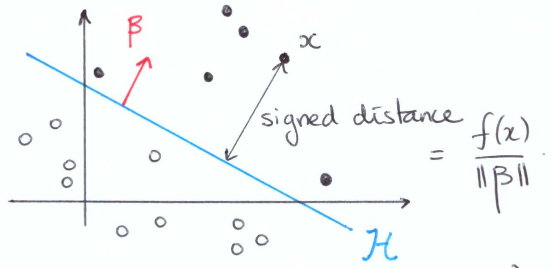
SL : NEURAL NETWORKS

I. PERCEPTRON

The perceptron is one of the oldest algorithms in machine learning (Rosenblatt (1958)). Let $\mathcal{L}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be our learning sample, where $y_i \in \{-1, +1\}$. The perceptron algorithm finds a separating hyperplane $\mathcal{H}$ to classify observations.

$$\mathcal{H} = \{ x \in \mathbb{R}^d \mid \beta_0 + \beta^t x = 0 \}$$

Put $f(x) = \beta_0 + \beta^t x$.



(See SL: LINEAR CLASSIFICATION)

$\Rightarrow$ The sign of $f(x) = \beta_0 + \beta^t x$ indicates on which side of the hyperplane x lies.

$\Rightarrow$ If $(x_i, y_i = +1)$ is mislassified, then $\beta_0 + \beta^t x_i < 0$ while

if $(x_i, y_i = -1)$ is misclassified, $\beta_0 + \beta^t x_i > 0$.

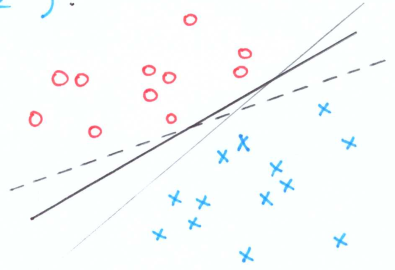
In both cases, $y_i(\beta_0 + \beta^t x_i) < 0$ if (x_i, y_i) is misclassified. We would like all observations to be such that $y_i(\beta_0 + \beta^t x_i) > 0$.

We incorporate the intercept β_0 into the vector β by adding an extra coordinate equal to 1 to the vector of predictors x_i.

$\Rightarrow$ Our goal is to find β such that for all i, $y_i \langle \beta, x_i \rangle > 0$

We assume that the data is linearly separable, so that such a β exists (but not unique).

[Different hyperplanes returning zero training error] $\rightarrow$



The objective we wish to optimize is

$$(*) \quad \boxed{\sum_{i=1}^{n} \mathbb{1}(y_i \langle \beta, x_i \rangle < 0)}$$

$\equiv$ empirical risk minimization over the space $\mathcal{F}$ of linear functions. The final classification rule is then $\text{sign}(\langle \beta, x_i \rangle)$.

The objective $(*)$ is discontinuous and non-convex (zero-one loss). We may rewrite our objective as minimizing

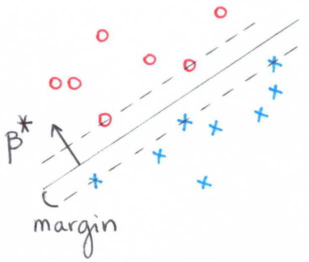
$$(**) \quad \boxed{\sum_{i=1}^{n} \max(0, -y_i \langle \beta, x_i \rangle)}$$

$\uparrow$ If (x_i, y_i) is misclassified, $-y_i \langle \beta, x_i \rangle$ is positive, making $(**) > 0$. Linear separability ensures the existence of a β^* such that $(**) = 0$

- By scaling β^*, we obtain many minimizers to (**). To fix this, we have two solutions:

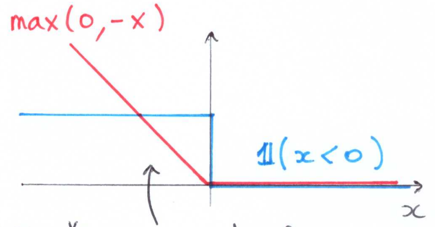
  (i) Fix $\|\beta^*\| = 1$, and let $\gamma > 0$ be the largest value (aka margin) such that $y_i \langle \beta^*, x_i \rangle \geq \gamma$.

  (ii) Return β^* = the smallest norm vector such that $y_i \langle \beta^*, x_i \rangle \geq 1$.



$\uparrow$
We consider (ii), but the two options are equivalent: rescale β^* by its norm in (ii) to get $\gamma = 1/\|\beta^*\|$.

- The criteria (**) is not smooth, but convex, unlike (*)



$\Rightarrow$ Use a SGD algorithm to minimize (**):

Not a surrogate loss: does not lie above $\mathbb{1}(x<0)$, see SL: CONVEX RELAXATION

$$\boxed{\beta_{j+1} \leftarrow \beta_j - \eta_j \tilde{\nabla}_\beta \, \ell(y_i, \langle \beta_j, x_i \rangle)}$$

estimate at iteration j

pick one observation (x_i, y_i) at random in $\mathcal{L}_n$, and update β_j in the opposite direction of a subgradient of $\ell(y_i, \langle \beta, x_i \rangle) := \max(0, -y_i \langle \beta, x_i \rangle)$ evaluated at β_j.

(see SL: GRADIENT DESCENT ALGORITHMS)

The vector

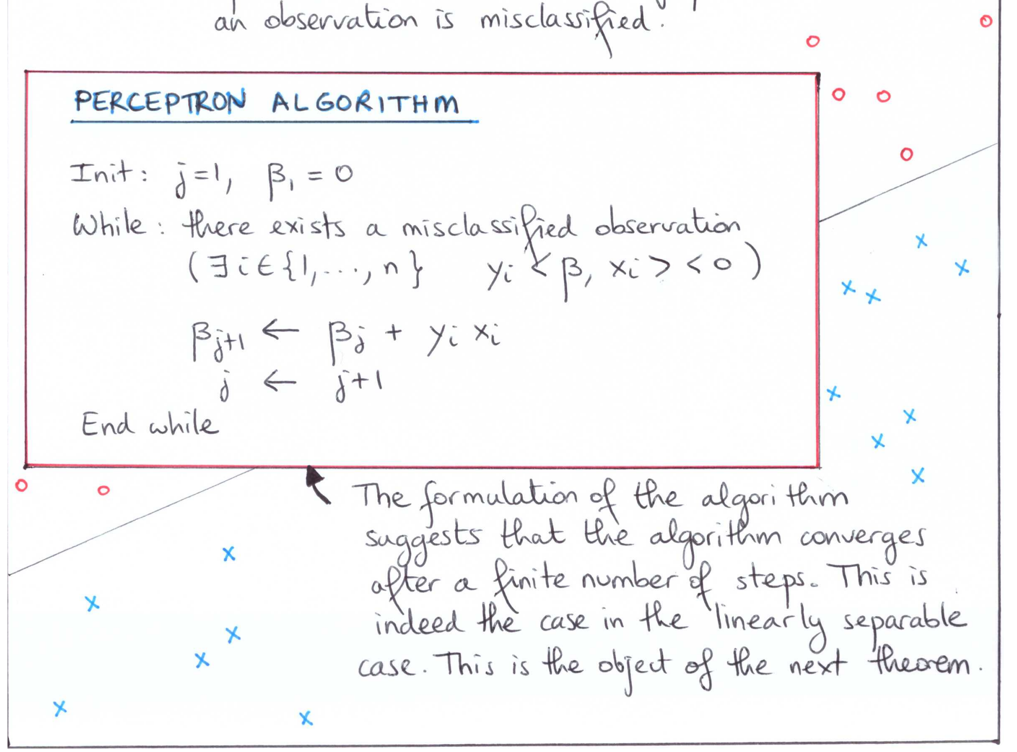
$$v = \begin{cases} 0 & \text{if} \quad y \langle \beta, x \rangle \geq 0 \quad \leftarrow \text{correctly classified} \\ -yx & \text{if} \quad y \langle \beta, x \rangle < 0 \quad \leftarrow \text{misclassified.} \end{cases}$$

is a subgradient of $\max(0, -y\langle \beta, x \rangle)$.

Taking $\eta_j \equiv 1$ (we justify this later), the SGD update becomes

$$\boxed{\beta_{j+1} \leftarrow \beta_j + \begin{cases} 0 & \text{if } (x_i, y_i) \text{ is correctly classified} \\ y_i x_i & \text{if } (x_i, y_i) \text{ is misclassified.} \end{cases}}$$

$\uparrow$
In practice, (x_i, y_i) is not chosen randomly: we circle through the data from $i = 1, \dots, n$ and update the current estimate of β each time an observation is misclassified.



PERCEPTRON ALGORITHM

```
Init: $j = 1$, $\beta_1 = 0$
While: there exists a misclassified observation
       $(\exists i \in \{1, \dots, n\} \quad y_i \langle \beta, x_i \rangle < 0)$
    $\beta_{j+1} \leftarrow \beta_j + y_i x_i$
    $j \leftarrow j + 1$
End while
```

The formulation of the algorithm suggests that the algorithm converges after a finite number of steps. This is indeed the case in the linearly separable case. This is the object of the next theorem.

Theorem: Let $\mathcal{L}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$, $y_i \in \{\pm 1\}$, and $B = \max_i \|x_i\|$.
Denote by β^* the vector of smallest norm such that $\forall i$, $y_i \langle \beta^*, x_i \rangle \geq 1$ holds. ($\Rightarrow$ data is linearly separable).
Then the perceptron algorithm stops after at most $B^2 \|\beta^*\|^2$ iterations.

proof: Let β^* such that $\forall i$ $y_i \langle \beta^*, x_i \rangle \geq 1$.
At iteration j, suppose (x_i, y_i) is misclassified:

$$\beta_{j+1} \leftarrow \beta_j + \eta_j x_i y_i$$

Take $B = \max_{1 \leq i \leq n} \|x_i\|$ & consider

$$\beta_{j+1} - \eta_j B^2 \beta^* = \beta_j - \eta_j B^2 \beta^* + \eta_j x_i y_i$$

$$\|\beta_{j+1} - \eta_j B^2 \beta^*\|^2 = \|\beta_j - \eta_j B^2 \beta^*\|^2 + \eta_j^2 \|x_i y_i\|^2 + 2 \langle \beta_j - \eta_j B^2 \beta^*, \eta_j x_i y_i \rangle$$

↑ Since (x_i, y_i) is misclassified, we have $y_i \langle \beta_j, x_i \rangle < 0$. Thus:

$$\leq \|\beta_j - \eta_j B^2 \beta^*\|^2 + \eta_j^2 \|x_i y_i\|^2) \leq B^2$$
$$+ -2 \eta_j^2 B^2 \underbrace{y_i \langle \beta^*, x_i \rangle}_{\geq 1}$$
$$\leq \|\beta_j - \eta_j B^2 \beta^*\|^2 - 2 \eta_j^2 B^2$$

$\Rightarrow$ The square distance between β_j and $\eta_j B^2 \beta^*$ is reduced by an amount $\eta_j^2 B^2$ at each iteration. $\Rightarrow$ The perceptron algorithm must terminate after no more than $\dfrac{\|\eta_j B^2 \beta^*\|^2}{\eta_j^2 B^2} = B^2 \|\beta^*\|^2$ iterations.

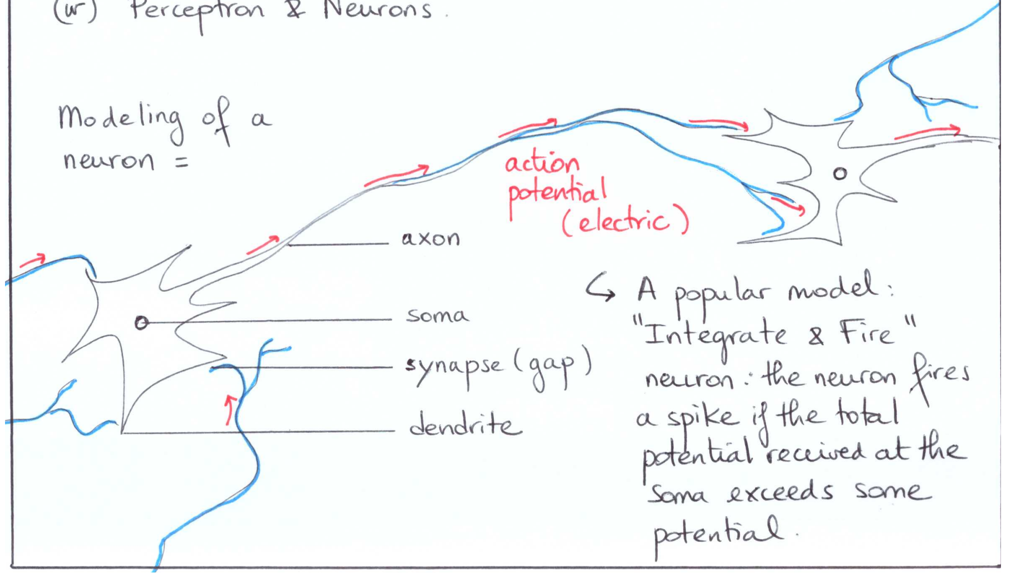
- Remark: (i) The learning rate η_j cancels out in the proof of the theorem $\Rightarrow$ we can set $\eta_j = 1$ without loss of generality in the update $\beta_j \leftarrow \beta_j + \eta_j x_i y_i$.

(ii) Equivalently, setting $\|\beta^*\| = 1$ and letting $\gamma > 0$ be the largest margin such that $y_i \langle \beta^*, x_i \rangle \geq \gamma$, the algorithm terminates in a maximum of B^2/γ^2 number of iterations $\Rightarrow$ the smaller the margin, the longer the algorithm may take to converge.

(iii) If the data is not linearly separable, the perceptron runs forever, and circles may develop. Instead, one opts for a surrogate loss such as $\max(0, 1-x)$ instead of the loss $\max(0, -x)$. The use of $\max(0, 1-x)$ leads to support vector machine (see SL: SVM), and returns a separating hyperplane maximizing the margin γ, allowing some observations to be misclassified.

(iv) Perceptron & Neurons.

Modeling of a neuron:



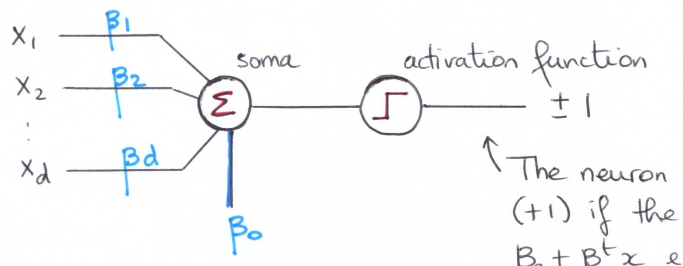
↳ A popular model: "Integrate & Fire" neuron: the neuron fires a spike if the total potential received at the soma exceeds some potential.

A basic model of a neuron:

- Background activity $= \beta_0$ (the neuron may fire a spike even if no message is received from neighboring neurons).
- Soma potential is $= \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d = \beta_0 + \beta^t x$

weighted sum of the potentials received at the synapses.

$X = (x_1, \dots, x_d) \in \mathbb{R}^d$



↑ The neuron fires a spike (+1) if the soma potential $\beta_0 + \beta^t x$ exceeds some threshold γ : if $\beta_0 + \beta^t x \geq \gamma$

$\Rightarrow$ Taking $\gamma = 0$, learning the parameters (β_0, β) is the same as separating the points $\{(x_1, y_1), \dots, (x_n, y_n)\}$

Ex: For the following three cases, can we determine weights (β_0, β) such that the neuron fires accordingly?

x_1	x_2	OR
0	0	-1
0	1	1
1	0	1
1	1	1

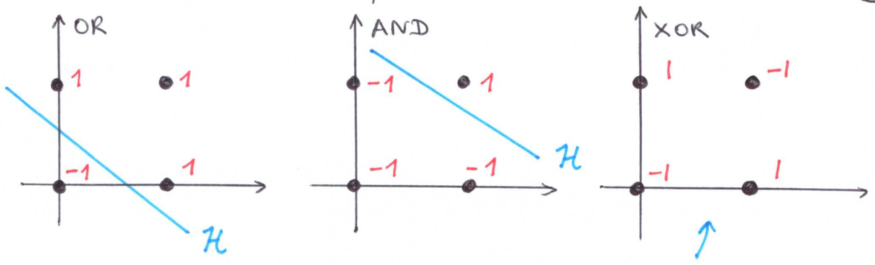
x_1	x_2	AND
0	0	-1
0	1	-1
1	0	-1
1	1	1

x_1	x_2	XOR
0	0	-1
0	1	1
1	0	1
1	1	-1

"Boolean" functions

↑ One neuron cannot reproduce this pattern.

This is better seen on a picture:



↑ points are not linearly separable.

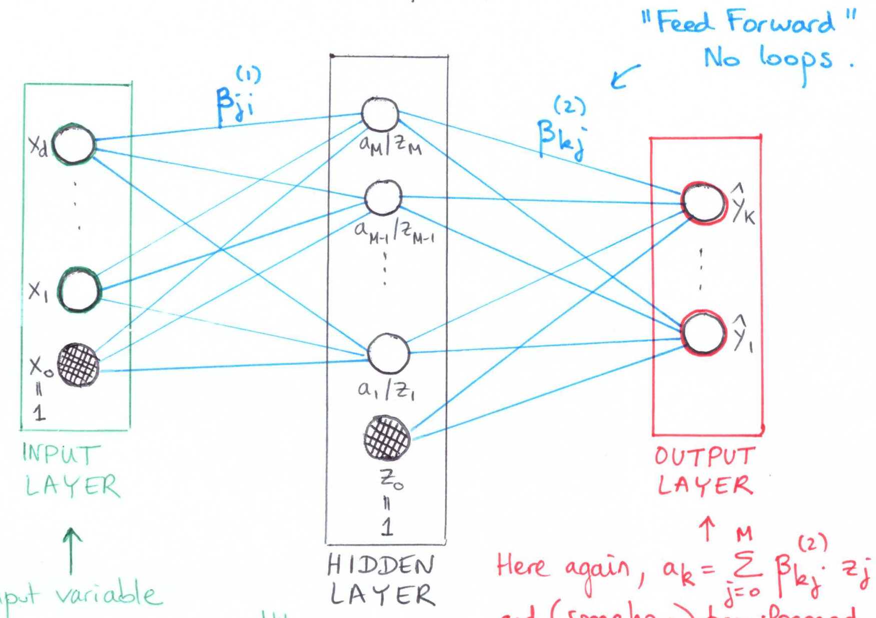
To learn more complex patterns (aka non linear), we combine / stack neurons togethers, leading to the "multi-layer perceptron", a special class of feed-forward neural network.

II_ FEED-FORWARD NEURAL NETWORK

II.1. Definitions / the model

The term 'Neural Network' refers to attempts to find reasonable mathematical descriptions of the flow of information in biological systems, such as in the human brain. The goal is not to make use of the best model of a neuron, but to use models that lead to efficient computations and estimation of the parameters when dealing with large datasets. Neural Network were studied in the 80s and the 90s, but with little practical success. Recent algorithmic advances and increase in computational power make these models extremely popular these days. ($\rightarrow$ "Deep Learning", "Convolutional Nets", ...).

A feedforward neural network is described using a directed acyclic graph, whose edges represent neurons, and vertices connections between neurons. The network is organised in layers, and typically look like:



Input variable $x = (x_0, x_1, -, x_d) \in \mathbb{R}^{d+1}$, $x_0 = 1$

feeds the network

Here again, $a_k = \sum_{j=0}^{M} \beta_{kj}^{(2)} z_j$, and (somehow) transformed to produce outputs $\hat{y}_k = \sigma(a_k)$.

Input values are linearly combined to produce:

$$a_j = \underset{\substack{\uparrow \\ \text{weights}}}{\sum_{i=0}^{d} \beta_{ji}^{(1)}} x_i \quad , \; j = 1, -, \underset{\substack{\uparrow \\ \text{\# neurons in the hidden layer.}}}{M}$$

Names of the Network:

"3-layer" Network (# layers)

"2-layer" Network (# layers of weights)

The a_j's are then non-linearly transformed to produce $z_j = h(a_j)$

aka: activation function.

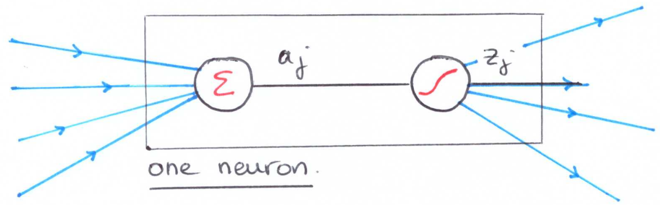
"smooth" (sigmoid)
differentiable (tanh)

Combining the two layers of the network, the k-th output is given by

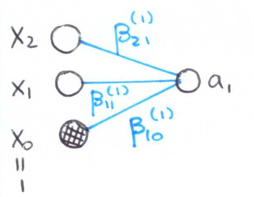
$$\hat{y}_k = f_k(x) = \sigma\left(\sum_{j=0}^{M} \beta_{kj}^{(2)} \, h\left(\sum_{i=0}^{d} \beta_{ji}^{(1)} x_i \right)\right)$$

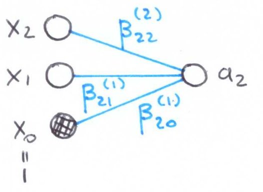
$$= \text{non-linear function of } x = (x_0, x_1, -, x_d).$$

× Remarks: Taking h linear reduces the neuron network (NN) to a linear model. The improvement of NNs lies in the presence of non-linear activation functions. Note that in a NN, the activation functions are taken continuous & differentiable, as opposed to the perceptron, which uses discontinuous step functions.



× Example: learning an XOR function with a three-layer network. Consider:





x_1	x_2	z_1	z_2
0	0	$\simeq$ 0	$\simeq$ 1
0	1	$\simeq$ 0	$\simeq$ 0
1	0	$\simeq$ 0	$\simeq$ 0
1	1	$\simeq$ 1	$\simeq$ 0

$\beta_{10}^{(1)} = -30$

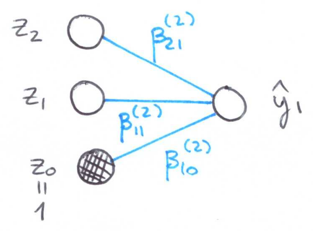
$\beta_{11}^{(1)} = \beta_{21}^{(1)} = +20$

$\beta_{20}^{(1)} = 20$

$\beta_{21}^{(1)} = \beta_{22}^{(2)} = -30$

$z_j = \sigma\left(\beta_{j0}^{(1)} + \beta_{j1}^{(1)} x_1 + \beta_{j2}^{(1)} x_2\right)$

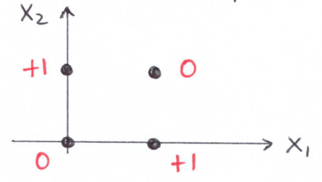
σ = sigmoid.



z_1	z_2	$\hat{y}_1$
0	1	0
0	0	1
0	0	1
1	0	0

$\beta_{10}^{(2)} = 10$

$\beta_{11}^{(2)} = \beta_{12}^{(2)} = -20$ $\Rightarrow$ We reproduced the pattern:



(points are not linearly separable)

II.2. Weak & Strong Consistency.

Given a learning sample $\mathcal{L}_n = \{(X_1,Y_1),\dots,(X_n,Y_n)\}$, we consider empirical risk minimization over the class of neural network, with a square loss:

$$\boxed{f_n \in \underset{f \in \mathcal{F}_n}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \left(Y_i - f(X_i)\right)^2,}$$

where

$$\mathcal{F}_n := \left\{ \sum_{j=1}^{k_n} \beta_j^{(2)} \sigma\left(\sum_{i=1}^{d} \beta_{ji}^{(1)} x_i + \beta_{j0}^{(1)} \right) + \beta_0^{(2)} \;\middle|\; k_n \geq 1,\ \sum_{j=0}^{k_n} |\beta_j^{(2)}| \leq \beta_n,\ \beta_j^{(2)} / \beta_{ji}^{(1)} \in \mathbb{R} \right\}$$

Consider one output value (drop subscript k)

The # of neurons in the hidden layer is allowed to depend on n.

Under some restrictions on k_n and β_n, we obtain universally consistent neural network estimates:

Theorem [Györfi et al - Theorem 16.1]

If k_n and β_n satisfy

(i) $k_n \to +\infty$, $\beta_n \to \infty$, $\dfrac{k_n \beta_n^4 \log(k_n \beta_n^2)}{n} \to 0$,

then

$$\mathbb{E} \int \left(f_n(x) - r(x) \right)^2 \mathbb{P}_X(dx) \longrightarrow 0 \quad \text{as } n \to +\infty$$

for all distributions of (X,Y) such that $\mathbb{E}\, Y^2 < +\infty$.
(weak universal consistency).

(ii) If in addition $\exists\, \delta > 0$ s.t. $\beta_n^4 / n^{1-\delta} \to 0$, then

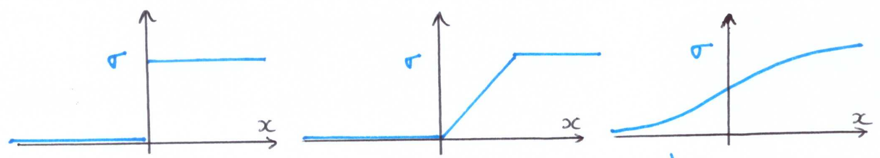
$$\int \left(f_n(x) - r(x) \right)^2 \mathbb{P}_X(dx) \longrightarrow 0 \quad \text{a.s. as } n \to +\infty$$

(strong universal consistency).

For this result to hold, we need σ in $\mathcal{F}_n$ to be such that $\lim_{x\to-\infty} \sigma(x) = 0$, and $\lim_{x\to+\infty} \sigma(x) = 1$.

As usual, $r(x) = \mathbb{E}(Y|X=x)$

Ex:



threshold: $\mathbb{1}(x \in [0,\infty))$

ramp: $x\,\mathbb{1}(x \in [0,1]) + \mathbb{1}(x \geq 1)$

arctan: $\frac{1}{2} + \frac{1}{\pi} \arctan x$

gaussian $\dfrac{1}{\sqrt{2\pi}} \displaystyle\int_{-\infty}^{x} e^{-u^2/2} du$, logistic $(1+e^{-x})^{-1}$

... / ...

x Remark: this result follows from the approximation power of neural networks: for every continuous function $f: \mathbb{R}^d \to \mathbb{R}$, $\forall \varepsilon > 0$, there exists a neural network

$$g(x) := \sum_{j=1}^{M} \beta_j^{(2)} \sigma\left(\sum_{i=1}^{d} \beta_{ji}^{(1)} x_i + \beta_{j0}^{(1)} \right) + \beta_0^{(2)}$$

such that

$$\sup_{x \in K} | f(x) - g(x) | < \varepsilon .$$

↖ for a given compact $K \subset \mathbb{R}^d$.

⇒ In theory, a neural network can learn any non-linear (but continuous) input-output relationship. NNs are said to be universal approximators.

The computation of the empirical risk minimizer is performed using approximating procedures, using gradient descent algorithms. By far the most popular algorithm is BACKPROPAGATION. It converges to a local minimum, with no theoretical guarantee that a global minimum is reached. (see section II.3).

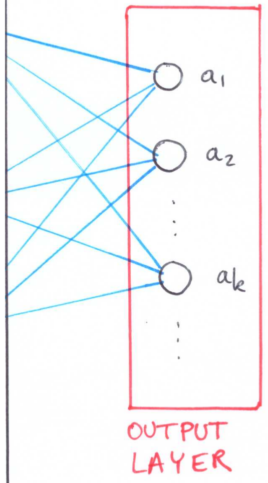
x Remark: rates of convergence may be obtained; see Section 16.3 in Györfi et al.

↳ In terms of estimation & approximation errors, the theorem on page 12 tells us that provided the number of neurons tends to ∞ at a right rate, and under some constraints on the weights, the approximation error tends to 0 as $n \to +\infty$. In addition, the estimation error vanishes as well, so that the class of neural networks is PAC learnable, with finite VC dimension, in the case of binary classification.

Specifically, for functions $f_n \in \mathcal{F}_n$, we construct the hard classifier as $\operatorname{sign} f_n$. The VC dimension of the class of hard classifiers $\{ \operatorname{sign} f_n \mid f_n \in \mathcal{F}_n \}$ can be seen to be of the order of the number N of coefficients to estimate in the network: $O(N \log N)$ [see Theorem 20.6 in Shalev-Schwartz & Ben-David (2014)].

II.3. Backpropagation.

Let a_k denote the k-th output of the neural network.



(see page 9)

Output $a_1, a_2, \dots$ are then transformed to produce a final estimate $\hat{y}_1, \hat{y}_2, \dots$. Depending on the nature of the prediction task, the transform of a_k takes a different expression. We discuss three important cases: (i) Regression, (ii) Binary Classification and (iii) K-class classification.

(i) Regression. For simplicity, we consider a single output value $y \in \mathbb{R}$: the neural network has a single neuron in its output layer. Given a learning sample $\mathcal{L}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ and a square loss function, the goal is to minimize over the parameters $\{\beta\}$ the empirical risk given by $\hat{R}_n(f) = \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$,

where $f \in$ neural network with weights $\{\beta\}$.

- Useful for later: $\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} E_i$, where $E_i = \frac{1}{2}(y_i - f(x_i))^2$. In this notation,

$$\frac{\partial E_i}{\partial a_1} = \frac{1}{2}\frac{\partial}{\partial a_1}(y_i - f(x_i))^2$$
$$= \frac{1}{2}\frac{\partial}{\partial a_1}(y_i - a_1)^2$$
$$= a_1 - y_i.$$
$$= f(x_i) - y_i.$$

Taking $\sigma(a_1) = a_1$, the identity activation function in the case of regression.

In the case where the response variable $Y \in \mathbb{R}^K$, the network has K output neurons $a_1, .., a_K$. The target values $Y_1, .., Y_K$ are estimated using $f_1(X), .., f_K(X)$, where as in the case of a single output, the identity activation function is taken for the output layer.

The empirical risk becomes $\frac{1}{n}\sum_{i=1}^{n} E_i$, with

$$E_i = \frac{1}{2}\|y_i - \underset{\uparrow}{f(x_i)}\|^2 = \frac{1}{2}\sum_{k=1}^{K}(\underset{\nwarrow}{y_{ik}} - f_k(x_i))^2$$

$f(x_i) := (f_1(x_i), \dots, f_K(x_i))^t$ $\quad$ $y_i = (y_{i1}, \dots, y_{iK})^t$

We compute similarly that

$$\frac{\partial E_i}{\partial a_k} = a_k - y_{ik} = f_k(x_i) - y_{ik} \quad \text{———} \quad (1)$$

RK: The objective E_i is usually a non-convex function of the model parameters $\beta_{jm}^{(\ell)}$.

(ii) Binary Classification. The network has a single output neuron a_1. We consider a logistic activation function $\sigma(a) = \frac{e^a}{1+e^a}$. It follows that

$$f(x_i) = \mathbb{P}(Y_i = 1 \mid X_i = x_i) = \sigma(a_1).$$

$$\Leftrightarrow \log\left\{\frac{\mathbb{P}(Y_i = 1 \mid X_i = x_i)}{\mathbb{P}(Y_i = 0 \mid X_i = x_i)}\right\} = a_1$$

$$\Leftrightarrow \mathbb{P}(Y_i = \underset{\in\{0,1\}}{\underset{\uparrow}{y_i}} \mid X_i = x_i) = \sigma(a_1)^{y_i}(1 - \sigma(a_1))^{1-y_i}$$
$$= f(x_i)^{y_i}(1 - f(x_i))^{1-y_i}.$$

If we have a total of K separate binary classifications to perform, then

$$\mathbb{P}(Y_i = \underset{=(y_{i1},..,y_{iK})\in\{0,1\}^K}{\underset{\uparrow}{y_i}} \mid X_i = x_i) = \prod_{k=1}^{K}\sigma(a_k)^{y_{ik}}(1 - \sigma(a_k))^{1-y_{ik}}$$

Taking negative the logarithm and summing over all observations in the learning sample, we obtain the empirical risk $\sum_{i=1}^{n} E_i$ with

$$E_i = -\sum_{k=1}^{K}\left(y_{ik}\log\sigma(a_k) + (1 - y_{ik})\log(1 - \sigma(a_k))\right).$$

- Useful for later:

$$\frac{\partial E_i}{\partial a_k} = -\sum_{\ell=1}^{K}\left\{y_{i\ell}\underbrace{\frac{\partial\log\sigma(a_\ell)}{\partial a_k}}_{k=\ell} + (1 - y_{i\ell})\underbrace{\frac{\partial\log(1-\sigma(a_\ell))}{\partial a_k}}_{k=\ell}\right\}$$

$$= \frac{\sigma'(a_k)}{\sigma(a_k)} = 1 - \sigma(a_k) \qquad = -\frac{\sigma'(a_k)}{1-\sigma(a_k)} = -\sigma(a_k)$$

$$\Rightarrow \boxed{\begin{aligned}\frac{\partial E_i}{\partial a_k} &= -\{y_{ik}(1-\sigma(a_k)) + (1-y_{ik})(-\sigma(a_k))\} \\ &= \sigma(a_k) - y_{ik} \qquad (2)\end{aligned}}$$

(iii) <u>K-class classification</u>. The network possesses K output neurons. We use a 1-to-K coding scheme for the response variable $Y = (Y_1, \dots, Y_K) \in \{0,1\}^K$, where $Y_k = 1$ iff X is in class k, and 0 otherwise.
The network outputs $f_1(x), \dots, f_K(x)$ are interpreted as $f_k(x) = \mathbb{P}(Y_k = 1 \mid X = x)$, so that the empirical risk is $\sum_{i=1}^n E_i$ with

$$E_i = -\sum_{k=1}^K y_{ik} \log \underset{\uparrow}{f_k(x_i)}$$

Functions f_k are related to a_k via a softmax activation function:

$$f_k(x) = \sigma(\underset{\substack{\uparrow \\ a=(a_1,\dots,a_K)}}{a}) = \frac{e^{a_k}}{\sum_{j=1}^K e^{a_j}},$$

so that $0 \leq f_k \leq 1$, and $\sum_{k=1}^K f_k = 1$

- <u>Useful for later</u>:

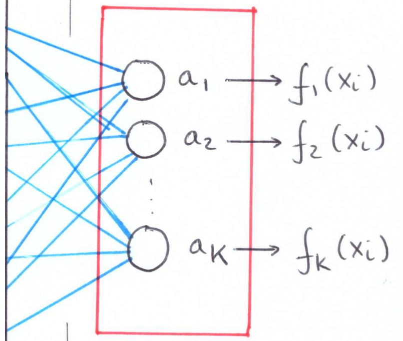
$$\frac{\partial E_i}{\partial a_k} = -\sum_{\ell=1}^K y_{i\ell} \frac{\partial \log f_\ell(x_i)}{\partial a_k}$$

$$= -\sum_{\ell=1}^K y_{i\ell} \left\{ \frac{\partial}{\partial a_k}\left(a_\ell - \log \sum_{j=1}^K e^{a_j}\right) \right\}$$

$$\frac{\partial E_i}{\partial a_k} = -y_{ik} - \sum_{\ell=1}^K y_{i\ell} \left\{ \underbrace{\frac{-e^{a_k}}{\sum_{j=1}^K e^{a_j}}}_{-f_k(x_i)} \right\}$$

$$\boxed{\frac{\partial E_i}{\partial a_k} = f_k(x_i) - y_{ik} \qquad (3)}$$

<u>Summarizing</u>: Network with K output neurons:



(i) <u>Regression</u> $f_k(x_i) = a_k$

(ii) <u>K indpt binary classification tasks</u>:

$$f_k(x_i) = \frac{e^{a_k}}{1+e^{a_k}} \quad \text{(logistic)}$$

(iii) <u>K-class classification</u>:

$$f_k(x_i) = \frac{e^{a_k}}{\sum_{j=1}^K e^{a_j}} \quad \text{(softmax)}$$

In each case, $\frac{\partial E_i}{\partial a_k} = f_k(x_i) - y_{ik}$, where

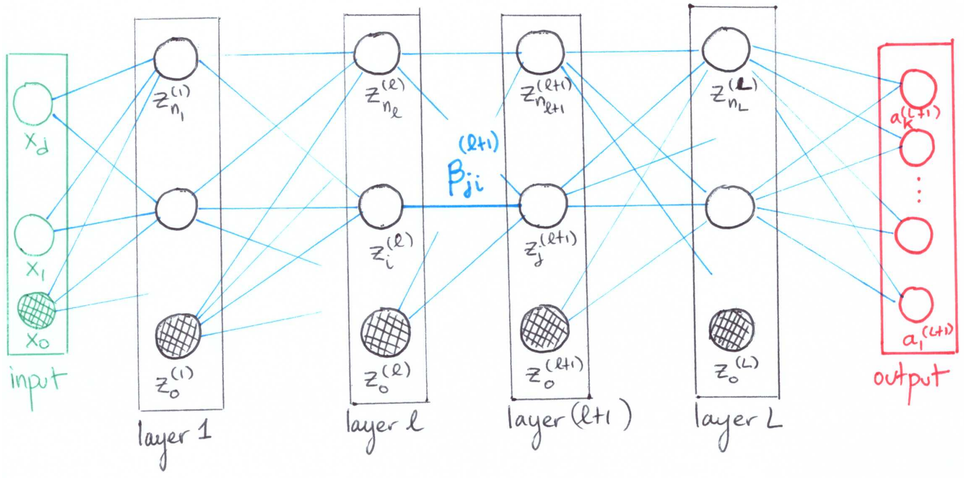
- $E_i = \frac{1}{2}\sum_{j=1}^K (y_{ij} - f_j(x_i))^2$ (case (i))
- $E_i = -\sum_{j=1}^K (y_{ij} \log f_j(x_i) + (1-y_{ij})\log(1-f_j(x_i)))$ (ii)
- $E_i = -\sum_{j=1}^K y_{ik} \log f_j(x_i)$ (iii)

(non-convex)

→ We use a SGD algorithm to minimize $\frac{1}{n}\sum_{i=1}^n E_i$: at each iteration, pick an observation (x_i, y_i) at random from $\mathcal{D}_n$, and update the current weight estimates β_j according to

$$\beta_{j+1} \leftarrow \beta_j - \gamma_j \nabla_\beta E_i .$$

Consider a general network with L hidden layers, $(d+1)$ input units, and K output units. Each hidden layer possesses n_ℓ neurons, $\ell = 1, \dots, L$.



• <u>Output</u>: $f_k(x) = \sigma\left(a_k^{(L+1)}\right)$, $1 \leq k \leq K$

$$a_k^{(L+1)} = \sum_{j=0}^{n_L} \beta_{kj}^{(L+1)} z_j^{(L)}$$

• <u>Hidden</u>: $z_j^{(\ell+1)} = h\left(a_j^{(\ell+1)}\right)$

$$a_j^{(\ell+1)} = \sum_{k=0}^{n_\ell} \beta_{jk}^{(\ell+1)} z_k^{(\ell)}, \quad \begin{matrix} 0 \leq \ell \leq L-1 \\ 1 \leq j \leq n_{\ell+1} \end{matrix}$$

We need to compute the partial derivatives $\dfrac{\partial E_i}{\partial \beta_{mj}^{(\ell)}}$ ← obs i

Weight between the m-th neuron in layer ℓ and the j-th neuron in layer $\ell-1$.

We start with the weights between layer L and the output layer, and then consider hidden layers.

• $$\frac{\partial E_i}{\partial \beta_{kj}^{(L+1)}} \overset{\uparrow \text{ chain rule}}{=} \underbrace{\frac{\partial E_i}{\partial a_k^{(L+1)}}}_{\text{Denote this term } \delta_{ik}^{(L+1)}} \frac{\partial a_k^{(L+1)}}{\partial \beta_{kj}^{(L+1)}}$$

$1 \leq k \leq K$
$0 \leq j \leq n_L$
$1 \leq i \leq n$

→ Since $a_k^{(L+1)} = \sum_{m=0}^{n_L} \beta_{km}^{(L+1)} z_m^{(L)}$, we have

$$\frac{\partial a_k^{(L+1)}}{\partial \beta_{kj}^{(L+1)}} = z_j^{(L)}$$

We saw page 18 that in the cases of (i) regression + square loss, (ii) binary classification + logistic loss and (iii) K-class classification + softmax, that $\delta_{ik}^{(L+1)} = f_k(x_i) - y_{ik}$.

$$\Rightarrow \boxed{\frac{\partial E_i}{\partial \beta_{kj}^{(L+1)}} = \delta_{ik}^{(L+1)} z_j^{(L)}}$$

• <u>Hidden units</u>: $$\frac{\partial E_i}{\partial \beta_{mj}^{(\ell)}} = \underbrace{\frac{\partial E_i}{\partial a_m^{(\ell)}}}_{\text{Denote this term } \delta_{im}^{(\ell)}} \frac{\partial a_m^{(\ell)}}{\partial \beta_{mj}^{(\ell)}}$$

$1 \leq i \leq n$
$1 \leq m \leq n_\ell$
$0 \leq j \leq n_{\ell-1}$
$1 \leq \ell \leq L$

→ $a_m^{(\ell)} = \sum_{s=0}^{n_{\ell-1}} \beta_{ms}^{(\ell)} z_s^{(\ell-1)}$

$$\Rightarrow \frac{\partial a_m^{(\ell)}}{\partial \beta_{mj}^{(\ell)}} = z_j^{(\ell-1)} .$$

Thus $$\boxed{\frac{\partial E_i}{\partial \beta_{mj}^{(\ell)}} = \delta_{im}^{(\ell)} z_j^{(\ell-1)}}$$

It remains to compute $\delta_{im}^{(\ell)}$.

Use the chain rule one more time:

$$\delta_{im}^{(\ell)} = \frac{\partial E_i}{\partial a_m^{(\ell)}} = \sum_{r=1}^{n_{\ell+1}} \underbrace{\frac{\partial E_i}{\partial a_r^{(\ell+1)}}}_{\delta_{ir}^{(\ell+1)}} \frac{\partial a_r^{(\ell+1)}}{\partial a_m^{(\ell)}}.$$

$1 \leq i \leq n$
$1 \leq m \leq n_\ell$
$1 \leq \ell \leq L$

$$\hookrightarrow a_r^{(\ell+1)} = \sum_{s=0}^{n_\ell} \beta_{rs}^{(\ell+1)} \underbrace{z_s^{(\ell)}}_{h(a_s^{(\ell)})}$$

$$\Downarrow$$

$$\frac{\partial a_r^{(\ell+1)}}{\partial a_m^{(\ell)}} = \beta_{rm}^{(\ell+1)} h'(a_m^{(\ell)})$$

$$\Rightarrow \delta_{im}^{(\ell)} = \sum_{r=1}^{n_{\ell+1}} \delta_{ir}^{(\ell+1)} \beta_{rm}^{(\ell+1)} h'(a_m^{(\ell)}).$$

$$(*) \quad \boxed{\delta_{im}^{(\ell)} = h'(a_m^{(\ell)}) \sum_{r=1}^{n_{\ell+1}} \beta_{rm}^{(\ell+1)} \delta_{ir}^{(\ell+1)}}$$

we backpropagate the error terms.

$1 \leq i \leq n$
$1 \leq m \leq n_\ell$
$1 \leq \ell \leq L$ $\quad (n_{L+1} = K)$

Summary: One iterate of the SGD algorithm can be decomposed into two successive steps: a FORWARD pass followed by a BACKWARD pass:

→ Forward pass: Given the current parameter estimates, feed observation x_i into the network: calculate the input & output of each neuron: $z_i^{(\ell)}$ / $a_i^{(\ell)}$, all the way to the output layer: $f_1(x_i), \ldots, f_K(x_i)$. Compare these values with $y_{i1}, \ldots, y_{iK}$, and compute the error terms $\delta_{i1}^{(L+1)}, \ldots, \delta_{iK}^{(L+1)}$, where $\delta_{ik}^{(L+1)} = f_k(x_i) - y_{ik}$.

→ Backpropagate the error terms via formula (*), all the way to the first layer: $\delta_{im}^{(\ell)}$, $\quad 1 \leq m \leq n_\ell$, $1 \leq \ell \leq L$.

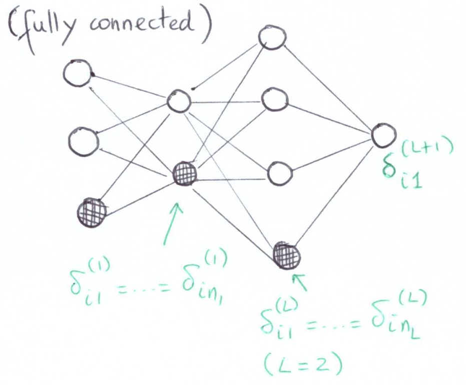
The weight between the m-th neuron in layer ℓ and the j-th neuron in layer $(\ell-1)$ is then updated using

$$\frac{\partial E_i}{\partial \beta_{mj}^{(\ell)}} = \delta_{im}^{(\ell)} z_j^{(\ell+1)} : \qquad \beta_{mj}^{(\ell)} \leftarrow \beta_{mj}^{(\ell)} - \gamma \frac{\partial E_i}{\partial \beta_{mj}^{(\ell)}}.$$

Remark: In mini-batch, select a subset of the training data, and update the parameters of the network using

$$\underset{\text{size of mini batch}}{\frac{1}{B}} \sum_{i \in \text{mini batch}} \frac{\partial E_i}{\partial \beta_{mj}^{(\ell)}}.$$

→ Weight Initialization. Consider a neural network with one output unit $(K=1)$. Suppose you initialize the network with equal weights (equal to zero, or not). Then all input (resp. output) values of each neuron within one layer are equal.


(fully connected)
$\delta_{i1}^{(L+1)}$
$\delta_{i1}^{(1)} = \ldots = \delta_{in_1}^{(1)}$
$\delta_{i1}^{(L)} = \ldots = \delta_{in_L}^{(L)}$
$(L=2)$

$z_1^{(1)} = \ldots = z_{n_1}^{(1)}$
$\vdots$
$z_1^{(L)} = \ldots = z_{n_L}^{(L)}$.

In addition, backpropagating the output error $\delta_{i1}^{(L+1)}$ through the network, we see that $\delta_{i1}^{(\ell)} = \ldots = \delta_{in_\ell}^{(\ell)}$ $\quad (\ell = 1, \ldots, L)$.

⇒ Weight updates within one hidden layer are all the same:

$$\frac{\partial E_i}{\partial \beta_{kj}^{(L+1)}} = \underbrace{\delta_{ik}^{(L+1)}}_{\text{indpt of } j} z_j^{(L)} \; (k=1); \qquad \frac{\partial E_i}{\partial \beta_{mj}^{(\ell)}} = \underbrace{\delta_{im}^{(\ell)}}_{\text{indpt of } j \text{ and } m} z_j^{(\ell-1)}$$

After one run of the SGD, weights within each layer are all equal; and this holds for subsequent runs of the algorithm aswell. To introduce diversity in the network, it is customary to use small random weights (such as $\mathcal{N}(0, \varepsilon^2)$, or $\mathcal{U}[-\varepsilon, \varepsilon]$) for initialization.

II.4. Mixture Density Networks.

Consider the regression problem, with response variable $Y_i \in \mathbb{R}^K$; $Y = (Y_{i1}, \dots, Y_{iK})$. We saw on page 15 that Y_{ik} is estimated using $f_k(x_i) = a_k$, where a_k denotes the level of the k-th neuron in the output layer. The criterion we minimized there is $\frac{1}{n}\sum_{i=1}^{n} E_i$, with

$$E_i = \frac{1}{2}\| y_i - f(x_i) \|^2 = \frac{1}{2}\sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 .$$

An equivalent probabilistic formulation is to assume the conditional density

(•) $\boxed{Y_i \mid X_i = x_i \sim \mathcal{N}(f(x_i), \sigma^2 I_K)}$,

$f(x_i) = (f_1(x_i), \dots, f_K(x_i))^t \in \mathbb{R}^K$

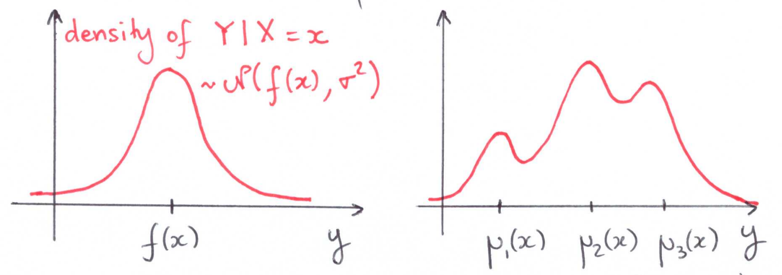
and to minimize minus the log-likelihood

$$\frac{n}{2}\log\sigma^2 + \frac{n}{2}\log\sigma^2 + \frac{1}{\sigma^2}\,\frac{1}{2}\sum_{i=1}^{n}\underbrace{\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2}_{i\text{-th error term.}}$$

$\Rightarrow$ We generalize (•) by allowing the conditional distribution of Y given $X = x$ to be a mixture of Gaussians, whose parameters are functions of x, and

corresponds to the (transformed) outputs of a NN.

(K=1)

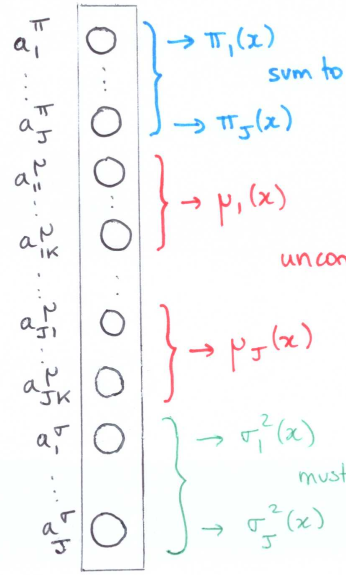


– Mixture of 3 normal distr. –

$$\boxed{Y_i \mid X_i = x_i \sim \sum_{j=1}^{J} \pi_j(x)\, \mathcal{N}(y_i \mid \mu_j(x), \sigma_j^2(x))}$$

↘ The output layer of the NN contains a total of $J(K+2)$ neurons, corresponding to:

- J prior probabilities : $\pi_1(x), \dots, \pi_J(x) \rightarrow J$ coef
- J means of dim K : $\mu_1(x), \dots, \mu_J(x) \rightarrow KJ$ coef
- J variances : $\sigma_1^2(x), \dots, \sigma_J^2(x) \rightarrow J$ coef.



sum to 1 : softmax : $\pi_j(x) = \dfrac{\exp(a_j^\pi)}{\sum_{\ell=1}^{J}\exp(a_\ell^\pi)}$.

unconstrained $\mathbb{R}^K$ vectors : $[\mu_j(x)]_k = a_{jk}^\mu$

must be >0 : take $\sigma_j(x) = \exp(a_j^\sigma)$.

$\Rightarrow$ We seek to minimize $\frac{1}{n}\sum_{i=1}^{n} E_i$, with

$$E_i = -\log\left\{ \sum_{j=1}^{J} \pi_j(x_i)\,\mathcal{N}(y_i \mid \mu_j(x_i), \sigma_j^2(x_i)) \right\},$$

using a SGD algorithm. The backprop algorithm runs as before, provided the error terms $\frac{\partial E_i}{\partial a^\pi}$, $\frac{\partial E_i}{\partial a^\mu}$ & $\frac{\partial E_i}{\partial a^\sigma}$ can be easily evaluated

$$\rightarrow \frac{\partial E_i}{\partial a_j^\pi} = \sum_{m=1}^{J} \underbrace{\frac{\partial E_i}{\partial \pi_m}}\ \underbrace{\frac{\partial \pi_m}{\partial a_j^\pi}} \quad (\text{chain rule})$$

$$\frac{\partial E_i}{\partial \pi_m} = -\frac{\mathcal{N}(y_i \mid \mu_m, \sigma_m^2)}{\sum_{l=1}^{J} \pi_l \mathcal{N}(y_i \mid \mu_l, \sigma_l^2)}$$

$$\frac{\partial \pi_m}{\partial a_j^\pi} = \frac{\partial}{\partial a_j^\pi}\left\{ \frac{e^{a_m^\pi}}{\sum_{l=1}^{J} e^{a_l^\pi}} \right\}$$

If $m \neq j$ $= -\pi_m \pi_j$

If $m = j$ $= \pi_j(1-\pi_j)$

$\Rightarrow\ = \pi_m\,(\mathbb{1}(j=m) - \pi_j)$

We obtain

$$\frac{\partial E_i}{\partial a_j^\pi} = -\sum_{m=1}^{J} \left\{ \underbrace{\frac{\pi_m \mathcal{N}(y_i \mid \mu_m, \sigma_m^2)}{\sum_{l=1}^{J} \pi_l \mathcal{N}(y_i \mid \mu_l, \sigma_l^2)}} \right\} (\mathbb{1}(j=m) - \pi_j)$$

$\parallel$ Posterior distribution that Y_i belongs to class m, given $Y_i = y_i$. Denote this term γ_{im}.

$$= \sum_{m=1}^{J} (\pi_j - \mathbb{1}(j=m))\,\gamma_{im}$$

$$= \sum_{m=1}^{J} \pi_j \gamma_{im} - \gamma_{ij}$$

since $\sum_{m=1}^{J} \gamma_{im} = 1$

$$\boxed{\frac{\partial E_i}{\partial a_j^\pi} = \pi_j - \gamma_{ij}}$$

$$\rightarrow \frac{\partial E_i}{\partial a_{jk}^\mu} = \frac{\partial E_i}{\partial \mu_{jk}} \qquad \text{where } \mu_j = (\mu_{j1}, \dots, \mu_{jK})^t$$

$\uparrow$ Since $\mu_{jk} = a_{jk}^\mu$, $1 \le j \le J$, $1 \le k \le K$.

$$= \frac{\partial}{\partial \mu_{jk}}\left\{ -\log \sum_{l=1}^{J} \pi_l \mathcal{N}(y_i \mid \mu_l, \sigma_l^2) \right\}.$$

$$= \frac{-\pi_j \mathcal{N}(y_i \mid \mu_j, \sigma_j^2)\ \cancel{-}\frac{1}{\cancel{2}\sigma_j^2}(\cancel{-2})(y_{ik} - \mu_{jk})}{\sum_{l=1}^{J} \pi_l \mathcal{N}(y_i \mid \mu_l, \sigma_l^2)}$$

$$\boxed{\frac{\partial E_i}{\partial a_{jk}^\mu} = \gamma_{ij}\,\frac{\mu_{jk} - y_{ik}}{\sigma_j^2}}$$

$$\rightarrow \frac{\partial E_i}{\partial a_j^\sigma} = \sum_{m=1}^{J} \frac{\partial E_i}{\partial \sigma_m} \underbrace{\frac{\partial \sigma_m}{\partial a_j^\sigma}}$$

$\sigma_m = e^{a_m}$

$\Rightarrow \frac{\partial \sigma_m}{\partial a_j^\sigma} = \sigma_m\,\mathbb{1}(j=m)$

$$= \frac{\partial E_i}{\partial \sigma_j} \times \sigma_j$$

Next, $\frac{\partial E_i}{\partial \sigma_j} = \frac{\partial}{\partial \sigma_j}\left\{ -\log \sum_{l=1}^{J} \pi_l \mathcal{N}(y_i \mid \mu_l, \sigma_l^2) \right\}$

$$= -\frac{\pi_j}{(2\pi)^{K/2}}\,\frac{-\frac{K}{\sigma_j^{K+1}} e^{-\frac{1}{2\sigma_j^2}\|y_i-\mu_l\|^2} + \frac{1}{\sigma_j^K} e^{-\frac{1}{2\sigma_j^2}\|..\|^2}\ \cancel{-}\frac{2}{\sigma_j^3}\ \cancel{-}\frac{1}{2}\|y_i - \mu_l\|^2}{\sum_{l=1}^{J} \pi_l \mathcal{N}(y_i \mid \mu_l, \sigma_l^2)}$$

$$\frac{\partial E_i}{\partial \sigma_j} = \frac{K}{\sigma_j}\gamma_{ij} - \frac{1}{\sigma_j^3}\gamma_{ij}\|y_i - \mu_\ell\|^2 .$$

Thus,

$$\boxed{\frac{\partial E_i}{\partial a_j^\sigma} = \gamma_{ij}\left(K - \frac{\|y_i-\mu_\ell\|^2}{\sigma_j^2}\right)}$$

Once the model is fitted, we may compute the posterior mean of Y given X = x, given by $\sum_{j=1}^{J}\pi_j(x)\,\mu_j(x)$, which is of limited value (why bother using a mixture density model, if it's to combine all components once the model is fitted). A better use of the model is to pick the most likely mixture (corresponding to the largest value of $\pi_j(x)$), and consider the associated $\mu_j(x)$ as a predictor.

II.5. Regularization in NN

To reduce the complexity of a NN, we may add a regulaization term to $\frac{1}{n}\sum_{i=1}^{n}E_i$, such as $\frac{\lambda}{2}\|\beta\|^2$, for β=(vector containing all the parameters, from all the layers). Unfortunately, this ℓ_2-regularizer is inconsistent with some scaling propeties of the network.

Consider for simplicity a NN with one hidden layer, and identity activation functions on the output layer:

- Suppose you feed $x = (\underset{1}{\underset{\shortparallel}{x_0}}, x_1, \dots, x_d)$ into the network:

$$\boxed{\begin{aligned} a_j &= \sum_{\ell=1}^{d}\beta_{j\ell}^{(1)}x_i + \beta_{j0}^{(1)} \\ z_j &= h(a_j) \\ f_k(x) &= \sum_{m=1}^{M}\beta_{km}^{(2)}z_j + \beta_{k0}^{(2)} \end{aligned}}$$

(page 9)

- Suppose now that you feed $ax+b$ $(a\neq 0,\ b\neq 0)$ into the same network, with weights $\gamma_{km}^{(\ell)}$:

$$a'_j = \sum_{\ell=1}^{d}\gamma_{j\ell}^{(1)}(ax_i+b) + \gamma_{j0}^{(1)}$$
$$= \sum_{\ell=1}^{d}\left\{a\gamma_{j\ell}^{(1)}\right\}x_i + \left\{\gamma_{j0}^{(1)} + b\sum_{\ell=1}^{d}\gamma_{j\ell}^{(1)}\right\}$$

~~&~~ so $a'_j = a_j$ provided we consider a linear transform of the weights, given by

$$\begin{cases} \gamma_{j\ell}^{(1)} = \beta_{j\ell}^{(1)}/a & 1\le \ell\le d \\ \gamma_{j0}^{(1)} = \beta_{j0}^{(1)} - \frac{b}{a}\sum_{\ell=1}^{d}\beta_{j\ell}^{(1)} . \end{cases}$$

containing weights in the first layer only.

⇒ When a penalty of type $\frac{\lambda}{2}\|\beta\|^2$ is included into the objective function, the regularizer remains unchanged provided we replace λ by $a^2\lambda$, if we feed $ax+b$ instead of x into the network. In other words, feeding x with a tuning parameter λ, a feeding $ax+b$ with tuning parameter $a^2\lambda$ yieds weights that are linear transform of one another.

- Suppose that one scales the output of the network by c, and translate it by an amount equal to d:

$$c f_k(x) + d = \sum_{j=1}^{M} \alpha_{kj}^{(2)} z_j + \alpha_{k0}^{(2)}$$

$$f_k(x) = \sum_{j=1}^{M} \left\{ \frac{\alpha_{kj}^{(2)}}{c} \right\} z_j + \left\{ \frac{\alpha_{k0}^{(2)} - d}{c} \right\}$$

Equal to $\sum_{j=1}^{M} \beta_{kj}^{(2)} z_j + \beta_{k0}^{(2)}$ provided we consider a rescaling of the weights equal to:

$$\begin{cases} \alpha_{kj}^{(2)} = c\, \beta_{kj}^{(2)} \\ \alpha_{k0}^{(2)} = d + c\, \beta_{k0}^{(2)} \end{cases}$$

$\Rightarrow$ If a ridge penalty is added to the cost function, with tuning parameter equal to λ in the case of no rescaling of the output, then scaling the output as $c f_k(x) + d$ yields a re-scaling of the weights, provided the tuning parameter λ is replaced with λ / c^2.

Consequence: If we shift both the input & output at the same time, a single tuning parameter is not enough to ensure invariance of the weights under linear transformations. Insted, we need to consider two tuning parameters λ_1 and λ_2, and a penalty of the form:

$$\frac{1}{2} \left\{ \lambda_1 \sum_{\text{layer } 1} \beta_j^2 + \lambda_2 \sum_{\text{layer } 2} \beta_j^2 \right\}.$$

Then use SGD by adding an extra $(\lambda_m \beta)$ to the gradient term.