Machine Learning and Neural Networks (MATH3431) Epiphany term, 2023

Draft Handout 5: Artificial neural networks

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce the Artificial neural network as a model and procedure in classical and Bayesian framework. Motivation, set-up, description, computation, implementation, tricks. We focus on the Feedforward network.

Reading list & references:

(1) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
- Ch. 20 Neural Networks

(2) Bishop, C. M., & Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: springer.
- Ch. 5 Neural Networks

(3) Neural Networks for Pattern Recognition
- Ch. 4 The multi-layer perceptron

(4) LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (2002). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9-50). Berlin, Heidelberg: Springer Berlin Heidelberg.

1. Intro and motivation [1]

Note 1. Artificial Neural Networks (NN) are statistical models which have mostly been developed from the algorithmic perspective of machine learning. They were originally created as an attempt to model the act of thinking by modeling neurons in a brain. In ML, NN are used as global approximators.

Note 2. The original biological motivation for feed-forward NN stems from McCulloch & Pitts (1943) who published a seminal model of a NN as a binary thresholding device in discrete time, i.e.

$$n_j(t) = 1\left(\sum_{\forall i \to j} w_{j,i} n_i(t-1) > \theta_i\right)$$

where the sum is over neuron i connected to neuron j; $n_j(t)$ is the output of neuron i at time t and $0 < w_{j,i} < 1$ are attenuation weights. Thus the effect is to threshold a weighted sum of the inputs at value θ_i. Perhaps, such a mathematical model involving compositions of interconnected non-linear functions could be able to mimic human's learning mechanism and be implemented in a computing environment (with faster computational abilities) with purpose to discover patters, make predictions, cluster, classify, etc...

[1] In this Section, formulas not needed to be memorized.

Remark 3. Mathematically, NN are rooted in the classical theorem by Kolmogorov stating (informally) that every continuous function $h(\cdot)$ on $[0,1]^d$ can be written as

$$h(x) = \sum_{i=1}^{2d+1} F_i \left(\sum_{j=1}^{d} G_{i,j}(x_j) \right) \tag{1.1}$$

where $\{G_{i,j}\}$ and $\{F_i\}$ are continuous functions whose form depends on f. Perhaps, one may speculate that functions $\{G_{i,j}\}$ and $\{F_i\}$ can be approximated by sigmoids or threshold functions of the form $\sigma\left(w^\top x\right)$ allowing the number of the tunable coefficients w to be high enough such that they can represent any function -hence the property of NN as global approximators.

Example 4. Consider a regression problem with predictive rule $h : \mathbb{R}^d \to \mathbb{R}^q$, suitable for cases where the examples (data) consist of input $x \in \mathbb{R}^d$, and output targets $y \in \mathbb{R}^q$. A 2 layer artificial neural network is

$$h_k(x) = \sigma_{(2)} \left(w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j} \sigma_{(1)} \left(w_{(1),j,0} + \sum_{\forall i} w_{(1),j,i} x_i \right) \right)$$

for $k = 1, ..., q$. One may choose with $\sigma_2(\alpha) = \alpha$, $\sigma_1(\alpha) = 1/(1 + \exp(-\alpha))$. The only left here is to learn unknown parameters $\left\{w_{(\cdot),\cdot,\cdot}\right\}$.

2. Feedforward Neural Network (Mathematical set-up)

Note 5. An (Artificial) Neural Network (NN) can be depicted as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes $\mathcal{V}$ correspond to neurons and edges $\mathcal{E}$ correspond to links between them.

Note 6. A Feed-Forward Neural Network (FFNN) or else multi-layer perceptron is a special case of NN which can be depicted by a directed acyclic graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. [2] (See Figure 2.1).

[2](its vertices can be numbered so that all connections go from one vertex to another with a higher number)

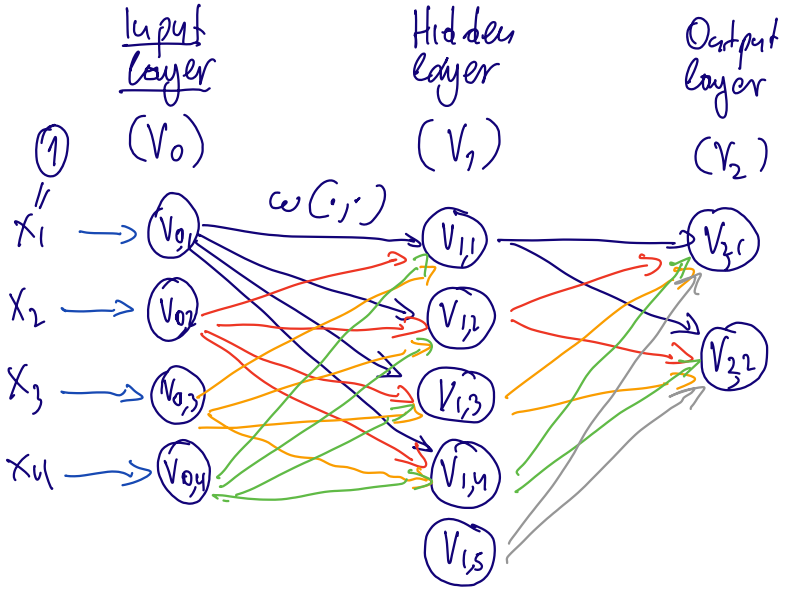

FIGURE 2.1. Feed forward neural network (1 hidden layer)

Note 7. We assume that the network is organized in layers. The set of nodes is decomposed into a union of (nonempty) disjoint subsets

$$V = \cup_{t=0}^{T} V_t$$

such that every edge in $\mathcal{E}$ connects a node from V_t to a node from V_{t+1}, for $t = 1, ..., T$.

Note 8. The first layer V_0 is called **input layer**. If x has d dimensions, then the first layer V_0 contains d nodes.

Note 9. The last layer V_T is called **output layer**. If y has q dimensions, then the last layer V_T contains q nodes.

Note 10. The intermediate layers $\{V_1, ..., V_{T-1}\}$ are called **hidden layers.**

Note 11. In a neural network, the nodes of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ correspond to neurons.

Notation 12. The **i-th neuron of the t-th layer** is denoted as $v_{t,i}$.

Note 13. The output of neuron i in the input layer V_0 is simply x_i that is $o_{0,i}(x) = x_i$ for $i = \{1, ..., d\}$.

Note 14. Each edge in the graph $(v_{t,j}, v_{t+1,i})$ links the output of some neuron $v_{t,j}$ to the input of another neuron $v_{t+1,i}$; i.e. $(v_{t,j}, v_{t+1,i}) \in \mathcal{E}$.

Note 15. We define a weight function $w : \mathcal{E} \to \mathbb{R}$ over the edges $\mathcal{E}$.

Note 16. Activation of neuron i at hidden layer 1 is the weighted sum of the outputs $o_{0,i}(x) = x_i$ of the neurons in V_0 which are connected to $v_{1,i}$ where weighting is according to function w, that is

$$\alpha_{1,i}(x) = \sum_{\forall j:(v_{0,j},v_{1,i})\in\mathcal{E}} w((v_{0,j},v_{1,i}))\, x_i \tag{2.1}$$

Note 17. Each single neuron $v_{t,i}$ is modeled as a simple scalar function, $\sigma_t(\cdot) : \mathbb{R} \to \mathbb{R}$, called **activation function** at layer t.

Note 18. We denote by $o_{t,i}(x) := \sigma_{t-1}(\alpha_{t-1,i}(x))$ **the output of neuron** $v_{t,i}$ when the network is fed with the input x.

Remark 19. Activation of neuron i at layer t is the weighted sum of the outputs $o_{t-1,j}(x)$ of the neurons in V_{t-1} which are connected to $v_{t,i}$ where weighting is according to function w, that is

$$\alpha_{t,i}(x) = \sum_{\forall j:(v_{t-1,j},v_{t,i})\in\mathcal{E}} w((v_{t-1,j},v_{t,i}))\, o_{t-1,j}(x) \tag{2.2}$$

Note 20. The input of a neuron is obtained by taking a weighted sum of the outputs of all the neurons connected to it, where the weighting is according to w.

Note 21. The input to $v_{t+1,i}$ is activation $\alpha_{t+1,i}(x)$ namely a weighted sum of the outputs $o_{t,j}(x)$ of the neurons in V_t which are connected to $v_{t+1,i}$, where weighting is according to w. The output of $v_{t+1,i}$ is the application of the activation function $\sigma_{t+1}(\cdot)$ on its input $\alpha_{t+1,i}(x)$. –That's why it is called **Feed forward Neural Network**.

Summary 22. The feed-forward NN formula in a layer by layer manner is performed (defined) according to the following recursion.

At $t = 0$**,:** for $i = 1, ..., |V_0|$

$$o_{0,i}(x) := x_i$$

At $t = 0, ..., T-1$**,:** for $i = 1, ..., |V_{t+1}|$

$$\alpha_{t+1,i}(x) = \sum_{\forall j:(v_{t,j},v_{t+1,i})\in\mathcal{E}} w((v_{t,j},v_{t+1,i}))\, o_{t,j}(x)$$

$$o_{t+1,i}(x) = \sigma_{t+1}(\alpha_{t+1,i}(x))$$

Note 23. Depth of the NN is the number of the layers T.

Note 24. Size of the network is the number $|V|$.

Note 25. Width of the NN is the number $\max_{\forall t}(|V_t|)$.

Note 26. The architecture of the neural network is defined by the triplet $(\mathcal{V}, \mathcal{E}, \sigma_t)$.

Note 27. The neural network can be fully specified by the quadruplet $(\mathcal{V}, \mathcal{E}, \sigma_t, w)$.

Example 28. Figure 2.1 denotes a NN with depth 2, size 11, width 5. The neuro with no incoming edges has$o_{1,5} = \sigma(0)$.

Notation 29. To easy the notation, we denote the weights as $w_{(t),i,j} := w\left((v_{t,j}, v_{t+1,i})\right)$. Using this notation, $w_{(t),i,j} = 0$ is equivalent in (2.2) to $(v_{t,j}, v_{t+1,i}) \notin \mathcal{E}$ and means that the link $v_{t,j} \to v_{t+1,i}$ is not in the network.

Note 30. Often a **constant neuron** $v_{t,0}$ (at each layer t and $i = 0$) which outputs 1; i.e. $o_{0,0}(x) = 1$ and $o_{t,0}(x) = 1$. The corresponding weight $w_{(t),k,0}$ is called **bias**. This resembles to the constant term in the linear regression.

Example 31. (Cont. Example 4) The 2 layer neural network

$$h_k(x) = \sigma_{(2)}\left(w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j}\sigma_{(1)}\left(w_{(1),j,0} + \sum_{\forall i} w_{(1),j,i}x_i\right)\right) \tag{2.3}$$

can be written according to the recursion in Summary 22 as

- Input layer
$$o_{(0),i}(x) = \begin{cases} 1 & i = 0 \\ x_i & i = 1, ..., d \end{cases}$$
- Hidden layer
$$\begin{aligned} \alpha_{(1),j}(x) &= w_{(1),j,0} + \sum_{\forall j} w_{(1),j,i}x_i \\ o_{(1),j}(x) &= \sigma_{(1)}\left(\alpha_{(1),j}(x)\right) \\ &= \sigma_{(1)}\left(w_{(1),j,0} + \sum_{\forall j} w_{(1),j,i}x_i\right) \end{aligned}$$
- Output layer
$$\begin{aligned} \alpha_{(2),k}(x) &= w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j}o_{(2),k}(x) \\ o_{(2),k}(x) &= \sigma_{(2)}\left(\alpha_{(2),k}(x)\right) \\ &= \sigma_{(2)}\left(w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j}o_{(2),k}(x)\right) \end{aligned}$$

If $h_k(x)$ returns values in $\mathbb{R}$, we can choose $\sigma_{(2)}$ as the identity function i.e., $\sigma_{(2)}(\alpha) = \alpha$. Note that (2.3) is also presented more compact

$$h_k(x) = \sigma_{(2)}\left(\sum_{\forall j} w_{(2),k,j}\sigma_{(1)}\left(\sum_{\forall i} w_{(1),j,i}x_i\right)\right)$$

by considering the constant neuron associated with first x which is set equal to 1, e.g $x_1 = 1$, similar to the linear regression models.

Note 32. Activation functions σ_t are non-increasing functions often sigmoids or threshold functions. Their choice is problem-dependent. some examples

- Identity function: $\sigma(\alpha) = \alpha$ cannot be used in hidden layers
- Threshold sigmoid: $\sigma(\alpha) = 1(\alpha > 0)$
- Logistic sigmoid: $\sigma(\alpha) = \frac{1}{1+\exp(-\alpha)}$
- Rectified linear unit: $\text{RELU}(\alpha) = \max(\alpha, 0)$

Example 33. Examples on the choice of the activation function at the output layer T:

- In the univariate regression problem with prediction rule $h(\cdot) \in \mathbb{R}$, and examples $z_i = (x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, we can choose $\sigma_T(\alpha) = \alpha$ to get $h(\alpha) = \sigma_T(\alpha) = \alpha$.
- In the binary logistic regression problem with prediction rule $h(\cdot) \in [0,1]$ and examples $z_i = (x_i, y_i) \in \mathbb{R}^d \times \{0,1\}$, we can choose $\sigma_T(\alpha) = \frac{1}{1+\exp(-\alpha)}$ to get $h(\alpha) = \sigma_T(\alpha) = \frac{1}{1+\exp(-\alpha)} = \frac{\exp(\alpha)}{1+\exp(\alpha)}$.

3. Learning Neural Networks

Note 34. Assume we are interested in a prediction rule $h_{\mathcal{V},\mathcal{E},\sigma,w} : \mathbb{R}^{|V_0|} \to \mathbb{R}^{|V_T|}$ which is modeled as a feed-forward Neural Network with $(\mathcal{V}, \mathcal{E}, \sigma, w)$; that is

$$h_{\mathcal{V},\mathcal{E},\sigma,w}(x) = o_T(x)$$

where $o_T = \left(o_{T,1}, ..., o_{T,|V_T|}\right)^\top$ is according to the summary 22.

Note 35. Learning the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of a neural network is a model selection task (similar to the variable selection in linear regression).

Note 36. We assume that the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of the neural network $(\mathcal{V}, \mathcal{E}, \sigma, w)$ is fixed (given), and that there is interest in learning the weight function $w : \mathcal{E} \to \mathbb{R}$ or equivalently in vector form the vector of weights $\left\{w_{(t),i,j}\right\}$ where $w_{(t),i,j} := w((v_{t,j}, v_{t+1,i}))$.

Note 37. The class of hypotheses is

$$\mathcal{H}_{\mathcal{V},\mathcal{E},\sigma} = \{h_{\mathcal{V},\mathcal{E},\sigma,w} : \text{ for all } w : \mathcal{E} \to \mathbb{R}\}$$

for given $(\mathcal{V}, \mathcal{E}, \sigma)$.

Notation 38. To simplify notation we will use h_w instead of $h_{\mathcal{V},\mathcal{E},\sigma,w}$.

Note 39. Assume that there is available a training set of examples (data-set) $\mathcal{S} = \{z_i = (x_i, y_i); i = 1, ..., n\}$ with $x_i \in \mathcal{X} = \mathbb{R}^{|V_0|}$ and $y_i \in \mathcal{Y}$.

Note 40. To learn the unknown w, we need to specify a loss function $\ell(w, z)$ at some value of weight vector $w \in \mathbb{R}^{|\mathcal{E}|}$ and at some example $z = (x, y)$.

Example 41. For instance, loss function can be

(1) $\ell(w, z) = \frac{1}{2}\|h_w(x) - y\|_2^2$ based on a norm
(2) $\ell(w, z) = -\log(f(y|w))$ based on a pdf/pmf of the sampling distribution $f(y|w)$ of the target y given the weight vector w; eg in a regression problem $y \sim \text{N}\left(h_w(x), \sigma^2\right)$.

Definition 42. Error function is a performance measure that can be defined as

$$\text{EF}\left(w|\left\{z_i^*\right\}\right)=\sum_{i=1}^{n}\ell\left(w,z_i^*\right) \tag{3.1}$$

where $\mathcal{S}^*=\left\{z_i^*=\left(x_i^*,y_i^*\right);i=1,...,n^*\right\}$ is a set of examples which does not necessarily need to be the training set $\mathcal{S}$.

Example 43. Following we present some paradigms of NN implemented in applications seen before and with respect to quantities Notes 32, 34, 40, 42

Case 1. (Regression problem) Assume we wish to predict the mapping $x\overset{h(\cdot)}{\to}y$, where $x\in\mathbb{R}^d$ and $y\in\mathbb{R}$. Consider a predictive rule $h_w:\mathbb{R}^d\to\mathbb{R}$. Assume a training data-set $\left\{z_i=\left(x_i,y_i\right)\right\}$.

- The output activation function is the identity function $\sigma_T\left(a\right)=a$. This is because $h_w\left(x\right)=o_T\left(x\right)=\sigma_T\left(\alpha_T\left(x\right)\right)$ by Summary 22 and Note 34. Since h_w returns in $\mathbb{R}$ and the output activation $\alpha_T\left(x\right)$ returns in $\mathbb{R}$, then mapping $\sigma_T\left(a\right)=a$ suffices.
- A suitable loss can be

$$\ell\left(w,z=\left(x,y\right)\right)=\frac{1}{2}\left(h_w\left(x\right)-y\right)^2$$

Alternatively, if I consider a statistical model as

$$y|x,w\sim\text{N}\left(h_w\left(x\right),\beta^{-1}\right)$$

for some fixed $\beta>0$, the loss can be

$$\begin{aligned}\ell\left(w,z=\left(x,y\right)\right)=&-\log\left(\text{N}\left(y|h_w\left(x\right),\beta^{-1}\right)\right)\\=&-\left(-\frac{1}{2}\log\left(\frac{1}{2\pi}\right)-\frac{1}{2}\log\left(\beta^{-1}\right)-\frac{1}{2}\frac{\left(h_w\left(x\right)-y\right)^2}{\beta^{-1}}\right)\\=&\frac{1}{2}\beta\left(h_w\left(x\right)-y\right)^2+\text{const...}\end{aligned}$$

- The Error function is

$$\text{EF}\left(w|z\right)=\sum_{i=1}^{n}\ell\left(w,z_i=\left(x_i,y_i\right)\right)=\frac{1}{2}\beta\sum_{i=1}^{n}\left(h_w\left(x_i\right)-y_i\right)^2$$

Case 2. (Multi-output regression problem) Assume we wish to predict the mapping $x\overset{h(\cdot)}{\to}y$, where $x\in\mathbb{R}^d$ and $y\in\mathbb{R}^q$. Consider a predictive rule $h_w:\mathbb{R}^d\to\mathbb{R}^q$. Assume a training data-set $\left\{z_i=\left(x_i,y_i\right)\right\}$.

- The output activation function is the identity function $\sigma_T\left(a\right)=a$. This is because $h_{w,k}\left(x\right)=o_{T,k}\left(x\right)=\sigma_T\left(\alpha_{T,k}\left(x\right)\right)$ for $k=1,...,q$ by Summary 22 and Note 34. Since h_w returns in $\mathbb{R}^q$ and the output activation is $\alpha_T\left(x\right)$ in $\mathbb{R}$, then mapping $\sigma_T\left(a\right)=a$ suffices.

- A suitable loss can be

$$\ell\left(w, z=(x,y)\right)=\frac{1}{2}\left\|h_w\left(x\right)-y\right\|_2^2=\frac{1}{2}\sum_{k=1}^{q}\left(h_{w,k}\left(x\right)-y_k\right)^2$$

Alternatively, if I consider a statistical model as

$$y|x,w\sim \mathrm{N}\left(h_w\left(x\right),\Sigma\right)$$

for some fixed $\Sigma>0$, the loss can be

$$\begin{aligned}\ell\left(w, z=(x,y)\right)=&-\log\left(\mathrm{N}\left(y|h_w\left(x\right),\Sigma\right)\right)\\ =&-\left(-\frac{q}{2}\log\left(\frac{1}{2\pi}\right)-\frac{1}{2}\log\left(\left|\Sigma\right|\right)-\frac{1}{2}\left(h_w\left(x\right)-y\right)^{\top}\Sigma^{-1}\left(h_w\left(x\right)-y\right)\right)\\ =&\frac{1}{2}\left(h_w\left(x\right)-y\right)^{\top}\Sigma^{-1}\left(h_w\left(x\right)-y\right)+\text{const...}\end{aligned}$$

- The Error function is

$$\begin{aligned}\mathrm{EF}\left(w|z\right)=&\sum_{i=1}^{n}\ell\left(w,z_i=(x_i,y_i)\right)\\ =&\frac{1}{2}\sum_{i=1}^{n}\left(h_{w,k}\left(x_i\right)-y_{k,i}\right)^{\top}\Sigma^{-1}\left(h_{w,k}\left(x\right)-y_{k,i}\right)\end{aligned}$$

where $y_{k,i}$ is the k-th dimension of the i-th example in the dataset.

Case 3. (Binary classification problem) Assume we wish to classify objects with features $x\in\mathbb{R}^d$ in 2 categories. Consider a predictive rule $h_w:\mathbb{R}^d\to(0,1)$ as a classification probability i.e, $h_w\left(x\right)=\Pr\left(x \text{ belongs to class 1}\right)$. Assume a training data-set $\{z_i=(x_i,y_i)\}$ with $y_i\in\{0,...,q\}$ labeling the class.

- A suitable output activation function can be the logistic sigmoid

$$\sigma_T\left(a\right)=\frac{1}{1+\exp\left(-a\right)}$$

This is because $h_w\left(x\right)=o_T\left(x\right)=\sigma_T\left(\alpha_T\left(x\right)\right)$ by Summary 22 and Note 34. Since h_w returns in $(0,1)$ and the output activation is $\alpha_T\left(x\right)$ in $\mathbb{R}$, then the aforesaid mapping suffices.

- If I consider a statistical model as

$$y_i|x_i,w\sim\text{Bernoulli}\left(h_w\left(x_i\right)\right)$$

with mass function

$$f\left(y|x,w\right)=h_w\left(x\right)^y\left(1-h_w\left(x\right)\right)^{1-y}$$

the loss can be

$$\begin{aligned}\ell\left(w, z=(x,y)\right)=&-\log\left(\text{Bernoulli}\left(y_i|h_w\left(x\right)\right)\right)\\ =&-\left(y\log\left(h_w\left(x\right)\right)+\left(1-y\right)\left(1-\log\left(h_w\left(x\right)\right)\right)\right)\end{aligned}$$

- The Error function given a set of examples $\{z_i^* = (x_i^*, y_i^*)\}$ is

$$\begin{aligned}\mathrm{EF}\left(w|z^*\right) &= \sum_{i=1}^{n} \ell\left(w, z_i^* = (x_i^*, y_i^*)\right) \\ &= -\sum_{i=1}^{n} y_i^* \log\left(h_w\left(x_i^*\right)\right) - \left(1 - y_i^*\right)\left(1 - \log\left(h_w\left(x_i^*\right)\right)\right)\end{aligned}$$

Case 4. (Multi-class classification problem) Assume we wish to classify objects with features $x \in \mathbb{R}^d$ in q categories. Consider a predictive rule $h_w : \mathbb{R}^d \to \mathcal{P}$, with $\mathcal{P} = \left\{\varpi \in (0,1)^q : \sum_{j=1}^{q} \varpi_j = 1\right\}$ and $h_w = \left(h_{w,1}, ..., h_{w,q}\right)^\top$, as a classification probability i.e, $h_{w,k}\left(x\right) = \Pr\left(x \text{ belongs to class } k\right)$. Assume a training data-set $\{z_i = (x_i, y_i)\}$ with $y_i \in \{0, 1\}$ labeling the class. Then it is

- A suitable output activation function can be the softmax function

$$\sigma_T\left(a_k\right) = \frac{\exp\left(a_k\right)}{\sum_{k'=1}^{q} \exp\left(a_{k'}\right)}, \text{ for } k = 1, ..., q \tag{3.2}$$

  This is because $h_{w,k}\left(x\right) = o_{T,k}\left(x\right) = \sigma_{T,k}\left(\alpha_{T,k}\left(x\right)\right)$ by Summary 22 and Note 34. Since h_w is essentially a probability vector and the output activation is $\alpha_{T,k}\left(x\right)$ in $\mathbb{R}$ the aforesaid mapping suffices. Unfortunately, 3.2 is invariant to additive transformations $a \leftarrow a + c$ i.e, $\sigma_T\left(a_k\right) = \sigma_T\left(a_k + \text{const}\right)$.
- Another suitable output activation function can be

$$\tilde{\sigma}_T\left(a_k\right) = \frac{\exp\left(a_k\right)}{1 + \sum_{k'=1}^{q-1} \exp\left(a_{k'}\right)}, \text{ for } k = 1, ..., q-1 \tag{3.3}$$

  This is because $h_{w,k}\left(x\right) = o_{T,k}\left(x\right) = \sigma_{T,k}\left(\alpha_{T,k}\left(x\right)\right)$ by Summary 22 and Note 34. Since h_w is essentially a probability vector and the output activation is $\alpha_{T,k}\left(x\right)$ in $\mathbb{R}$ the aforesaid mapping suffices as $h_{w,k}\left(x\right) = o_{T,k}\left(x\right)$ for $k = 1, ..., q-1$ and $h_{w,q}\left(x\right) = 1 - \sum_{k=1}^{q-1} h_{w,k}\left(x\right)$. The advantage of (3.3) compared to (3.2) is that the former uses one less output neurons (less unknown parameters to learn). Also (3.3) is not invariant to additive transformations $a \leftarrow a + c$ unlike 3.2i.e, $\tilde{\sigma}_T\left(a_k\right) \neq \tilde{\sigma}_T\left(a_k + \text{const}\right)$.
- If I consider a statistical model as

$$y|x, w \sim \text{Multinomial}\left(h_w\left(x\right)\right)$$

  with mass function

$$f\left(y|x, w\right) = \prod_{k=1}^{q} h_{w,k}\left(x\right)^{y_k}$$

  the loss can be

$$\begin{aligned}\ell\left(w, z = (x, y)\right) &= -\log\left(\text{Multinomial}\left(y|h_w\left(x\right)\right)\right) \\ &= -\sum_{k=1}^{q} y_k \log\left(h_{w,k}\left(x\right)\right)\end{aligned}$$

- The Error function given a set of examples $\{z_i^* = (x_i^*, y_i^*)\}$ is

$$\begin{aligned} \mathrm{EF}\left(w|z^*\right) &= \sum_{i=1}^{n} \ell\left(w, z_i^* = (x_i^*, y_i^*)\right) \\ &= -\sum_{i=1}^{n}\sum_{k=1}^{q} y_{k,i} \log\left(h_{w,k}\left(x_i\right)\right) \end{aligned}$$

where $y_{k,i}$ is the k-th dimension of the i-th example in the dataset.

4. Classical learning of Neural Network

Note 44. Our purpose is to find optimal $h_w \in \mathcal{H}_{\mathcal{V},\mathcal{E},\sigma}$ under loss $\ell(\cdot,\cdot)$ and against training data-set $\mathcal{S} = \{z_i = (x_i, y_i); i = 1, ..., n\}$.

4.1. **Standard.**

Essentially, this is an optimization problem, where the objective is to minimize either the risk function $R_g(w)$ or the empirical risk function $\hat{R}_S(w)$.

Problem 45. Compute optimal $w^* \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the risk function $R_g(w)$

$$w^* = \arg\min_{w\in\mathcal{H}} \left(R_g(w)\right) = \arg\min_{w\in\mathcal{H}} \left(\mathrm{E}_{z\sim g}\left(\ell(w,z)\right)\right) \tag{4.1}$$

Problem 46. Compute optimal $w^* \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the empirical risk function $\hat{R}_S(w)$

$$w^* = \arg\min_{w\in\mathcal{H}} \left(\hat{R}_S(w)\right) = \arg\min_{w\in\mathcal{H}} \left(\frac{1}{n}\sum_{i=1}^{n} \ell(w, z_i)\right) \tag{4.2}$$

4.2. **Shrinkage.**

Note 47. Often neural network models are over-parameterized, in the sense that the dimensionality of $w \in \mathbb{R}^{|\mathcal{E}|}$ is too large, with negative consequences in its predictability. This can be addressed by learning the architecture NN, precisely by learning which of the edges of the FFNN significantly contribute to the NN model and should be kept, and which do not contribute and may be removed (or be inactive).

Note 48. To address Note 47, one can resort to shrinkage methods e.g., LASSO, Ridge, which indirectly allow edge/weight selection/elimination by shrinking the values of the weights $\left\{w_{(t),i,j}\right\}$ towards zero, and setting some of them as $w_{(t),i,j} = 0$ if their absolute value is small enough. This is based on the observation in (2.2) i.e., $w_{(t),i,j} = 0$ is equivalent to $(v_{t,j}, v_{t+1,i}) \notin \mathcal{E}$ which implies that the link $v_{t,j} \to v_{t+1,i}$ is not active (essentially not in the neural network).

Problem 49. Find $h_w \in \mathcal{H}_{\mathcal{V},\mathcal{E},\sigma}$ (essentially compute $w \in \mathbb{R}^{|\mathcal{E}|}$) under loss $\ell(\cdot,\cdot)$ and shrinkage term $J(w;\lambda)$, and against training data-set $\mathcal{S} = \{z_i = (x_i, y_i); i = 1, ..., n\}$. Compute $w^* \in \mathbb{R}^{|\mathcal{E}|}$, according to the minimization:

$$w^* = \arg\min_{w\in\mathcal{H}} \left(R_g(w) + J(w;\lambda)\right) \tag{4.3}$$

$$= \arg\min_{w\in\mathcal{H}} \left(\mathrm{E}_{z\sim g}\left(\ell(w,z) + J(w;\lambda)\right)\right) \tag{4.4}$$

and set $w^*_{t,i,j} = 0$ if $w^*_{t,i,j}$ is less than a threshold user specific value $\xi > 0$ i.e. $\left|w^*_{t,i,j}\right| < \xi$.

Problem 50. Popular shrinkage terms $J(w;\lambda)$ are

- Ridge: $J(w;\lambda) = \lambda \|w\|_1$ Term 1
- LASSO: $J(w;\lambda) = \lambda \|w\|_2^2$ Term 1
- Elastic net: $J(w;\lambda = (\lambda_1, \lambda_2)) = \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$

4.3. **Stochastic gradient implementation.**

Note 51. Training a neural network model is usually a high-dimensional problem (essentially w has high dimensionality to provide a better approximation) and a big-data problem (essentially we need a large number of training examples to learn a large number of weights). For this reason, Stochastic Gradient Descent (and its variations) is a suitable stochastic learning algorithm.

Note 52. To address Problem 45, the recursion of the SGD with batch size m is

$$w^{(t+1)} = w^{(t)} - \eta_t \frac{1}{m} \sum_{j=1}^{m} \partial_w \ell\left(w^{(t)}, z_j^{(t)}\right)$$

Note 53. To address the Problem 49, the recursion of the SGD with batch size m is

$$w^{(t+1)} = w^{(t)} - \eta_t \left[\frac{1}{m} \sum_{j=1}^{m} \partial_w \ell\left(w^{(t)}, z_j^{(t)}\right) + \partial_w J(w;\lambda)\right]$$

for some positive λ which is user specified, or chosen via cross validation.

Note 54. The learning problem associated to the Neural network model is (almost always) non-convex due to the non-convex loss with respect to the w's. Upon implementing SGD in the learning problem of Neural Network, the theoretical results in Section 4 (Handout 2) and Section 4 (Handout 3) will not be effective due to the violation of the assumptions.

Note 55. Practical guidelines for the use of SGD in the learning problem of NN:

(1) Utilize a learning rate η_t that changes over the iterations. The choice of the sequence η_t is more significant. In practice, it is tuned by a trial and error manner: given a validation data-set $\mathcal{S}^*$ you may perform cross validation based on Error Function (3.1).
(2) Re-run the SGD procedure several times and by using different settings (learning rate η_t, batch size m) and different seed $w^{(0)}$ (randomly choosen) each time. Possibly, by luck, at some trial, we will initialize the SGD process with a random seed $w^{(0)}$ producing a trace leading to a good local minimum w^*.
(3) The output w^*_{SGD} returned by SGD is the best discovered w tested by using a performance measure (Error Function) using a validation set $\mathcal{S}^* = \{z_i^* = (x_i^*, y_i^*)\, ; i = 1, ..., n^*\}$; Eg.

$$w^*_{\text{SGD}} = \arg\min_{w^{(t)}} \left(\text{EF}\left(w^{(t)} | \{z_i^*\}\right)\right).$$

5. Error backpropagation (to compute $\nabla_w \ell(w, z)$)

Appendix A. About graphs

Definition 56. A directed graph is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprising

- a set of nodes $\mathcal{V}$ (or vertices), where a nodes are abstract objects, and
- a set of edges $\mathcal{E} = \{(v, u) \,|\, v \in \mathcal{E}, u \in \mathcal{E}, v \neq u\}$ (or directed edges, directed links, arrows) which are ordered pairs of vertices (that is, an edge is associated with two distinct vertices).

Definition 57. An edge-weighted graph or a network is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ equipped with a weight function $w : \mathcal{E} \to \mathbb{R}$ that assigns a number (the weight) to each edge $e \in \mathcal{E}$.

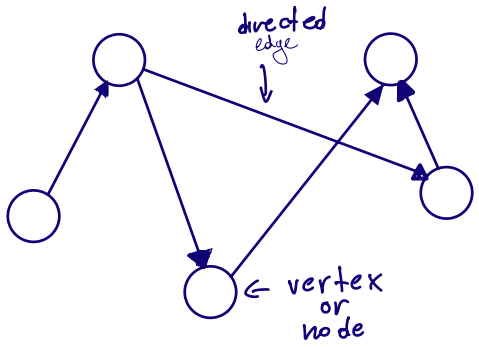

Figure A.1. A directed graph