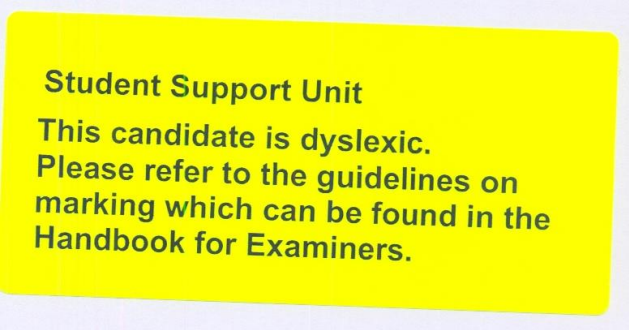
Student Number: 213123 & Kaggle Team Name: 213123



1.1. Introduction

To be able to classify between memorable and non-memorable images, I have created a feed-forward neural network with the use of the Keras libraries that run on top of TensorFlow. I have written three scripts of python code one that selects what data I want to use as a training set, another that shows the tuning of a model with a method of normalising the data set and lastly a script that can run the best performing model. One of the most significate issues to overcome was the fact that the dataset chosen was small and therefore prone to overfitting secondly was that there was a domain adaption issue which is difficult to resolve when given a supervised learning approach.

Please note that all the equations were converted to images once being saved.

2. Cleaning data

Before implementing a model, the data needs to be treated in a way that will increase the effectiveness of a chosen model. Initially, the priority was to balance the dataset, as known there is a domain adaption problem between the test set and the training set. Where the test set holds 61% false classifications and the training set is around 12.5% false when including the additional training data.

To balance the training set, I extracted all rows in the training set, and the additional data set which had a false classification. With the samples that held true classifications, I took all values from the training data and then supplemented with true samples from the additional training set to bring the balance to be equal to that test sets domain.

Downscaling the data in this way left just over 1000 samples for both training and validation, an option to get a more extensive dataset would be to upscale by adding copies of the falsely classified data and to then extract more true values from the additional data set. However, this would mean making the training set not as unique for samples with a false prediction and adding more samples with NAN values for samples with a true prediction.

To fill in the cells that hold a value of NAN, Pandas have a function called interpolate, which also performs extrapolation. To try to fit a best-case estimation to a cell which value is dependent on the adjacent cells. In this instance, however, only interpolation was implemented as extrapolation gave less desirable results for when preforming against the validation set. After cleaning the data set that which is left is small, with 847 training samples and 212 validation samples with an 80:20 split.

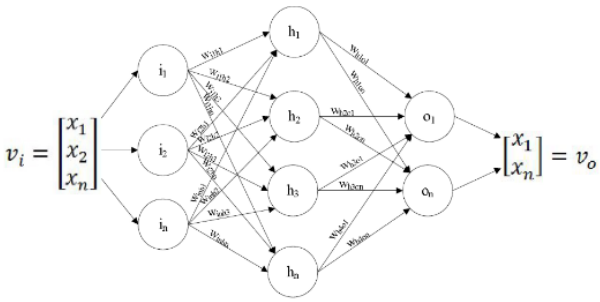
The preprocessing normalisation function by default uses L2 normalisation works on the rows of the dataset, unlike the standard scaler that works on the columns.

$$Z = \left(\frac{x - Min(x)}{\max(x) - \min(x)}\right)$$

This means that any value of z will be either in between or equal to zero or one. The purpose that is looking to be achieved by transforming the data in this way is to make "training less sensitive to the scale of features, so we can better solve for coefficients" [4]

2.1. Feed-forward pass

A feed-forward neural network consists of an input layer, output layer and a number of hidden layers in between. Any more than a single hidden layer is recognised as a deep neural network where the number of layers to use is dependent on the complexity of the problem that the network is attempting to solve.



Each layer of the network consists of weights and neurons, where the weight represents the strength of the connection between neurons. Therefore the higher the value of the weight, the more significant the magnitude is to the next neuron. As this network is fully connected each node in a layer has a connection to every node in the next layer. The input into a neuron can be described as.

$$z = b + \sum_{i=1}^{N} a_i w_i$$

Where b is a bias which ensures that the activation of a neuron cannot be zero, using Keras the bias does not initialise by default and are set on all hidden layers.

During the feed-forward pass, each layer will have an activation function. These activations allow us to fit non-linear data by adding curvature to the outputs of a node which allows solving for more complex problems. For the output layer, the sigmoid activation function has been employed. This will make any output to get infinitely close to the values of one or zero. Due to the exponential component, it is less likely for a value to output around 0.5. The activation equation where a3 represents the third activation and net3 being the output of the first hidden layer.

$$a3 = \frac{1}{1+e^{-net3}}$$

For the input and hidden layer, a ReLu activation is used where the output of the activation is clipped at zero. This, however, can cause a node to become inactive, a solution to this is too similar functions such as the Leaky ReLu where and a negative value will be multiplied by a small alpha parameter. From trial and error using Tanh, sigmoid, LeakyRelu, PReLu, Tanh, sigmoid and ELU, I found ReLu to be the most effective in terms of accuracy on the validation set.

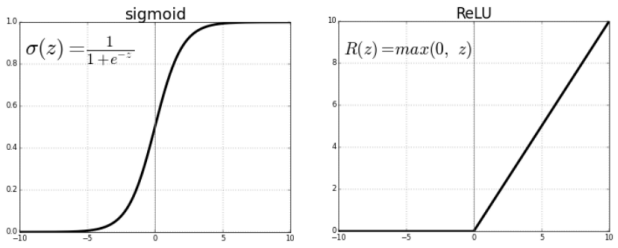


Fig: ReLU v/s Logistic Sigmoid

[1]

2.2. Error gradient and backpropagation

This is a technique that is used to adjust weights and biases in all layers of the neural network. This strengthens the effects by the relative weights and biases on the error gradient.

As there is one hidden layer in this neural network, there are two steps for backpropagation let the propagation from $net3$ back to $\hat{a}2 = delta3$ (top layer backpropagation) and from $\hat{a}2$ back to the input $X = delta2$ (lower layer backpropagation). As the top layer does not have an activation $delta3$ is not difficult to solve.

Error gradient looks at the change in error with respect to $W1$ and $W2$ respectively and can be shown as follows:

$\frac{\partial e}{\partial W_2}$ and $\frac{\partial e}{\partial W_1}$

Where the error e is equal to the target with the output subtracted.

$$(t - o)$$

However, for a normalised dataset, the sum of squared errors can be used instead (SSE), which is $(t-0)^2$.

For convenience the $\frac{1}{2}$ can be added which cancels out after subsequent differentiation, this gives.

$$e = \frac{1}{2}(t-0)^2$$

$$U = t - 0$$

$$e = \frac{1}{2}U^2$$

After applying the chain rule:

$$\frac{\partial e}{\partial W_2} = -(t-0)\frac{\partial 0}{\partial W_2}$$

Where

$$delta3 = -(t-0)$$

To calculate $\frac{\partial e}{\partial W_1}$ a similar approach is taken however as the output is dependant net3 then the derivative of the output with respect to $net3 = \frac{\partial o}{\partial net_3} = f'(net_3)$.

Once again solving with the chain rule, the result is given as:

$$\frac{\partial e}{\partial W_1} = -(t-0)\cdot O(1-0)\cdot W_2^T \cdot a_2(1-a_2)\cdot X^T$$

Where $delta3 = -(t-0)$ without activation in the top layer.

$$delta2 = W_2^T \cdot delta_3 \cdot a_2(1-a_2)$$

Therefor with each additional layer, the calculating delta can be done within a loop. With the error calculated, the weights can be adjusted, by subtracting the error from the weights the output of the network can learn to reduce the cost. Before this can be achieved, the error needs to be scaled by a defined hyperparameter which is the learning rate. The equations are as followed:

$$W_1 \leftarrow W_1 \alpha\left(\frac{\partial e}{\partial w_1}\right)$$

$$W_2 \leftarrow W_2 \alpha\left(\frac{\partial e}{\partial w_2}\right)$$

The learning rate a small number that is less than one. The reason being that if the learning rate is large, the global minimum can be overstepped. However, if the learning rate is too small, it could take many iterations to converge or become stuck in a local minimum.

Using the Adam optimiser a momentum and acceleration terms are added to the gradient descent, and the error is calculated dependent on the defined minibatch size, in this case, the update is performed every 32 samples.

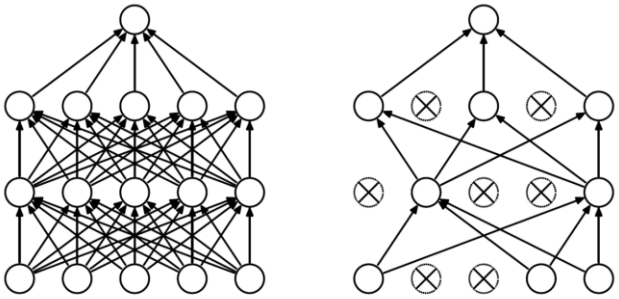
By default, Keras will initialise weights as Glorot uniform in the attempt to prevent the gradients from vanishing or exploding quickly by trying to keep the variance of the input of layer equal to the variance of the outputs of a layer.

$$\text{Xaveir Initialisation} = \sqrt{\frac{1}{number\ of\ inputs + number\ of\ targets}}$$

2.3. Regularizers

As the dataset is small, the neural network can become efficient at being able to solve the training set. However, this has made it so that the model does not perform well when generalising. Two methods have been applied to attempt to mitigate the overfitting of the neural network.

The first is known as drop out where the hyperparameter is a value between zero and one which determines the probability of a node being inactive. Furthermore, therefore meaning that an input vector will less likely become dependant totally on a single feature as it could randomly disappear. As the weights do not become dependent on single features, the weights shrink, and the values spread out. If given a high dropout of 0.5, this means that hidden layers are around half the size of what the optimiser has set them to be.



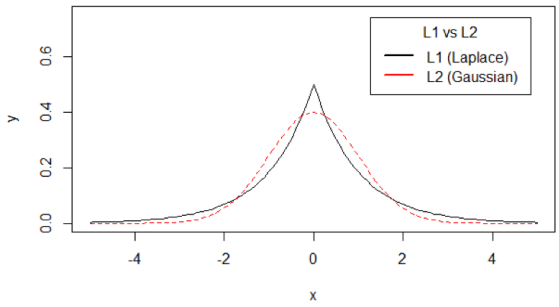
(a) Standard Neural Net (b) After applying dropout.

The second regulariser that I used was L2 regularisation, also known as ridge regression. Ridge regression takes the cost function and adds a regularisation term to it, which is highlighted in red below.

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n}\theta_i^2$$

Where λ is the regularisation parameter which can be any value between zero and one and represents the regularisation strength. And where n being the number of features and the sum of theta squared being the slope squared. Therefore the sum of the slope squared is being scaled by lambda values and added to the cost as a penalty. Because of this penalty the weights get driven closer to the origin by adding the regularisation value. [5]

The benefit of using L2 regularisation over L1 is that L1 peaks at zero as a Laplacian distribution rather than a Gaussian distribution.



[6]

A paper from Toronto university shows that combining regularisers has proven to work well on the MNIST data set with max-norm incorporated with dropout showing to have the most significant results.

Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

[3]

2.4. Optimisation

Neural networks are black boxes, and knowing exactly how to set the hyperparameters without trial and error is impossible. However, there are guideline rules like the layers in a neural network tend to do well if the number of nodes in a layer is two thirds that of the previous one. This is a crude method.

However, the Keras libraries contain several methods to optimise the hyperparameters that do not require the individual to follow the rule of thumb guidelines or set nested loops.

A standard method is known as a Naïve grid search, which will attempt to trial every possible combination of hyperparameters. The issue here is that the dimensionality multiplies were to have millions of combinations of neural networks from the range of hyperparameters is not unrealistic. The second method of optimisation would be a random search which will randomly select the values of the hyperparameters within the range and choices of which are previously defined. When a good solution is found, it will save as the best solution until another is found.

For this model, Bayesian optimisation has been selected as a means of tuning the neural network by using Bayes theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

By setting hyperparameters by hand, it becomes very easy to fail to notice things. Gaussian processes are very good at noticing trends in the data. This is using machine learning to take the task of figuring out what the hyperparameters should be by predicting the regions of the hyperparameter space in which we expect to get good results.

The way that this is achieved is by keeping track of the best hyperparameters that have worked well so far. After each trial, our best trial may be replaced by our new trial if it gives better performance, or it might stay the same. If a set of parameters performs very poorly this not absolutely a bad thing as something is learnt from this as Bayesian optimisation also has exploration and exploitation features.

References

[1] Medium. 2020. *Activation Functions In Neural Networks*. [online] Available at: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6> [Accessed 29 May 2020].

[2] Goodfellow, I., Bengio, Y. and Courville, A., 2017. *Deep Learning*. Cambridge, Mass: The MIT Press.

[3] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2020. *Dropout: A Simple Way To Prevent Neural Networks From Overfitting*. [online] Jmlr.org. Available at: <http://jmlr.org/papers/v15/srivastava14a.html> [Accessed 29 May 2020].

[4] Medium. 2020. *Standardise Or Normalise? — Examples In Python*. [online] Available at: <https://medium.com/@rrfd/standardize-or-normalize-examples-in-python-e3f174b65dfc> [Accessed 29 May 2020].

[5] Neukart, F., 2017. *Reverse Engineering The Mind*. Wiesbaden: Springer Fachmedien Wiesbaden.

[6] Jmlr.org. 2020. [online] Available at: <http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf> [Accessed 29 May 2020].