Durham University

Introduction to Neural Networks

Daniel Zack Brodtman

Supervisor: Marija Zamaklar

Introduction

Neural networks are a powerful and accessible application of machine learning, with uses ranging from driverless cars to medical diagnostics. They mimic neurons and synapses in the brain to learn the patterns underpinning a dataset, and use this to make predictions about new inputs. This poster introduces the training process for building a neural network, with an emphasis on the applications in image classification.

Outline of a Neural Network

Neural networks are organized into layers, and each layer is populated by neurons. Each neuron has a weight, w, and bias, b, and these are the *parameters* of the network. The goal of training the network is to find the parameters that describe the underlying relationships between the data.

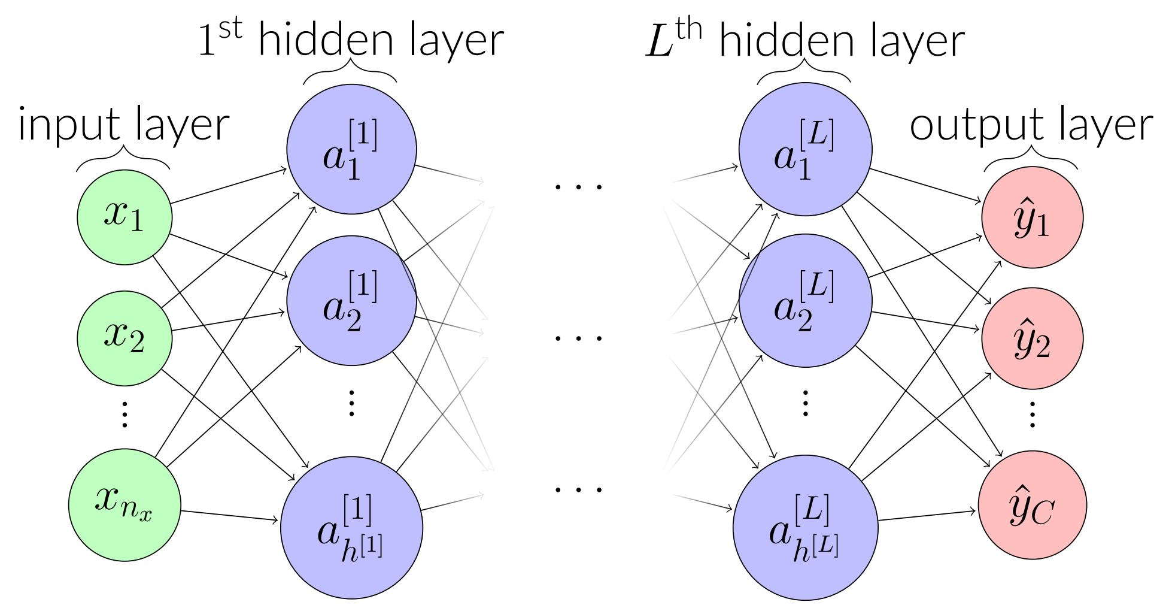


Figure: Network graph of a fully connected $(L+1)$-layer neural network with n_x input units and C output units. The l^{th} hidden layer contains $h^{[l]}$ hidden units.

Any choices we make about the network's architecture are called *hyperparameters*. This includes the number of layers or nodes per layer, the activation functions of these nodes and the learning rate.

Training a Neural Network

In general, the process of a neural network is as follows:

- An m-size training set is fed through a network with randomly initialized parameters. This is *forward propagation*.
- A prediction $\hat{y}$ is made and the difference between that and the true value, y, is calculated as the *loss*. A *cost function*, J, is calculated using the loss, e.g. mean square error

$$J_{MSE}(y^i,\hat{y}^i) = \frac{\sum_{i=1}^m (y^i - \hat{y}^i)^2}{m}$$

- New parameters are chosen by minimising the cost function with *gradient descent* (shown below). This is *backpropagation*.
- The network is evaluated using a validation data set before the hyperparameters are tweaked and the entire process repeated to improve performance.
- We can now input unseen data into the network and it will make a prediction about it.

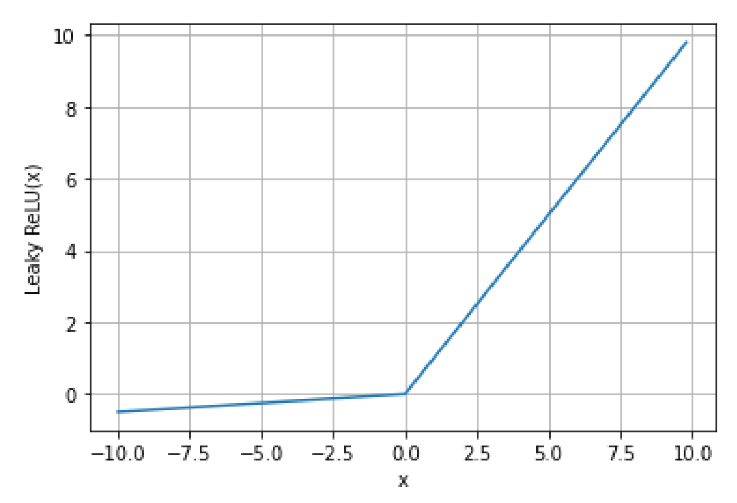
Figure: The red arrows demonstrate two instances of (simplified) gradient descent on the surface of the mountain. Realistically cost, J, is a million-dimensional function which is extremely difficult to minimize.

Forward Propagation

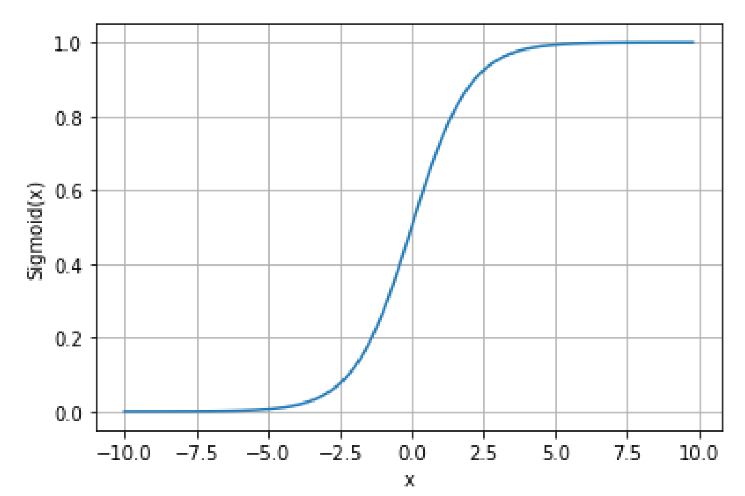
Forward propagation is the process by which data is passed through the network to produce an output. The data is fed through each node of a layer, $[l]$, where it is multiplied by a weight matrix and added to a bias before being passed through an *activation function*, $g(z)$, to give an *activation value*, a.

$$z_j^{[l]} = \sum_k^{n_x} W_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]} \quad \text{and} \quad a_j^{[l](i)} = g(z_j^{[l]})$$

The purpose of the activation function is to introduce nonlinearity into the system, because most complex data cannot be modelled by linear functions. Activation functions can take several forms depending on the context. For the hidden layers of a network, the most popular choice of activation function is the Leaky ReLU function because it has no flat areas which slow gradient descent (vanishing gradient problem). It is also cheap to compute and has a mean activation of approximately 0 which has been shown to speed up learning. A more complex function, the sigmoid function is useful as the output neuron's activation function in binary classification because it has a codomain of $(0,1)$ which can be interpreted as a probability of the label. We can generalise this to classification problems with more than two target labels using the Softmax function, $\sigma(z_i) = \frac{\exp(z_i)}{\sum_{c=1}^{C}\exp(z_c)}$. This outputs a vector of probabilities for each label (summing to 1).



(a) Leaky ReLu: $f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$



(b) Sigmoid: $f(x) = (1+e^{-x})^{-1}$

The activation value is passed as the input to nodes in the next layer. The process is repeated for all layers through the network and then an output is produced for each example in the training dataset. The reviewing of these outputs is backpropagation.

Backpropagation

Backpropagation is the process of moving backwards through the network to update the parameters (weights and bias). The idea is to find the parameters that minimise the cost function, which we can think of as a surface. The neural network calculates the gradient of the cost function with respect to the parameters of each node. We set a *learning rate*, α, multiply it by the gradient of our cost with respect to the parameter we are optimizing, and take this away from the parameter to edge closer to the minimum cost. We then forward propagate and backpropagate again for a chosen number of iterations. This works because $-\nabla J$ points in the direction of the nearest minimum of the surface, i.e. the location of the parameters that give the lowest cost.

$$w_{new} = w_{old} - \alpha\frac{\partial J(w,b)}{\partial w} \quad \text{and} \quad b_{new} = b_{old} - \alpha\frac{\partial J(w,b)}{\partial b}$$

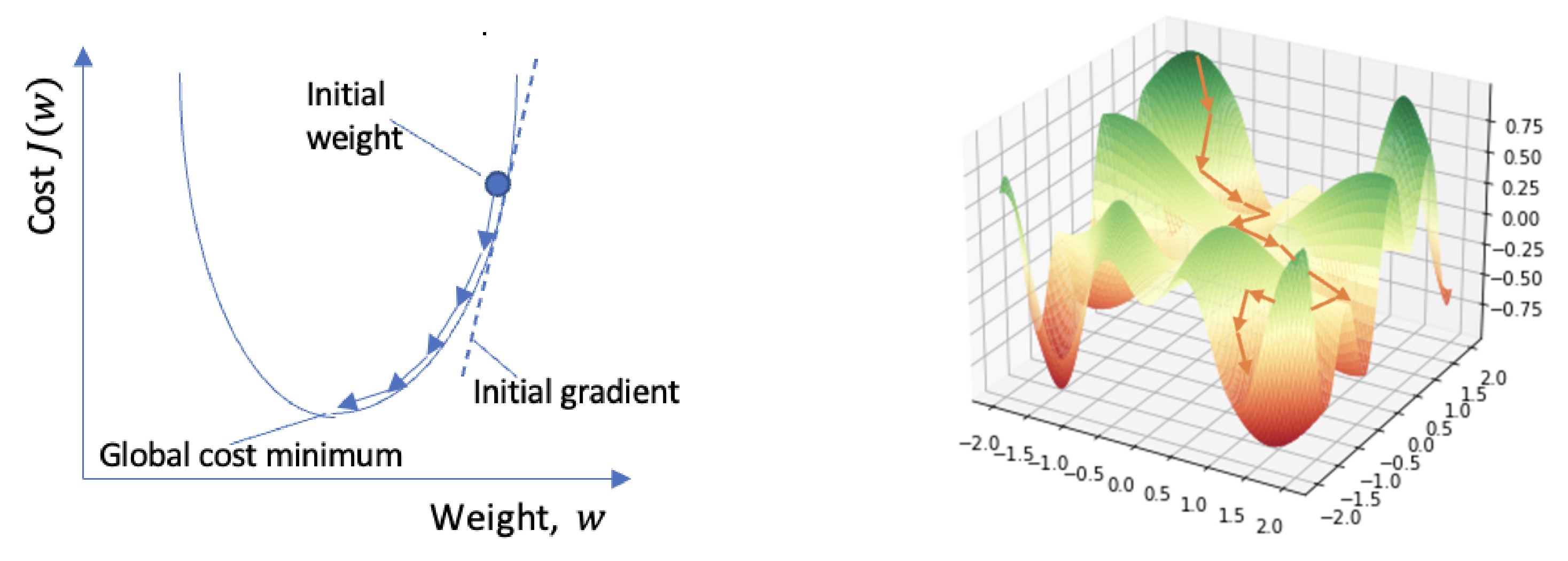


Figure: Gradient descent in one and many dimensions

We can speed up backpropagation with *optimization*. Examples include *learning rate decay*, where we reduce our learning rate as we get closer to our optimum parameters so we don't overshoot the minimum cost, or using *momentum* to take into account the previous parameter values to move in the direction of greatest slope towards the minimum.

Improving Network Performance

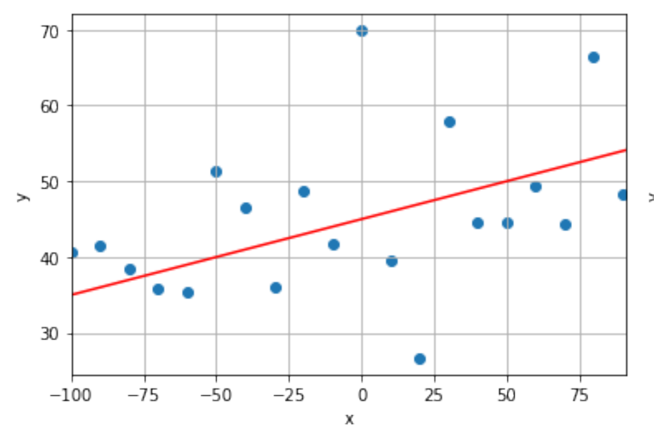
Improving neural networks is a balancing act between bias and variance. If the model is too complex it will generalise poorly to new data. On the other hand, simple models don't learn enough about the training data to give accurate predictions for new data. To find out which of these problems we have, we separate a small portion of our training data, called *validation data*, and run this through our network after each training iteration to see how the loss of the network is changing. Depending on the performance, we adjust our hyperparameters and repeat.

Bias

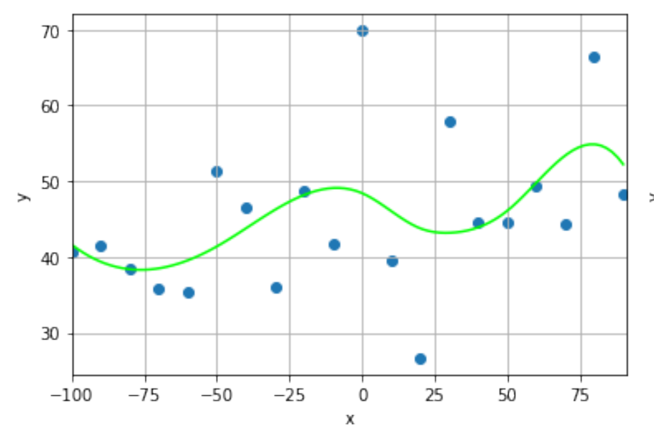
High bias or underfitting is when the network is too simple to grasp the distribution of the data. It has a low accuracy when it comes to using it on real-world examples. If validation loss is decreasing as we train the network then we know we have underfitting because we have not yet found the best model for the data. We need to make the model more complex by either iterating the training for longer or increasing the number of layers and nodes of the network.

Variance

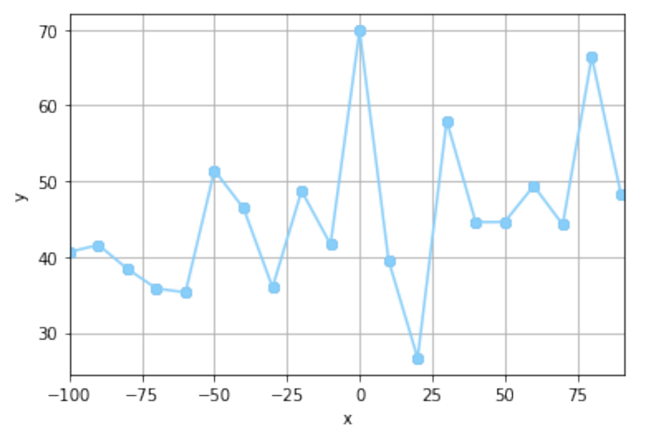
High variance or overfitting is when the network is too sensitive to the training data, and cannot generalise to real-world data well. If validation loss is increasing as we train the network then we know we have overfitting because the model is becoming too specific to the training data. We need to simplify our model and reduce the influence of training data outliers (noise) on our network by using more training data to reduce the amount of noise relative to the total amount of data, or *regularisation*. This can involve adding a penalty term, $\lambda f(w)$, to the cost function or using *dropout* to randomly turn off some nodes during each training iteration. These techniques reduce the likelihood of high weights which are a sign that the network has become too complex.



(a) High bias & low variance: Underfitting



(b) Medium variance & medium bias



(c) High variance & low bias: Overfitting

Future Research: Convolutional Neural Networks (CNNs)

As CCTV and social media become omnipresent in modern society, creating endless streams of visual data, the problem of image classification has become a principal driver in machine learning developments. CNNs use filters, the parameters of which are learnt like weights, to detect features such as edges or blur. These filters use the convolution operation $*$, where the matrix of pixels (which each have a brightness value on a color model such as RGB, CMYK, etc.) is convoluted with a filter matrix using the dot product.

$$\begin{pmatrix} 0 & 1 & 1 & 1_{\times 1} & 0_{\times 0} & 0_{\times 1} & 0 \\ 0 & 0 & 1 & 1_{\times 0} & 1_{\times 1} & 0_{\times 0} & 0 \\ 0 & 0 & 0 & 1_{\times 1} & 1_{\times 0} & 1_{\times 1} & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 3 & \boxed{4} & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{pmatrix}$$

CNNs also use pooling layers to reduce the dimensions of an image by splitting it up into regions and using the average or maximum brightness value of each region. Repeated use of convolution and pooling layers results in a much smaller data matrix which now only has the determining features of the image. This is then flattened into a vector which is passed through a series of fully connected layers. CNNs allow for training on high-resolution images with much less pre-processing than standard neural networks. This opens the door to exciting applications in computer vision such as object detection for self-driving cars and biometric authentication, and the best CNNs have been shown to outperform human experts in medical tasks [1].

References

[1] W Zhou, Y Yang, C Yu, J Liu, X Duan, Z Weng, D Chen, Q Liang, Q Fang, J Zhou, et al. Ensembled deep learning model outperforms human experts in diagnosing biliary atresia from sonographic gallbladder images. nat. commun. 12 (1), 1259 (2021), 2021.