Recurrent Neural Networks for Language Modeling

N-grams Vs. Sequence models

Tradition Language models

N-grams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)$$

- Large N-grams to capture dependencies between distant words
- Need a lot of space and RAM

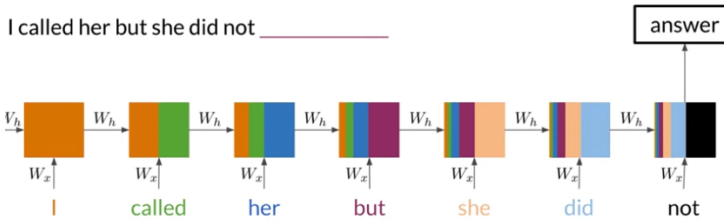
Recurrent Neural Networks (RNN)

RNNs look at every previous word.

RNNs model relationships among distant words.

A lot of computations share parameters.

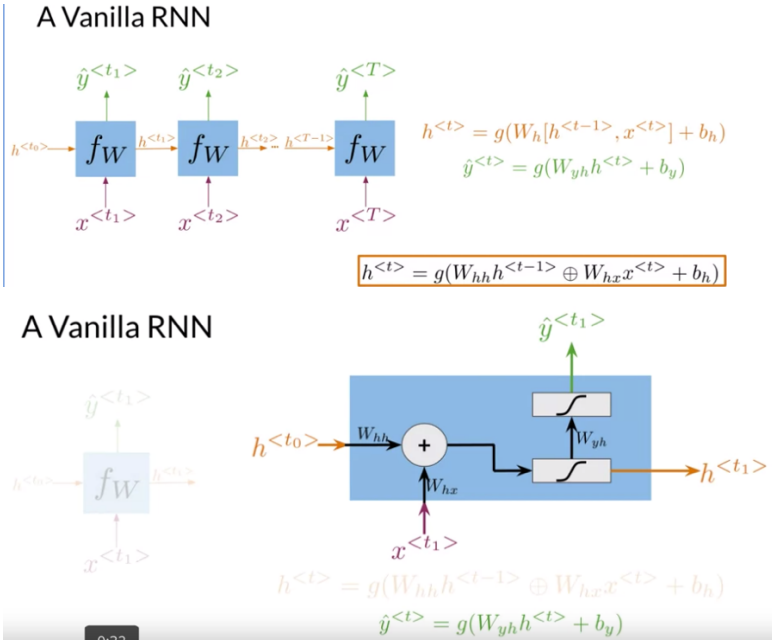
RNNs Basic Structure



Applications of RNNS

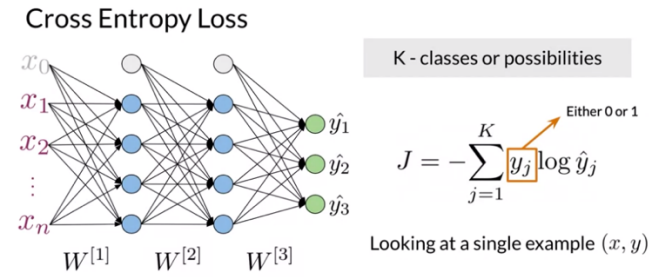
- One to one
- One to many (image to many words)
- Many to one (many words to sentiment)
- Many to Many (translation French words -> English words)

Math in Simple RNNs

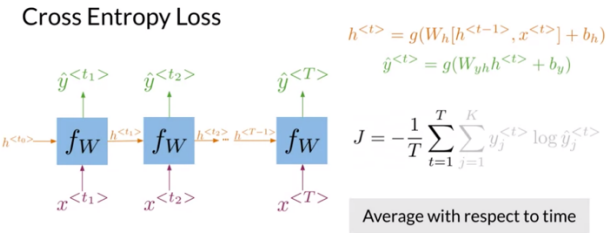


Cost Function for RNNs

For RNNs the loss function is just an average through time.



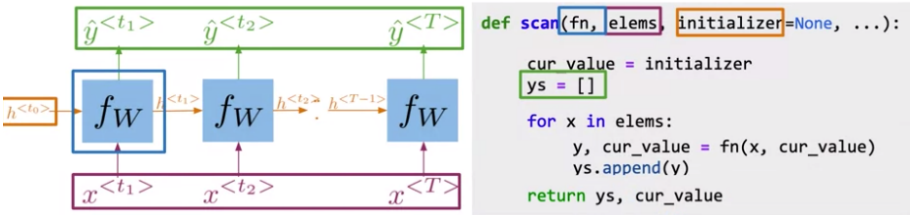
Loss for RNN



Implementation Notes

- Scan() functions def scan (fn, elems, initializer=None,)
- Frameworks like TF need this type of abstraction
- Parallel computations and GPU usage

tf.scan() function



Gated Recurrent Units (GRUs)

Gated Recurrent Units

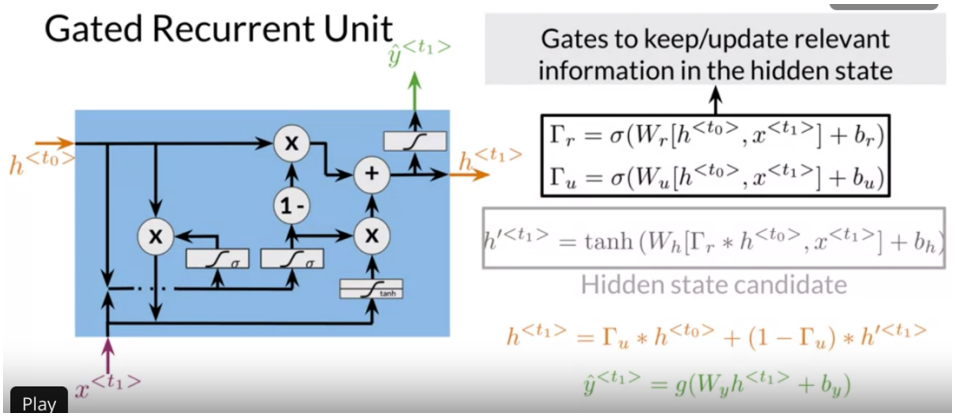
"Ants are really interesting. They are everywhere."

Plural

Relevance and update gates to remember important prior information

Gated Recurrent Unit

Gates to keep/update relevant information in the hidden state

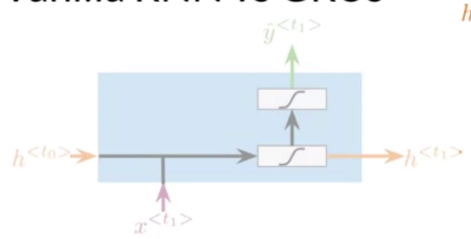


$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$

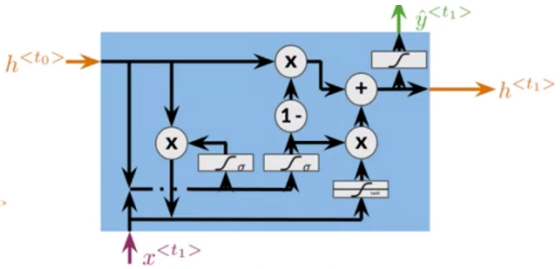
Hidden state candidate

$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

Vanilla RNN vs GRUs



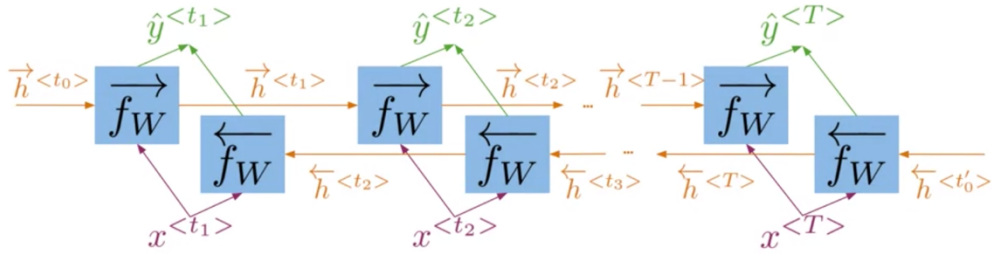
$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh} h^{<t>} + b_y)$$



$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$
$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

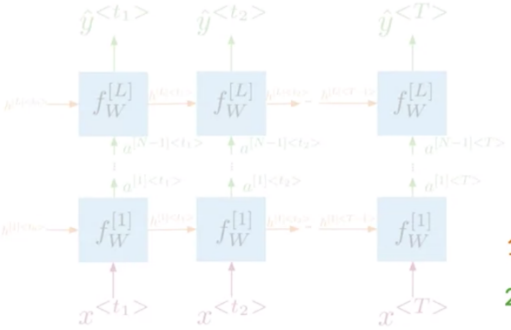
Deep and BiDirectional RNNs

Bi-directional RNNs



Information flows from the past and from the future **independently**

Deep RNNs



$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$

$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

1. Get hidden states for current layer
2. Pass the activations to the next layer

Calculating perplexity

Calculating Perplexity

The perplexity is a metric that measures how well a probability model predicts a sample and it is commonly used to evaluate language models. It is defined as:

$$P(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1,\ldots,w_{n-1})}}$$

As an implementation hack, you would usually take the log of that formula (to enable us to use the log probabilities we get as output of our `RNN`, convert exponents to products, and products into sums which makes computations less complicated and computationally more efficient). You should also take care of the padding, since you do not want to include the padding when calculating the perplexity (because we do not want to have a perplexity measure artificially good). The algebra behind this process is explained next:

$$logP(W) = log\Big(\sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1,\ldots,w_{n-1})}}\Big)$$

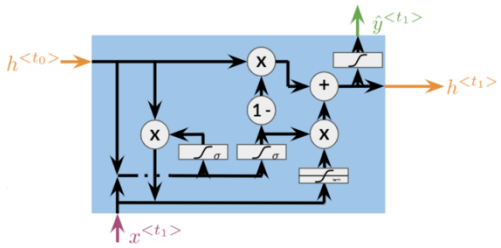
$$= log\Big(\prod_{i=1}^{N} \frac{1}{P(w_i|w_1,\ldots,w_{n-1})}\Big)^{\frac{1}{N}}$$

$$= log\Big(\prod_{i=1}^{N} P(w_i|w_1,\ldots,w_{n-1})\Big)^{-\frac{1}{N}}$$

$$= -\frac{1}{N} log\Big(\prod_{i=1}^{N} P(w_i|w_1,\ldots,w_{n-1})\Big)$$

$$= -\frac{1}{N}\Big(\sum_{i=1}^{N} logP(w_i|w_1,\ldots,w_{n-1})\Big)$$

A GRU cell have more computations than the ones that vanilla RNNs have. You can see this visually in the following diagram:



As you saw in the lecture videos, GRUs have relevance Γ_r and update Γ_u gates that control how the hidden state $h^{<t>}$ is updated on every time step. With these gates, GRUs are capable of keeping relevant information in the hidden state even for long sequences. The equations needed for the forward method in GRUs are provided below:

$$\Gamma_r = \sigma(W_r[h^{<t-1>}, x^{<t>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[h^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \tanh(W_h[\Gamma_r * h^{<t-1>}, x^{<t>}] + b_h)$$

$$h^{<t>} = \Gamma_u * c^{<t>} + (1 - \Gamma_u) * h^{<t-1>}$$