**Experimenting with the Hyperparameters of a Recurrent Neural Network**

**About RNNs**

A key advantage to using an RNN is their ability to work with variable length sequences, both as inputs and outputs so their relationship can be anything from ‘One-to-One’ to ‘Many-to-Many’. In Natural Language Processing, there are two common use cases for an RNN. Sentiment Analysis uses a ‘Many-to-One’ RNN as multiple words are passed as input, whereas the output is a single classification. In Translators, a ‘Many-to-Many’ RNN is used as both the input and output are a collection of words.

**What does the Maths look like?**

* Matrices
* Vectors
* Trigonometric Functions

Consider a ‘Many-to-Many’ RNN; X0, X1 … Xn are the inputs and Y0, Y1 … Yn are the outputs. X and Y are *vectors*. In the hidden layer, there is a *vector* ‘h’.

hn is calculated using hn-1 and Xn.

Yn is calculated using ht.

A typical RNN uses the same 3 weights for each step. All of these are *matrices*.

Wxh is used for xn to hn  links (Input Layer to Hidden Layer)

Whh is used for hn-1 to hn links (Inside Hidden Layer)

Why is ised for hn to yn links (Hidden Layer to Output Layer)

There are also two bias neurons.

bh is used when calculating hn

by is used when calculating Yn

*Formulae:*

hn = tanh((Wxh\*xn)+(Whh\*hn-1)+ bh)

yn = (Why\*hn) + by

**Optimising an RNN: Phase 1**

‘Learning Rate’ refers to by how much the network adjusts a parameter

Aim of Phase 1 is to modify the *Learning Rate* and observe its effect on the accuracy of the model at given intervals. Optimiser is defaulted to Adam for this phase.

Vary the Learning Rate from 0.0001, 0.001, 0.01 and 0.1.

**Hypothesis:**

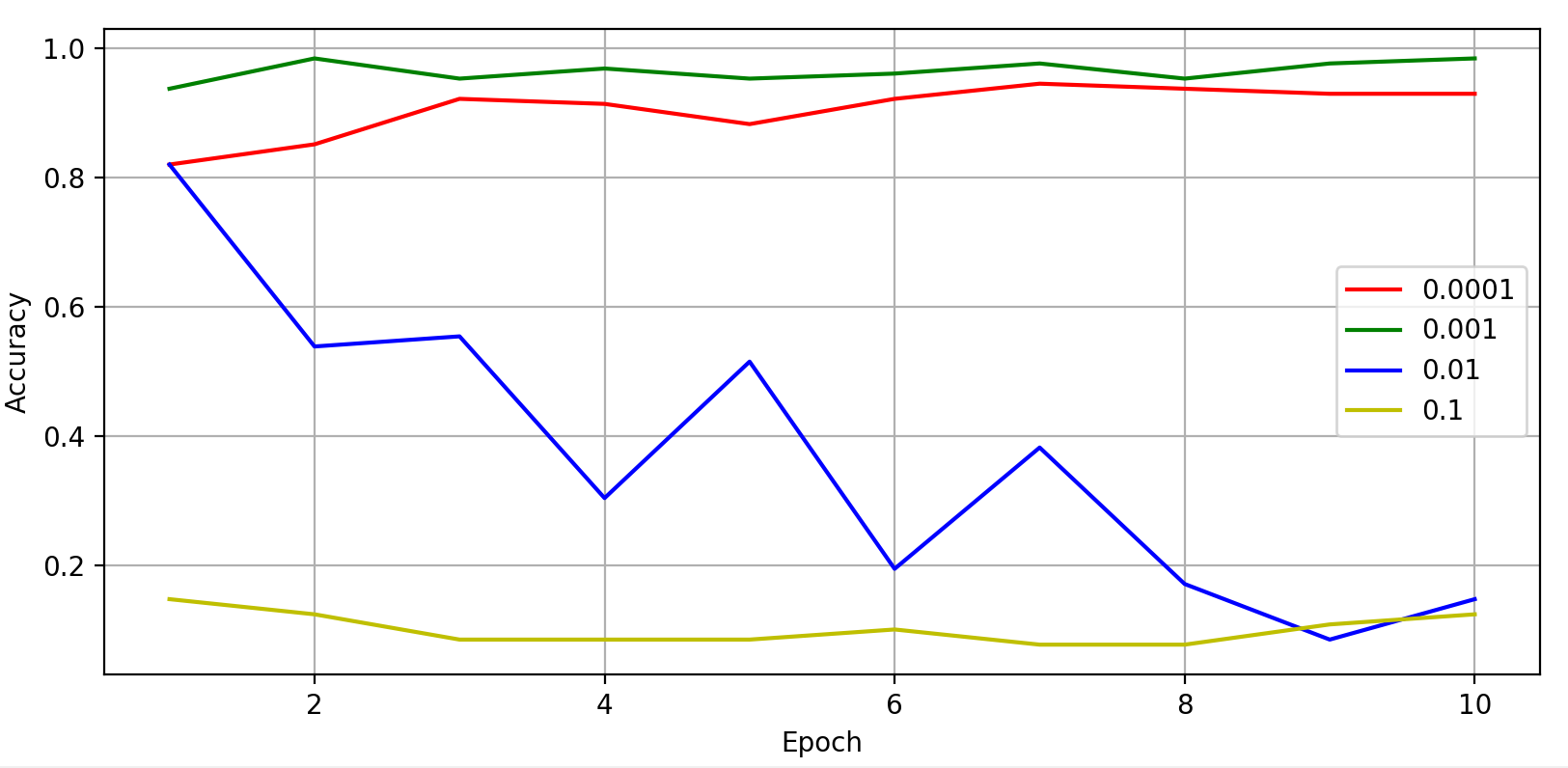
0.0001: Converges in later epochs.

0.001: Converges in later epochs

0.01: Diverges quickly.

0.1: No improvement or detriment to performance.

**Results:**



**Phase 2 – Adversarial Examples**

<https://ml.berkeley.edu/blog/posts/adversarial-examples/>

Now that the model is producing results, it’s important to see if there is any commonality between the images it incorrectly categorised. Are there pairs of numbers which are often mistaken for each other? If so, can we isolate enough of these incorrect answers to produce an adversarial training dataset which may further improve the accuracy achieved on the testing dataset?

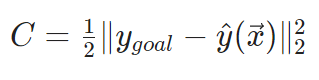
*Targeted Adversarial Examples* add carefully constructed noise over an image which would normally fall correctly into Class A with a high confidence, but due to the noise is now incorrectly categorised as Class B with a very high confidence.

Non-Targeted Adversarial Examples are ones without any additional noise which simply trick the Neural Network into incorrectly categorising an image. To do this, we generate an image that has been designed to make the network give a certain output.

Cost Function: ∑(xn – yn)2 takes the sum of (actual output – expected output)2. A high output shows a low accuracy and the inverse of that is true. The average of all outputs over the full training process is indictative of the quality of the network. Finding the local minimum of a cost function involves beginning at a random x co-ordinate on a graph and working out if the gradient is positive or negative. A positive gradient means that the local minimum is to the left of the current x position and a negative gradient means that the local minimum is to the right. As you move closer to the local minimum, your movement reduces so that you don’t overshoot and end up passing it. This process is then scaled up for each neuron in the network.

Consider the magnitude of each value in the column vector to be an instruction. If the value of w3 corresponds to the adjustment ‘-0.13’, then w3 needs to decrease slightly. However if the value of w4 correspondes to ‘+1.34’, then w4 must increase greatly. This relative importance shows which neurons need to be adjusted to have the greatest overall positive effect on the result.

1. Consider the image you want to make to be a 784-dimensional vector (28\*28)
2. Define a cost function:

RNN is not ideal for MNIST as it is a classification task. Implementing LSTM instead of SimpleRNN in Keras yields an improved accuracy of +5% by simply changing the type of network and no other code.

LSTMs can learn long-term dependencies.