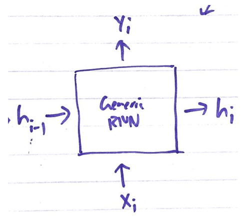
6.S198 Assignment 4

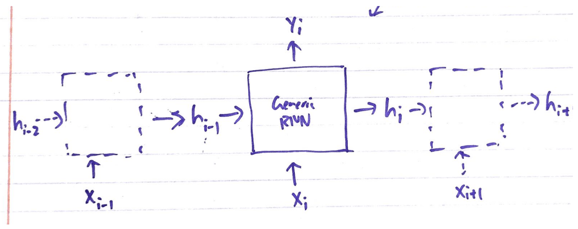
Recurrent Neural Networks

In this unit, we'll learn about a new type of neural architecture which will allow us to effectively model sequence data. They have been successfully used for natural language processing tasks such as sentiment analysis and machine translation, and can be adapted to work for many other machine learning tasks from music and image generation.



The above figure is a popular way of representing how data flows through a single cell in a recurrent neural network. The input (x) flows in from the bottom while the previous hidden state (h) flow in from the left, producing a new hidden state (h') on the right and an output (y).

Given a series of inputs, we apply our RNN cell sequentially to each input, updating the hidden state as we go along.



Let's examine two concrete implementations of recurrent network cells.

Simple RNN

The original recurrent neural network (RNN) proposed by Jeff Elman in 1990 has largely been superseded by the long-short term memory (LSTM) and gated recurrent units (GRU), but can still be used as a simple baseline.

$$h^t = tanh(W^{h,x}x^t + W^{h,h}h^{t-1})$$
$$y^t = softmax(W^{o,h}h^t)$$

We see that in the above formulation, the new hidden state is a function of the previous hidden state and the current input. By applying this recursively, we see that the hidden state is a function of all previous inputs.

In practice, however, simple RNNs often fail to learn long-term dependencies[1], which is why researchers have proposed alternatives such as the LSTM and GRU.

LSTM

The original equations for the long-short term memory network are shown below.

$$f^t = sigmoid(W^{f,x}x^t + W^{f,h}h^{t-1})$$
$$i^t = sigmoid(W^{i,x}x^t + W^{i,h}h^{t-1})$$
$$o^t = sigmoid(W^{o,x}x^t + W^{o,h}h^{t-1})$$
$$c^t = f^t c^{t-1} + i^t tanh(W^{c,x}x^t + W^{c,h}h^{t-1})$$
$$h^t = o^t\ tanh(c^t)$$

The *f*, *i*, and *o* variables are referred to as the forget, input, and output gates. They allow the model to explicitly control how much of the hidden state to forget, how much of the hidden state to replace with new inputs, and how much of the hidden state to show the "world".

By explicitly allowing our model to control what to remember and what to forget, we see much better performance on tasks that require long term memory.

A Word of Warning

Recurrent neural networks are a powerful tool for processing sequential data.... but they are also very difficult to train.

Due to their sequential nature, the outputs can only be computed after the entire sequence of inputs has been processed, which means training is very time consuming and resource intensive.

Furthermore, exploding and vanishing gradients are a much bigger problem in recurrent networks than feedforward or convolutional networks since the gradients need to be propagated through many timesteps.

[1] In other words, they often fail to remember inputs from more than a few timesteps ago.

Exercise 1

In this exercise, you will use deeplearnjs to build a model for classifying text[2]. Specifically, our model will learn to classify emails as being related to one of the following topics: cryptography, electronics, medicine, or space.

This exercise can be completed online through the editor at https://6s198.kevz.me/ or offline by cloning https://github.com/k15z/6s198.kevz.me/ and using your standard IDE setup.

We will train and evaluate 2 types of models: a traditional bag of words (BoW) model and a LSTM.

1.1 Bag-of-Words

In this section, you will modify a standard bag-of-words model to try and achieve better performance.

Recall that a binary bag-of-words encoding maps every unique word to a different dimension. When given a sentence with 10 distinct words, the encoding is a high-dimensional vector with 10 ones indicating the words present in the sentence.

This binary feature vector is fed into a single-layer neural network which is equivalent to multiclass logistic regression.

Navigate to the online editor (or clone the repository and set it up locally) and make sure you understand the template code. Run it for 100 epochs and report the performance below.

Baseline: Multiclass logistic regression on a binary bag-of-words, provided by the staff.	
Train Loss	
Test Loss	
Test Accuracy	
Training Time	
Number of Epochs	

[2] http://qwone.com/~jason/20Newsgroups/

Now, try modifying the template by adding hidden layers and regularization (dropout or weight decay) and report your results below.

Model 1: ______________________	
Train Loss	
Test Loss	
Test Accuracy	
Training Time	
Number of Epochs	

Model 2 (optional): _______________	
Train Loss	
Test Loss	
Test Accuracy	
Training Time	
Number of Epochs	

Now that we have some reasonable baselines[3], let's see if we can do better with a recurrent network.

1.2 LSTM

Coming soon...

[3] Typically you would use ngrams as opposed to "words"; we choose to only use unigrams in this implementation due to time constraints.

Additional Questions

1. Name a few key differences between our bag-of-words model and our recurrent neural network. (Hint: What information does our bag-of-words model lack?)

2. Why are LSTMs typically more successful than simple RNNs?

3. Find and describe an interesting application or modification of recurrent neural networks. Some useful keywords include PixelRNN, attention mechanisms, and reinforcement learning. Feel free to explore /r/machinelearning and arXiv for more ideas.