Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 9: 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) Bishop, C. M., & Bishop, H. (2024). Deep learning: foundations and concepts. New York: Springer. [a]
- Ch. 5, 6, 8, 9

(2) Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.
- Ch. 4 The multi-layer perceptron

(3) 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.

[a] As alternative & more compact see Ch. 5 Neural Networks from "Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: Springer."

1. Intro and motivation

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.

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

$$h_{k}\left(x\right)=\sigma_{2}\left(w_{2,k,0}+\sum_{j}w_{2,k,j}\sigma_{1}\left(w_{1,j,0}+\sum_{i}w_{1,j,i}x_{i}\right)\right)\text{, for } k=1,...,q$$

where $\{\sigma_{j}\left(\cdot\right)\}$ are known functions and $\{w_{\cdot,\cdot,\cdot}\}$ are unknown weights/coefficients. E.g., one may choose with $\sigma_{2}\left(\alpha\right)=\alpha$, $\sigma_{1}\left(\alpha\right)=1/\left(1+\exp\left(-\alpha\right)\right)$.

Note 3. 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.

$$n_{j}\left(t\right)=1\left(\sum_{\forall i\rightarrow j}w_{j,i}n_{i}\left(t-1\right)>\theta_{i}\right)$$

no need to memorize the formula

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...

Note 4. 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

no need to memorize the formula

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

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 5. Recall soft-SVM in (Example 14 in Handout 8: Kernel methods); the predictive rule h^ψ had a formula

$$h^\psi(x) = \text{sign}\left(b + \langle w, \psi(x)\rangle\right) = \text{sign}\left(b + \sum_j w_j \psi_j(x)\right), \tag{1.2}$$

with some embedding $\psi(x) = (\psi_1(x), \psi_2(x) ...)^\top$ and unknown coefficients $\{w_j\}$. Assume there is no prior info about how to specify the formula of the "ideal" embedding ψ; then the scientist could try to parameterize ψ such as

$$\psi_j(x) = \zeta\left(\omega_{j,0} + \sum_i \omega_{j,i} x_i\right), \tag{1.3}$$

for some chosen function $\zeta(\cdot)$ and unknown coefficients $\{\omega_{j,i}\}$, hopping that the unknown coefficients $\{\omega_{j,i}\}$ can be successfully tuned during the learning procedure against the dataset and hence result in an adaptive semi-parametric way to construct the the embedding ψ from the training data without his/her interference. Based on Note 4, and assuming that a linear combination of $\{\zeta(\cdot)\}$, specified as sigmoids, could acceptably enough represent the fluctuations of $\{\psi_j(x)\}$ in (1.2), such an attempt to "adaptively construct the embedding ψ" might be successful.

Example 6. (Cont Example 5) Having this in the mind, and while wanting to improve expressiveness, the scientist could become even more greedy, and consider the use of another embedding in (1.3) as

$$\psi_j(x) = \zeta\left(\omega_{j,0} + \sum_i \omega_{j,i} \phi_i(x)\right),$$

whose bases $\phi(x) = (\phi_1(x), \phi_2(x) ...)^\top$ were parameterised as

$$\phi_i(x) = \xi\left(\varpi_{i,0} + \sum_s \varpi_{i,s} x_s\right)$$

for some chosen sigmoid $\xi(\cdot)$ and unknown coefficients $\{\varpi_{i,s}\}$. This recursive basis function composition creates a deep hierarchical structure which may improve approximation of the ideal embedding $\{\psi_j(x)\}$, and expressiveness of predictive rule h^ψ by sacrificing "interpretability" and "parsimony" of the "model".

2. Feedforward Neural Network (Mathematical set-up)

Note 7. An (Artificial) Neural Network (NN) is an interconnection of several sigmoids or threshold functions associated and 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 8. 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[1], $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. (See Figure 2.1).

Note 9. Here we restrict ourselves to FFNN only.

Note 10. 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 11. The first layer V_0 is called **input layer**. If x has d dimensions, then the first layer V_0 contains d nodes.

Note 12. The last layer V_T is called **output layer**. If y has q dimensions, then the last layer V_T contains q nodes.

Note 13. The intermediate layers $\{V_1, ..., V_{T-1}\}$ are called **hidden layers.**

Note 14. In a neural network, the nodes of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ correspond to neurons.

Note 15. The **i-th neuron of the t-th layer** is denoted as $v_{t,i}$.

Note 16. The output of neuron i in the input layer V_0 is simply x_i that is $o_{0,i}(x) = x_i$ for $i = \{1, ..., d\}$.

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

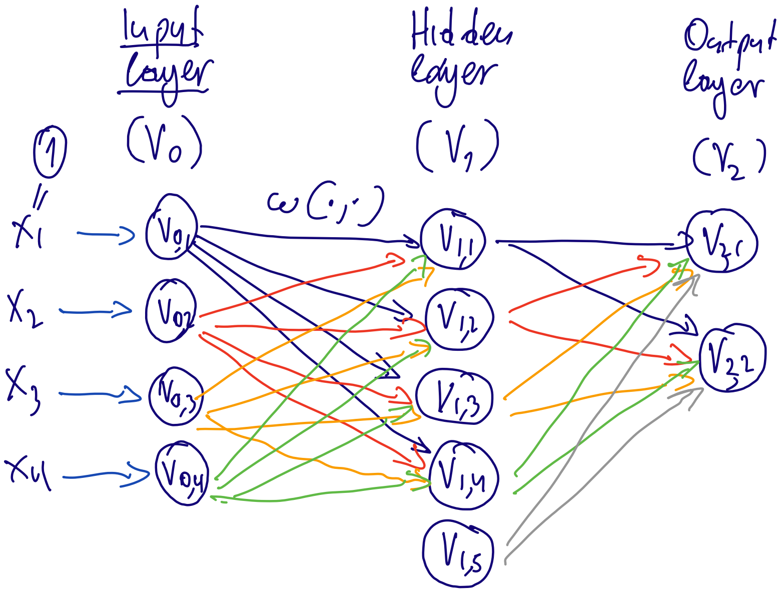


FIGURE 2.1. Feed forward neural network (1 hidden layer)

Note 17. 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 18. We define a **weight function** $w : \mathcal{E} \to \mathbb{R}$ over the edges $\mathcal{E}$.

Note 19. **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

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

Note 20. 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 21. **The output of neuron** $v_{t,i}$ when the network is fed with the input x is denoted as $o_{t,i}(x) := \sigma_{t-1}(\alpha_{t-1,i}(x))$.

Note 22. 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

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

Note 23. 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 $\alpha_{t+1,i}(x)$.

Algorithm 24. *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* $i = 1, ..., |V_{t+1}|$

$$\begin{aligned}\alpha_{t+1,i}(x) =& \sum_{\forall j:(v_{t,j},v_{t+1,i})\in\mathcal{E}} w\left(\left(v_{t,j}, v_{t+1,i}\right)\right) o_{t,j}(x)\\ o_{t+1,i}(x) =& \sigma_{t+1}\left(\alpha_{t+1,i}(x)\right)\end{aligned}$$

Note 25. **Depth** of the NN is the number of the layers excluding the input layer; i.e. T.

Note 26. **Size** of the network is the number $|V|$.

Note 27. **Width** of the NN is the number $\max_{\forall t}(|V_t|)$.

Note 28. The **architecture** of the neural network is defined by the triplet $(\mathcal{V}, \mathcal{E}, \sigma_t)$.

Note 29. The neural network can be fully specified by the quadruplet $(\mathcal{V}, \mathcal{E}, \sigma_t, w)$.

Example 30. 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 31. 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 $(v_{t,j}, v_{t+1,i}) \notin \mathcal{E}$ and means that the link $v_{t,j} \rightarrow v_{t+1,i}$ is not in the network.

Note 32. 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 33. (Cont. Example 2) 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 Algorithm 24 as

- Input layer

$$o_{(0),i}(x) = \begin{cases} 1 & i = 0 \\ x_i & i = 1, ..., d \end{cases}$$

- Hidden layer

$$\begin{aligned}\alpha_{(1),j}\left(x\right) &= w_{(1),j,0}+\sum_{j}w_{(1),j,i}x_{i}\\ o_{(1),j}\left(x\right) &= \sigma_{(1)}\left(\alpha_{(1),j}\left(x\right)\right)\\ &= \sigma_{(1)}\left(w_{(1),j,0}+\sum_{j}w_{(1),j,i}x_{i}\right)\end{aligned}$$

- Output layer

$$\begin{aligned}\alpha_{(2),k}\left(x\right) &= w_{(2),k,0}+\sum_{j}w_{(2),k,j}o_{(1),k}\left(x\right)\\ o_{(2),k}\left(x\right) &= \sigma_{(2)}\left(\alpha_{(2),k}\left(x\right)\right)\\ &= \sigma_{(2)}\left(w_{(2),k,0}+\sum_{j}w_{(2),k,j}o_{(2),k}\left(x\right)\right)\end{aligned}$$

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

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

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 34. Activation functions σ_{t} are non-increasing functions often sigmoids or threshold functions. Their choice is problem-dependent. some examples

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

Example 35. Examples on the choice of the activation function at the output layer T:

- In the univariate regression problem with prediction rule $h\left(\cdot\right)\in\mathbb{R}$, and examples $z_{i}=\left(x_{i},y_{i}\right)\in\mathbb{R}^{d}\times\mathbb{R}$, we can choose $\sigma_{T}\left(\alpha\right)=\alpha$ to get $h\left(\alpha\right)=\sigma_{T}\left(\alpha\right)=\alpha$.
- In the binary logistic regression problem with prediction rule $h\left(\cdot\right)\in\left[0,1\right]$ and examples $z_{i}=\left(x_{i},y_{i}\right)\in\mathbb{R}^{d}\times\left\{0,1\right\}$, we can choose $\sigma_{T}\left(\alpha\right)=\frac{1}{1+\exp(-\alpha)}$ to get $h\left(\alpha\right)=\sigma_{T}\left(\alpha\right)=\frac{1}{1+\exp(-\alpha)}=\frac{\exp(\alpha)}{1+\exp(\alpha)}$.

3. Learning Neural Networks

Note 36. Assume we are interested in a prediction rule $h_{\mathcal{V},\mathcal{E},\sigma,w}:\mathbb{R}^{|V_{0}|}\rightarrow\mathbb{R}^{|V_{T}|}$ which is modeled as a feed-forward Neural Network with $(\mathcal{V},\mathcal{E},\sigma,w)$; that is

$$h_{\mathcal{V},\mathcal{E},\sigma,w}\left(x\right)=o_{T}\left(x\right)$$

where $o_T = \left(o_{T,1}, ..., o_{T,|V_T|}\right)^\top$ is according to the Algorithm 24.

Note 37. Learning the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of a neural network is a model selection task (recall the variable selection in linear regression).

Note 38. We assume that the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of the neural network $(\mathcal{V}, \mathcal{E}, \sigma, w)$ is fixed (given/known), 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 $\left\{w_{(t+1),i,j}\right\}$ where $w_{(t+1),i,j} := w\left(\left(v_{t,j}, v_{t+1,i}\right)\right)$.

Note 39. The class of hypotheses is

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

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

Notation 40. To simplify notation we will use h_w instead of $h_{\mathcal{V},\mathcal{E},\sigma,w}$ as $\mathcal{V}, \mathcal{E}, \sigma$ is fixed here.

Note 41. 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 42. 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 $z = (x, y)$.

Definition 43. Error function is a performance measure that can be defined as

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

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

Example 44. (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 $\{z_i = (x_i, y_i)\}$.

- The output activation function can be the identity function $\sigma_T\,(a) = a$. This is because $h_w\,(x) = o_T\,(x) = \sigma_T\,(\alpha_T\,(x))$ by Algorithm 24 and Note 36. Since h_w returns in $\mathbb{R}$ and the output activation $\alpha_T\,(x)$ returns in $\mathbb{R}$, then mapping $\sigma_T\,(a) = a$ suffices.
- A reasonable loss can be the Euclidean distance
$$\ell\,(w, z = (x, y)) = \frac{1}{2}\left(h_w\,(x) - y\right)^2$$
- Alternatively, if I consider a statistical model as

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

for some fixed $\beta > 0$, the implied loss is

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

- The Error function is

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

Example 45. (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 Algorithm 24 and Note 36. 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 reasonable loss can be an ellipsoidal norm

$$\ell\left(w, z=(x, y)\right)=\left(h_w\left(x\right)-y\right)^\top P\left(h_w\left(x\right)-y\right),\ \text{for some } P>0$$

- Alternatively, if I consider a statistical model as

$$y_i|x_i, w\overset{\text{ind}}{\sim}\mathrm{N}\left(\mu, P^{-1}\right),\ \text{where}\ \ \mu_i=h_w\left(x_i\right) \tag{3.3}$$

for some fixed precision matrix $P>0$, the implied loss is

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

- The Error function is

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

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

Example 46. (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 $\left\{z_i=\left(x_i, y_i\right)\right\}$ with $y_i\in\{0,...,q\}$ labeling the class.

- A suitable output activation function can be the logistic sigmoid
$$\sigma_T(a) = \frac{1}{1+\exp(-a)}$$
This is because $h_w(x) = o_T(x) = \sigma_T(\alpha_T(x))$ by Algorithm 24 and Note 36. Since h_w returns in $(0,1)$ and the output activation is $\alpha_T(x)$ in $\mathbb{R}$, then the aforesaid mapping suffices.
- A reasonable loss (among others) can be the cross entropy
$$\ell(w, z=(x,y)) = -y\log(h_w(x)) - (1-y)\log(1-h_w(x))$$
- If I consider a statistical model as
$$y_i|x_i, w \overset{\text{ind}}{\sim} \text{Bernoulli}(p_i), \text{ where } p_i = h_w(x_i) \tag{3.4}$$
with mass function
$$f(y|x,w) = h_w(x)^y (1-h_w(x))^{1-y}$$
the implied loss is
$$\begin{aligned}\ell(w, z=(x,y)) &= -\log(f(y|x,w)) \\ &= -y\log(h_w(x)) - (1-y)\log(1-h_w(x))\end{aligned}$$
- The Error function given a set of examples $\{z_i^* = (x_i^*, y_i^*)\}$ is
$$\begin{aligned}\text{EF}(w|z^*) &= \sum_{i=1}^{n} \ell(w, z_i^* = (x_i^*, y_i^*)) \\ &= -\sum_{i=1}^{n} y_i^* \log(h_w(x_i^*)) - (1-y_i^*)\log(1-h_w(x_i^*))\end{aligned}$$

Example 47. (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 $h_w = (h_{w,1}, ..., h_{w,q})^\top$, as a classification probability i.e, $h_{w,k}(x) = \Pr(x \text{ belongs to class } k)$. Assume a training data-set $\{z_i = (x_i, y_i)\}$ where y_i, with $y_{i,k} \in \{0,1\}$ and $\sum_{k=1}^{q} y_{i,k} = 1$, labeling the class to which the i-the example belongs. Then it is

- A suitable output activation function can be the softmax function
$$\sigma_T(a_k) = \frac{\exp(a_k)}{\sum_{k'=1}^{q} \exp(a_{k'})}, \text{ for } k = 1, ..., q \tag{3.5}$$
This is because $h_{w,k}(x) = o_{T,k}(x) = \sigma_{T,k}(\alpha_{T,k}(x))$ by Algorithm 24 and Note (36). Since h_w is essentially a probability vector and the output activation is $\alpha_{T,k}(x)$ in $\mathbb{R}$ the aforesaid mapping suffices. Unfortunately, (3.5) is invariant to additive transformations $a \leftarrow a + c$ i.e, $\sigma_T(a_k) = \sigma_T(a_k + \text{const})$.
- Another suitable output activation function can be
$$\tilde{\sigma}_T(a_k) = \frac{\exp(a_k)}{1 + \sum_{k'=1}^{q-1} \exp(a_{k'})}, \text{ for } k = 1, ..., q-1 \tag{3.6}$$

This is because $h_{w,k}(x) = o_{T,k}(x) = \sigma_{T,k}(\alpha_{T,k}(x))$ by Algorithm 24 and Note 36. Since h_w is essentially a probability vector and the output activation is $\alpha_{T,k}(x)$ in $\mathbb{R}$ the aforesaid mapping suffices as $h_{w,k}(x) = o_{T,k}(x)$ for $k = 1, ..., q-1$ and $h_{w,q}(x) = 1 - \sum_{k=1}^{q-1} h_{w,k}(x)$. 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 (3.5)i.e, $\tilde{\sigma}_T(a_k) \neq \tilde{\sigma}_T(a_k + \text{const})$.

- A reasonable loss can be the cross entropy

$$\ell(w, z = (x, y)) = -\sum_{k=1}^{q} y_k \log(h_{w,k}(x))$$

- If I consider a statistical model as

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

with mass function

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

the implied loss is

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

which is the cross-entropy.

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

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

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

4. Classical learning of Neural Network

Note 48. 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 learning problem.

Note 49. 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 50. Compute optimal $w^* \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the risk function $R_g(w)$

$$w^* = \arg\min_w (R_g(w)) = \arg\min_w (\text{E}_{z\sim g}(\ell(w, z))) \tag{4.1}$$

Problem 51. Compute optimal $w^{*} \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the empirical risk function $\hat{R}_S(w)$

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

4.2. Regularized loss optimization.

Note 52. 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 53. To address Note 52, 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 $v_{t,j} \rightarrow v_{t+1,i}$ is not active (essentially not in the neural network).

Problem 54. Compute $h_w \in \mathcal{H}_{\mathcal{V},\mathcal{E},\sigma}$ (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:

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

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

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

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

5. Stochastic gradient descent for classical training of NN

Note 56. 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 computational learning tool.

Note 57. To address Problem 50, 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 58. To address the Problem 54, the recursion of the SGD with batch size m is

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

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

Note 59. 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 3 (Handout 3) will not be effective due to the violation of the assumptions.

Note 60. Practical guidelines for the use of SGD in the learning problem of NN: Ref [3]

(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.

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

5.1. Error backpropagation.

Note 61. The error back-propagation 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 to train the FFNN.

Note 62. Error backpropagation assumes that $\ell (w, z)$ is differentiable at w for each value of z; $\nabla_w \ell (w, z)$ exists –however extensions exist.

Notation 63. 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 $t = 1, ...T$.

Algorithm 64. *(Error back-propagation)*

```
Requires: The FFNN $(\mathcal{V}, \mathcal{E}, \sigma, w)$ 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:
    Set $w_{t+1,j,i} = \begin{cases} w (v_{t,i}, v_{t+1,j}) & \text{if } (v_{t,i}, v_{t+1,j}) \in \mathcal{E} \\ 0 & \text{if } (v_{t,i}, v_{t+1,j}) \notin \mathcal{E} \end{cases}$
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) For $t = T$ ; For $i = 1, ..., k_T$
            Compute:
(5.1)           $\delta_{T,i} = \frac{\partial \ell_T}{\partial \alpha_{T,i}} (\alpha_T)$
        where $\ell_T$ is the loss as a function of the vector of activations $\alpha_T = (\alpha_{T,1}, ..., \alpha_{T,k_T})^\top$.
    (2) For $t = T - 1, ..., 1$ ; For $i = 1, ..., k_t$
            Compute:
(5.2)           $\delta_{t,i} = \frac{d}{d\alpha_{t,i}} \sigma_t (\alpha_{t,i}) \sum_{j=1}^{k_{t+1}} w_{t+1,j,i} \delta_{t+1,j}$
Output:
    For $t = 1, ..., T$; $i = 1, ..., k_t$ ; $j = 1, ..., k_{t-1}$
        If edge $(v_{t-1,j}, v_{t,i}) \in \mathcal{E}$, set:
(5.3)           $\frac{\partial \ell}{\partial w_{t,i,j}} (w, z) = \delta_{t,i} \, o_{t-1,j}$
```

Note 65. In (5.1) and (5.2), $\{\delta_{t,i}\}$ are called errors.

Example 66. 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:

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

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

comp:

$$\delta_{1,j} = \frac{\mathrm{d}}{\mathrm{d}\xi} \sigma_1(\xi) \Big|_{\xi = \alpha_{1,j}} \sum_{k=1}^{q} w_{2,k,j} \delta_{2,k} = \left(1 - \left(\underbrace{\sigma_1(\alpha_{1,j})}_{= o_{1,j}} \right)^2 \right) \sum_{k=1}^{q} w_{2,k,j} \delta_{2,k}$$

$$= \left(1 - (o_{1,j})^2 \right) \sum_{k=1}^{q} w_{2,k,j} \delta_{2,k}$$

Output:

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

Note 67. We show that the Algorithm 64 is valid. The Forward pass is valid due to Algorithm 24. Now, we work on the Backward pass.

- 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.
- Let $\ell_t(\cdot)$ be the loss function $\ell(w,z)$ as a function of the vector of the activations α_t at t-th layer.

- It is

$$\frac{\partial \ell}{\partial w_{t,i,j}}(w,z) = \frac{\partial \ell_t}{\partial \alpha_{t,i}}(\alpha_t)\frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} = \delta_{t,i}\frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} \tag{5.4}$$

  where

$$\delta_{t,i} = \frac{\partial \ell_t}{\partial \alpha_{t,i}}(\alpha_t)$$

  for $i = 1, ..., k_t$, $j = 1, ..., k_{t-1}$ and $t = 1, ..., T$.
- Regarding the second part of (5.4), it is

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

  for $t = 1, ..., T$ and $i = 1, ..., k_t$.
- I will try to find a way to compute δ_t given δ_{t+1} recursively from V_T to V_0.
  - For $t = T$, and $i = 1, ..., k_T$, it is

$$\delta_{T,i} = \frac{\partial \ell_T}{\partial \alpha_{T,i}}(\alpha_T)$$

  - For $t < T$, and $i = 1, ..., k_t$, it is

$$\delta_{t,i} = \frac{\partial \ell_t}{\partial \alpha_{t,i}}(\alpha_t) = \sum_{j=1}^{k_{t+1}} \frac{\partial \ell_{t+1}}{\partial \alpha_{t+1,j}}(\alpha_{t+1})\frac{\partial \alpha_{t+1,j}}{\partial \alpha_{t,i}} = \sum_{j=1}^{k_{t+1}} \delta_{t+1,j}\frac{\partial \alpha_{t+1,j}}{\partial \alpha_{t,i}}$$

    and

$$\begin{aligned}\frac{\partial \alpha_{t+1,j}}{\partial \alpha_{t,i}} &= \frac{\partial}{\partial \alpha_{t,i}}\sum_{s=1}^{k_t} w_{t+1,j,s}o_{t,s} = \frac{\partial}{\partial \alpha_{t,i}}\sum_{s=1}^{k_t} w_{t+1,j,s}\sigma_t(\alpha_{t,s}) \\ &= w_{t+1,j,i}\frac{\mathrm{d}}{\mathrm{d}\alpha_{t,i}}\sigma_t(\alpha_{t,i})\end{aligned}$$

    hence

$$\delta_{t,i} = \frac{\mathrm{d}}{\mathrm{d}\alpha_{t,i}}\sigma_t(\alpha_{t,i})\sum_{j=1}^{k_{t+1}} \delta_{t+1,j}w_{t+1,j,i}$$

Note 68. 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 from V_T to V_0.

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

Note 69. Recall (Handout 3, Algorithm 45), 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 70. The computation of the Hessian H_t can be done by using error propagation ideas for

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

We will not go further to exact computations.

Note 71. 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 72. Consider the regression problem with predictive rule $h : \mathbb{R}^d \rightarrow \mathbb{R}$ 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

$$\begin{aligned} H &= \frac{\mathrm{d}(\nabla_w \ell)}{\mathrm{d}w} = \frac{\mathrm{d}}{\mathrm{d}w}\left(\nabla_w h_w(x)(h_w(x) - y)^\top\right) \\ &= \left(\frac{\mathrm{d}}{\mathrm{d}w}\nabla_w h_w(x)\right)(h_w(x) - y)^\top + \nabla_w h_w(x)(\nabla_w h_w(x))^\top \end{aligned} \tag{5.5}$$

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 $\mathrm{EF}(w|\{z_i\}) = \sum_{i=1}^n \ell(w, z_i = (x_i, y_i))$ is

$$H_n = \sum_{i=1}^n \frac{\mathrm{d}(\nabla_w \mathrm{EF})}{\mathrm{d}w} \approx \sum_{i=1}^n \nabla_w h_w(x_i)(\nabla_w h_w(x_i))^\top \tag{5.6}$$

Note 73. (Cont. Note 72) To efficiently compute the inverse H_n^{-1} of (5.6) I can utilize Woodbury identity

$$\left(M + vv^\top\right)^{-1} = M^{-1} - \frac{(M^{-1}v)(v^\top M^{-1})}{1 + v^\top M^{-1}v} \tag{5.7}$$

No need to memorize Woodbury identity

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

$$(H_n)^{-1} = \left(\sum_{i=1}^{n} v_i v_i^\top\right)^{-1} = \left(\sum_{i=1}^{n-1} v_i v_i^\top + v_n v_n^\top\right)^{-1} = \left(H_{n-1} + v_n v_n^\top\right)^{-1} \tag{5.8}$$

$$= H_{n-1}^{-1} - \frac{\left(H_{n-1}^{-1} v_n\right)\left(v_n^\top H_{n-1}^{-1}\right)}{1 + v_n^\top H_{n-1}^{-1} v_n}$$

In practice I start with $H_0 = \epsilon I$ with $\epsilon > 0$ small, and iterate (5.8).

Note 74. Likewise and by modifying (5.5), one can compute the corresponding approximations for the classification problems or problems with different loss functions.

6. Bayesian Artificial Neural Networks

Note 75. Consider a feed-forward Neural Network with $(\mathcal{V}, \mathcal{E}, \sigma, w)$. Assume the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of a neural network is fixed/known, and interest lies in learning the weights $\{w_{t,i,j}\}$.

Problem 76. Assume 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 $y_i \in \mathcal{Y}$. The Bayesian NN model is then

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

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 77. 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 78. One could specify the sampling distribution based on the loss function $\ell(h_w(x), z = (x, y))$ as

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

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 79. The prior density of the weights may be specified based on some shrinkage term $J(w; \lambda)$ (Note 55) as

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

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 88, careless training of NN tends to produce over-fitted NN with large weights (in abs values).

Note 80. 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 81. The corresponding posterior (according to the Bayes theorem) is

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

and given 6.1 and 6.2, in log scale I get

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

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

Example 82. (Regression problem) Sampling distribution can be specified as in (6.1) based on (Euclidean) loss

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

which (based on (3.4)) can build a density

$$f\left(y_i|x_i, w\right) \propto \exp\left(-\ell\left(h_w\left(x_i\right), \left(x_i, y_i\right)\right)\right) = -\frac{\beta}{2}\left(h_w\left(x_i\right) - y_i\right)^2,$$

This implies a sampling distribution

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

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

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

which is based on the Ridge shrinkage term. This implies a prior

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

The resulted posterior density is

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

Example 83. (Binary classification problem) Sampling distribution can be specified as in (6.1) based on (cross-entropy) loss

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

which (based on (3.4)) can build a density

$$\begin{aligned} f\left(y_i|x_i, w\right) &\propto \exp\left(-\ell\left(h_w\left(x_i\right), \left(x_i, y_i\right)\right)\right) \\ &= -y_i\log\left(h_w\left(x_i\right)\right) - \left(1 - y_i\right)\left(1 - \log\left(h_w\left(x_i\right)\right)\right), \end{aligned}$$

This implies a sampling distribution

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

To encourage sparsity, prior can be specified as in (6.2) by using the LASSO shrinkage term

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

The resulted posterior density is

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

Note 84. 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

$$w^{(t+1)} = w^{(t)} + \eta_t\left(\frac{n}{m}\sum_{j\in\mathcal{J}^{(t)}} \nabla_w \log\left(f\left(z_j^{(t)}|w^{(t)}\right)\right) + \nabla_w \log\left(f\left(w^{(t)}\right)\right)\right) + \sqrt{2\eta_t\tau}\epsilon_t, \tag{6.6}$$

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 85. 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

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

(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

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

and compute

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

7. Theoretical aspects

Note 86. 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 87. Neural Networks can be highly "over-parametrized". E.g. they may consist of a large number of layers each of them having a large number of neurons with purpose to represent each feature/characteristic of the pattern to be recognized –this increases the number of unknown weights to be learned. Careless training may produce NN models with bad generalization predictive properties.

Note 88. Training of NN models corresponds to an iterative reduction of the Error Function defined on the training data-set. On the other hand 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 (while NN is being tuned), followed by an increase (while the NN starts to overfit). The reason is because the NN training is a a high-dimensional and big-data problem; hence during the NN training against the EF defined on the training data, the NN memorises too much info from the specific training dataset, overfits to it, and presents bad generalization performance. This overfitting is often associated to the production of weight values larger than needed for the NN learning rule to have a good generalization performance. It is desirable to avoid this over-fitting with purpose to obtain a NN based learning rule with good generalization performance.

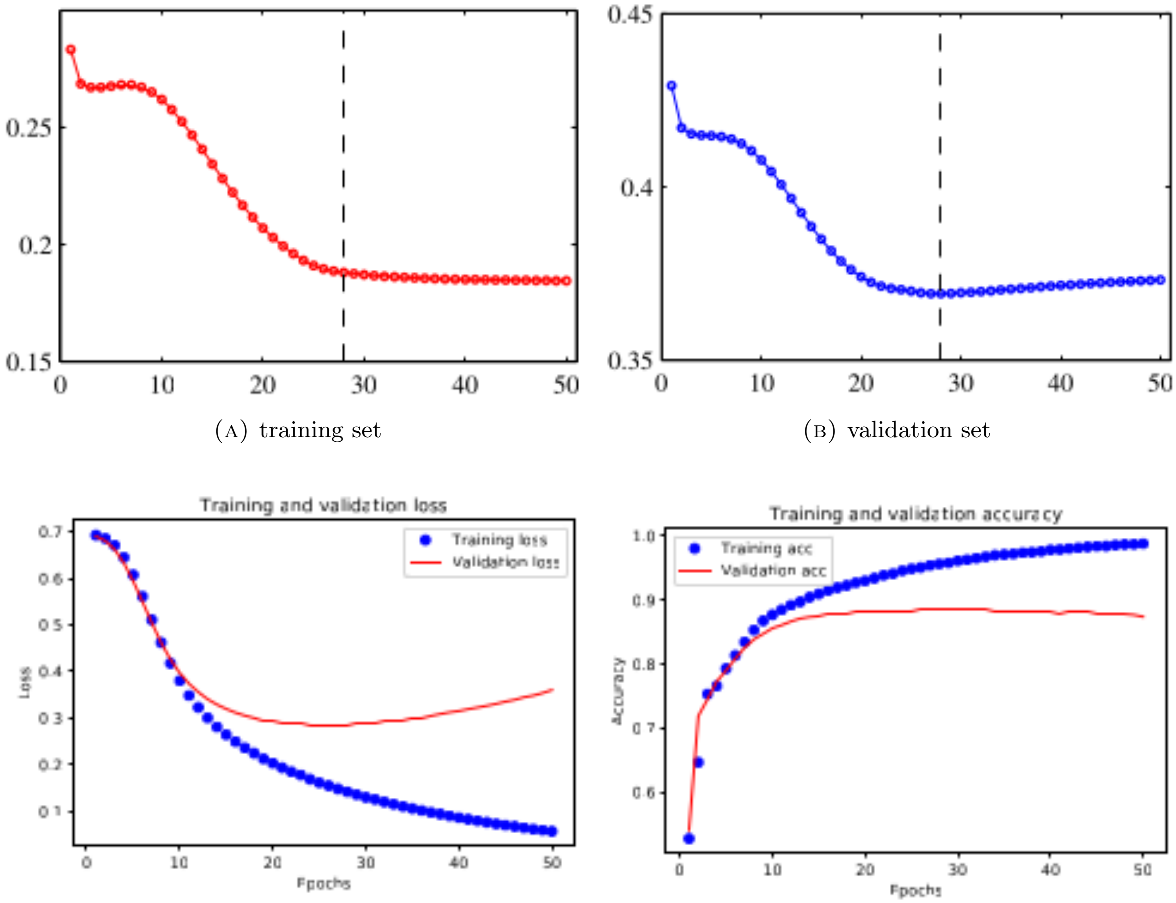
Note 89. **Early stopping** is a way of limiting the effective network complexity by halting training before a minimum of the training error has been reached. During training of NN against the MM defined on the training set, we monitor the NN performance against the EF defined on the validation data-set, and stop the training procedure just before the EF defined on the validation data set starts increasing (aka before overfitting) to prevent the model memorizing too much information about the training set.

Note 90. **Regularization** (Section 4.2) can 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 91. NN Learning Problems (E.g, Problems 50, 51, 54, and 76) are often non-identifiable with respect to weights $\{w_{t,i,j}\}$.

Example 92. 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.

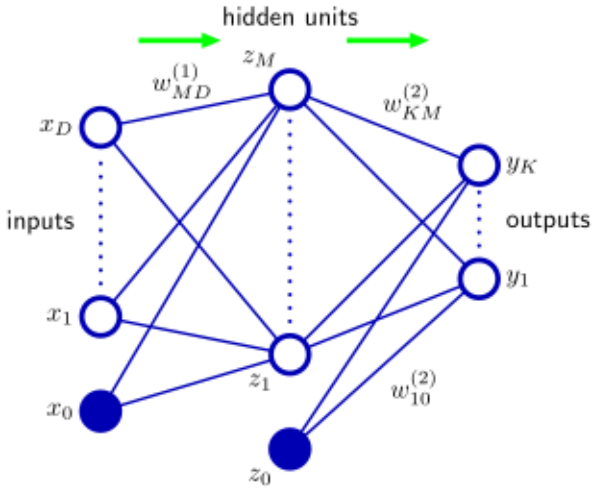


FIGURE 8.2. A FFNN

8.3. **Non-convexity.**

Note 93. 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 91. 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 94. To address local optima issue, run SGD/SGLD learning algorithms multiple times by initializing them with different seeds each time.

8.4. **Vanishing and exploding gradient.**

Note 95. In the Error back-propagation Algorithm 64, for the computation of $\left\{\frac{\partial \ell}{\partial w_{t,i,j}}\left(w, z\right)\right\}$ crucial quantities are the δ-derivatives the propagation of which follows the recursive rule (5.2), i.e.

$$\delta_{t,i} = \sum_{j=1}^{k_{t+1}} w_{t+1,j,i}\delta_{t+1,j}\sigma_t'\left(\alpha_{t,i}\right); \; t = T-1, ..., 1; \; i = 1, ..., k_t$$

δ_t-derivatives depend on the respective δ_{t+1}-derivatives, weights, and derivatives of the sigmoid in a multiplicative manner. Keeping on

$$\delta_{t,i} = \left[\sum_{j=1}^{k_{t+1}}\left(\sum_{s=1}^{k_{t+2}} \delta_{t+1,s} w_{t+2,s,j}\right)\sigma_{t+1}'\left(\alpha_{t+1,j}\right) w_{t+1,j,i}\right]\sigma_t'\left(\alpha_{t,i}\right)$$

we see that as the backpropagation algorithm flows backwards, the derivatives of the sigmoids as well as the weights are multiplied and the number of the involved products grows. This may cause the two problems: vanishing or exploding gradient.

Note 96. This is more powerful in lower layers.

Note 97. This exaggerates in deeper structures of NN.

8.4.1. *Vanishing gradient.*

Note 98. As $\sigma_t'\left(\cdot\right)$ are often smaller than 1 if the corresponding weights are not large enough, the gradients of the loss wrt the weights of the lower layers can take vanishingly small values and make training via SGD too slow.

Note 99. A remedy to mitigate vanishing gradient is to use rectified linear unit (ReLU) activation functions

$$\sigma_t\left(\alpha\right) = \max\left(0, \alpha\right) + \xi \min\left(0, \alpha\right)$$

where $\xi \in \mathbb{R}$ is a user specified parameter. For instance: if $\xi = 0$, we observe that $\sigma_t'\left(\alpha\right) = 1$ when $\alpha > 0$ hence it does not suffer from saturation. However the training may freeze when $\alpha < 0$ since $\sigma_t'\left(\alpha\right) = 0$, and hence we often consider some $|\xi| \approx$ small depending on the application.

8.4.2. *Exploding gradient.*

Note 100. If the values of the weights get very large values, this can lead to an explosion of the gradient estimates. This can affect the learning by pushing the estimates of the weights to the wrong region in the parameter space when SGD is used.

Note 101. Remedies of this are clipping the gradient in SGD (Note 39 in Handout 6: Bayesian Learning via Stochastic gradient and Stochastic gradient Langevin dynamics), the use of shrinkage terms in the learning problem (Section 4.2), or the use of Bayesian framework with appropriate priors (Section 6).

8.5. **Skip-layers.**

Note 102. 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 103. 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 33 and (2.4), (8.1) links the input to the output.

Note 104. In skip-layer FFNN cases, the error back propagation (Section 5.1) has to be adjusted properly.

Note 105. 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 106. 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 107. 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 $e \in \mathcal{E}$.

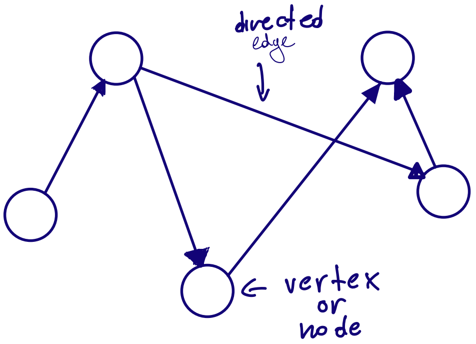


Figure A.1. A directed graph

Appendix B. About Partial derivatives

Note 108. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$.

Definition 109. The partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at the point $a = (a_1, \dots, a_n) \in U \subseteq \mathbb{R}^n$ 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 110. Essentially, Definition 109 says that the partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at the point $a = (a_1, \dots, a_n) \in U \subseteq \mathbb{R}^n$ 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 $g\left(h\right):=f\left(a_1,...,a_{i-1},a_i+h,a_{i+1},...,a_n\right)$ at value 0.

Example 111. Consider a function f with $f\left(x_1,x_2\right)=x_1^2+x_1x_2^3$. Comput is partial derivatives at $a=\left(2,3\right)^\top$; i.e. $\frac{\partial f}{\partial x_1}\left(a\right)$ and $\frac{\partial f}{\partial x_2}\left(a\right)$.

Solution. It is

$$\begin{aligned}\frac{\partial f}{\partial x_1}\left(a\right)&=\left.\frac{\partial f}{\partial x_1}\left(x\right)\right|_{x=a}=\left.\frac{\mathrm{d}}{\mathrm{d}x_1}\left(x_1^2+x_1x_2^3\right)\right|_{x=a}=\left.2x_1+x_2^3\right|_{x=a}\\&=2a_1+a_2^3=4+27=31\\\frac{\partial f}{\partial x_2}\left(a\right)&=\left.\frac{\partial f}{\partial x_2}\left(x\right)\right|_{x=a}=\left.\frac{\mathrm{d}}{\mathrm{d}x_2}\left(x_1^2+x_1x_2^3\right)\right|_{x=a}=\left.2x_1x_2^2\right|_{x=a}\\&=2a_1a_2^2=4+27=72\end{aligned}$$