Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are mostly used in applications of natural language processing and speech recognition.

One of the main motivations for RNNs is to derive insights from text and do better than “bag of words” implementations. Ideally, each word is processed or understood in the appropriate context.

Words should be handled differently depending on “context”. Also, each word should update the context.

Under the notion of recurrence, words are input one by one. This way, we can handle variable lengths of text. This means that the response to a word depends on the words that preceded it.

These are the two main outputs of an RNN:

- Prediction: What would be the prediction if the sequence ended with that word.
- State: Summary of everything that happened in the past.

Mathematical Details

Mathematically, there are cores and subsequent dense layers

current state = function1(old state, current input).

current output = function2(current state).

We learn function1 and function2 by training our network!

r = dimension of input vector

s = dimension of hidden state

t = dimension of output vector (after dense layer)

U is a $s \times r$ matrix

W is a $s \times s$ matrix

V is a $t \times s$ matrix

In which the weight matrices U, V, W are the same across all positions

Practical Details

Often, we train on just the ”final” output and ignore intermediate outputs.

Slight variation called Backpropagation Through Time (BPTT) is used to train RNNs.

Sensitive to length of sequence (due to “vanishing/exploding gradient” problem).

In practice, we still set a maximum length to our sequences. If the input is shorter than maximum, we “pad” it. If the input is longer than maximum, we truncate it.

RNN Applications

RNNs often focus on text applications, but are commonly used for other sequential data:

- Forecasting: Customer Sales, Loss Rates, Network Traffic.
- Speech Recognition: Call Center Automation, Voice Applications.
- Manufacturing Sensor Data
- Genome Sequences

Long-Short Term Memory RNNs (LSTM)

LSTMs are a special kind of RNN (invented in 1997). LSTM has as motivation solve one of the main weaknesses of RNNs, which is that its transitional nature, makes it hard to keep information from distant past in current memory without reinforcement.

LSTM have a more complex mechanism for updating the state.

Standard RNNs have poor memory because the transition Matrix necessarily weakens signal.

This is the problem addressed by Long-Short Term Memory RNNs (LSTM).

To solve it, you need a structure that can leave some dimensions unchanged over many steps.

- By default, LSTMs remember the information from the last step.
- Items are overwritten as an active choice.

The idea for updating states that RNNs use is old, but the available computing power to do it sequence to sequence mapping, explicit memory unit, and text generation tasks is relatively new.

Augment RNNs with a few additional Gate Units:

- Gate Units control how long/if events will stay in memory.
- Input Gate: If its value is such, it causes items to be stored in memory.
- Forget Gate: If its value is such, it causes items to be removed from memory.
- Output Gate: If its value is such, it causes the hidden unit to feed forward (output) in the network.

Gated Recurrent Units (GRUs)

GRUs are a gating mechanism for RNNs that is an alternative to LSTM. It is based on the principle of Removed Cell State:

- Past information is now used to transfer past information.
- Think of as a “simpler” and faster version of LSTM.

These are the gates of GRU:

Reset gate: helps decide how much past information to forget.

Update gate: helps decide what information to throw away and what new information to keep.

LSTM vs GRU

LSTMs are a bit more complex and may therefore be able to find more complicated patterns.

Conversely, GRUs are a bit simpler and therefore are quicker to train.

GRUs will generally perform about as well as LSTMs with shorter training time, especially for smaller datasets.

In Keras it is easy to switch from one to the other by specifying a layer type. It is relatively quickly to change one for the other.

Sequence-to-Sequence Models (Seq2Seq)

Thinking back to any type of RNN interprets text, the model will have a new hidden state at each step of the sequence containing information about all past words.

Seq2Seq improve keeping necessary information in the hidden state from one sequence to the next.

This way, at the end of a sentence, the hidden state will have all information relating to past words.

The size of the vector from the hidden state is the same no matter the size of the sentence.

In a nutshell, there is an encoder, a hidden state, and a decoder.

Beam Search

Beam search is an attempt to solve greedy inference.

- Greedy Inference, which means that a model producing one word at a time implies that if it produces one wrong word, it might output a wrong entire sequence of words.

- Beam search tries to produce multiple different hypotheses to produce words until <EOS> and then see which full sentence is most likely.

These are examples of common enterprise applications of LSTM models:

- Forecasting: (LSTM among most common Deep Learning models used in forecasting).
- Speech Recognition
- Machine Translation
- Image Captioning
- Question Answering
- Anomaly Detection
- Robotic Control