E0 306: Deep Learning: Theory and Practice 17^{th} January 2019

Scribe Notes for Lecture 4

Introduction to Neural Networks

Lecturer: Navin Goyal *Scribe*: S Gaurang

1 Logistic Regression for Binary Classification

Logistic regression is a simple and elegant method commonly used for classification tasks, and not as a model to tackle regression problems. We will see how the model assigns a continuous, monotonic confidence level for its predictions of an instance belonging to some class, assuming that the values of different instances are independent of one-another.

It has seen widespread use, and is an important classification technique in its own right. It serves as a stepping stone in understanding Neural Networks, as heuristically it can be considered as a Neural Net with zero hidden layers.

To begin, for compactness, we fix some notation:
Denote the Training Set $\mathcal{S} = \left\{ \left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right) \right\}$

where $x^{(i)} \in \mathbb{R}^{\mathrm{d}}$, denoted by $x^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_{\mathrm{d}}^{(i)}\right)$, and $y^{(i)} \in \{0, 1\}$

1.1 Classification Function

Motivation: For the purpose of classification, we would like a model that gives a measure of its confidence for its prediction of a given instance belonging to a some class or label, rather than hard, discrete outputs of 0 or 1. To be precise, given such a Model M that assigns a conditional probability p_M for an outcome, given the instance, we can predict the label of a new data point.
Observe that by the Law of Total Probability: $p_M(y = 0 \mid x) \; + \; p_M(y = 1 \mid x) \; = 1$

We thus obtain the binary classifier, the Bayes-optimal Predictor by predicting:

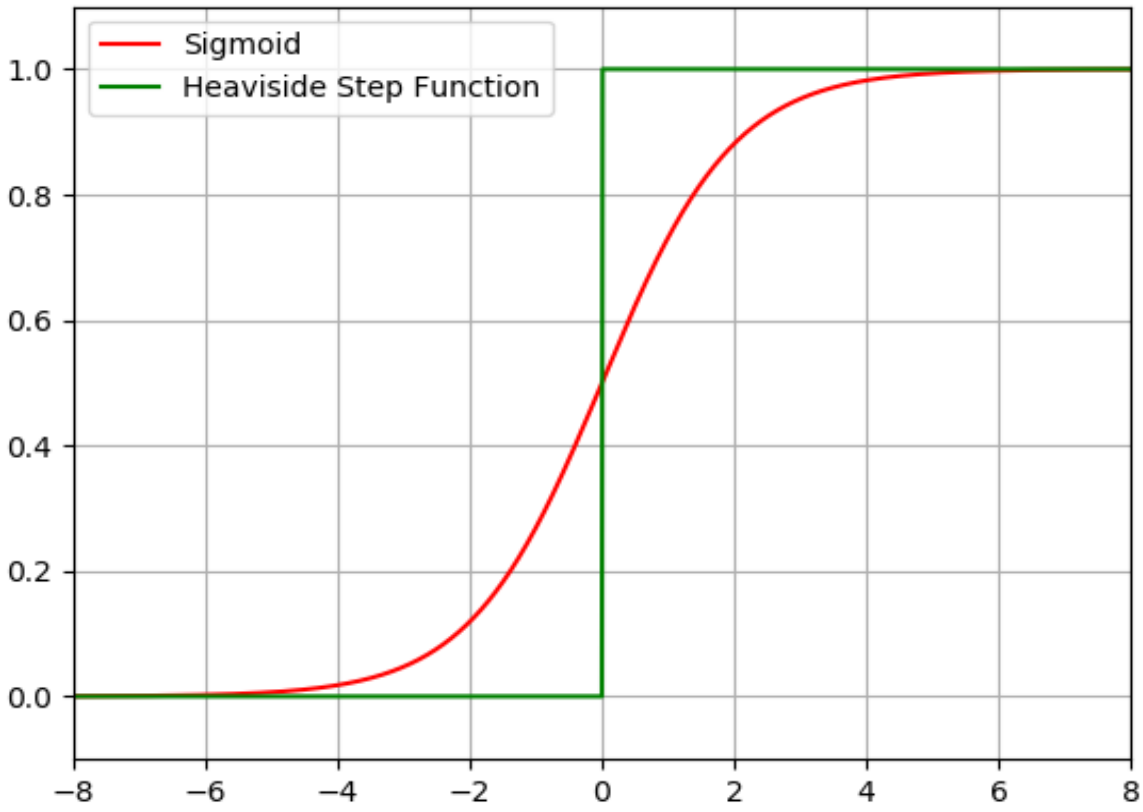
$$h(x) = \begin{cases} 1 & \text{if} \quad p_M(y = 1 \mid x) \; \geq \; 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Now, if restrict our model M to be linear, it is completely defined by the parameters (W, b): weights $W \in \mathbb{R}^{\mathrm{d}}$ and bias $b \in \mathbb{R}$.
We then can obtain a probability from the real-valued, linear function $\langle W, x \rangle + b$ by utilizing the sigmoid function $\sigma(t)$, used commonly to convert real numbers to probabilities, i.e. to real numbers between 0 and 1.

1.2 Sigmoid Function

The sigmoid function, also called the Logistic function, is defined as $\sigma(t) := \dfrac{1}{1+e^{-t}}$

Figure 1: Plot of the Sigmoid Function. Note that $\sigma(0) = 0.5$



Few Important Properties:

$$\sigma(t) = \frac{e^t}{1+e^t} = 1 - \frac{1}{1+e^t} = 1 - \sigma(-t)$$

$$\frac{d}{dt}\Big(\sigma(t)\Big) = \frac{e^{-t}}{(1+e^{-t})^2} = \left(\frac{1}{1+e^{-t}}\right)\left(\frac{e^{-t}}{1+e^{-t}}\right) = \sigma(t)\big(1-\sigma(t)\big)$$

The sigmoid function is a smooth (differentiable) analogue of the Heaviside step function, which is defined to be 1 on $\mathbb{R}^+$, and 0 for negative inputs.
Thus we have the probability density function $p_{W,b}(y = 1 \mid x) = \sigma(\langle W, x\rangle + b)$, and thus the binary classifier is given by:

$$h_{W,b}(x) = \begin{cases} 1 & \text{if } \ \sigma\big(\langle W, x\rangle + b\big) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Remark: Since the derivative $\sigma'(t) = \sigma(t)\big(1 - \sigma(t)\big)$ exists for all $t \in \mathbb{R}$, and is always positive as $0 < \sigma(t) < 1$, the sigmoid function is a strictly increasing (and thus injective) function.

1.3 Learnability

From our above remarks, $\sigma(t) \geq 0.5 \iff t \geq 0$, and thus we have that the hypothesis $h_{W,b} \geq 0.5 \iff \langle W, x \rangle + b \geq 0$.
Hence, our Hypothesis class $\mathcal{H} = \big\{h_{W,b} \mid w \in \mathbb{R}^{\mathrm{d}}, b \in \mathbb{R}\big\}$ is the same as the class of all Half-Spaces of $\mathbb{R}^{\mathrm{d}}$, as the final decision boundary is still a hyperplane in $\mathbb{R}^{\mathrm{d}}$.

The VC dimension of $\mathcal{H}$ i.e. class of Half-Spaces in $\mathbb{R}^{\mathrm{d}}$ is finite, and is d+1 (was discussed previously in Lecture 2).
Thus, from the Fundamental Theorem of Statistical Learning, $\mathcal{H}$ is agnostically PAC-learnable with at most $O\Big(\frac{VC(\mathcal{H}) + \log\frac{1}{\delta}}{\epsilon^2}\Big)$ samples, and also that the samples are ϵ-representative with probability at least $1 - \delta$, $L_D(h) - L_S(h) \leq \epsilon \ \forall h \in \mathcal{H}$
As a result, by the Empirical Risk Minimisation (ERM) Rule, the generalisation error is small if the number of samples, $m >> d$. This implies that it is sufficient to minimise the empirical loss $L_S(h)$, preferably over as a large a training set as possible in order to minimise the *test error* $L_D(h_{W,b})$.

Problem: Find $W \in \mathbb{R}^{\mathrm{d}}$ and $b \in \mathbb{R}$ such that it minimises empirical loss $L_S(h)$:

$$L_S(h_{W,b}) = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{1}\big(y^{(i)} \neq h_{W,b}(x^{(i)})\big)$$

Case 1: We first assume that the data is linearly separable, and hence that the problem is realisable, i.e, there actually exist $W_0 \in \mathbb{R}^{\mathrm{d}}$ and $b_0 \in \mathbb{R}$ such that the true label y of any data point $x \in \mathbb{R}^{\mathrm{d}}$ is given by $y = h_{W_0,b_0}(x)$.
Hence, in order to minimise the empirical loss on the Training Samples $\mathcal{S}$, ideally we want to choose (W, b) such that $\forall (x^{(i)}, y^{(i)}) \in \mathcal{S}$:

$$\sigma\big(\langle W, x^{(i)} \rangle + b\big) \geq 0.5 \ \text{ if } \ y^{(i)} = 1 \qquad \sigma\big(\langle W, x^{(i)} \rangle + b\big) < 0.5 \ \text{ if } \ y^{(i)} = 0$$

And as we have seen above, this is equivalent to finding (W, b) such that:

$$\langle W, x^{(i)} \rangle + b \geq 0 \ \text{ if } \ y^{(i)} = 1 \qquad \langle W, x^{(i)} \rangle + b < 0 \ \text{ if } \ y^{(i)} = 0$$

Thus, this is a linear programming problem in (W, b). There are efficient methods known for solving this problem, which have polynomial time complexity in the number of samples m, and the dimension d of the domain.

Case 2: Assume the problem is not realisable, that is there is no perfect linear classifier that achieves zero error, i.e, the data is not linearly separable.

This is truly a hard problem to solve, it is not efficiently solvable: the problem has been shown to be NP-hard. This is discussed in a little more detail in Section 6. Furthermore, even finding a linear classifier which approximately minimises the empirical error is a hard problem.

Possible Approaches: In practice, the data is often not linearly separable. Thus many formulations have been put forth, in an attempt to solve the problem, generally based on several heuristics. One such approach is to formulate a Least Mean Squares problem:

$$\underset{W,b}{\operatorname{argmin}} \sum_{i\in[m]} \left(\langle W, x^{(i)}\rangle + b - y^{(i)}\right)^2$$

Since it is a Convex Optimisation problem, as the underlying domain is convex and the squaring map is a convex function, it is efficiently solvable. However it solves a regression problem, and by imposing more stringent conditions on it, we may not get the desired output for a classification problem.
Instead, we can try yet another optimisation method to tackle the problem:

$$\underset{W,b}{\operatorname{argmin}} \sum_{i\in[m]} \left(\sigma\big(\langle W, x^{(i)}\rangle + b\big) - y^{(i)}\right)^2$$

This is no longer a convex optimisation problem, as the function we are attempting to minimise is no longer convex in the variables (W, b). This optimisation problem has been shown to be hard to solve computationally.

2 Maximum Likelihood Estimate and Cross-Entropy Loss

We explore yet another method to tackle the problem. We assume that the events $(x^{(i)}, y^{(i)})$ are independent and identically distributed. We now shift out attention to finding (W, b) such that the conditional probability assigned by the model to the observed label set given the data points in the domain, is maximised over the whole sample set $\mathcal{S}$.
Define the Likelihood Function :

$$\mathsf{Likelihood}(W, b, \mathcal{S}) = p_{W,b}\left(y^{(1)}|x^{(1)}\right) \times p_{W,b}\left(y^{(2)}|x^{(2)}\right) \times \cdots \times p_{W,b}\left(y^{(m)}|x^{(m)}\right)$$

Since it is easier to work with sums of terms rather their products, we wish to cast the problem in the more tractable setting, as in the former case. To do so, we utilise the logarithm function, leveraging the property $\log(ab) = \log(a) + \log(b)$, and that the $\log(t)$-function is differentiable at every point in its domain, with (positive) derivative $1/t$. Hence, the log-function is strictly increasing (and thus injective) function on $\mathbb{R}^+$. Hence, maximising a function $f(t)$ is equivalent to maximising the function $\log(f(t))$.
Thus, the problem can be re-formulated as finding (W, b) that maximises the log-likelihood function:

$$\mathsf{LogLikelihood}(W, b, \mathcal{S}) = \log\left(p_{W,b}(y^{(1)}|x^{(1)})p_{W,b}(y^{(2)}|x^{(2)})\dots p_{W,b}(y^{(m)}|x^{(m)})\right) = \sum_{i\in[m]} \log\left(p_{W,b}(y^{(i)}|x^{(i)})\right)$$

Motivating Cross-Entropy Loss: Consider an arbitrary training example $(x, y) \in \mathcal{S}$. We introduce a new notation for conciseness: denote the predicted probability that $y = 1$, given x, by $\hat{y} = \sigma\big(\langle W, x\rangle + b\big)$. This enables us to cast the conditional probability in a more useful form:

$$p_{W,b}(y|x) = \hat{y}^y(1-\hat{y})^{1-y} = \begin{cases} \hat{y} & \text{if} \quad y = 1 \\ 1-\hat{y} & \text{if} \quad y = 0 \end{cases}$$

Taking the logarithm on both sides, we obtain the log-likelihood for this training example:

$$\log\Big(p_{W,b}(y|x)\Big) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$

Moreover, maximising the log-likelihood is the same as minimising the negative log-likelihood, which is called the Cross Entropy Loss for this example:

$$L_{CE}\big(W, b; (x, y)\big) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Cross Entropy Loss: The Cross Entropy Loss in general, is defined on a set of training examples $\mathcal{S}$ as follows:

$$L_{CE}\big(W, b; \mathcal{S}\big) = \sum_{i\in[m]} L_{CE}\big(W, b; (x, y)\big) = \sum_{i\in[m]} -y^{(i)}\log\big(\sigma\big(\langle W, x^{(i)}\rangle + b\big)\big) - (1-y^{(i)})\log\big(1-\sigma\big(\langle W, x^{(i)}\rangle + b\big)\big)$$

Hence, we see that to maximise the likelihood function, it is equivalent to finding (W, b) such that it minimises the Cross Entropy Loss on the samples $\mathcal{S}$.
This is now a tractable problem, as it is a continuous optimisation problem of a convex function. Minimisation of convex functions can be tackled using methods like Gradient Descent or Newton's method (will be discussed later on in the course). Logistic regression thus uses Cross-Entropy Loss as a surrogate-loss function.

Remark: In general, a convex function need not have a minimum: the Exponential function is one such example. Also in general, the global minimiser may not exist, and we may have to be content with approximate solutions.

3 Logistic Regression for Multiclass

We can extend the method of Logistic regression for binary classification to the setting where multiple classes are present, and is also known as Multinomial Regression.
Consider the case where the labels can take values from k-classes $\{1, 2, \ldots, k\}$. Thus our training examples $\mathcal{S}$ is now of the form:
$\mathcal{S} = \left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$ where $x^{(i)} \in \mathbb{R}^{\mathrm{d}}$, and $y^{(i)} \in \left\{1, 2, \ldots, k\right\}$
Approach: We use multiple linear functions, k of them to exact, one for each class, to obtain a k-dimensional vector for each example.

$$\big(f_1(x), f_2(x), \ldots, f_k(x)\big) = \big(\langle W_1, x\rangle + b_1, \langle W_2, x\rangle + b_2, \ldots, \langle W_k, x\rangle + b_k\big)$$

where each $W_i \in \mathbb{R}^{\mathrm{d}}$ and $b_i \in \mathbb{R}$. To be concise, denote $W = (W_1, W_2, \ldots W_k)$ and $b = (b_1, b_2, \ldots, b_k)$.
We encounter the same situation as before with a minor modification: we wish to convert this vector into a probability function. To do so, we utilize the Softmax function.

$$\mathsf{Softmax}(t_1, t_2, \ldots, t_k) = \left(\frac{e^{t_1}}{\sum_{i\in[k]} e^{t_i}}, \frac{e^{t_2}}{\sum_{i\in[k]} e^{t_i}}, \ldots, \frac{e^{t_k}}{\sum_{i\in[k]} e^{t_i}}\right)$$

Observe that each coordinate in the output of the Softmax function is a positive number between 0 and 1, such that the sum of all the coordinates is 1. Thus we can think of this vector to represent the probabilities that the model assigns for the labels amongst the classes $1, 2, \ldots k$. Thus, our conditional probability assigned by the model for class c is :

$$p_{W,b}(y = c|x) = \frac{e^{t_c}}{\sum_{i\in[k]} e^{t_i}}$$

Just as how the sigmoid function was chosen because it was a smooth approximation to the Heaviside Step function for binary classification, the Softmax function is the smooth analogue to the non-differentiable function $\text{argmax}(t_1, t_2, \ldots, t_k)$.
In this setting, the Likelihood function is of the form:

$$\mathsf{Likelihood}(W, b, \mathcal{S}) = p_{W,b}\left(y^{(1)}|x^{(1)}\right) p_{W,b}\left(y^{(2)}|x^{(2)}\right) \ldots p_{W,b}\left(y^{(m)}|x^{(m)}\right)$$

Consider a single training example, $\mathcal{S} = \{(x, y)\}$. Then we have:

$$p_{W,b}(y|x) = p_{W,b}(1|x)^{\mathbf{1}(y=1)} \times p_{W,b}(2|x)^{\mathbf{1}(y=2)} \times \cdots \times p_{W,b}(k|x)^{\mathbf{1}(y=k)}$$

Hence, the Cross Entropy Loss on this training example is given by:

$$L_{CE}\big(W, b; (x, y)\big) = -\log\big(p_{W,b}(y|x)\big) = -\sum_{i\in[k]} \boldsymbol{1}\left(y = i\right) \log\left(\frac{e^{\left\langle W_i, x\right\rangle + b_i}}{\sum_{j\in[k]} e^{\left\langle W_j, x\right\rangle + b_j}}\right)$$

4 Introduction to Feedforward Neural Networks

Motivation: A Neural Network can be thought of as being obtained by stacking layers of logistic regression classifiers, such that the output of one classifier is used in the input of one or more classifiers of the next layer.
At present, we restrict ourselves to fully-connected neural networks. They are the simplest kind of neural networks, and for conciseness, they are also denoted as NN.
Formally, a Neural Network is function from $\mathbb{R}^{\mathrm{d}}$ to $\mathbb{R}^{\mathrm{k}}$.

4.1 Two-Layer Neural Network

We begin with the simplest NN, a two-layer NN, with one hidden-layer and one output layer. These are also called 1-hidden layer NN's.

Let the number of hidden neurons be m. Hence, denote the input as $x \in \mathbb{R}^{\mathrm{d}}$, and let the weights between the input and hidden layer be $W_r \in \mathbb{R}^{\mathrm{d}}$ and biases $b_r \in \mathbb{R}$ for $r \in [m]$. Then the output of the neural network is given by $f = (f_1, f_2, \ldots, f_k)$, where for $i \in [k]$:

$$f_i(x) = \sum_{r \in [m]} a_{ir} \rho\Big(\langle W_r, x \rangle + b_r\Big)$$

Here, a_{ir} represents the weight between the output of the r^{th} hidden neuron and the input of the i^{th} output neuron
$\rho(t) : \mathbb{R} \to \mathbb{R}$ is called the activation function and it is a non-linear function, which is usually differentiable (required for backpropagation).

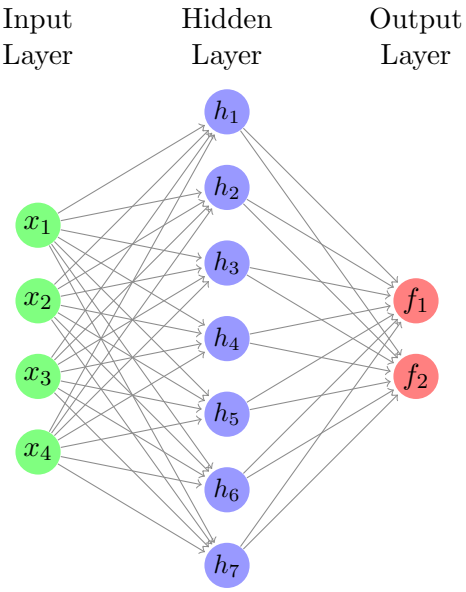
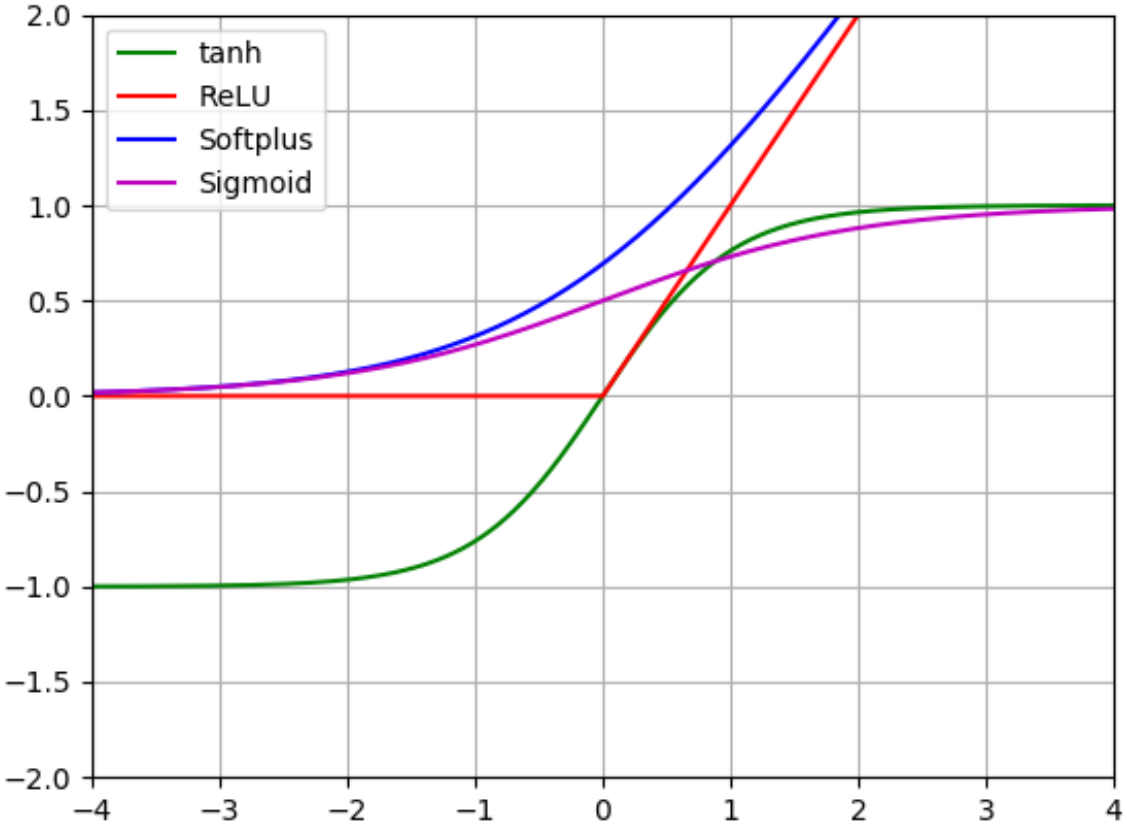


Figure 2: This is an example of a fully-connected, feedforward two-layer Neural Network, with $d = 4$ input neurons, $m = 7$ hidden neurons and $k = 2$ output neurons.

The activation function plays a pivotal role, as without this non-linearity, the expressive power of NN's is dramatically reduced and the NN would essentially become a linear function from $\mathbb{R}^d$ to $\mathbb{R}^k$. It can be shown that with a non-linear activation function, an NN with a single hidden layer is a universal function approximator.
Some standard activation functions used in practice include the sigmoid function, hyperbolic tangent, arctangent, softplus and the Rectified Linear Unit or ReLU.
The ReLU function is not differentiable at the origin, but for backpropagation, it is generally defined to be 1 or 0. Even with this correction, ReLU does not have a continuous derivative. However, it has been found that NN's which use ReLU tend to learn better with lower generalisation error, and several variants of the ReLU function like PReLU(parametrised linear unit), SeLU(scaled exponential linear unit) are in active use today.

Figure 3: Plots of Commonly Used Activation Functions



We can also succinctly describe the two-layer neural network using Matrix notation:

$$f(x) = A\rho\Big(Wx + B\Big)$$

where ρ is applied element-wise $\rho(v_1, v_2, \ldots, v_d) = \big(\rho(v_1), \rho(v_2), \ldots, \rho(v_d)\big)$ and where W, B and A represent the matrices:

$$W = \begin{bmatrix} --- W_1 --- \\ --- W_2 --- \\ \cdots \cdots \cdots \cdots \cdots \\ \cdots \cdots \cdots \cdots \cdots \\ --- W_m --- \end{bmatrix} \quad B = \begin{bmatrix} --- b_1 --- \\ --- b_2 --- \\ \cdots \cdots \cdots \cdots \cdots \\ \cdots \cdots \cdots \cdots \cdots \\ --- b_m --- \end{bmatrix} \quad A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ a_{k1} & a_{12} & \ldots & a_{km} \end{bmatrix}$$

Another modification commonly seen in practice is the application of a sigmoid or Softmax function on the linear output layer, so that the output represents probabilities instead of real numbers.

4.2 Neural Network With Multiple Hidden Layers

Now we consider an NN with l layers: one output layer and $l-1$ hidden layers. Suppose there are d^i neurons in the i^{th} layer of the NN (which are stacked on the input layer of d neurons). Hence, denote the weights and biases of each layer of the NN as:

$$W^{(1)} \in \mathbb{R}^{d_1 \times d} \ , \ W^{(2)} \in \mathbb{R}^{d_2 \times d_1} \ , \ \ldots \ , \ W^{(l-1)} \in \mathbb{R}^{d_l \times d_{l-1}} \ , \ W^{(l)} \in \mathbb{R}^{k \times d_l} \ ,$$

$$b^{(1)} \in \mathbb{R}^{d_1} \ , \ b^{(2)} \in \mathbb{R}^{d_2} \ , \ \ldots \ , \ b^{(l)} \in \mathbb{R}^{d_l} \ ,$$

Then the k-dimensional vector output of this Neural Network is as follows:

$$f(x) = W^{(l)} \rho\Big(\ldots \rho\Big(W^{(2)} \rho\Big(W^{(1)} x + b^{(1)} \Big) + b^{(2)} \Big) \ldots \Big) + b^{(l)}$$

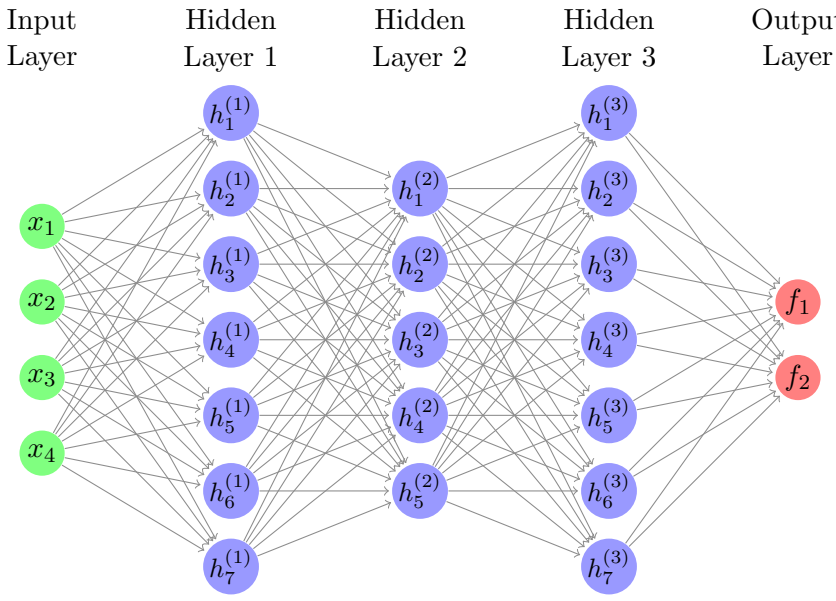


Figure 4: This is an example of a fully-connected, feedforward Neural Network with $l = 4$ layers, with $d = 4$ input neurons and $k = 2$ output neurons.

5 VC-Dimension of Neural Networks

We need all the weights and biases of a Neural Network to specify it completely. Each weight matrix of size (d_i, d_j) has $d_i \times d_j$ number of parameters, and each bias vector in $\mathbb{R}^{d_i}$ has d_i number of parameters. Thus the total number of parameters for a NN of l layers is given by: $P = (d_1 \times d) + (d_2 \times d_1) + \cdots + (d_k \times d_l) + d_1 + d_2 + \ldots d_l$
The VC-dimension for such class of NN (assuming a binary output for the NN, using ReLU activation function) is quite well controlled:

$$VC\big(\mathcal{H}\big) = O(Pl \log(Pl))$$

Hence, if the number of samples m is much larger than $VC\big(\mathcal{H}\big)$, we apply ERM, and finally expect a small generalisation error for the trained Neural Network.

6 Difficulty of Training Neural Networks

Here are presented a small collection of results regarding the computational difficulty for Empirical Risk Minimisation for NN's, and in general it is seen that the problem is hard when the network is not over-parameterised.

1. Blum & Rivest [3]: They considered a two -layer NN, with only two neurons in the hidden layer, two output neurons and an n-dimensional input layer. The NN used the Threshold function for activation, defined to be 1 if $\sum_{i\in[n]} a_i x_i + b_i > 0$, and 0 otherwise.
   They showed that the problem of fitting the NN to the training set of size $O(n)$ is NP-hard.

2. Bartlett & David [2]: They considered that in a similar setting as [3], but with k-hidden neurons. It was shown that for a realisable sample set $\mathcal{S}$, training an NN that makes an error on at most $\frac{C}{k^2}$ examples(C being any constant), is NP-hard.

3. L.K. Jones [5], Van H. Vu [8]: They proved that for the case where the NN uses the sigmoid function as activation, and the number of hidden layers was allowed to be as large as $O(poly(n))$, with a linear output layer with positive weights, the problem of minimising the square loss within an additive error of $4k^{-3}$ is NP-Hard.

4. Boob *et al.* [4], Bakshi *et al.* [1], Manurangsi & Reichman [6]: A two-layer NN with ReLU activation was considered, having two hidden units, and linear or ReLU output. It was shown that minimising the square loss is NP-Hard.

5. Jiří Sima [7]: It was shown that given $\epsilon > 0$ and a training set $\mathcal{S}$, the problem of finding fitting a single sigmoid unit (i.e. finding parameters (W,b)) so as to have at most ϵ-square loss is an NP-Hard.

Even with these results, it is still possible that with a larger NN, it might become easier to fit well to the data. Moreover, these results do not address the case when the NN has more than one hidden layer, or what happens when the positivity constraints are dropped.
In the next class, we will see that it is easy to fit an NN to data, if the the network is over-parameterised, that is, one can efficiently solve Empirical Risk Minimisation.

References

[1] Ainesh Bakshi, Rajesh Jayaram, and David P. Woodruff. Learning two layer rectified neural networks in polynomial time. *CoRR*, abs/1811.01885, 2018.

[2] Peter L. Bartlett and Shai Ben-David. Hardness results for neural network approximation problems. *Theoretical Computer Science*, 284(1):53 – 66, 2002. Computing Learining Theory.

[3] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117 – 127, 1992.

[4] Digvijay Boob, Santanu S. Dey, and Guanghui Lan. Complexity of training relu neural network. *CoRR*, abs/1809.10787, 2018.

[5] L. K. Jones. The computational intractability of training sigmoidal neural networks. *IEEE Transactions on Information Theory*, 43(1):167–173, Jan 1997.

[6] Pasin Manurangsi and Daniel Reichman. The computational complexity of training relu(s). *CoRR*, abs/1810.04207, 2018.

[7] Jiří Sima. Training a single sigmoidal neuron is hard. *Neural Comput.*, 14(11):2709–2728, November 2002.

[8] Van H. Vu. On the infeasibility of training neural networks with small squared errors. In *Proceedings of the 10th International Conference on Neural Information Processing Systems*, NIPS'97, pages 371–377, Cambridge, MA, USA, 1997. MIT Press.