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

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. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: Springer.
- Ch. 5 Neural Networks

(3) Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.
- 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) is an interconnection of several sigmoid or threshold functions associated arranged in layers, such that the outputs of one layer form the input of the next layer. Formally the structure of these interconnections can be depicted as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes $\mathcal{V}$ correspond to vertices/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 whose vertices can be numbered so that all connections go from a neuron (vertex) to one with a higher number. Hence, neurons (vertices) have one-way connections to other neurons such that the output of a lower numbered neuron feeds the input of the higher numbered neuron (in a forward manner). The neurons can be arranged in layers so that connections go from one layer to a later layer. FFNN can be depicted by a directed acyclic graph[2], $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. (See Figure 2.1).

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

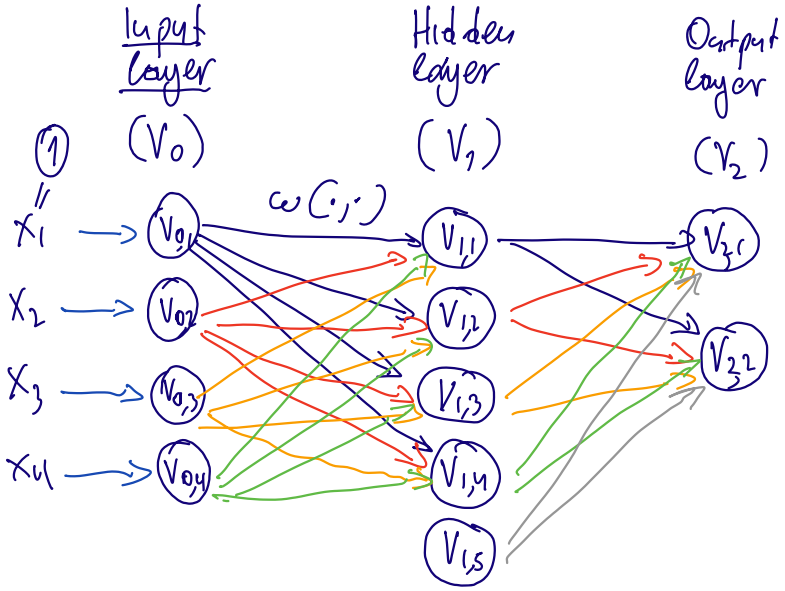


FIGURE 2.1. Feed forward neural network (1 hidden layer)

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 excluding the input layer; i.e. 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 neuron with no incoming edges has $o_{1,5} = \sigma(0)$.

Notation 29. To easy the notation, we denote the weights as $w_{(t+1),i,j} := w\left(\left(v_{t,j}, v_{t+1,i}\right)\right)$. Using this notation, $w_{(t+1),i,j} = 0$ is equivalent in (2.2) to $\left(v_{t,j}, v_{t+1,i}\right) \notin \mathcal{E}$ 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

$$h_k(x) = \sigma_{(2)}\left(\sum_{\forall j} w_{(2),k,j}\sigma_{(1)}\left(\sum_{\forall i} w_{(1),j,i}x_i\right)\right) \tag{2.4}$$

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 $\{w_{(t+1),i,j}\}$ where $w_{(t+1),i,j} := w((v_{t,j}, v_{t+1,i}))$.

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

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

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

$$y_i|x,w\sim\mathrm{N}\left(\mu_i,\beta^{-1}\right),\ \text{ where }\ \mu_i=h_w\left(x_i\right) \tag{3.2}$$

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

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

- The Error function is

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

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

$$y_i|x_i, w\sim \mathrm{N}\left(\mu,\Sigma\right),\ \text{ where }\ \mu_i=h_w\left(x_i\right) \tag{3.3}$$

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

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

- 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

$$y_i|x_i, w\sim \text{Bernoulli}\left(p_i\right),\ \text{ where }\ p_i=h_w\left(x_i\right) \tag{3.4}$$

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

$$\sigma_T\left(a_k\right) = \frac{\exp\left(a_k\right)}{\sum_{k'=1}^{q} \exp\left(a_{k'}\right)}, \text{ for } k = 1, ..., q \tag{3.5}$$

  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, 3.5 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

$$\tilde{\sigma}_T\left(a_k\right) = \frac{\exp\left(a_k\right)}{1 + \sum_{k'=1}^{q-1} \exp\left(a_{k'}\right)}, \text{ for } k = 1, ..., q-1 \tag{3.6}$$

  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 (3.6) compared to (3.5) is that the former uses one less output neurons (less unknown parameters to learn). Also (3.6) is not invariant to additive transformations $a \leftarrow a + c$ unlike 3.5i.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

$$y_i|x, w \sim \text{Multinomial}\left(p_i\right), \text{ where } p_i = h_w\left(x_i\right)$$

  with mass function

$$f\left(y_i|x, w\right) = \prod_{k=1}^{q} h_{w,k}\left(x_i\right)^{y_{i,k}}$$

  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. **Regularization.**

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 (or weight decay) $J(w;\lambda)$, 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. $\left|w^*_{t,i,j}\right| < \xi$.

Note 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: $J(w;\lambda = (\lambda_1, \lambda_2)) = \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$

5. Stochastic gradient implementation for training NN

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.

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

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

Note 56. The error backpropagation procedure is an efficient algorithm for the computation of the gradient $\nabla_w \ell(w,z)$ of the loss function $\ell(w,z)$ at some value of w and some example $z=(x,y)$ as required for the implementation of stochastic gradient based algorithm for the learning problems under consideration.

Assumption 57. *Error backpropagation assumes that $\ell(w,z)$ is differentiable at w for each value of z; $\nabla_w \ell(w,z)$ exists.*

Notation 58. Let $V_t = \{v_{t,1}, ..., v_{t,k_t}\}$ be the t-th layer of a NN and $k_t = |V_t|$ the number of neuron in layer $t = 1, ...T$.

Notation 59. Let $o_t = (o_{t,1}, ..., o_{t,k_t})^\top$ be the vector of the outputs of the t-th layer of a NN.

Notation 60. Let $\alpha_t = (\alpha_{t,1}, ..., \alpha_{t,k_t})^\top$ be the vector of the activations of the t-th layer of a NN.

Notation 61. We denote as $\ell_t(\cdot)$ the loss function $\ell(w,z)$ as a function of the output o_t at t-th layer.

- E.g it is $\ell_T(\xi) = \frac{1}{2}(\xi - y)^2$ if $\ell(w,z) = \frac{1}{2}(h_w(x) - y)^2$ since $h_w(x) = o_T(x)$.

Algorithm 62. *(Error backpropagation)*

Requires: *The NN* $(\mathcal{V}, \mathcal{E}, \sigma, w)$ *with the values of the a weight vector* $w \in \mathbb{R}^{|\mathcal{E}|}$*, and example value* $z = (x, y)$

Returns: $\nabla_w \ell(w, z) = \left(\frac{\partial}{\partial w_{t,i,j}} \ell(w, z) \, ; \, \forall t, i, j \right)$

Initialize:

$$\text{Set } w_{t+1,j,i} = \begin{cases} w(v_{t,j}, v_{t+1,j}) & \text{if } (v_{t,j}, v_{t+1,j}) \in \mathcal{E} \\ 0 & \text{if } (v_{t,j}, v_{t+1,j}) \notin \mathcal{E} \end{cases}$$

Forward pass:

(1) For $i = 1, ..., d$

Set:

$$o_{0,i} = x_i$$

(2) For $t = 1, ..., T$

For $i = 1, ..., k_t$

Compute:

$$\alpha_{t,i} = \sum_{j=1}^{k_{t-1}} w_{t,i,j} o_{t-1,j}$$

Compute:

$$o_{t,i} = \sigma_t(\alpha_{t,i})$$

Backward pass:

(1) Set[a]*:*

$$\delta_T = \frac{d\ell_T}{do_T}(o_T)$$

(2) For $t = T - 1, ..., 1$

For $i = 1, ..., k_t$

Compute:

$$\delta_{t,i} = \sum_{j=1}^{k_{t+1}} w_{t+1,j,i} \delta_{t+1,j} \, \frac{d}{da} \sigma_{t+1}(a) \Big|_{a=\alpha_{t+1,j}}$$

Output:

for each edge $(v_{t-1,j}, v_{t,i}) \in \mathcal{E}$

Set,

$$\frac{\partial}{\partial w_{t,i,j}} \ell(w, z) = \delta_{t,i} \, \sigma_t'(\alpha_{t,i}) \, o_{t-1,j}$$

for $i = 1, ..., k_t$ *and* $j = 1, ..., k_{t-1}$.

---

[a] Eg if $\ell(w, z) = \frac{1}{2} \sum_{k=1}^{q} (h_k(x) - y_k)^2$ then $\delta_T = \frac{\mathrm{d}\ell_T}{\mathrm{d}o_T}(o_T) = o_T - y$, or otherwise $\frac{\partial \ell_T}{\partial o_{T,k}}(o_T) = o_{T,k} - y_k$ for all $k - 1, ..., q$

Remark 63. Several sources in the literature use the notation

$$\tilde{\delta}_t := \frac{\mathrm{d}\ell_t}{\mathrm{d}\alpha_t} = \frac{\mathrm{d}\ell_t}{\mathrm{d}o_t}\frac{\mathrm{d}o_t}{\mathrm{d}\alpha_t}$$

instead of

$$\delta_t = \frac{\mathrm{d}\ell_t}{\mathrm{d}o_t}$$

used in Algorithm 62. Then the backward pass and output steps in Algorithm 62 can be equivalently restated as

Backward pass:

(1) For $t = T$,

Set:

$$\tilde{\delta}_T = \frac{\mathrm{d}\ell_T}{\mathrm{d}\alpha_T}(o_T)$$

(2) For $t = T-1, ..., 1$

For $i = 1, ..., k_t$

Compute:

$$\tilde{\delta}_{t,i} = \sigma_t'(\alpha_{t,k}) \sum_{j=1}^{k_{t+1}} w_{t+1,j,i}\tilde{\delta}_{t+1,j}$$

Output:

- for each edge $(v_{t-1,j}, v_{t,i}) \in \mathcal{E}$

Set

$$\frac{\partial}{\partial w_{t,i,j}}\ell(w,z) = \tilde{\delta}_{t,i}o_{t-1,j}$$

Example 64. Consider the multi-output regression problem, assume a predictive rule $h_w : \mathbb{R}^d \to \mathbb{R}^q$ with $h_w = (h_{w,1}, ..., h_{w,q})$ and

$$h_k(x) = \sigma_2\left(\sum_{j=1}^{c} w_{2,k,j}\sigma_1\left(\sum_{i=1}^{d} w_{1,j,i}x_i\right)\right)$$

with activation functions $\sigma_2(a) = a$, and $\sigma_1(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$, and loss $\ell(w,z) = \frac{1}{2}\sum_{k=1}^{q}(h_k(x) - y_k)^2$ for example $z = (x, y)$. Perform the error back propagation steps to compute the elements of $\nabla_w \ell(w,z)$ at some w and z.

Solution.

Forward pass:

Set: $o_{0,i} = x_i$ for $i = 1, ..., d$

Compute:

at $t = 1$**:** for $j = 1, ...c$

comp: $\alpha_{1,j} = \sum_{i=1}^{d} w_{1,j,i}x_i$

comp: $o_{1,j} = \tanh(\alpha_{1,j})$

at $t = 2$**:** for $k = 1, ...q$

comp: $\alpha_{2,k} = \sum_{j=1}^{c} w_{2,k,j}o_{1,j}$

comp: $o_{2,k} = \alpha_{2,k}$

get: $h_k = o_{2,k}$

Note that $\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_1(\xi) = 1 - (\sigma_1(\xi))^2$ and that $\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_2(\xi) = 1$.

Backward pass:

at $t = 2$**:** for $k = 1, ...q$

comp:

$$\tilde{\delta}_{2,k} = \frac{\partial}{\partial \alpha_{2,k}}\ell_T = \sum_{j=1}^{q} \frac{\partial \ell_T}{\partial o_{2,j}}(o_{2,j}) \frac{\partial o_{2,j}}{\partial \alpha_{2,k}}(\alpha_{2,k}) = \frac{\partial \ell_T}{\partial o_{2,k}}(o_{2,k}) \frac{\partial o_{2,k}}{\partial \alpha_{2,k}}(\alpha_{2,k}) = h_k - y_k$$

at $t = 1$**:** for $j = 1, ...c$

comp:

$$\begin{aligned}\tilde{\delta}_{1,j} &= \frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_1(\xi)\Big|_{\xi=\alpha_{1,j}} \sum_{k=1}^{q} w_{2,j,k}\tilde{\delta}_{2,k} \\ &= \left(1 - (o_{1,j})^2\right) \sum_{k=1}^{q} w_{2,j,k}\tilde{\delta}_{2,k}\end{aligned}$$

Output:

$$\frac{\partial \ell(w,z)}{\partial w_{1,j,i}} = \tilde{\delta}_{1,j}x_i \text{ and } \frac{\partial \ell(w,z)}{\partial w_{2,k,j}} = \tilde{\delta}_{2,k}o_{2,j}$$

5.2. **Design of Error back-propagation (Algorithm 62).**

Notation 65. Let $\ell_t : \mathbb{R}^{k_t} \to \mathbb{R}$ be the loss function of the sub-network defined by layers $\{V_1, ..., V_T\}$ as a function of the outputs o_t of the neurons in V_t.

Notation 66. Let W_t be the $k_{t-1} \times k_t$ matrix of the weights of the t-th layer of a NN such as $[W_t]_{i,j} := w_{t,i,j} = w(v_{t-1,j}, v_{t,i})$.

Notation 67. We introduce notation, such that if $\sigma : \mathbb{R} \to \mathbb{R}$ and $a \in \mathbb{R}^d$, then $\sigma(a) \in \mathbb{R}^d$ is a d-dimensional vector such that $\sigma(a) = (\sigma(a_1), ..., \sigma(a_d))$.

Notation 68. The vector of the outputs $o_t \in \mathbb{R}^{k_t}$ of the neurons of V_t can be written as $o_t = \sigma_t(\alpha_t)$; aka $o_{t,j} = \sigma_t(\alpha_{t,j})$, for $j = 1, ..., k_t$.

Notation 69. The vector of activations, aka vector of the inputs, of the neurons of V_t can be written as $\alpha_t = W_t o_{t-1}$.

Note 70. Hence,

$$\ell_t(o_t) = \ell_t(\sigma_t(\alpha_t)) = \ell_t(\sigma_t(W_t o_{t-1})) \tag{5.1}$$

Notation 71. To facilitate differential calculus, let O_{t-1} be the $k_t \times (k_{t-1}k_t)$ matrix

$$O_{t-1} = \begin{bmatrix} o_{t-1}^\top & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & o_{t-1}^\top \end{bmatrix}$$

and let $w_t \in \mathbb{R}^{k_{t-1}k_t}$ is the vector $w_t = \left(W_{t,1}, ..., W_{t,k_{t-1}}\right)^\top$ by concatenating the rows of matrix W_t. Hence

$$W_t o_{t-1} = O_{t-1} w_{t-1} \tag{5.2}$$

Note 72. Hence, from (5.1) and (5.2), it is

$$\ell_t(o_t) = \ell_t\left(\underbrace{\sigma_t\left(\overbrace{O_{t-1}w_t}^{=\alpha_t}\right)}_{=o_t}\right)$$

Note 73. By chain rule it is

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}w_t}\ell_t(o_t) =& \frac{\mathrm{d}\ell_t}{\mathrm{d}o_t}\frac{\mathrm{d}o_t}{\mathrm{d}\alpha_t}\frac{\mathrm{d}\alpha_t}{\mathrm{d}w_t} \\ =& \frac{\mathrm{d}\ell_t}{\mathrm{d}o_t}\mathrm{diag}\left(\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_t(\xi)\Big|_{\xi=\alpha_t}\right) O_{t-1}\end{aligned} \tag{5.3}$$

Notation 74. Let as define

$$\delta_t = \frac{\mathrm{d}}{\mathrm{d}o_t}\ell_t(o_t)$$

and name them as 'errors'.

Note 75. Then (5.3), becomes

$$\begin{aligned}\frac{\mathrm{d}\ell_t}{\mathrm{d}w_t} &= \delta_t \mathrm{diag}\left(\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_t(\xi)\Big|_{\xi=\alpha_t}\right) O_{t-1} \\ &= \left(\delta_{t,1}\sigma'(\alpha_{t,1})\, o_{t-1,1}^\top, ..., \delta_{t,k_t}\sigma'(\alpha_{t,k_t})\, o_{t-1,k_t}^\top\right)\end{aligned}$$

Note 76. I will find a way to compute $\{\delta_t\}$ recursively from V_T to V_0. For $t = T$, it is

$$\delta_T = \frac{\mathrm{d}}{\mathrm{d}o_T}\ell_T(o_T) = \frac{\mathrm{d}}{\mathrm{d}o_T}\ell(o_T, z)$$

Note that

$$\ell_t(o_t) = \ell_{t+1}\left(\underbrace{\sigma_{t+1}\left(\overbrace{W_{t+1}o_t}^{=\alpha_{t+1}}\right)}_{=o_{t+1}}\right)$$

Then for $t = T-1, ..., 1$, it is

$$\begin{aligned}\delta_t =& \frac{\mathrm{d}\ell_t}{\mathrm{d}o_t} = \frac{\mathrm{d}\ell_{t+1}}{\mathrm{d}o_t} \\ =& \frac{\mathrm{d}\ell_{t+1}}{\mathrm{d}o_{t+1}}\frac{\mathrm{d}o_{t+1}}{\mathrm{d}\alpha_{t+1}}\frac{\mathrm{d}\alpha_{t+1}}{\mathrm{d}o_t} \\ =& \delta_{t+1}\mathrm{diag}\left(\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_{t+1}(\xi)\Big|_{\xi=\alpha_{t+1}}\right) W_{t+1}\end{aligned}$$

Note 77. Hence, for $t=1,...,T$ it is

$$\frac{\mathrm{d}\ell(w,z)}{\mathrm{d}w_t} = \delta_t \mathrm{diag}\left(\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_t(\xi)\Big|_{\xi=\alpha_t}\right) O_{t-1}$$

or element wise, for $t=1,...,T$, $j=1,...,k_{t-1}$, $i=1,...,k_t$ it is

$$\frac{\partial\ell(w,z)}{\partial w_{t,i,j}} = \delta_{t,i}\,\sigma_t'(\alpha_{t,i})\; o_{t-1,j}$$

Remark 78. Error back-propagation idea can also be used to efficiently compute the gradient $\nabla_x \ell(w, z=(x,y))$ of the loss function $\ell(w, z=(x,y))$ with respect to the inputs x. The idea is the same, use chain rule to find a recursive procedure.

5.3. **Preconditioning and computation of the Hessian.**

Note 79. Recall (Handout 3, Algorithm 43), that SGD may be improved by using a suitable preconditioner $P_t > 0$ as

$$w^{(t+1)} = w^{(t)} - \eta_t P_t \nabla_w \ell\left(w^{(t)}, z^{(t)}\right)$$

such a preconditioner can be the $P_t := [H_t + \epsilon I_d]^{-1}$ where H_t is the Hessian of $\ell\left(w^{(t)}, z^{(t)}\right)$ and $\epsilon > 0$.

Note 80. The computation of the Hessian H_t can be done by using error propagation ideas for

$$[H_t]_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} f(w)\Big|_{w=w^{(t)}}$$

We will not go further to exact computations.

Note 81. The exact computation of the Hessian H_t of the loss $\ell(\cdot, z)$ in NN setting can be computationally expensive and hence approximations are often used (with hope to work well). One can implement the general purpose AdaGrad (Section 7.1, Handout 3). In what follows we present other alternatives tailored to the NN model.

Note 82. Consider the regression problem with predictive rule $h: \mathbb{R}^d \to \mathbb{R}$ with $h_w(x) = o_T(x) = \alpha_T(x)$, and loss function $\ell(w, z=(x,y)) = \frac{1}{2}(h_w(x) - y)^2$. Then the Hessian of ℓ is

$$\begin{aligned} H &= \frac{\mathrm{d}(\nabla_w \ell)}{\mathrm{d}w} = \frac{\mathrm{d}}{\mathrm{d}w}\left(\nabla_w h_w(x)\,(h_w(x)-y)^\top\right) \\ &= \left(\frac{\mathrm{d}}{\mathrm{d}w}\nabla_w h_w(x)\right)(h_w(x)-y)^\top + \nabla_w h_w(x)\,(\nabla_w h_w(x))^\top \end{aligned} \tag{5.4}$$

We can expect that $h_w(x) \approx y$ provided that the network is well trained due to the global approximation ability of the FFNN. Yet, we can expect that $h_w = \mathrm{E}(y)$ provided that we train the network under the quadratic loss and because $\arg\min_h \mathrm{E}_{y\sim g}(h-y)^2 = \mathrm{E}_{y\sim g}(y)$. Hence a reasonable approximation can be

$$\begin{aligned} H &\approx \nabla_w h_w(x)\,(\nabla_w h_w(x))^\top \\ &\overset{\sigma_T(a)=a}{=} \nabla_w \alpha_T(x)\,(\nabla_w \alpha_T(x))^\top \end{aligned}$$

Consequently, the approximation of the Hessian of the Error function $\text{EF}\left(w|\left\{z_i\right\}\right)=\sum_{i=1}^{n}\ell\left(w,z_i=\left(x_i,y_i\right)\right)$ is

$$H_n=\sum_{i=1}^{n}\frac{\mathrm{d}\left(\nabla_w\text{EF}\right)}{\mathrm{d}w}\approx\sum_{i=1}^{n}\nabla_w h_w\left(x_i\right)\left(\nabla_w h_w\left(x_i\right)\right)^\top \tag{5.5}$$

Note 83. (Cont. Note 82) To efficiently compute the inverse H_n^{-1} of (5.5) I can utilize Woodbury identity

$$\left(M+vv^\top\right)^{-1}=M^{-1}-\frac{\left(M^{-1}v\right)\left(v^\top M^{-1}\right)}{1+v^\top M^{-1}v}$$

offering a way to avoid the computational expensive task of directly inverting the high dimensional H_n. Given $v_i=\nabla_w h_w\left(x\right)$, it is

No need to memorize Woodbury identity

$$\begin{aligned}\left(H_n\right)^{-1}=&\left(\sum_{i=1}^{n}v_iv_i^\top\right)^{-1}=\left(\sum_{i=1}^{n-1}v_iv_i^\top+v_nv_n^\top\right)^{-1}=\left(H_{n-1}+v_nv_n^\top\right)^{-1}\\ =&H_{n-1}^{-1}-\frac{\left(H_{n-1}^{-1}v_n\right)\left(v_n^\top H_{n-1}^{-1}\right)}{1+v_n^\top H_{n-1}^{-1}v_n}\end{aligned} \tag{5.6}$$

In practice I start with $H_0=\epsilon I$ with $\epsilon>0$ small, and iterate (5.6).

Note 84. Likewise and by modifying (5.4), one can compute the corresponding approximations for the classification problems or problems with different loss functions.

6. Bayesian Artificial Neural Networks

Note 85. Consider a feed-forward Neural Network with $(\mathcal{V},\mathcal{E},\sigma,w)$. We use the same notation and structure as in the classical treatment above. Assume the architecture $(\mathcal{V},\mathcal{E},\sigma)$ of a neural network is fixed/known, and interest lies in learning the weights $\{w_{t,i,j}\}$.

Note 86. Assume there is available a training set of examples (data-set) $\mathcal{S}=\left\{z_i=\left(x_i,y_i\right);i=1,...,n\right\}$ with $x_i\in\mathcal{X}=\mathbb{R}^{|V_0|}$ and $y_i\in\mathcal{Y}$. The Bayesian NN model is then

$$\begin{cases}y_i|x_i,w & \overset{\text{ind}}{\sim}f\left(y_i|x_i,w\right),\ i=1,...,n\ \text{(sampling distribution)}\\ w & \sim f\left(w\right)\qquad\qquad\qquad\text{(prior distribution)}\end{cases}$$

where the sampling distribution is specified either according to the experimental design (how $\{z_i\}$) are collected or based on subjective judgments, while the prior distribution is specified based on subjective manner.

Note 87. In most of the real applications, it is difficult (almost impossible) to specify the sampling distribution based on the experimental design or judgments, and almost impossible to specify the prior of the weights based on subjective judgments.

Note 88. One could specify the sampling distribution based on the loss function $\ell\left(h_w\left(x\right),z=\left(x,y\right)\right)$ as

$$f\left(y_i|x_i,w\right)\propto\exp\left(-\ell\left(h_w\left(x_i\right),\left(x_i,y_i\right)\right)\right), \tag{6.1}$$

based on the argument that sampling distribution in the posterior distribution contraindicates how far the theoretical model $h_w(x_i)$ (as casted in a NN) is from the corresponding observation y_i via the likelihood.

Note 89. The prior density of the weights may be specified based on some shrinkage term $J(w;\lambda)$ (Note 50) as

$$f(w) \propto \exp(-J(w;\lambda)) \tag{6.2}$$

based on the argument that I often model the predictive rule $h_w(\cdot)$ with an over-parametrized NN (with many weighs, more than needed) and many of them should be shrunk to zero. Also as we see in Note 97, careless training of NN tends to produce over-fitted NN with large weights (in abs values).

Note 90. Regarding prior it is often, $w \sim \mathrm{N}\left(0, I\lambda^{-1}\right)$ for some small λ controlling prior uncertainty about weights, i.e. 50% chances for the weight to be above or below zero.

Note 91. The corresponding posterior (according to the Bayes theorem) is

$$f(w|\{z_i\}) = \frac{\prod_{i=1}^n f(z_i|w) f(w)}{\int \prod_{i=1}^n f(z_i|w') f(w') \mathrm{d}w'} \propto \prod_{i=1}^n f(z_i|w) f(w)$$

and given 6.1 and 6.2, in log scale I get

$$\log(f(w|\{z_i\})) = -\sum_{i=1}^n \ell(h_w(x_i), (x_i, y_i)) - J(w;\lambda) + \text{const}$$

that resembles the EF with a shrinkage term in classical learning.

Example 92. (Cont. Example 43)

Case 1. (Regression problem) Sampling distribution can be specified as in (6.1) based on loss

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

Note this is equivalent to as if we had considered a sampling distribution

$$y_i|x_i, w \sim \mathrm{N}\left(\mu_i, \beta^{-1}\right), \text{ where } \mu_i = h_w(x_i) \tag{6.3}$$

for some fixed $\beta > 0$ based on (3.2). Prior can be specified with density (6.2) where

$$J(w;\lambda) = \frac{\lambda}{2}\|w\|_2^2$$

is the Ridge shrinkage term. Note this is equivalent to as if we had considered

$$w \sim \mathrm{N}\left(0, I\lambda^{-1}\right) \tag{6.4}$$

The resulted posterior density is

$$f(w|\{z_i\}) \propto \exp\left(-\frac{\beta}{2}\sum_{i=1}^n (h_w(x_i) - y_i)^2 - \frac{\lambda}{2}\|w\|_2^2\right) \tag{6.5}$$

Case 2. (Multi-output regression problem) Try to do it by yourself

Case 3. (Binary classification problem) Sampling distribution can be specified as in (6.1) based on loss

$$\ell\left(w,(x,y)\right)=y\log\left(h_w\left(x\right)\right)+\left(1-y\right)\left(1-\log\left(h_w\left(x\right)\right)\right)$$

which is equivalent to as if we had considered a sampling distribution

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

based on (3.4). Prior can be specified as in (6.2) with the LASSO shrinkage term

$$J\left(w;\lambda\right)=\lambda\left\|w\right\|_1;$$

The resulted posterior density is

$$f\left(w|\left\{z_i\right\}\right)\propto\exp\left(-\sum_{i=1}^{n}\left[y_i\log\left(h_w\left(x_i\right)\right)+\left(1-y_i\right)\left(1-\log\left(h_w\left(x_i\right)\right)\right)\right]-\lambda\left\|w\right\|_1\right)$$

Case 4. (Multi-class classification problem) Try to do it by yourself.

Note 93. Sampling from the posterior can be performed via SGLD due to the high-dimensionality in the weights w and the big size of the training data set. Recall the recursion of the SGLD with batch size m is

$$w^{(t+1)}=w^{(t)}+\eta_t\left(\frac{n}{m}\sum_{j\in\mathcal{J}^{(t)}}\nabla_w\log\left(f\left(z_j^{(t)}|w^{(t)}\right)\right)+\nabla_w\log\left(f\left(w^{(t)}\right)\right)\right)+\sqrt{\eta_t}\sqrt{\tau}\epsilon_t, \tag{6.6}$$

for $\epsilon_t\sim\text{N}\left(0,1\right)$. (See Algorithm 21 in Handout 4: Bayesian Learning via Stochastic gradient and Stochastic gradient Langevin dynamics).

Note 94. Given sample values $\left\{w^{(t)}\right\}_{t=1}^{T}$ produced from SGLD recursions (6.6), learning of any function $h_w\left(x\right)$ of the w's can be performed via

(1) Monte Carlo estimation namely averaging the samples values $\left\{w^{(t)}\right\}_{t=1}^{T}$ as

$$\widehat{h_w}\left(x\right)=\frac{1}{T}\sum_{t=1}^{T}h_{w^{(t)}}\left(x\right),\ \ \text{(Monte Calro estimator)}$$

(2) Maximum-A-posteriori (MAP) estimation namely find the best $\hat{w}^*$ among the samples values $\left\{w^{(t)}\right\}_{t=1}^{T}$ that maximizes the posterior i.e

$$\hat{w}^*=\arg\max_{\left\{w^{(t)}\right\}_{t=1}^{T}}\left(\log f\left(\left\{z_i\right\}|w\right)+\log f\left(w\right)\right)$$

and compute

$$\widehat{h_w}\left(x\right)=h_{\hat{w}^*}\left(x\right),\ \ \text{(MAP estimator)}$$

7. Theoretical aspects

Note 95. We will not go this direction here. For the interested student, I suggest for starters, Ch 5 from "Ripley, B. D. (2007). Pattern recognition and neural networks. Cambridge university press.";

and for advanced Ch 30 from "Devroye, L., Györfi, L., & Lugosi, G. (2013). A probabilistic theory of pattern recognition (Vol. 31). Springer Science & Business Media."

8. Comments, guidelines, and discussions

8.1. **Over-fitting issues.**

Note 96. Neural Networks can be "over-parametrized", for instance by consisting of a large number of layers each of them having a large number of neurons able to represent each feacture/characteristic of the pattern of interest to be learned. Careless training may produce NN models with bad generalization predictive properties.

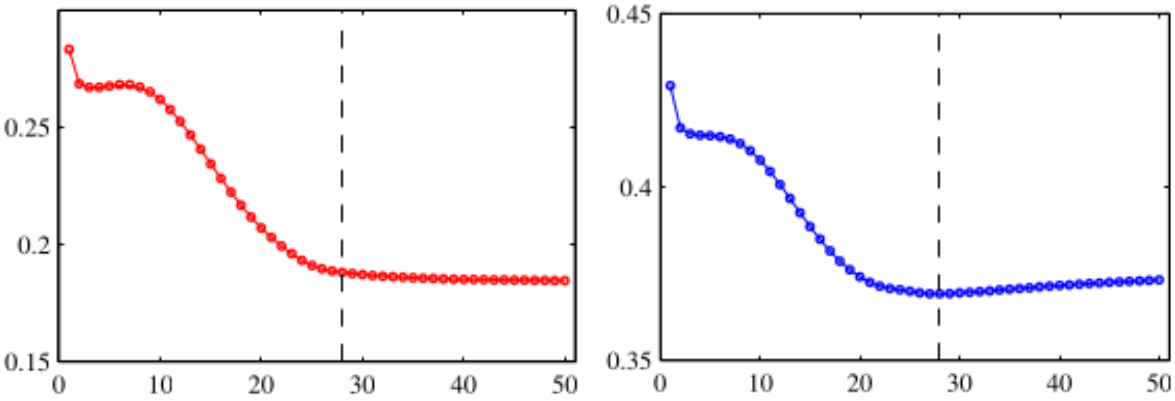
Note 97. Training of non-linear (non-convex) NN models corresponds to an iterative reduction of the Error Function defined on the training data-set. It has been observed that the Error Function defined on the validation data-set (independent to the training data set) often shows a decrease at first, followed by an increase as the NN starts to overfit. This overfit is often associated to the production of weight values larger than needed to be generalised. It is desirable to avoid this over-fitting with purpose to obtain a NN with good generalization performance.

Note 98. **Early stopping** is a way of limiting the effective network complexity by halting training before a minimum of the training error has been reached. During training, we monitor the NN performance against the EF defined on the validation data-set, and stop the training procedure when just before the EF defined on the validation data set start increasing (aka before overfitting signs) to prevent the model memorizing too much information about the training set.

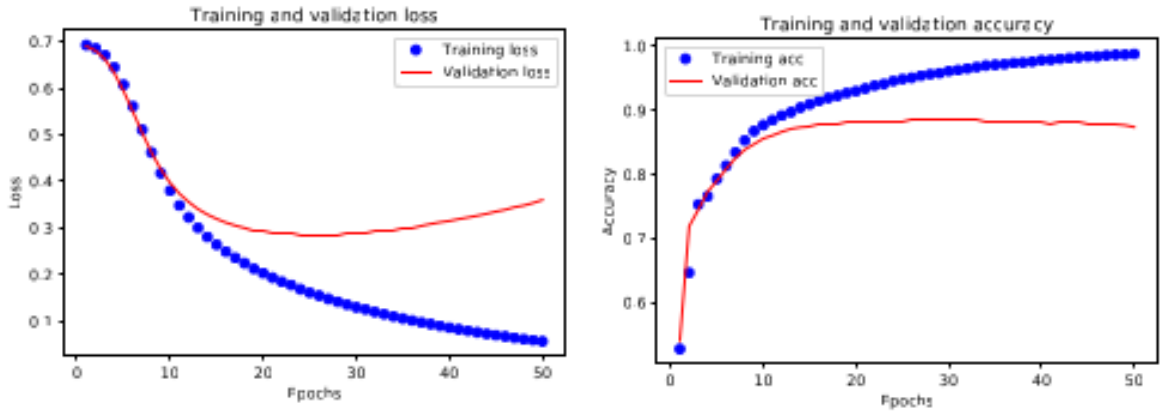
Note 99. **Regularization** as in Section 4.2 can alternatively be used to address the above issue by using a shrinkage term preventing the weights to grow too much away from zero.

8.2. **Non-identifiabilities.**

Note 100. The weights $\{w_{t,i,j}\}$ have non-identifiabilities. Consider the FFNN in Figure 8.2 with 1 hidden layer of M neurons, activation function $\sigma_1(a) = \tanh(a)$ and and full connectivity in both layers. If we flip the sign of all of the w's feeding into a particular hidden unit (for a given input pattern) then the sign of the activation of the hidden unit will be reversed because $\tanh(-a) = -\tanh(a)$. This transformation can be compensated by changing the sign of all of the w's leading out of that hidden unit. By changing the signs of a particular group of w's, the input-output mapping function represented by the FFNN is unchanged, and so there are two different weight vectors resulting the same mapping function. I can do M such flips (there are M hidden nuerons) leading to 2^M equivalent parameter settings. Similarly, we can permulte the labels of the neurons in hidden layer without changing the loss function; there are $M!$ such permutations. Hence the equivalent parameter settings in total are $2^M M!$. –This is not harmful in training or predictive rul computation as we just need to find one such parameter setting.



(A) training set (B) validation set



(C) loss function (D) accuracy

FIGURE 8.1. Behavior of the Error Function wrt the iterations, against a training set and against a validation set.

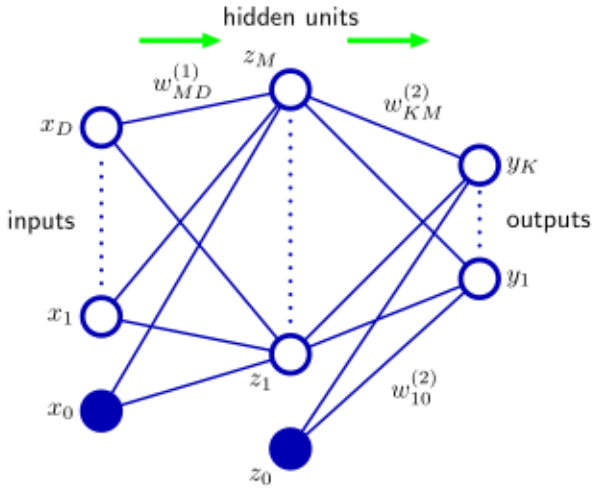


FIGURE 8.2. A FFNN

8.3. **Non-convexity.**

Note 101. Learning problems with NN are non-convex, hence the loss function to be minimized has many local minima. The consequence is that the SGD/SGLD learning algorithms may be trapped in a local optima and never reach any of the global ones. The number of local minima is exaggerated due to the symmetries discussed in Note 100. Due to the large number of data (big-data) used to train the NN (in real life) such local minima become more shallow while the global minima more picky. Due to this and the stochastic nature of SGD/SGLD algorithms, SGD/SGLD may be able (by chance) to escape from such local minima/maxima and reach the global ones.

Note 102. To address local optima issue, run SGD/SGLD learning algorithms multiple times by initializing them with different seeds each time.

8.4. **Skip-layers 2.4.**

Note 103. The general definition of feed forward Neural Network allows "skip-layer" connections that is the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in NN architecture includes edges which may not necessarily connect neurons between consecutive layers, namely $(v_{t,j}, v_{t+k,i}) \in \mathcal{E}$ for some $k > 1$.

Example 104. For instance, a fully connected FFNN with 1 hidden layer is

$$h_k(x) = \sigma_{(2)}\left(\underbrace{\sum_{\forall i} w^{\text{skip}}_{(1),k,i} x_i}_{\text{skip layer terms}} + \sum_{\forall j} w_{(2),k,j}\sigma_{(1)}\left(\sum_{\forall i} w_{(1),j,i} x_i\right)\right) \tag{8.1}$$

Compared to (2.4) in Example 31 and (2.4), (8.1) links the input to the output.

Note 105. In skip-layer FFNN cases, the error back propagation (Section 5.1) has to be adjusted properly.

Note 106. FFNN are mainly used without skip-layers because theory (Section 7) states that FFNN are global approximators even without skip layers, as well as because of computational inconvenience and modeling parsimony.

Appendix A. About graphs

Definition 107. 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 108. 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 $e \in \mathcal{E}$.

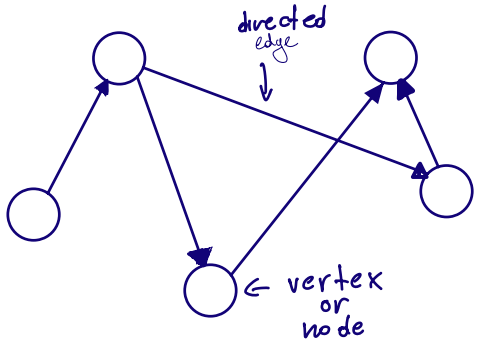


Figure A.1. A directed graph

Appendix B. About Partial derivatives

Note 109. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$.

Definition 110. The partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at the point $a = (a_1, \ldots, a_n) \in U \subseteq \mathbb{R}^n$ with respect to the i-th variable is denoted as

$$\frac{\partial f}{\partial x_i}(a) \text{ or } \left.\frac{\partial f}{\partial x_i}(x)\right|_{x=a}$$

and defined as

$$\begin{aligned}\left.\frac{\partial f}{\partial x_i}(x)\right|_{x=a} &= \lim_{h\to 0} \frac{f(a_1, ..., a_{i-1}, a_i + h, a_{i+1}, ..., a_n) - f(a_1, ..., a_{i-1}, a_i, a_{i+1}, ..., a_n)}{h} \\ &= \lim_{h\to 0} \frac{f(a + he_i) - f(a)}{h}\end{aligned}$$

where e_i is a $0-1$ vector with only one ace in the i-th location.

Remark 111. Essentially, Definition 110 says that the partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at the point $a = (a_1, \ldots, a_n) \in U \subseteq \mathbb{R}^n$ with respect to the i-th variable is

$$\frac{\partial f}{\partial x_i}(a) = \left.\frac{\mathrm{d}g}{\mathrm{d}h}(h)\right|_{h=0}$$

the derivative of function $g(h) := f(a_1, ..., a_{i-1}, a_i + h, a_{i+1}, ..., a_n)$ at value 0.

Example 112. Consider a function f with $f(x_1, x_2) = x_1^2 + x_1 x_2^3$. Comput is partial derivatives at $a = (2,3)^\top$; i.e. $\frac{\partial f}{\partial x_1}(a)$ and $\frac{\partial f}{\partial x_2}(a)$.

Solution. It is

$$\begin{aligned}\frac{\partial f}{\partial x_1}(a) &= \frac{\partial f}{\partial x_1}(x)\Big|_{x=a} = \frac{\mathrm{d}}{\mathrm{d}x_1}\left(x_1^2 + x_1 x_2^3\right)\Big|_{x=a} = 2x_1 + x_2^3\big|_{x=a} \\ &= 2a_1 + a_2^3 = 4 + 27 = 31 \\ \frac{\partial f}{\partial x_2}(a) &= \frac{\partial f}{\partial x_2}(x)\Big|_{x=a} = \frac{\mathrm{d}}{\mathrm{d}x_2}\left(x_1^2 + x_1 x_2^3\right)\Big|_{x=a} = 2x_1 x_2^2\big|_{x=a} \\ &= 2a_1 a_2^2 = 4 + 27 = 72\end{aligned}$$

Appendix C. About Jacobian

Note 113. Consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$.

Note 114. The Jacobian of f at $w \in \mathbb{R}^n$, is denoted as $J_w(f)$ and it is an $m \times n$ matrix with (i,j) elements such as

$$[J_w(f)]_{i,j} = \frac{\mathrm{d}}{\mathrm{d}w_j} f_i(w)$$

for $i = 1, ..., m$ and $j = 1, ..., n$.

Note 115. To align with standard notation in statistics, we will use the notation

$$\frac{\mathrm{d}f}{\mathrm{d}w}, \text{ or } \frac{\mathrm{d}}{\mathrm{d}w} f(w)$$

for $J_w(f)$.

Note 116. Some properties

- Let functions $f(w) = Aw$, for matrix $A \in \mathbb{R}^{m,n}$ and vector $w \in \mathbb{R}^n$. Then $\frac{\mathrm{d}f}{\mathrm{d}w}(w) = A$.
- Let functions $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^k \to \mathbb{R}^n$. The Jacobian of the composition function $f \circ g : \mathbb{R}^k \to \mathbb{R}^m$ with $f(w) = f(g(w))$ at w is

  $$\frac{\mathrm{d}}{\mathrm{d}\xi} f \circ g(\xi)\Big|_{\xi=w} = \frac{\mathrm{d}f}{\mathrm{d}g}(g)\Big|_{g=g(w)} \frac{\mathrm{d}g}{\mathrm{d}\xi}(\xi)\Big|_{\xi=w}$$

  or more compactly

  $$\frac{\mathrm{d}}{\mathrm{d}w} f \circ g = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}w}$$

Example 117. Let $g : \mathbb{R}^k \to \mathbb{R}^n$ with $g(w) = Aw$ where matrix $A \in \mathbb{R}^{n,k}$. Let $\sigma : \mathbb{R} \to \mathbb{R}$. Compute $\frac{\mathrm{d}}{\mathrm{d}w}\sigma \circ g(w)$.

Hint: Adopt notation $\sigma(a) = (\sigma(a_1), ..., \sigma(a_n))$ when $a \in \mathbb{R}^d$.

Solution. It is

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}w}\sigma\circ g\left(w\right)&=\left.\frac{\mathrm{d}}{\mathrm{d}g}\sigma\left(g\right)\right|_{g=g(w)}\frac{\mathrm{d}}{\mathrm{d}w}g\left(w\right)\\&=\left.\frac{\mathrm{d}}{\mathrm{d}g}\sigma\left(g\right)\right|_{g=Aw}\frac{\mathrm{d}}{\mathrm{d}w}g\left(w\right)\\&=\mathrm{diag}\left(\sigma\left(Aw\right)\right)\,A\end{aligned}$$

This is because

$$\left[\frac{\mathrm{d}}{\mathrm{d}w}g\left(w\right)\right]_{i,j}=\left[\frac{\mathrm{d}}{\mathrm{d}w}Aw\right]_{i,j}=\frac{\partial}{\partial w_j}\sum_{k=1}^{k}A_{i,k}w_k=A_{i,j}$$

and because

$$\left[\frac{\mathrm{d}}{\mathrm{d}g}\sigma\left(g\right)\right]_{i,j}=\left[\frac{\mathrm{d}\left(\sigma\left(g_1\right),...,\sigma\left(g_n\right)\right)}{\mathrm{d}\left(g_1,...,g_n\right)}\right]_{i,j}=\frac{\partial}{\partial g_j}\sigma\left(g_i\right)=\begin{cases}\sigma\left(g_i\right), & i=j\\0, & i\neq j\end{cases}$$