Document Classification with Tensorflow

An interactive tutorial using RNNs to classify text.

Chris Ormandy

July 18, 2017

Outline

Outline

Prerequisites & Goals

Knowledge is a brick wall that you raise line by line forever

Prerequisites

- Basic Tensorflow terminology
- Basic neural network terminology

Goals

- A bit more insight into RNNs and Tensorflow
- Some key techniques in Deep NLP

Outline

1. Introduction
2. RNN theory
3. Practical

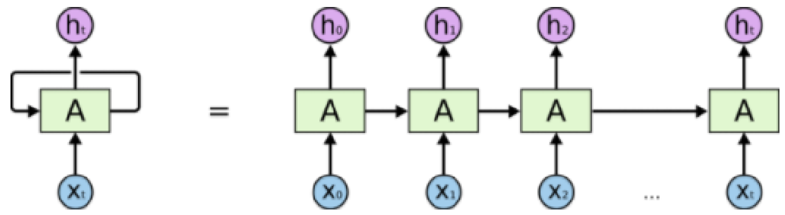
RNNs

From feed forward to RNNs

- An RNN takes a sequence as input.
- RNNs can still be trained using backprop.
- Time dependency modelled by recursive layer definition.

Example

Visual illustration of RNN (Colah)



RNNs

Feed forward networks

- $h = f(Xw + b)$

Simple Recurrent networks

- $h_t = f(X_t w + h_{t-1} u + b)$

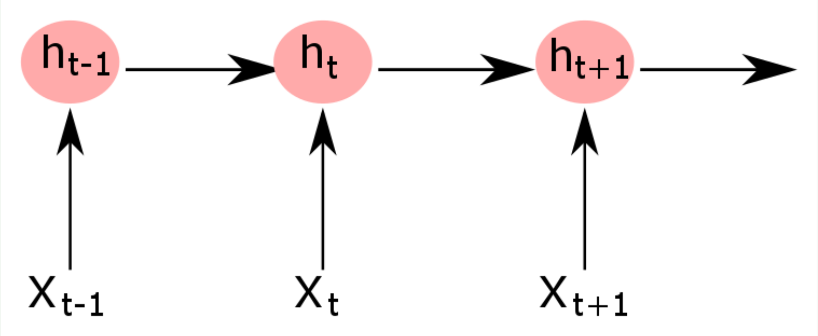
General Recurrent networks

- $h_t = cell(X_t, h_{t-1})$

RNN Recap

Example

Visual illustration of RNN



RNN Numerical Example

Example

Using a simple RNN to count 1s in a sequence

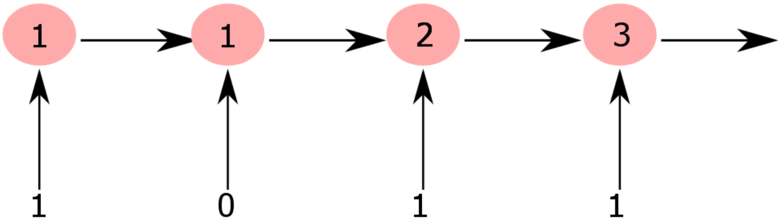
$$h_t = f(x_t W + h_{t-1} U + b)$$

f = identity function (no activation function)

W = 1

U = 1

b = 0



Representing Words As Vectors

Word2Vec etc...

Intuition

- We could think of words as representing a collection of features relating to different general concepts or topics.
- E.g. The word "Queen" would have a high value for the features that correspond to Female, Royalty, Person etc. and low values for features such as Male, Machine, Vegetable etc.
- Similar words are 'closer'. For example "King" is close to "Queen" for most features.
- In practice, we don't define these latent features, we just tell a neural network how many there are.
- More detail on COlah.

Outline

A basic Deep NLP pipeline

Simple steps

- Build vocabulary
- Map words to word embeddings (pre-trained or learnt during training)
- Represent text as matrix
- Feed into RNN
- Operate on hidden states of RNN

Different types of cell.

- Long short term memory cells and Gated Recurrent units form backbone of current methods.
- Simple RNN cell remembers every input.
- GRUs and LSTMs have an internal classifier which decides how much of an input to remember.
- This means it can learn to ignore uninformative words like 'the'.
- Also have an internal classifier that can decide whether to forget everything it has learnt so far.
- Useful if current input is really discriminative.

Beyond a single RNN

Why just using the final state is not good enough

- Even with complex cells like LSTMs, storing information for a long time is hard.
- This means our model 'forgets' what it saw early in the sequence.
- One approach is to use two RNNs, one which sees the sequence in order, and one which sees it in reverse and combine their final hidden states.

Bidirectional RNNs

Bidirectional Networks - why do they work?

- Imagine your RNN cell can store information for 500 words.
- For 1000 word document, can't represent it all using this cell.
- However if we use 2, one forwards and one backwards we can.
- Forward RNNs final state represents last 500 words.
- Reverse RNNs final hidden state represents first 500 words.

Beyond Final States

Hidden states

- Recall - an RNN has a hidden state for each step in the sequence.
- Each hidden state summarises the sequence up to an including that timestep.
- A simple method would be to average all the h_ts instead of taking the last one.

Beyond Final States

Combining Hidden States

- Plain average isn't usually great for combining hidden states in practice (unless you have very short text).
- **Sometimes better:** elementwise Min or Max or similar non-linear function.
- **Best:** Weighted average

Attention

Weighted Average to Attention

- Make the weighting of each hidden state a model parameter.
- Weightings are computed as function of the hidden state, trained by backprop.
- Compute weighted average of all hidden states using these weightings.

Remember!

- Remember more complex models aren't always better!
- More parameters requires more data and more time!

State of the Art

Pipeline

Most state of the art currently follows this pipeline:

- Embed - turn documents into matrix of word embeddings.
- Encode - Encode the documents into document matrices (like we did with RNN).
- Attend - Use attention to compute a dense document representation.
- Predict - Use the document representation to do inference.

Taken from Matthew Honnibal

State of the Art

Document Classification

What we have done today is pretty close to state of the art approaches for this type of problem.

- Even more sophisticated - Hierarchical Attention Networks.
- Divide documents into sentences.
- Run the same pipeline we have used today, to generate an attended sentence representation.
- Then do the same on the sentence embeddings to get a document vector.

State of the Art

Character Level Models

- Word embeddings are hard to generalise to unseen words.
- This is because the number of words is very, very large.
- Can try to use character embeddings - same training procedure as word embeddings.
- Much smaller space - a large character vocabulary is approx. 150 characters.
- However, no free lunch. This makes sequences much longer - a sequence of 50 words is several hundred characters.
- Need more complex models to encode long sentences.
- Offbit has an example Char level classification.

Even Further

Sequence to Sequence and NMT

- Once you understand text classification, ideas like sequence to sequence are a natural generalisation.
- Instead of using final state as input to a classifier, use it as the starting state of another RNN, and generate multiple predictions - for example translated text.
- All revolves around computing dense representations of text!
- Text classification a good pre-training tool for these architectures too!

Finally

Questions?

Where can I find more?

Resources

- Accompanying code on my github.
- Stanford CS224n - Excellent course to get started, with videos Cs224n.
- Matthew Honnibal's blog - explosion.ai.
- A great list of papers by topic on Deep NLP - andrewt3000
- General tensorflow - CS20SI