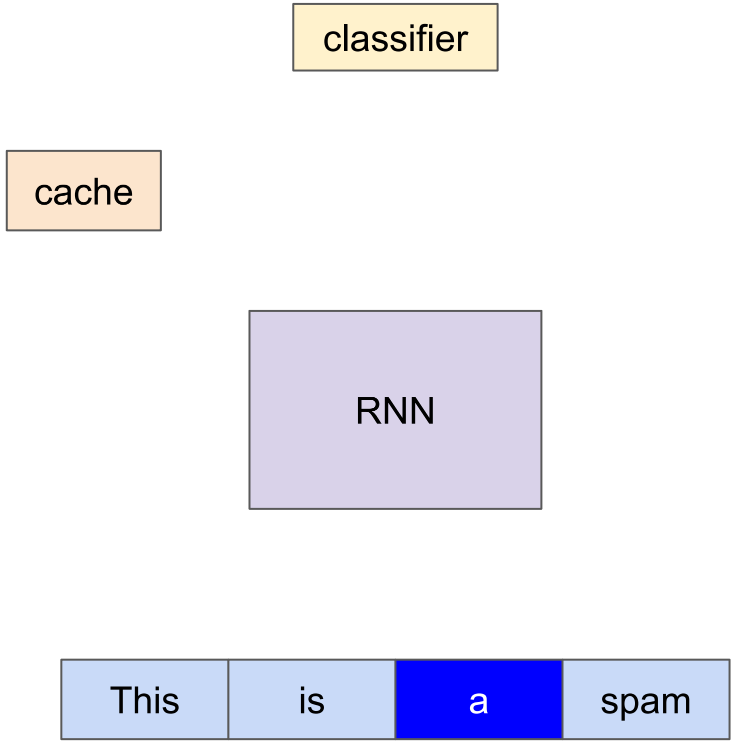
If the current word is between the first and the last, the model

1. takes the token as input and combines it with the most recently stored feature vector,
2. obtains the new "feature vector" from the last hidden layer,
3. and stores it (without necessarily sending it to the classifier layer).



RNN Forward Propagation

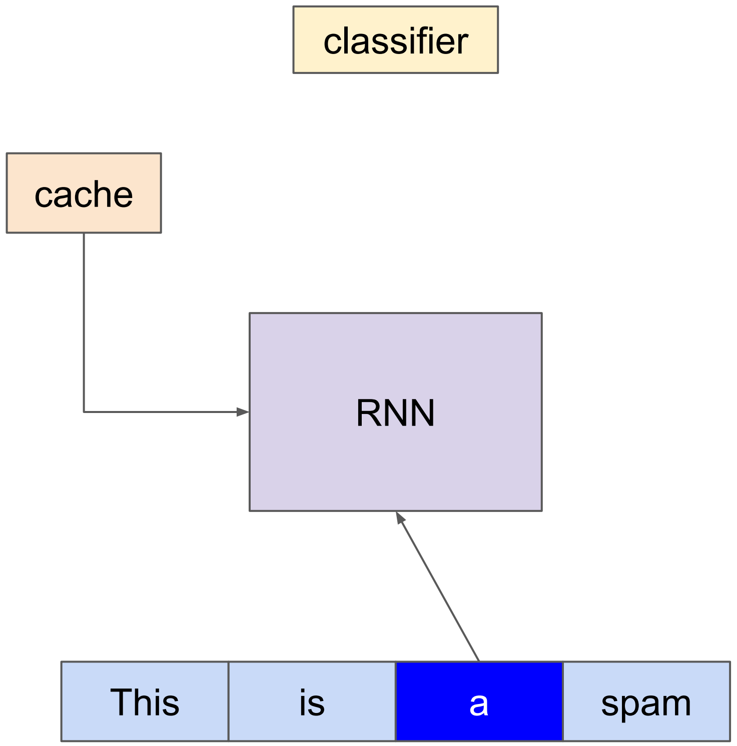
If the current word is between the first and the last, the model



RNN Forward Propagation

If the current word is between the first and the last, the model

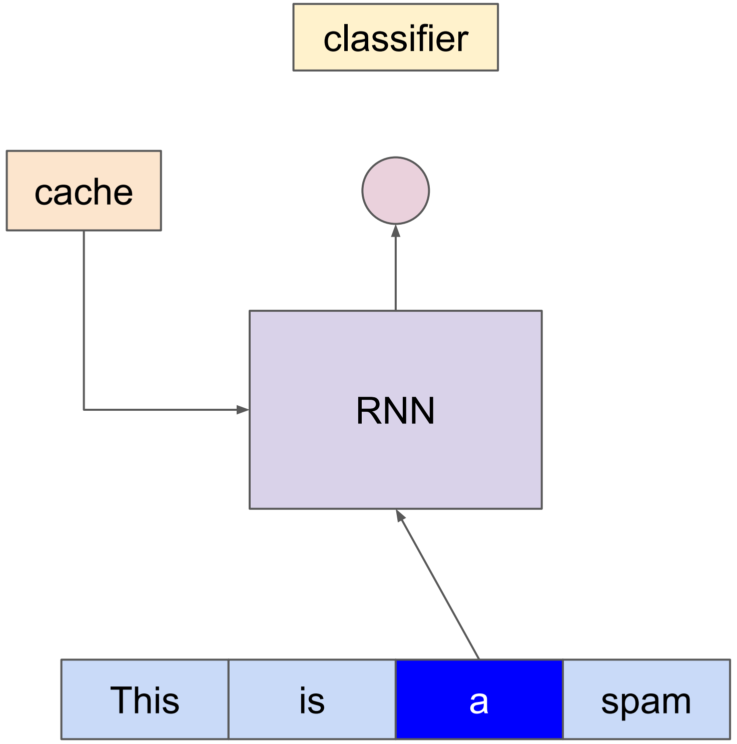
1. takes the token as input and combines it with the most recently stored feature vector,



RNN Forward Propagation

If the current word is between the first and the last, the model

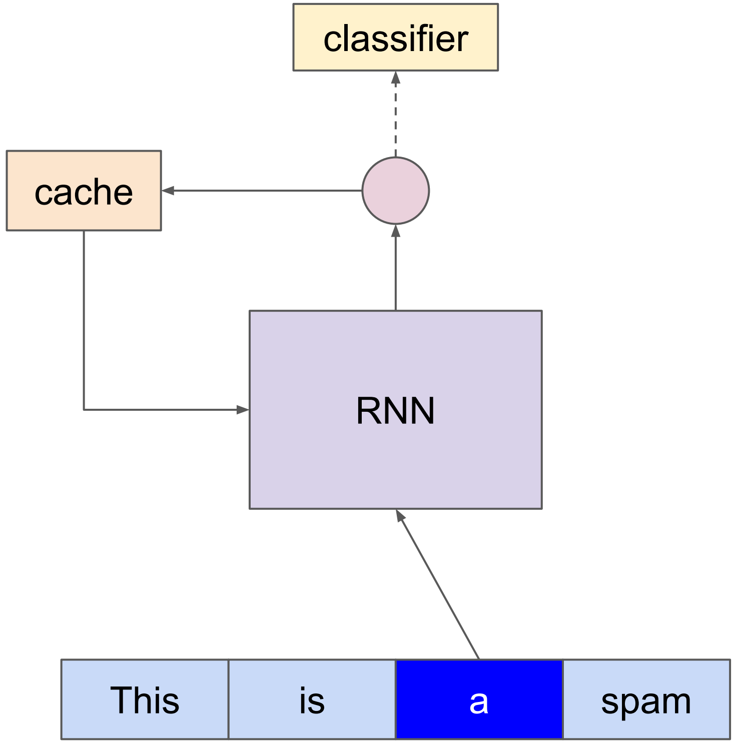
1. takes the token as input and combines it with the most recently stored feature vector,
2. obtains the new “feature vector” from the last hidden layer,



RNN Forward Propagation

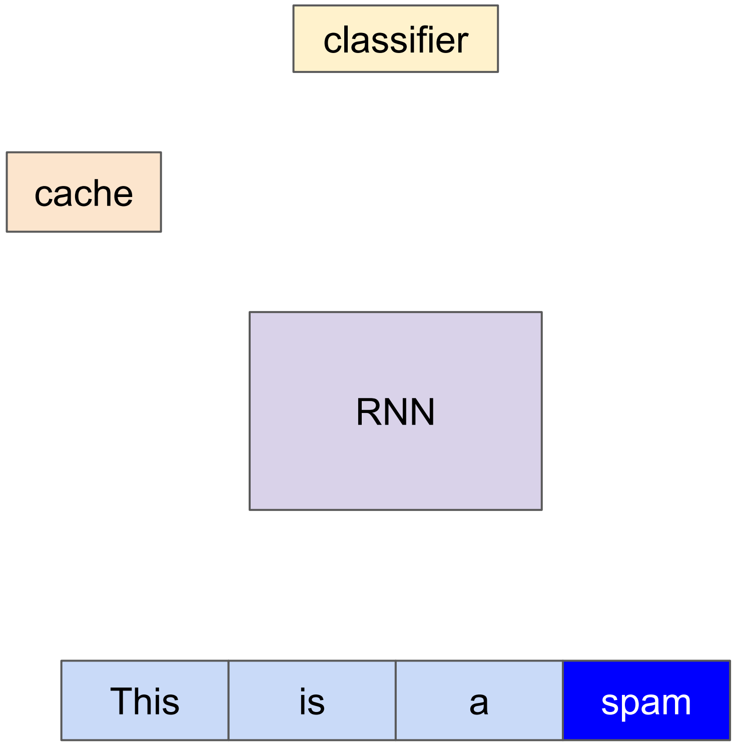
If the current word is between the first and the last, the model

1. takes the token as input and combines it with the most recently stored feature vector,
2. obtains the new “feature vector” from the last hidden layer,
3. and stores it (without necessarily sending it to the classifier layer).



RNN Forward Propagation

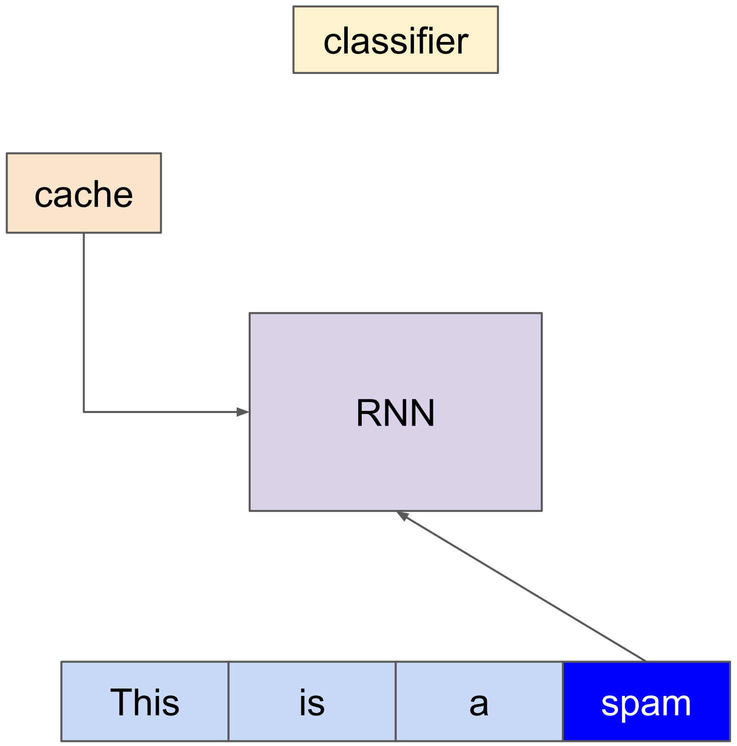
If the current word is the last in the sentence, the model



RNN Forward Propagation

If the current word is the last in the sentence, the model

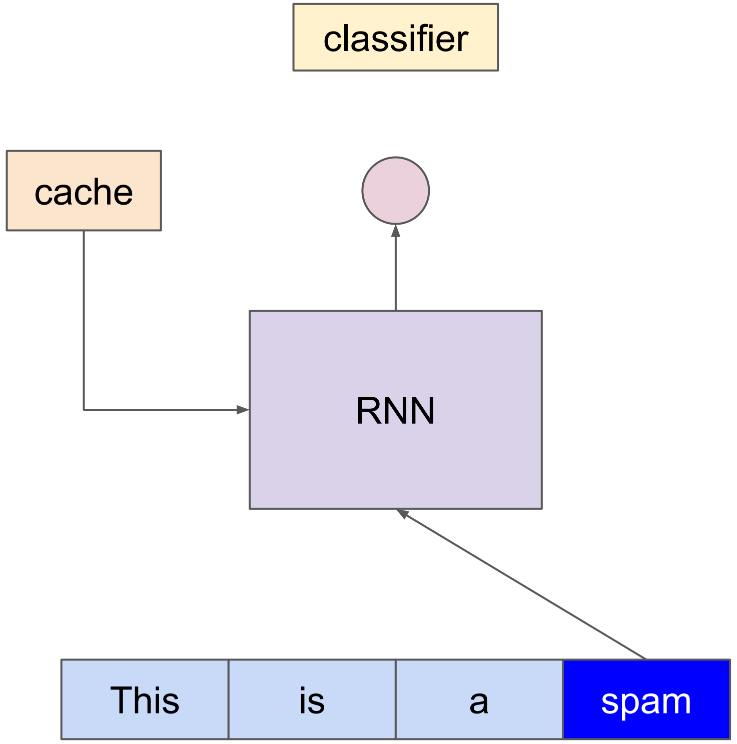
1. takes the token as input and combines it with the most recently stored feature vector,



RNN Forward Propagation

If the current word is the last in the sentence, the model

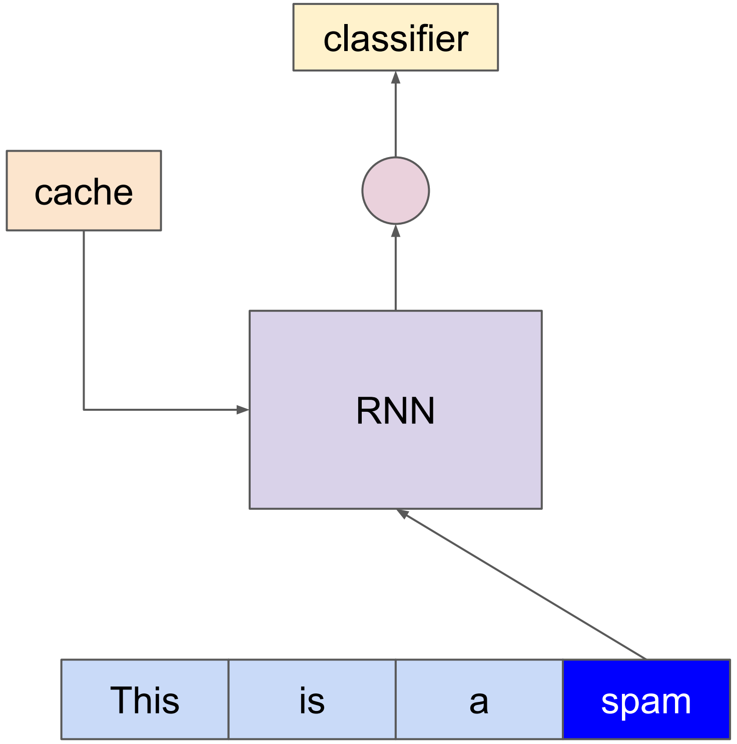
1. takes the token as input and combines it with the most recently stored feature vector,
2. obtains the new “feature vector” from the last hidden layer,



RNN Forward Propagation

If the current word is the last in the sentence, the model

1. takes the token as input and combines it with the most recently stored feature vector,
2. obtains the new "feature vector" from the last hidden layer,
3. and sends it to the classifier layer.



RNN Backward Propagation

One way to imagine how RNN works for a single token is that the token is sent to a sequence of functions.

Therefore, the backward propagation regarding that token is just computing a long chain of derivatives.

The number of functions is determined by the length of the sentence.

Such backward propagation is done for each token and each propagation yields a gradient.

The accumulated gradient is used to update the network.

RNN Backward Propagation

The disadvantage of RNN lies in the backward propagation: the longer the sentence, the longer the chained derivatives (which is a long product).

To address this, researchers made the network a lot more confusing (at least by the first glance) by introducing a “gate” mechanism. The “gate” can control if the previous token’s outputs may go to the next token’s pass.

The most famous modified variant is called **long short-term memory networks**, or **LSTM**.

Text Embedding

- In Natural Language Processing community, a **text embedding** is a numerical vector representation of the text.
- This vector can be any vector: the count vector, the ID vector, etc..
- Note that the feature vector from RNN’s last hidden layer is also embedding.
- In fact, it is very common to use the model’s feature vectors as embeddings.
    - This is because embeddings from a network carry more <u>context</u> and <u>semantic</u> information than just counts and IDs.

Attention - History

RNN/LSTM, despite of their issues, was the best approach for NLP.

That changed in 2014, when the concept of **attention** was proposed.

Three years later, the network structure called **Transformer** was built based on attention mechanism, and has been the base for all state-of-the-art models up to today.

For example, ChatGPT(ransformer).

Attention - Motivation

Consider the following news sentence from CNN:

"The legislative road ahead is tough, despite the bipartisan majority in committee, but this bill could redefine the business of marijuana."

Attention - Motivation

Consider the following news sentence from CNN:

“The legislative road ahead is tough, despite the bipartisan majority in committee, but this <u>bill</u> could redefine the business of marijuana.”

Attention - Motivation

Consider the following news sentence from CNN:

"The legislative road ahead is tough, despite the bipartisan majority in committee, but this bill could redefine the business of marijuana."

If we feed the sentence to an RNN model and obtain the embedding for the word "bill," which carries information of all words before "bill."

Attention - Motivation

Consider the following news sentence from CNN:

"The legislative road ahead is tough, despite the bipartisan majority in committee, but this <u>bill</u> could redefine the business of marijuana."

If we feed the sentence to an RNN model and obtain the embedding for the word "bill," which carries information of all words before "bill."

We can also feed the reverse sentence to the same RNN to obtain an embedding carrying information of all words after "bill."

Attention - Motivation

Consider the following news sentence from CNN:

"The legislative road ahead is tough, despite the bipartisan majority in committee, but this <u>bill</u> could redefine the business of marijuana."

If we feed the sentence to an RNN model and obtain the embedding for the word "bill," which carries information of all words before "bill."

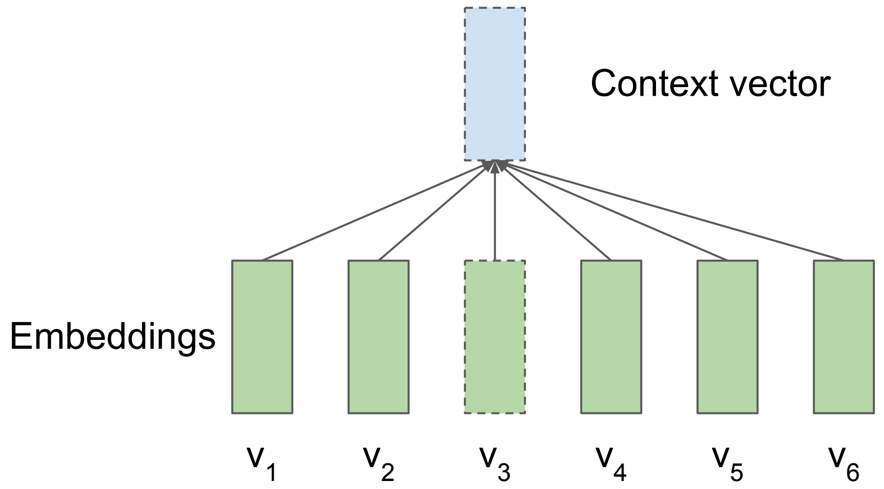
We can also feed the reverse sentence to the same RNN to obtain an embedding carrying information of all words after "bill."

However, is every word equally important in providing context for "bill?"

Attention - Single Target

- Given word embeddings and a target word
- Compute the similarities between the target word all words
- With the (normalized) similarity values as weights, compute the weighted sum of all embeddings
- The sum is called **context vector**, which is the new embedding for the target word

$$v'_3 = \sum_i a_i v_i = \sum_i \text{Similarity}(v_3, v_i) v_i$$



Attention - Self-Attention

Attention is, in fact, just similarity.

By taking the weighted sum, we are introducing context information to the target word embedding with more similar words contributing more information.

This is called **self-attention**. The same mechanism can be applied to different inputs.

Self-Attention - Overview

Let $V = (v_1, v_2, \ldots)$ be a sentence.

- For each v_i in V,
    - Compute $a_j = \text{Similarity}(v_i, v_j)$ for all v_j in V and normalize them
    - Compute context vector v_i' as a weighted sum of all v_j in V, with $\{a_j\}$ being the weights

Self-Attention - Overview

Let V = (v_1, v_2, …) be a sentence.

- For each v_i in **V**,
  - Compute a_j = Similarity(v_i, v_j) for all v_j in **V** and normalize them
  - Compute context vector v_i’ as a weighted sum of all v_j in **V**, with {a_j} being the weights

Query

Key

Value

Self-Attention - Network Version

Recall the forward propagation in matrix form (without bias)

$$y = W^T x$$

or, if x is a row vector

$$y = xW$$

Multiply a vector with a matrix means projection.

If the matrix corresponds to a network layer, then the projection is trainable (that is, can be optimized).

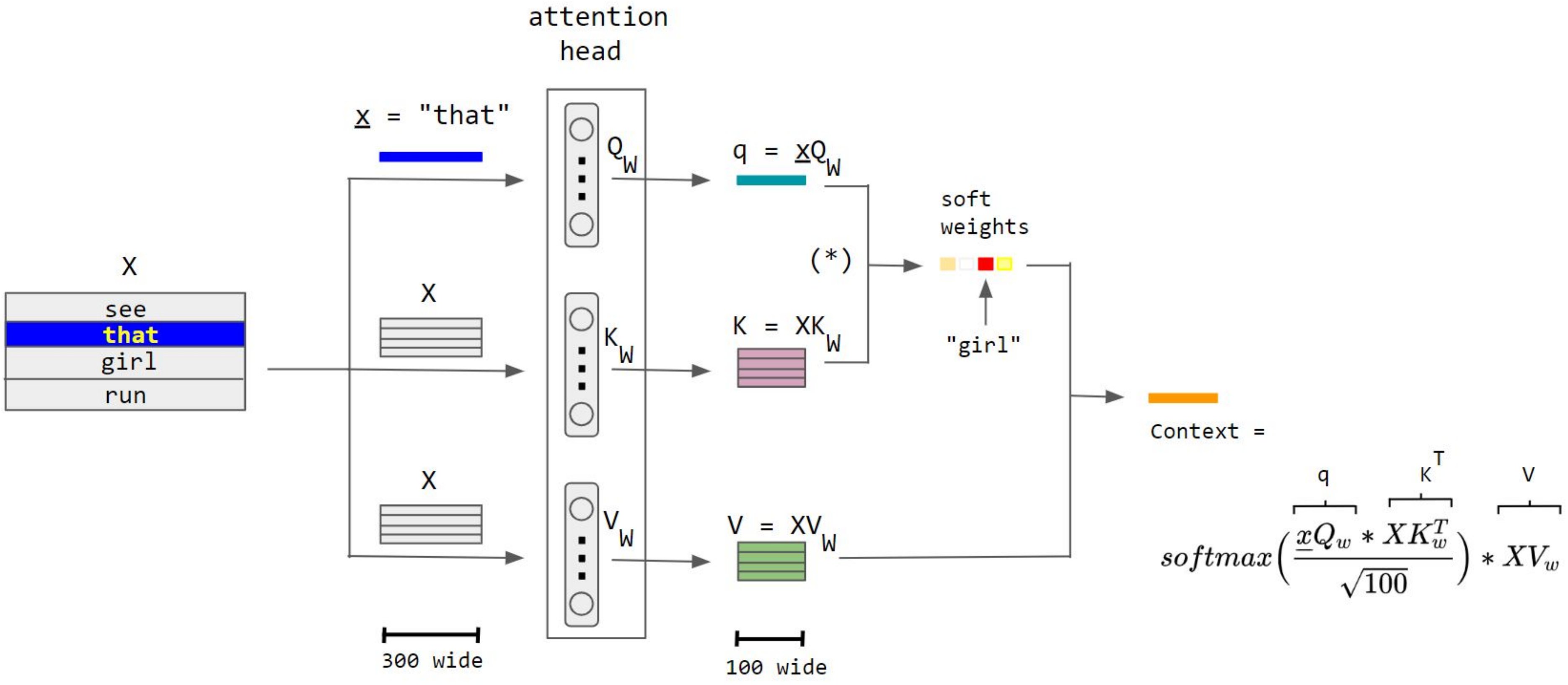
So, to bring the attention into a network, we can project the query, key, and value.

Self-Attention - Overview

Let $V = (v_1, v_2, \ldots)$ be a sentence.

- For each v_i in $(\mathbf{VQ_W})$,
    - Compute $a_j = \text{Similarity}(v_i, v_j)$ for all v_j in $(\mathbf{VK_W})$ and normalize them
    - Compute context vector v_i' as a weighted sum of all v_j in $(\mathbf{VV_W})$, with $\{a_j\}$ being the weights

Self-Attention - Overview



$$\mathrm{softmax}\left(\frac{\overbrace{\underline{x}Q_w}^{q} * \overbrace{XK_w^T}^{K^T}}{\sqrt{100}}\right) * \overbrace{XV_w}^{V}$$

Source: Wikipedia

Multi-Head Attention

Each instance of such attention network is called an **attention head**.

Usually multiple heads are computed in parallel for better performance.

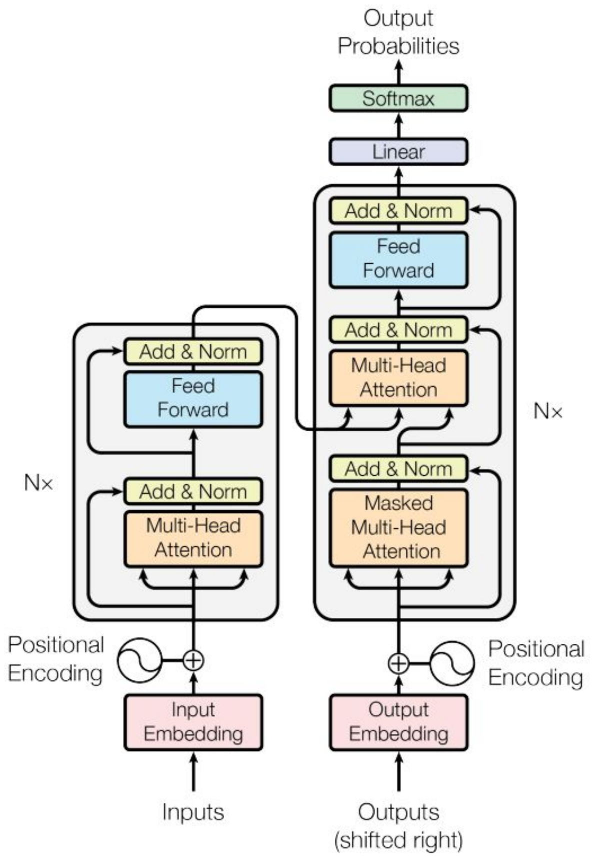
Multiple parallel heads form a multi-head **attention block** in a network.

A **Transformer** model is built based on the **attention block**.

Transformer

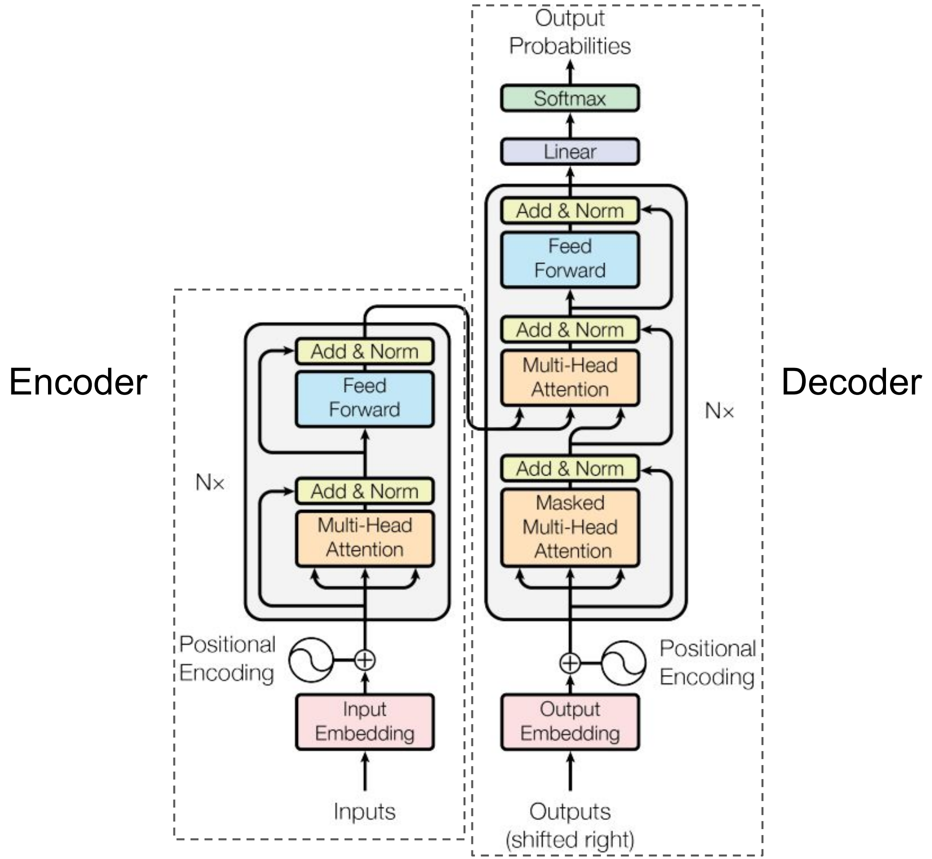
Source: Collider

Transformer



Source: Attention Is All You Need

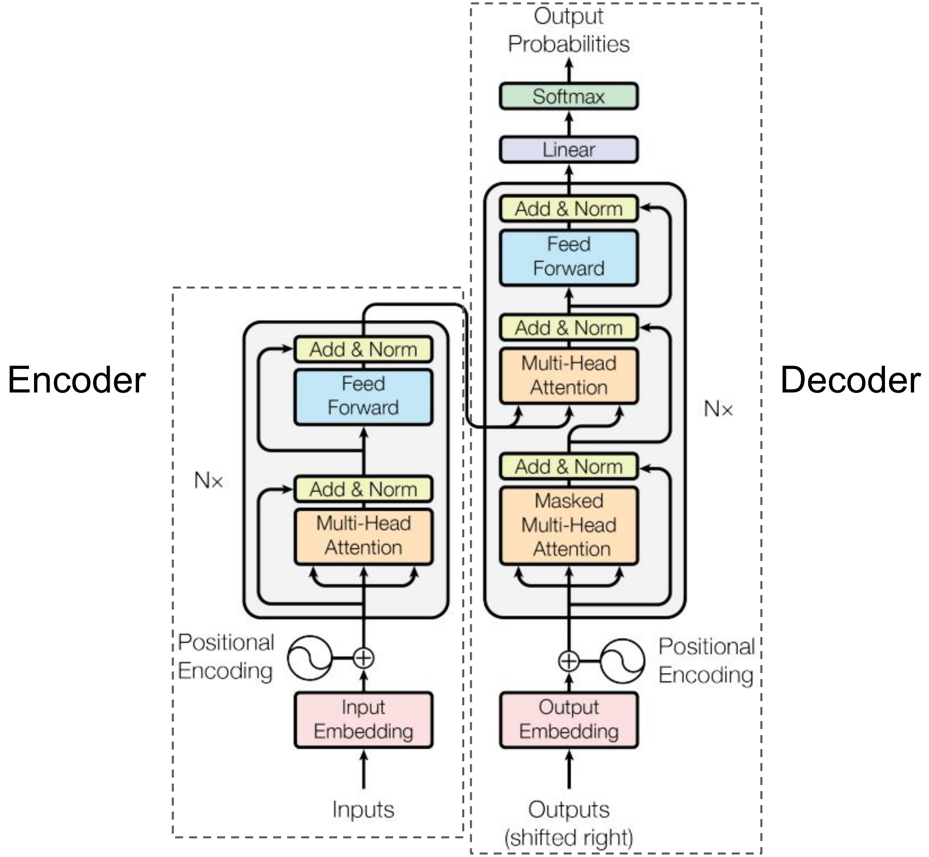
Transformer



Source: [Attention Is All You Need]

Transformer

- Encoder: text → feature vector
- Decoder: feature vector → text
- The model was originally designed for machine translation, hence the encoder-decoder structure
- For general purposes, encoder is sufficient



Source: Attention Is All You Need

Summary

- Shallow and deep networks are trained differently
- The output vector from the last hidden layer is commonly used as feature vector
- Text embedding = numerical vector representation of text
  - It can be built with different means, but most commonly with the feature vector from a network
- RNN/LSTM: processes one word at a time, allows previous information to flow into current and future steps
  - It has trouble with long text input
- Attention mechanism uses a weighted combination of context information to represent each word
- Transformer is built on top of attention and has been the core of many SOTA models in NLP

Summary

- Shallow and deep networks are trained differently
- The output vector from the last hidden layer is commonly used as feature vector
- Text embedding = numerical vector representation of text
    - It can be built with different means, but most commonly with the feature vector from a network
- RNN/LSTM: processes one word at a time, allows previous information to flow into current and future steps
    - It has trouble with long text input
- Attention mechanism uses a weighted combination of context information to represent each word
- Transformer is built on top of attention and has been the core of many SOTA models in NLP