COSC420 Assignment 1

Shane Mulligan, University of Otago

May 1, 2018

Abstract

Given commonly accepted rules of thumb for selecting activation and error functions both individually and in combination, in this report, some of these will be tested to see how well the hold up against my generalized delta rule network.

In particular, the Relu, sigmoid, tanh and softmax activation functions are be tested for their performance on the input, hidden and output layers.

Two error functions, sum of squares and negative log likelihood, are tested for their performance on fitting the data.

1 Introduction / Background

There are many activation and error functions available to choose from when designing a Neural Network to solve a particular task. One could even design their own if they wanted.

There are rules of thumb for selecting functions that have been shown empiracally to give you better results. I've decided to explore the difference that choice can make on the time it takes to train my delta rule network. To test this I've trained my network on the given datasets using each function under observation against a set of controlled values. To provide more evidence I've also run each experiment over a range of values for regularization and learing rate decay.

Finally I compare the different functions results by interpretting the data plotted side by side.

2 Experiments

In all experiments the regularization factor and decay rate parameters were iterated from 0.0 to 0.5 (inclusive) to provide a larger set of data for us to compare the effect of the functions under investigation.

2.1 Experiment 1 of 5 – Comparing activation functions for the hidden layer

2.1.1 Description

In this experiment, the Relu, softmax, sigmoid and tanh functions will be compared for their performance as the activation function of the hidden layer.

2.1.2 Questions

1. Do we see evidence supporting any of these rules of thumb in our results?
   (a) ReLU is used usually for hidden layers as it avoids vanishing gradient problem.
   (b) softmax will provide greater accuracy multiclass logistic regression datasets (Iris).

2.1.3 Hypotheses

1. We will see evidence of of the vanishing gradient problem when ussing sigmoid or tanh.
   See appendix 1.
2. softmax will provide greater accuracy and faster training than relu, sigmoid or tanh.
3. Tanh vs sigmoid results in faster learning, generally.

2.1.4 Method

I chose to implement a simple fully connected feedforward network with just 1 hidden layer and to stick to the given values from the param.txt file, modifying only the regularization factor and decay rate.

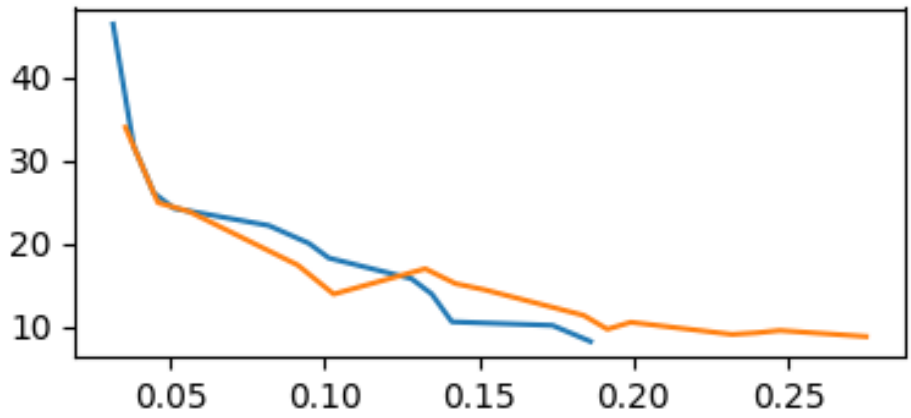
1. Independent variables
   These are the variables under investigation. For this experiment, there is only one, the activation function.
   (a) **Activation function**
   We observe what side effects result from changing activation function to each of the following:
   Relu function $f(x) = max(0, x)$
   Sigmoid function $f(x) = 1/(1 + exp(-x))$
   Softmax function $f(x) = e/sum(e), e = exp(x - amax(x))$
   Tanh function $f(x) = tanh(x)$
2. Controlled variables
   - Dataset: Iris
   - Input units: 4
   - Hidden units: 4
   - Output units: 3
   - Learning constant: 0.5
   - Momentum constant: 0.5
   - Error criterion: 0.02
   - Max epochs: 1000

- Regularization factor: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
- Decay rate: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]

3. Results

   (a) Figure 1: Sigmoid vs Softmax on Iris dataset

```
from shanepy import *

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

fig=plt.figure(figsize=(4,2))

data = np.genfromtxt("regularization-0-0-1000i-iris-sigmoid.time-vs-error.log",
delimiter=',', skip_header=0, skip_footer=0, names=['time', 'error'])

p = plt.plot(data['time'], data['error'])

data = np.genfromtxt("regularization-0-0-1000i-iris-softmax.time-vs-error.log",
delimiter=',', skip_header=0, skip_footer=0, names=['time', 'error'])

p = plt.plot(data['time'], data['error'])

fig.tight_layout()

fig.savefig('figures/regularization-0-0-1000i-iris-sigmoid-vs-softmax-time-vs-error.png')
return 'figures/regularization-0-0-1000i-iris-sigmoid-vs-softmax-time-vs-error.png'
```



2.1.5 Discussion

When regularization is enabled, the error immediately goes up to 75. This is clearly a bug in my NN.

It appears that systemic problems have caused the results to be unreliable. I suspect there is something wrong with my code.

Disabling regularization, I'm able to at least get two meaningful graphs from my experiment.

I am unable to come to any conclusinos.

I would have liked to have seen the vanishing gradient problem when testing sigmoid and tanh as the activation function for the hidden layer.

Best practice however says that Relu is a better than function for hidden layers.

2.1.6 Conclusions

The experiment was inconclusive.

2.2 Experiments 2 to 4

I have decided to abandon the remaining experiments but here they are for posterity.

2.2.1 Comparing softmax to tanh for the output layer in an encoder

1. Questions
   (a) Do we see evidence supporting any of these rules of thumb in our results?
      i. Softmax is important if you have multiple groups for classification.
      How does tanh compare to softmax as the activation function for the output layer?
   (b) For output layer, use softmax to get probabilities for possible outputs.
   (c) Is it possible to encode the desired behaviour of the training set?
2. Controlled variables
   - Dataset: Encoder
3. Method I need fewer hidden nodes than the input and output for the encoder.
4. Hypotheses
   (a) tanh is better suited to be the activation for the output layer of a generator network?

2.2.2 Comparing softmax to sigmoid with the Iris dataset

1. Hypothesis
   (a) Softmax is important if you have multiple groups for classification
2. Conclusions
   (a) Use softmax for both hidden and output.
   (b) Softmax should be used for hidden.

2.2.3 Comparing cost functions: Sum of squares vs Negative log likelihood

As we seek are the minimum of the cost function during training, selecting a cost function that is accurate is improtant.

1. Method

   The tasks I will test my network on are:

   (a) 3 bit parity :: 3:3:1 network

   (b) 4 bit parity :: 4:4:1 network

   (c) Encoder :: 3:3:8 network

   (d) XOR :: 2:2:1 network

   We set the population error function to be the sum of squares.

2. Possible discussion The shape of negative log-likelihood curves becomes steeper and more compressed as sample size increases, indicating greater certainty in our parameter estimate.

3. sum of squares The lowest sum of squares corresponds to the least error.

   One advantage to using negative log likelihoods is that we might have multiple observations, and we might want to find their joint probability

2.3 Future experiments

2.3.1 tan hidden vs output layer

1. Questions

   (a) Do we see evidence supporting any of these rules of thumb in our results?

      i. Hyperbolic tangent activation function is more suitable to the hidden layer than the output layer.

3 Appendix

3.1 Appendix 1 :: Using the neural network

3.1.1 Files

path	description
test.sh	An example shell invokation of the neural network for a one-off fit.
perceptron.py	The bulk of the code for the Neural Network.
mynumpy.py	Some supporting code for the Neural Network.
lab/	This is the directory where invocatinos of the NN should take place.
datasets/	This is the directory where data sets are stored.
lab/do-experiments.sh	This is the script where you can describe multiple experiments and gather the results.
lab/results	This is where the result logs appear.
lab/results/*.time-vs-error.log	These logs contain time,error tuples that can be used to make charts.
lab/results/*.1.log	standard out for each experiment run. these files are usually empty.
lab/results/*.2.log	standard error for each experiment run. this will contain a report of the experiment

1. Example of a report in lab/results/*.2.log

   - These files are comprised of
     (a) Command that was run
     (b) The parameters that were set
     (c) The population error every 100 epochs
     (d) The result of the criterion

```
python2 ../perceptron.py
-e1000 -l0.5 -d0 -a sigmoid
-t "../datasets/Iris"
-o results/learning-rate-0-0-1000i-iris-sigmoid.time_vs_error.log

Running with parameters:
------------------------
1000 max epochs
0.02 g_criterion
4 input units and 1 bias unit
4 hidden units
3 output units
learning rate = 0.5
regularization factor = 0.5
momentum = 0.5
decay rate = 0.0
Epoch 100, Population error 0.12
Epoch 200, Population error 0.12
Epoch 300, Population error 0.12
Epoch 400, Population error 0.12
Epoch 500, Population error 0.12
Epoch 600, Population error 0.12
```

```
Epoch 700, Population error 0.12
Epoch 800, Population error 0.12
Epoch 900, Population error 0.12
Epoch 1000, Population error 0.12
Input Weights
[[-0.00049395 -0.00049395 -0.00049395 -0.00049395]
 [-0.00064174 -0.00064174 -0.00064174 -0.00064174]
 [-0.00040358 -0.00040358 -0.00040358 -0.00040358]
 [-0.00034673 -0.00034673 -0.00034673 -0.00034673]
 [-0.00169323 -0.00169323 -0.00169323 -0.00169323]]
Output Weights
[[ 0.02201845 -0.04136443 -0.0920555 ]
 [ 0.02201845 -0.04136443 -0.0920555 ]
 [ 0.02201845 -0.04136443 -0.0920555 ]
 [ 0.02201845 -0.04136443 -0.0920555 ]]
Input activations
[ 0.667  0.459  0.627  0.584]
Hidden activations
[ 0.50004923  0.50004923  0.50004923  0.50004923]
Epoch 1000, Population error 0.12
0.122454076358 >= 0.02; Error criterion was not reached
```

3.2 Appendix 2 :: Vanishing gradient problem explained

If your input is on a higher side (where sigmoid goes flat) then the gradient will be near zero. This will cause very slow or no learning during backpropagation as weights will be updated with really small values.

3.3 Stochastic gradient descent algorithm for a 3-layer network

Pseudocode

```
  init network weights (often small random values)
  do
     forEach example named ex
        prediction = neural-net-output(network, ex)  // forward pass
        actual = teacher-output(ex)
        compute error (prediction - actual) at the output units

        // backward pass
        compute {\displaystyle \Delta w_{h}} \Delta w_h for all w from hidden to output
```

```
        // backward pass continued
        compute {\displaystyle \Delta w_{i}} \Delta w_i for all w from input to hidden
        update network weights // input layer not modified by error estimate
  until all examples classified correctly or another stopping criterion satisfied
  return the network
```

3.4 Appendix 2 :: Logistic and Softmax Regression

binary classification that y can take two values

3.5 Learning rate vs Momentum

We take steps proportional to the size of the gradient vector. The constant of proportionality is called the learning rate. momentum - the algorithm remembers its last step, and adds some proportion of it to the current step. This way, even if the algorithm is stuck in a flat region, or a small local minimum, it can get out and continue towards the true minimum.

When performing gradient descent, learning rate measures how much the current situation affects the next step, while momentum measures how much past steps affect the next step.

3.6 Delta Rule / Least Mean Square

Training is done by subtracting from each weight a small fraction of its corresponding derivative

3.7 Decay function

```
# Every epoch, do this
learn_rate = learn_rate * (learn_rate / (learn_rate + (learn_rate * decay_rate)))
```

3.8 Error functions

- See: ljvmiranda921.github.io

3.9 Delta rule and backpropagation

The core idea behind back-propagation is backward differentiation or reverse mode differentiation.

3.9.1 See:

How to understand the Delta in Back Propagation Algorithm - Quora

$dy/d(y) = 1 = dy/d[(x1 + x2) * x3]$

Previously we applied d/dx, now we apply dy/d.

This is only three inputs with two nodes but NNs usually consist of millions of inputs. Back-prop is "computationally cheap". One forward differentiation gives the derivative of one output with respect to "one" input but one backward differentiation can give you the derivative of one output with respect to all inputs including weights and biases.

3.9.2 Disambiguation

1. Delta Rule / Least Squares Criterion is not the same delta rule Best linear model has smallest value for D. $D = (d_1)^2 + (d_2)^2 ... (d_n)^2$ D is the sum of vertical diffs between points on the graph and the linear model.

3.10 [Neural] Transfer function

Transfer functions calculate a layers output from its net input.

3.10.1 Examples

1. tansig :: $n = 2/(1 + exp(-2 * n)) - 1$

3.10.2 Transfer function vs activation function

Two functions determine the way signals are processed by neurons; ie. Two functions determine a neurons signal processing.

Activation function determines the total signal a neuron receives. The value of the activation function is usually scalar and the arguments are vectors.

Output function o(I) operating on scalar activations and returning scalar values. Typically a squashing function is used to keep the output values within specified bounds.

These two functions together determine the values of the neuron outgoing signals.
The composition of the activation and the output function is called the transfer function o(I(x)).

3.10.3 Reference

Tansig (Neural Network Toolbox)

3.11 Activation functions

3.11.1 See:

1. Activation function - Wikipedia There is a great table describing various activation functions..
2. CS231n Convolutional Neural Networks for Visual Recognition

3.11.2 Sigmoid

Sigmoid function is a special case of the Logistic function.

1. Tanh Function — The Clever Machine The logistic sigmoid has a nice biological interpretation but it turns out that the logistic sigmoid can cause a neural network to get "stuck" during training in part because a strongly-negative input is provided to the logistic sigmoid, it outputs values very near zero.

3.11.3 tanh

1. There are two reasons for tanh instead of sigmoid (assuming you have normalized your data, and this is very important):

   (a) Having stronger gradients Since data is centered around 0, the derivatives are higher. To see this, calculate the derivative of the tanh function and notice that its range (output values) is [0,1]. The range of the tanh function is [-1,1] and that of the sigmoid function is [0,1]

   (b) Avoiding bias in the gradients
   This is explained very well in "Efficient Backprop by LeCun et al".

2. Tanh() in the output layer of generator network machine learning - use of Tanh() in the output layer of generator network - Stack Overflow

3.11.4 tan-sigmoid

The most exact and accurate prediction of neural networks is made using tan-sigmoid function for hidden layer neurons and purelin function for output layer neurons.I

3.11.5 relu

problem with ReLu is that some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. Simply saying that ReLu could result in Dead Neurons.

3.11.6 leaky relu

Leaky ReLu to fix the problem of dying neurons. It introduces a small slope to keep the updates alive.