Neural Networks: Part I

Fundamentals

Daniel Yukimura

yukimura@impa.br

September 4, 2018

Neural Networks I: Fundamentals

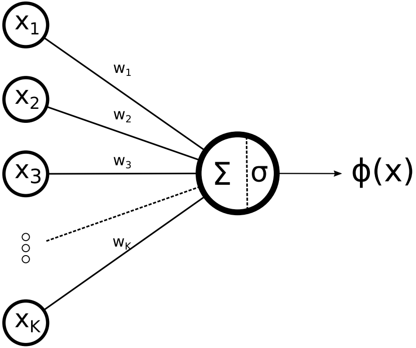
Goal: Learn a Parametric Function.

$$\mathcal{X} \longrightarrow \boxed{f_\theta} \longrightarrow \mathcal{Y}$$

- $\theta \in \Theta$: function parameters (**these are learned**).
- $\mathcal{X}$: input space.
- $\mathcal{Y}$: outcome space.

The Perceptron

The Fundamental Building Block of Deep Learning



Rosenblatt, 1957

The Perceptron

The Fundamental Building Block of Deep Learning

Processing units biologically inspired in **neurons**.

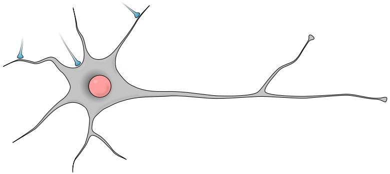


Figure: **Neuron**

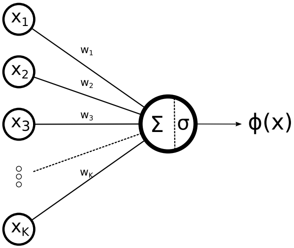


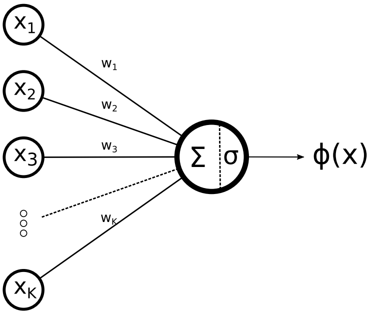
Figure: **(Artificial) Neuron**

- There is no clear correspondence between Deep Learning and how the human brain works!

Rosenblatt, 1957

The Perceptron

Model: A parametric function $\phi : \mathbb{R}^k \to \mathbb{R}$, given by

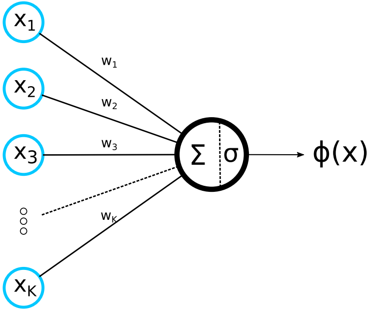


$$\phi(x) = \sigma\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

- **activation function**: $\sigma : \mathbb{R} \to \mathbb{R}$ (usually non-linear).
- **parameters**: $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ and $b \in \mathbb{R}$

The Perceptron

Model: A parametric function $\phi : \mathbb{R}^k \to \mathbb{R}$, given by
Is useful to look at it as a **feedforward flow!**

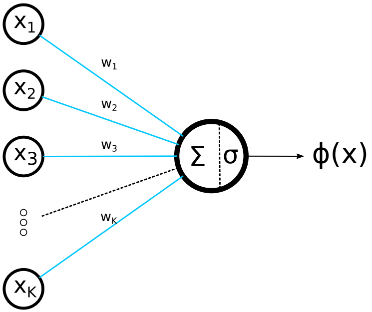


$$\phi(x) = \sigma\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

- **activation function**: $\sigma : \mathbb{R} \to \mathbb{R}$ (usually non-linear).
- **parameters**: $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ and $b \in \mathbb{R}$

The Perceptron

Model: A parametric function $\phi : \mathbb{R}^k \to \mathbb{R}$, given by
Is useful to look at it as a **feedforward flow!**

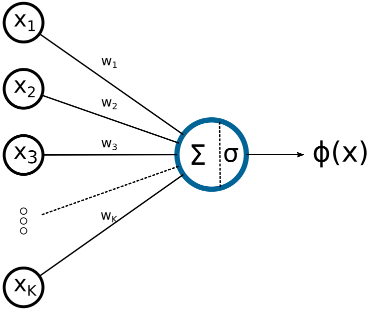


$$\phi(x) = \sigma\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

- **activation function**: $\sigma : \mathbb{R} \to \mathbb{R}$ (usually non-linear).
- **parameters**: $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ and $b \in \mathbb{R}$

The Perceptron

Model: A parametric function $\phi : \mathbb{R}^k \to \mathbb{R}$, given by
Is useful to look at it as a **feedforward flow!**

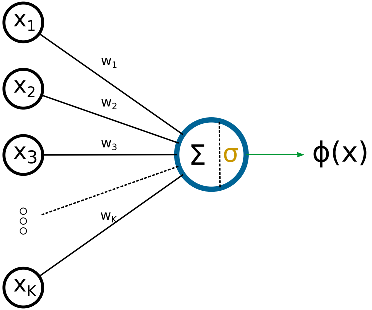


$$\phi(x) = \sigma\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

- **activation function**: $\sigma : \mathbb{R} \to \mathbb{R}$ (usually non-linear).
- **parameters**: $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ and $b \in \mathbb{R}$

The Perceptron

Model: A parametric function $\phi : \mathbb{R}^k \to \mathbb{R}$, given by
Is useful to look at it as a **feedforward flow!**

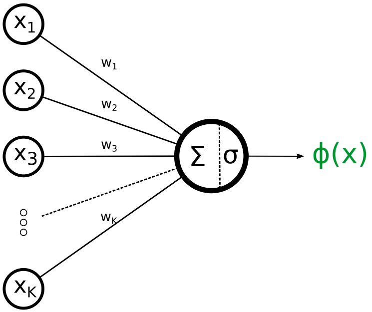


$$\phi(x) = \sigma\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

- **activation function**: $\sigma : \mathbb{R} \to \mathbb{R}$ (usually non-linear).
- **parameters**: $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ and $b \in \mathbb{R}$

The Perceptron

Model: A parametric function $\phi : \mathbb{R}^k \to \mathbb{R}$, given by
Is useful to look at it as a **feedforward flow!**



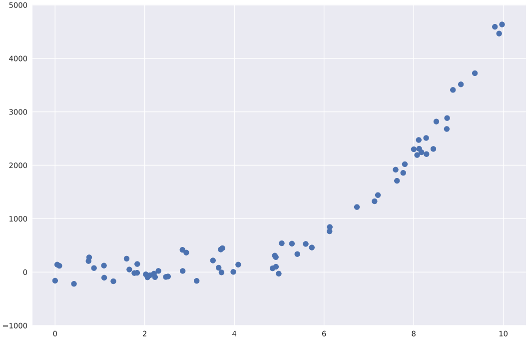
$$\phi(x) = \sigma\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

- **activation function**: $\sigma : \mathbb{R} \to \mathbb{R}$ (usually non-linear).
- **parameters**: $w = (w_1, \ldots, w_k) \in \mathbb{R}^k$ and $b \in \mathbb{R}$

The Perceptron

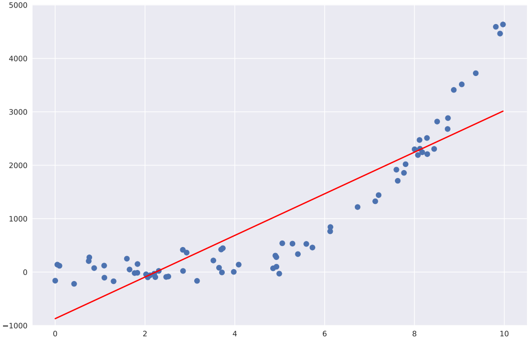
Reassessing Linear Models:

The addition of an **activation function** is the first step on rising model capacity.



Reassessing Linear Models:

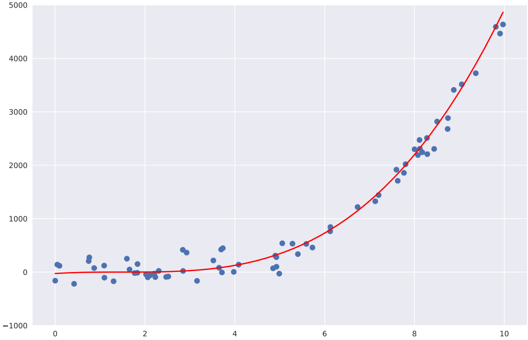
The addition of an **activation function** is the first step on rising model capacity.



The Perceptron

Reassessing Linear Models:

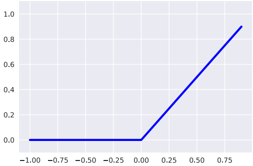
The addition of an **activation function** is the first step on rising model capacity.



Common Activation Functions

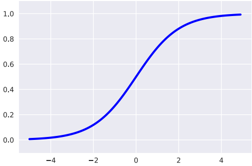
Rectified Linear Unit (ReLU)

$$\sigma(z) = \max(0, z)$$



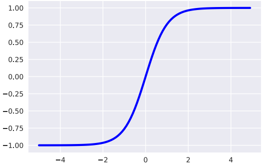
Sigmoid Function

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



Hyperbolic Tangent

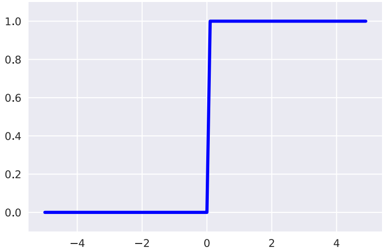
$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

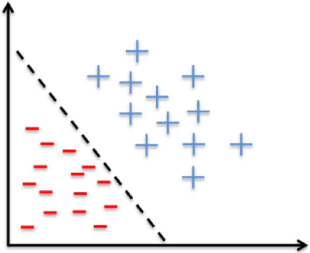


The Perceptron

Example: Binary Classification/Logistic Regression

The Perceptron was proposed as a model for **binary classification**. Originally it used the **step function** as activation.

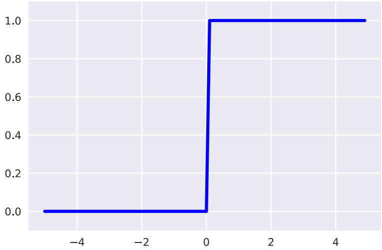


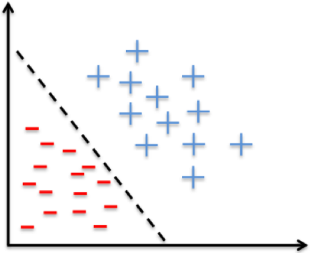


The Perceptron

Example: Binary Classification/Logistic Regression

The Perceptron was proposed as a model for **binary classification**. Originally it used the **step function** as activation.



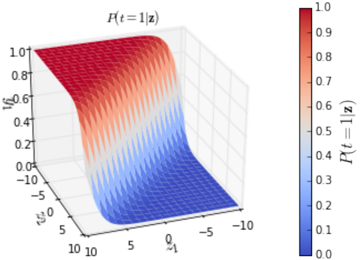


Is hard to learn without differentiability!

Example: Binary Classification/Logistic Regression

In **logistic regression** we model the posterior distribution $p(y \mid x)$ by smoothly squeezing the linear model into a probability distribution.

$$
\begin{aligned}
p_w(y = 1 \mid x) &= \mathrm{sigm}(w^T x) \\
&= \frac{1}{1 + e^{-w^T x}}
\end{aligned}
$$



meaning: The probability that x belongs to the class 1.

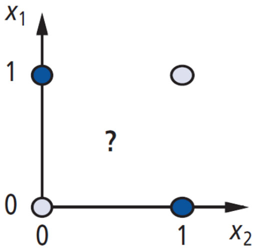
The Perceptron

Example: The XOR function

- The Perceptron is unnable to learn the exclusive or (XOR) function!
- The classes can't be separated by half-spaces (linear models).

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

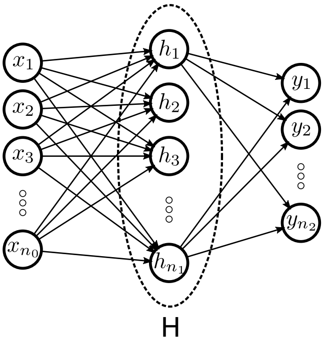
Table: $y = x_1 \oplus x_2$



Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

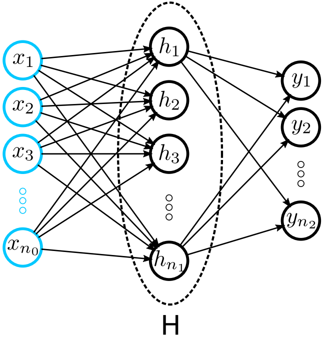


- $h_j = \sigma\left(\sum_{i=1}^{n_0} w_{i,j}^{(1)} x_i + b_j\right)$
- $y_k = \sum_{j=1}^{n_1} w_{j,k}^{(2)} h_i$

Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

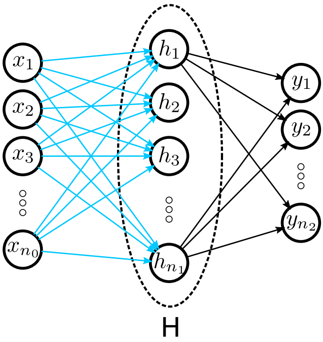


- $h_j = \sigma\left(\sum_{i=1}^{n_0} w_{i,j}^{(1)} x_i + b_j\right)$
- $y_k = \sum_{j=1}^{n_1} w_{j,k}^{(2)} h_i$
- **Forward propagation**

Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

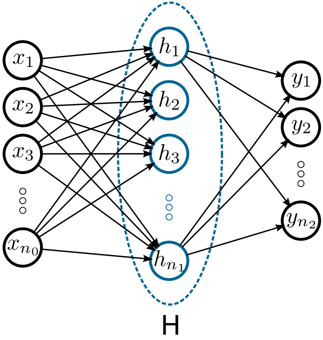


- $h_j = \sigma\left(\sum_{i=1}^{n_0} w_{i,j}^{(1)} x_i + b_j\right)$
- $y_k = \sum_{j=1}^{n_1} w_{j,k}^{(2)} h_i$
- **Forward propagation**

Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

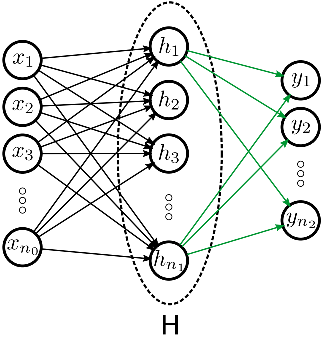


- $h_j = \sigma\left(\sum_{i=1}^{n_0} w_{i,j}^{(1)} x_i + b_j\right)$
- $y_k = \sum_{j=1}^{n_1} w_{j,k}^{(2)} h_i$
- **Forward propagation**

Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

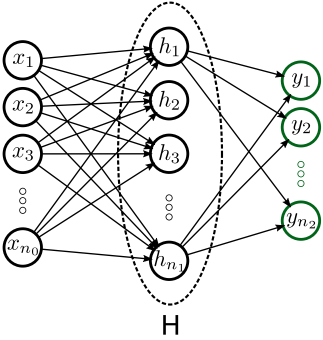


- $h_j = \sigma\left(\sum_{i=1}^{n_0} w_{i,j}^{(1)} x_i + b_j\right)$
- $y_k = \sum_{j=1}^{n_1} w_{j,k}^{(2)} h_i$
- **Forward propagation**

Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

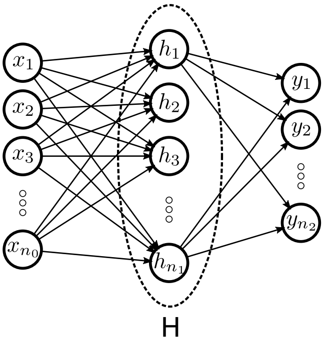


- $h_j = \sigma\left(\sum_{i=1}^{n_0} w_{i,j}^{(1)} x_i + b_j\right)$
- $y_k = \sum_{j=1}^{n_1} w_{j,k}^{(2)} h_i$
- **Forward propagation**

Neural Networks

How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.

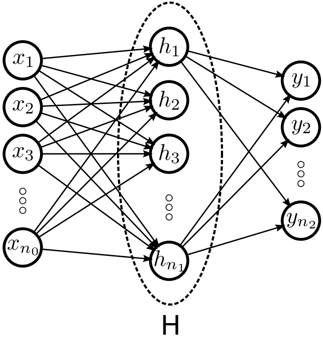


- $h = \sigma\left({W^{(1)}}^T x + b\right)$
- $y = {W^{(2)}}^T h$
- **Matrix notation is useful!**

Neural Networks

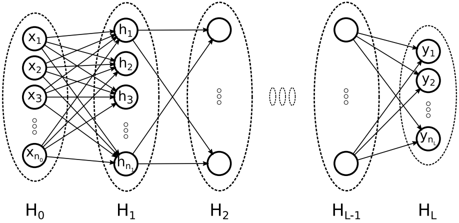
How to combine neurons to build more expressive models?

Feedforward Neural Network (FNN): We combine neurons layerwise as vertices of a directed graph.



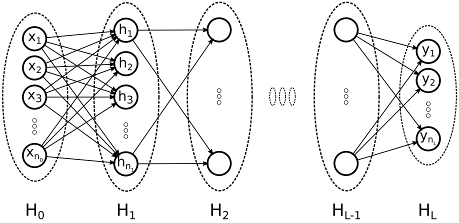
- $h = \sigma\left({W^{(1)}}^T x + b\right)$
- $y = {W^{(2)}}^T h$
- **Universal Approximation Theorem:** Given enough neurons in a hidden layer, and a non-linear increasing activation function, one can approximate any Borel measurable function (see [ref]).

Do we need more layers?



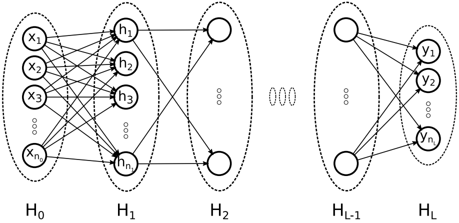
- Using more layers seems to allow more capacity while using fewer neurons, see [ref].
- There are many cases of success by using more layers.
- Deeper networks are harder to train!

Do we need more layers?



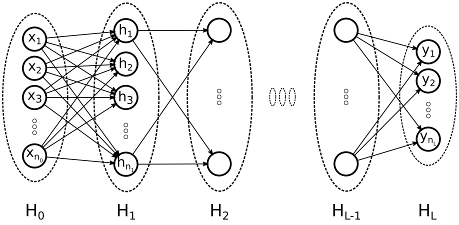
- Using more layers seems to allow more capacity while using fewer neurons, see [ref].
- There are many cases of success by using more layers.
- Deeper networks are harder to train!

Do we need more layers?



- Using more layers seems to allow more capacity while using fewer neurons, see [ref].
- There are many cases of success by using more layers.
- Deeper networks are harder to train!

Do we need more layers?



- $h^{(0)} = x$, $h^{(\ell)} = \sigma\left(W^{(\ell)^T} h^{(\ell-1)} + b^{(\ell)}\right)$, $\ell \in [L-1]$
- $\hat{y} = f(x, \theta) = W^{(L)^T} h^{(L-1)}$ (sometimes $\hat{y} = \sigma(\dots)$).
- We denote $\theta_\ell = (W^{(\ell)}, b^{(\ell)})$ the parameters of layer ℓ, and $\theta = (\theta_1, \dots, \theta_L)$

Risk Minimization

Recall:

We want to find the network weights that ***achieve the lowest risk value****.*

$$\begin{aligned}\hat{\theta} &= \operatorname*{argmin}_{\theta \in \Theta} R_n(\theta) \\ &= \operatorname*{argmin}_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \ell\left(f\left(x_i, \theta\right), y_i\right)\end{aligned}$$

Example: For L_2 regression we have

$$R_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \|f(x_i, \theta) - y_i\|_2^2$$

Maximum Likelihood Estimation

- When modelling posterior distributions $p_\theta(y|x)$ is useful to look at the likelihood function

$$\mathcal{L}_n(\theta) = p_\theta(\mathcal{D}) = \prod_{i=1}^{n} p_\theta(x_i, y_i)$$

- Maximizing $\mathcal{L}_n(\theta)$ means finding p_θ that best represents the data.
- But, in the supervised problem we can consider the alternative form

$$\mathcal{L}_n(\theta) = \prod_{i=1}^{n} p_\theta(y_i \mid x_i).$$

Maximum Likelihood Estimation

- When modelling posterior distributions $p_\theta(y|x)$ is useful to look at the likelihood function

$$\mathcal{L}_n(\theta) = p_\theta(\mathcal{D}) = \prod_{i=1}^{n} p_\theta\left(x_i, y_i\right)$$

- Maximizing $\mathcal{L}_n(\theta)$ means finding p_θ that best represents the data.
- But, in the supervised problem we can consider the alternative form

$$\mathcal{L}_n(\theta) = \prod_{i=1}^{n} p_\theta\left(y_i \mid x_i\right).$$

Maximum Likelihood Estimation

- When modelling posterior distributions $p_\theta(y|x)$ is useful to look at the likelihood function

$$\mathcal{L}_n(\theta) = p_\theta(\mathcal{D}) = \prod_{i=1}^{n} p_\theta\left(x_i, y_i\right)$$

- Maximizing $\mathcal{L}_n(\theta)$ means finding p_θ that best represents the data.
- But, in the supervised problem we can consider the alternative form

$$\mathcal{L}_n(\theta) = \prod_{i=1}^{n} p_\theta\left(y_i \mid x_i\right).$$

Maximum Likelihood Estimation

- The negative log-likelihood translates into the risk problem

$$-\frac{1}{n}\log\left(\mathcal{L}_n(\theta)\right) = \frac{1}{n}\sum_{i=1}^{n} -\log p_\theta\left(y_i \mid x_i\right)$$

- Therefore, the Maximum Likelihood Estimator (MLE) can be obtained through minimizing such risk

$$\hat{\theta} = \operatorname*{argmin}_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} -\log p_\theta\left(y_i \mid x_i\right)$$

Maximum Likelihood Estimation

- The negative log-likelihood translates into the risk problem

$$-\frac{1}{n}\log\left(\mathcal{L}_n(\theta)\right) = \frac{1}{n}\sum_{i=1}^{n} -\log p_\theta\left(y_i \mid x_i\right)$$

- Therefore, the Maximum Likelihood Estimator (MLE) can be obtained through minimizing such risk

$$\hat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}}\,\frac{1}{n}\sum_{i=1}^{n} -\log p_\theta\left(y_i \mid x_i\right)$$

Maximum Likelihood Estimation

Example: Binary Classification/Logistic Regression

- Observe that in the binary classification case $y_i \in \{0, 1\}$ we can write the posterior as

$$p_\theta\left(y_i \mid x_i\right) = f(x_i, \theta)^{y_i}(1 - f(x_i, \theta))^{(1-y_i)}$$

- Implying

$$\log p_\theta\left(y_i \mid x_i\right) = y_i \log\left(f(x_i, \theta)\right) + (1 - y_i)\log\left(1 - f(x_i, \theta)\right)$$

Maximum Likelihood Estimation

Example: Binary Classification/Logistic Regression

- Observe that in the binary classification case $y_i \in \{0, 1\}$ we can write the posterior as

$$p_\theta\left(y_i \mid x_i\right) = f(x_i, \theta)^{y_i}(1 - f(x_i, \theta))^{(1-y_i)}$$

- Implying

$$\log p_\theta\left(y_i \mid x_i\right) = y_i \log\left(f(x_i, \theta)\right) + (1 - y_i) \log\left(1 - f(x_i, \theta)\right)$$

Maximum Likelihood Estimation

Example: Binary Classification/Logistic Regression

- The resulting loss is called the **Cross Entropy Loss**

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} y_i \log\left(f(x_i,\theta)\right) + (1-y_i)\log\left(1-f(x_i,\theta)\right)$$

- The next question is how to actually optimize such functions.

Maximum Likelihood Estimation

Example: Binary Classification/Logistic Regression

- The resulting loss is called the **Cross Entropy Loss**

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} y_i \log\left(f(x_i,\theta)\right) + (1-y_i)\log\left(1-f(x_i,\theta)\right)$$

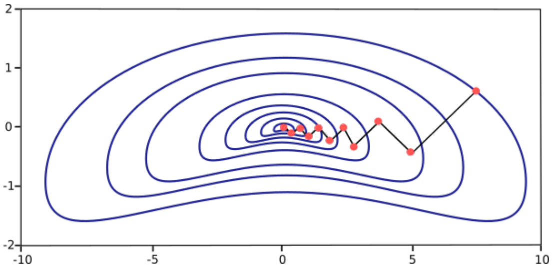
- The next question is how to actually optimize such functions.

Gradient Descent

The classical **gradient descent** (GD) consists on the iteration

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t)$$

for some initial configuration θ_0 and learning rate $\alpha > 0$.



Computing Gradients: Backpropagation

Backpropagation is an efficient algorithm for computing risk gradients of NN models (Is essentially just chain rule).

- Let $L(\theta) = c(f(x, \theta))$, where the cost function c might depend on the label y or other parameters, but for the derivation purpose they are omitted.
- How does a small change in the parameters θ_ℓ affect the loss L?
- Observe that $L(\theta) = L(h^{(\ell)}(h^{(\ell-1)}, \theta_\ell), \theta_{\ell+1}^L)$, then

$$\frac{\partial L}{\partial \theta_\ell} = \sum_{j=1}^{|H_\ell|} \frac{\partial L}{\partial h_j^{(\ell)}} \cdot \frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$$

Computing Gradients: Backpropagation

Backpropagation is an efficient algorithm for computing risk gradients of NN models (Is essentially just chain rule).

- Let $L(\theta) = c(f(x, \theta))$, where the cost function c might depend on the label y or other parameters, but for the derivation purpose they are omitted.
- How does a small change in the parameters θ_ℓ affect the loss L?
- Observe that $L(\theta) = L(h^{(\ell)}(h^{(\ell-1)}, \theta_\ell), \theta_{\ell+1}^L)$, then

$$\frac{\partial L}{\partial \theta_\ell} = \sum_{j=1}^{|H_\ell|} \frac{\partial L}{\partial h_j^{(\ell)}} \cdot \frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$$

Computing Gradients: Backpropagation

Backpropagation is an efficient algorithm for computing risk gradients of NN models (Is essentially just chain rule).

- Let $L(\theta) = c(f(x, \theta))$, where the cost function c might depend on the label y or other parameters, but for the derivation purpose they are omitted.
- How does a small change in the parameters θ_ℓ affect the loss L?
- Observe that $L(\theta) = L(h^{(\ell)}(h^{(\ell-1)}, \theta_\ell), \theta_{\ell+1}^{L})$, then

$$\frac{\partial L}{\partial \theta_\ell} = \sum_{j=1}^{|H_\ell|} \frac{\partial L}{\partial h_j^{(\ell)}} \cdot \frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$$

Computing Gradients: Backpropagation

- Observe that $L(\theta) = L(h^{(\ell)}(h^{(\ell-1)}, \theta_\ell), \theta^L_{\ell+1})$, then

$$\frac{\partial L}{\partial \theta_\ell} = \sum_{j=1}^{|H_\ell|} \frac{\partial L}{\partial h_j^{(\ell)}} \cdot \frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$$

- $\frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$ can be computed directly from the definition.

Computing Gradients: Backpropagation

- Observe that $L(\theta) = L(h^{(\ell)}(h^{(\ell-1)}, \theta_\ell), \theta^L_{\ell+1})$, then

$$\frac{\partial L}{\partial \theta_\ell} = \sum_{j=1}^{|H_\ell|} \frac{\partial L}{\partial h_j^{(\ell)}} \cdot \frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$$

- $\frac{\partial h_j^{(\ell)}}{\partial \theta_\ell}$ can be computed directly from the definition.

Computing Gradients: Backpropagation

- The vector $\delta_\ell = \frac{\partial L}{\partial h_j^{(\ell)}}$ can be computed through a recursion on the network, on the opposite direction, starting from L
  - $\delta_L = \frac{\partial L}{\partial h^{(L)}}$ is just the gradient of the cost function $c(\cdot)$.
  - For $\ell \in [L-1]$

$$\delta_\ell = \frac{\partial L}{\partial h^{(\ell+1)}} \frac{\partial h^{(\ell+1)}}{\partial h^{(\ell)}} = \delta_{\ell+1} \frac{\partial h^{(\ell+1)}}{\partial h^{(\ell)}}$$

  - The values of $\frac{\partial h^{(\ell+1)}}{\partial h^{(\ell)}}$ can also be computed directly.

Stochastic Gradient Descent (SGD)

- On each iteration $t > 0$ we choose uniformly at random an $S-$set $\mathcal{S} \subseteq [N]$ of indices ($|\mathcal{D}| = N$) and compute the *minibatch* gradient as

$$\hat{L}_S(\theta) = \frac{1}{S} \sum_{i \in \mathcal{S}} \ell(\theta, Z_i)$$

$$\hat{g}_S(\theta) = \nabla \hat{L}_S(\theta)$$

- The iteration is given as before

$$\theta_{t+1} = \theta_t - \alpha \hat{g}_S(\theta)$$

Stochastic Gradient Descent (SGD)

Remark: The noise resulting from working with minibatches actually helps on avoiding bad minimas and to escape saddle points.