ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Perceptron and Logistic Regression

Lê Anh Cường

2020

Outline

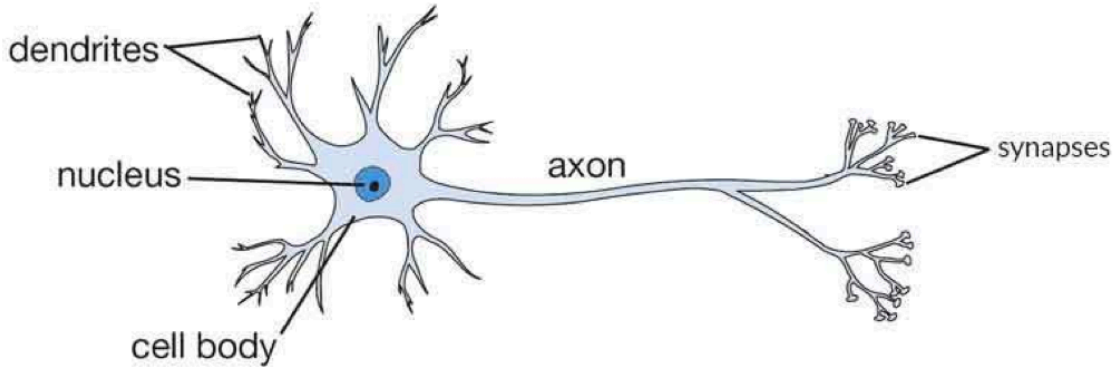
1. Biological Neuron and Artificial Neural Network
2. Perceptron and Learning Perceptron
3. Information Theory
4. Logistic function
5. Logistic classification with Cross Entropy Loss

References

- https://www.xenonstack.com/blog/artificial-neural-network-applications/

- https://peterroelants.github.io/posts/cross-entropy-logistic/
- https://peterroelants.github.io/posts/cross-entropy-softmax/

Biological Neuron



- **Function of Dendrite** (đuôi gai)

It receives signals from other neurons.

- **Soma (cell body)**

It sums all the incoming signals to generate input.

- **Axon Structure** (cấu trúc trục)

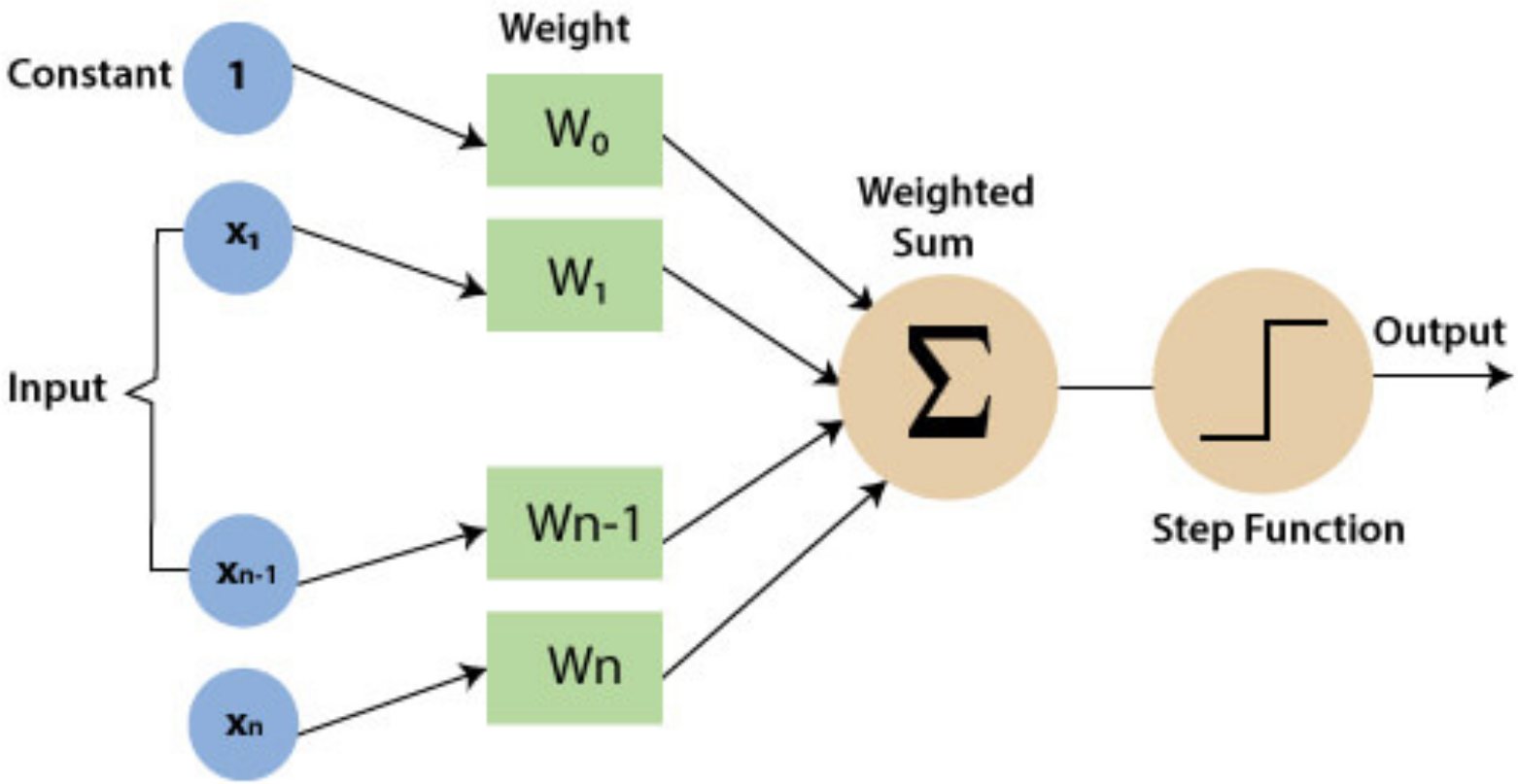
When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons.

- **Synapses Working (khớp thần kinh)**

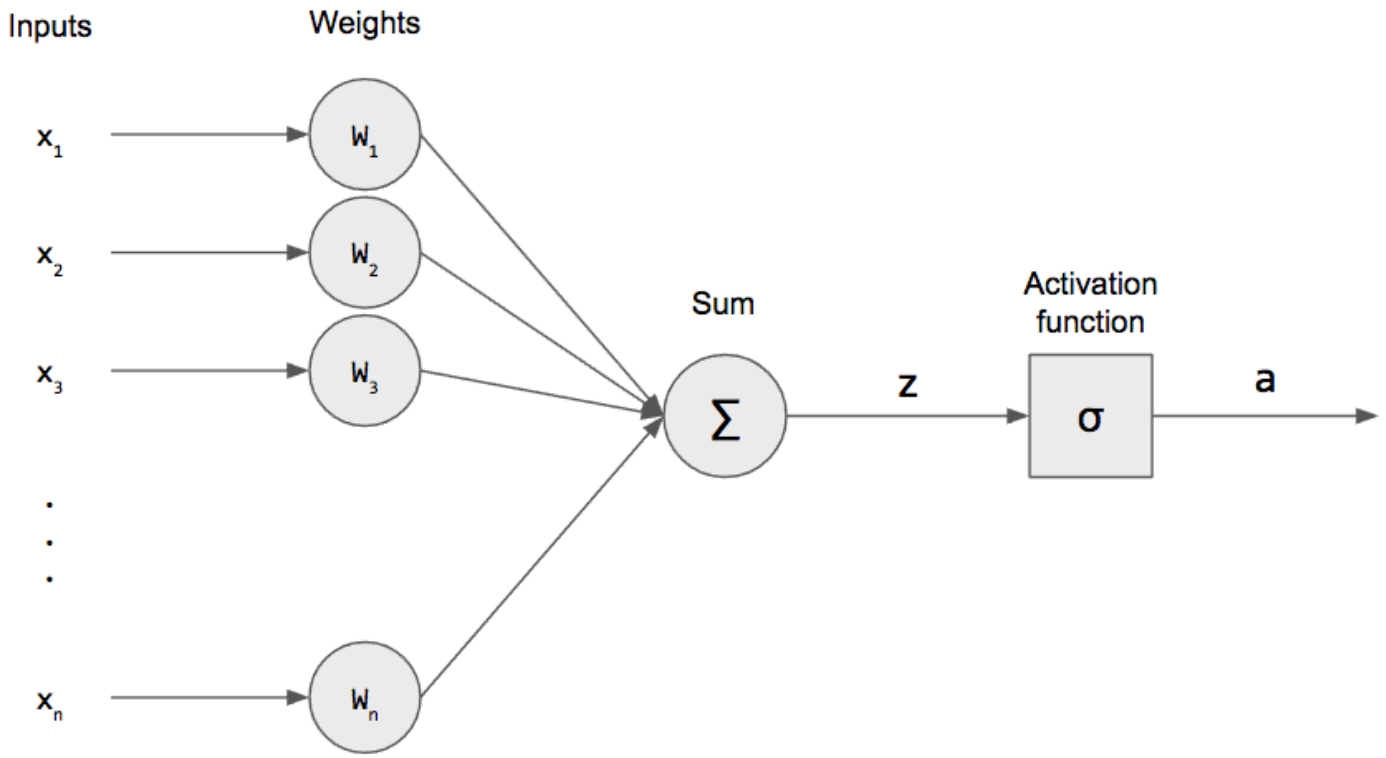
The point of interconnection of one neuron with other neurons. The amount of signal transmitted depend upon the strength (synaptic weights) of the connections.

The connections can be inhibitory (decreasing strength) or excitatory (increasing strength) in nature.

Artificial Neural or Perceptron

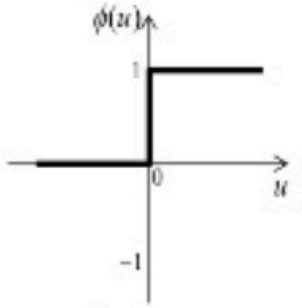


Perceptron


Inputs
Weights
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_n
Sum
Σ
z
Activation function
σ
a

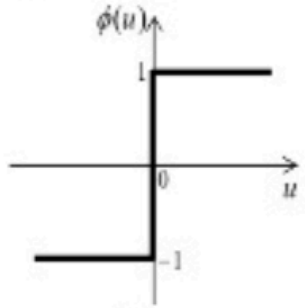
Activation functions

step function



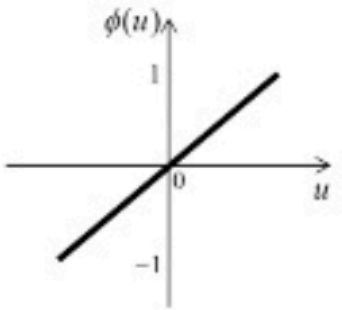
$$\phi_{step}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

sign function

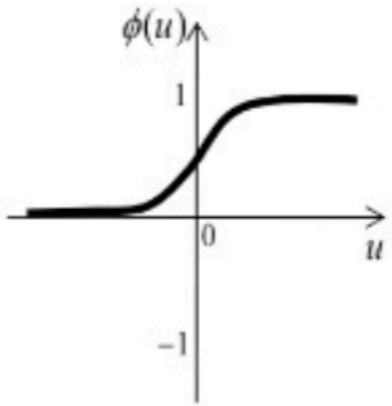


$$\phi_{sign}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

identity function

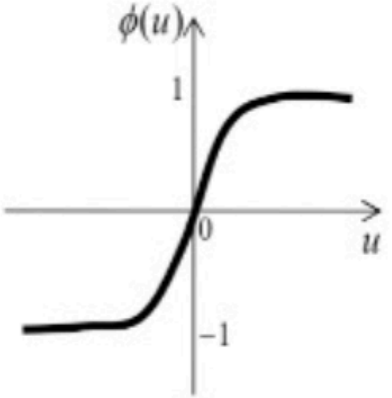


$$\phi_{id}(u) = u$$



$$\phi_{sig}(u) = \frac{1}{1+e^{-u}}$$

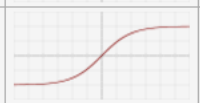
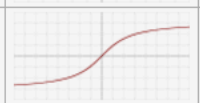
sigmoid function



$$\phi_h = \frac{e^u - 1}{e^u + 1}$$

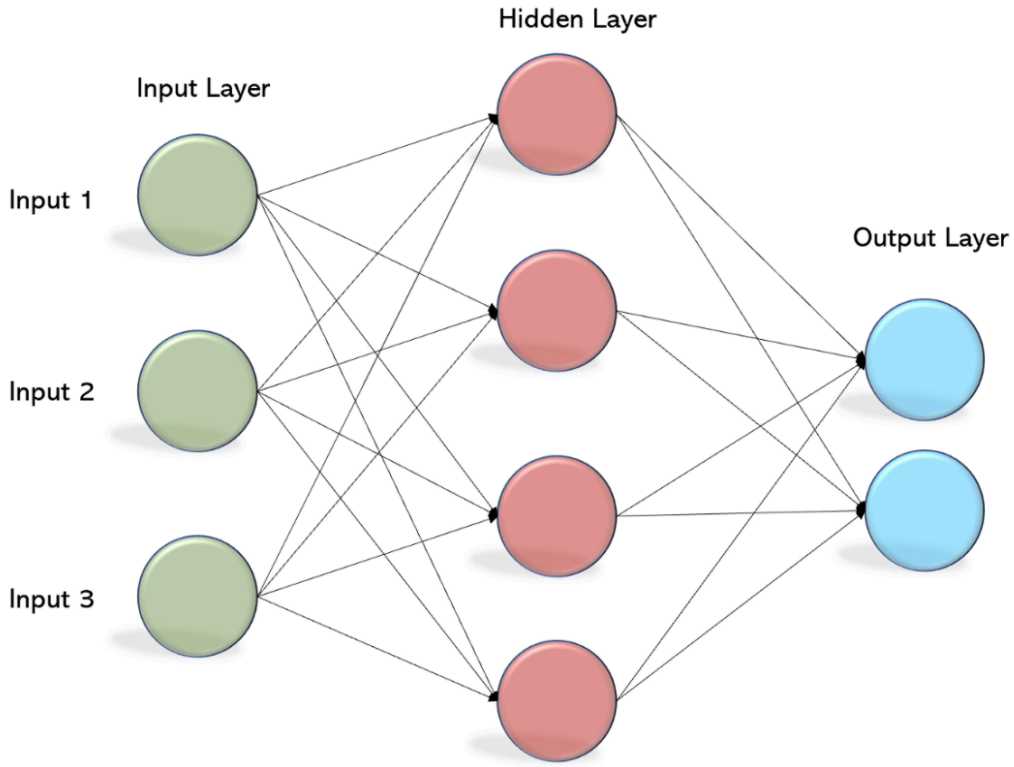
hyper tangent function

Activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Multi Layer Perceptron

- A multi layer perceptron (MLP) is a class of feed forward artificial neural network. MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training.



Learning Perceptron

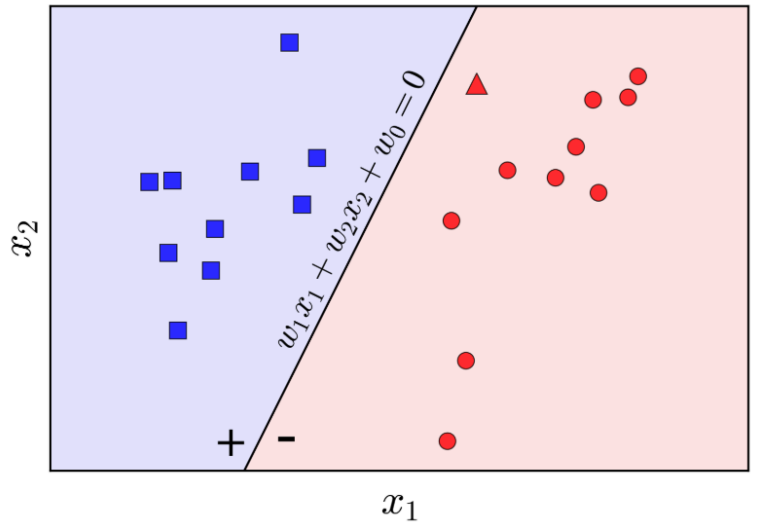
$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$$

$$\mathbf{y} = [y_1, y_2, \ldots, y_N] \in \mathbb{R}^{1 \times N},$$

$$f_{\mathbf{w}}(\mathbf{x}) = w_1 x_1 + \cdots + w_d x_d \quad \text{label}(\mathbf{x}) = 1 \text{ if } \mathbf{w}^T\mathbf{x} \geq 0, \text{otherwise} - 1$$
$$= \mathbf{w}^T\bar{\mathbf{x}} = 0$$

$$\text{label}(\mathbf{x}) = 1 \text{ if } \mathbf{w}^T\mathbf{x} \geq 0, \text{otherwise} - 1$$

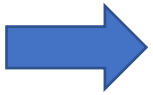
$$\text{label}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x})$$



Learning Perceptron: Loss function

$$J_1(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \mathrm{sgn}(\mathbf{w}^T \mathbf{x_i}))$$

Loss function counts the missed classified, in that case y_i and $\mathrm{sgn}(w^T x)$ are opposite signs, therefore: $-y_i \mathrm{sgn}(w^T xi)=1$.
However, it is discrete then difficult to calculate derivative



$$J(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \mathbf{w}^T \mathbf{x_i})$$

Learning Perceptron: update weight

$$J(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \mathbf{w}^T \mathbf{x_i})$$

Loss function

$$\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{x}_i; y_i) = -y_i \mathbf{x}_i$$

derivative

$$\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$$

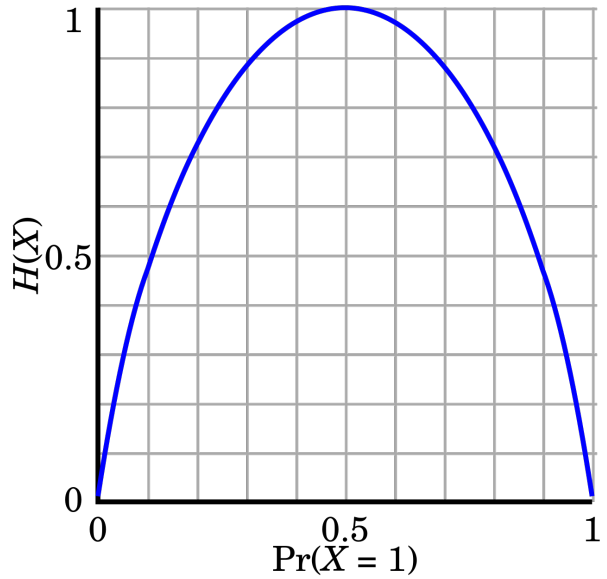
Update weight

Logistic Regression

Entropy

Entropy is a **measure** of the unpredictability of the state, or equivalently, of its average **information** content.

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$



p=0.7, then

$$\begin{aligned}
\mathrm{H}(X) &= -p \log_2(p) - q \log_2(q) \\
&= -0.7 \log_2(0.7) - 0.3 \log_2(0.3) \\
&\approx -0.7 \cdot (-0.515) - 0.3 \cdot (-1.737) \\
&= 0.8816 < 1
\end{aligned}$$

$$\begin{aligned}
\mathrm{H}(X) &= -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i) \\
&= -\sum_{i=1}^{2} \frac{1}{2} \log_2 \frac{1}{2} \\
&= -\sum_{i=1}^{2} \frac{1}{2} \cdot (-1) = 1
\end{aligned}$$

Kullback–Leibler divergence

In mathematical statistics, the **Kullback–Leibler divergence** (also called **relative entropy**) is a measure of how one probability distribution is different from a second, reference probability distribution.

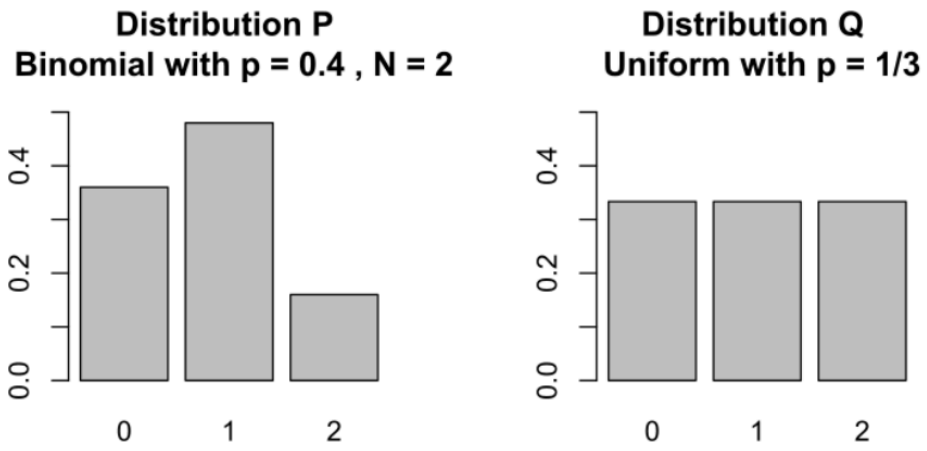
For discrete probability distributions P and Q defined on the same probability space, $\mathcal{X}$, the Kullback–Leibler divergence from Q to P is defined[4] to be

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right).$$

which is equivalent to

$$D_{\mathrm{KL}}(P \parallel Q) = -\sum_{x \in \mathcal{X}} P(x) \log\left(\frac{Q(x)}{P(x)}\right)$$

Kullback–Leibler divergence



x	0	1	2
Distribution *P*(x)	0.36	0.48	0.16
Distribution *Q*(x)	0.333	0.333	0.333

$$
\begin{aligned}
D_{\mathrm{KL}}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right) \\
&= 0.36 \ln\left(\frac{0.36}{0.333}\right) + 0.48 \ln\left(\frac{0.48}{0.333}\right) + 0.16 \ln\left(\frac{0.16}{0.333}\right) \\
&= 0.0852996
\end{aligned}
$$

Kullback–Leibler divergence

x	0	1	2
Distribution *P*(x)	0.36	0.48	0.16
Distribution *Q*(x)	0.333	0.333	0.333

$$
\begin{aligned}
D_{\mathrm{KL}}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right) \\
&= 0.36 \ln\left(\frac{0.36}{0.333}\right) + 0.48 \ln\left(\frac{0.48}{0.333}\right) + 0.16 \ln\left(\frac{0.16}{0.333}\right) \\
&= 0.0852996
\end{aligned}
$$

Interpretations [edit]

The Kullback–Leibler divergence from Q to P is often denoted $D_{\mathrm{KL}}(P \parallel Q)$.

In the context of machine learning, $D_{\mathrm{KL}}(P \parallel Q)$ is often called the information gain achieved if Q is used instead of P. By analogy with information theory, it is also called the **relative entropy** of P with respect to Q. In the context of coding theory, $D_{\mathrm{KL}}(P \parallel Q)$ can be constructed by measuring the expected number of extra bits required to code samples from P using a code optimized for Q rather than the code optimized for P.

Cross Entropy

In information theory, the **cross entropy** between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p.

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p \| q) ,$$

where $H(p)$ is the entropy of p.

Cross Entropy

The cross entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over the discrete variable x and is given by

$$H(p, q) = -\sum_{\forall x} p(x) \log(q(x))$$

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p \| q)$$

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right)$$

$$H(p) = -\sum_{x} P(x) \cdot \log P(x)$$

$$= \sum_{x} P(x) \cdot \log\left(\frac{1}{P(x)}\right)$$

Cross Entropy: especial case

Having set up our notation, $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can use cross entropy to get a measure of dissimilarity between p and q:

$$H(p,q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

Cross Entropy: machine learning

The cross entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over the discrete variable x and is given by

$$H(p, q) = -\sum_{\forall x} p(x) \log(q(x))$$

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p \| q)$$

Cross Entropy: machine learning

$$H(p, q) = -\sum_{\forall x} p(x) \log(q(x))$$

try to minimize

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p \| q)$$

constant

try to minimize

Cross Entropy: especial case and machine learning

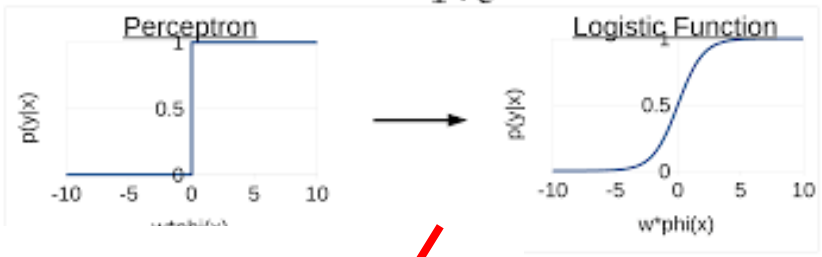
Having set up our notation, $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can use cross entropy to get a measure of dissimilarity between p and q:

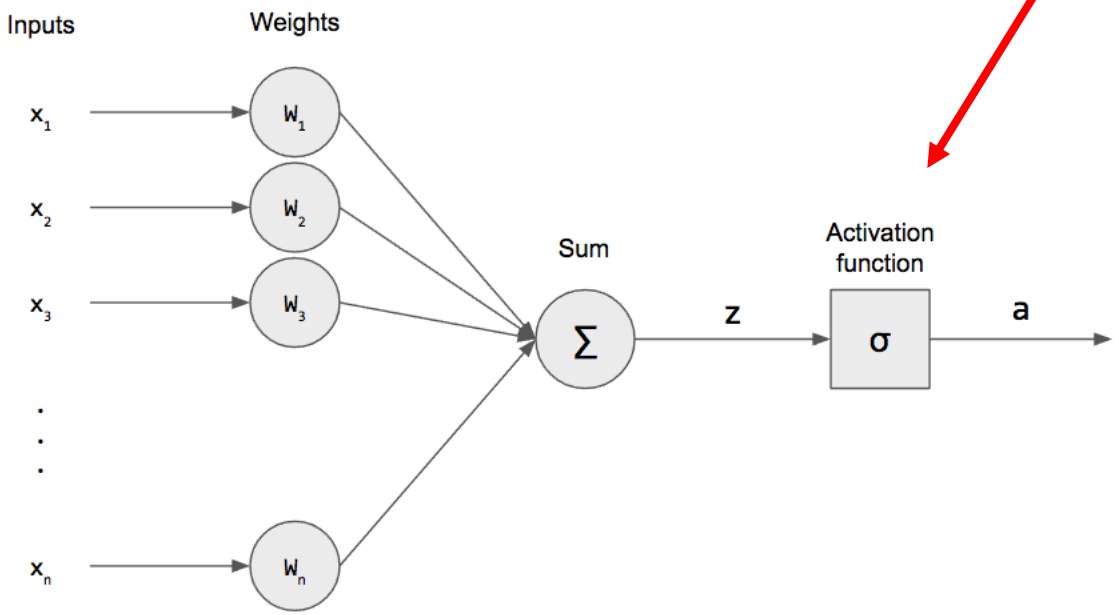
$$H(p,q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

learning = minimize

Logistic Regression

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$





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

Logistic Function

The goal is to predict the target class t from an input z. The probability $P(t = 1|z)$ that input z is classified as class $t = 1$ is represented by the output y of the logistic function computed as $y = \sigma(z)$. The logistic function σ is defined as:

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

We can write the probabilities that the class is $t = 1$ or $t = 0$ given input z as:

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

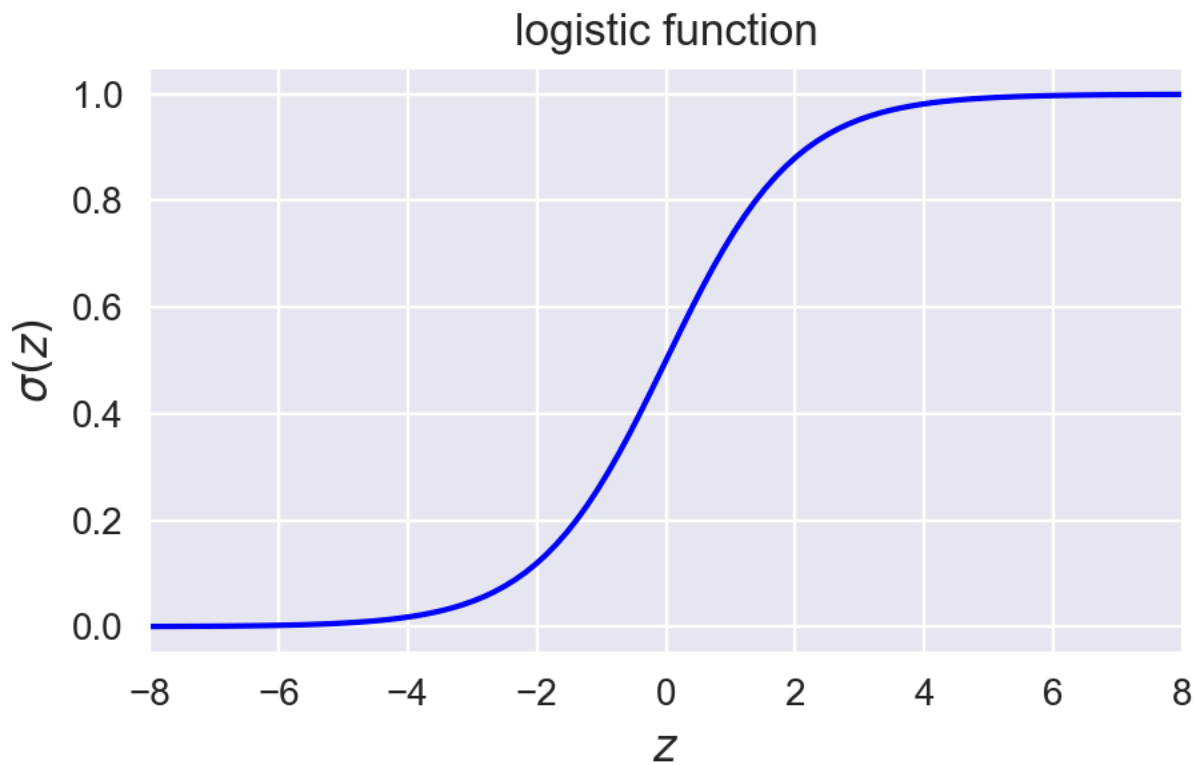
$$P(t = 0|z) = 1 - \sigma(z) = \frac{e^{-z}}{1 + e^{-z}}$$

Note that input z to the logistic function corresponds to the log odds ratio of $P(t=1|z)$ over $P(t=0|z)$.

$$\begin{aligned}\log \frac{P(t=1|z)}{P(t=0|z)} &= \log \frac{\frac{1}{1+e^{-z}}}{\frac{e^{-z}}{1+e^{-z}}} = \log \frac{1}{e^{-z}} \\ &= \log(1) - \log(e^{-z}) = z\end{aligned}$$

This means that the logg odds ratio $\log(P(t=1|z)/P(t=0|z))$ changes linearly with z. And if $z = x \cdot w$ as in neural networks, this means that the logg odds ratio changes linearly with the parameters w and input samples x.

```python
def logistic(z):
    """Logistic function."""
    return 1 / (1 + np.exp(-z))
```



Derivative of the logistic function

Since neural networks typically use gradient based opimization techniques such as gradient descent it is important to define the derivative of the output y of the logistic function with respect to its input z. $\partial y/\partial z$ can be calculated as:

$$\frac{\partial y}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{\partial \frac{1}{1+e^{-z}}}{\partial z} = \frac{-1}{(1+e^{-z})^2} \cdot e^{-z} \cdot -1 = \frac{1}{1+e^{-z}} \frac{e^{-z}}{1+e^{-z}}$$

And since $1 - \sigma(z)) = 1 - 1/(1+e^{-z}) = e^{-z}/(1+e^{-z})$ this can be rewritten as:

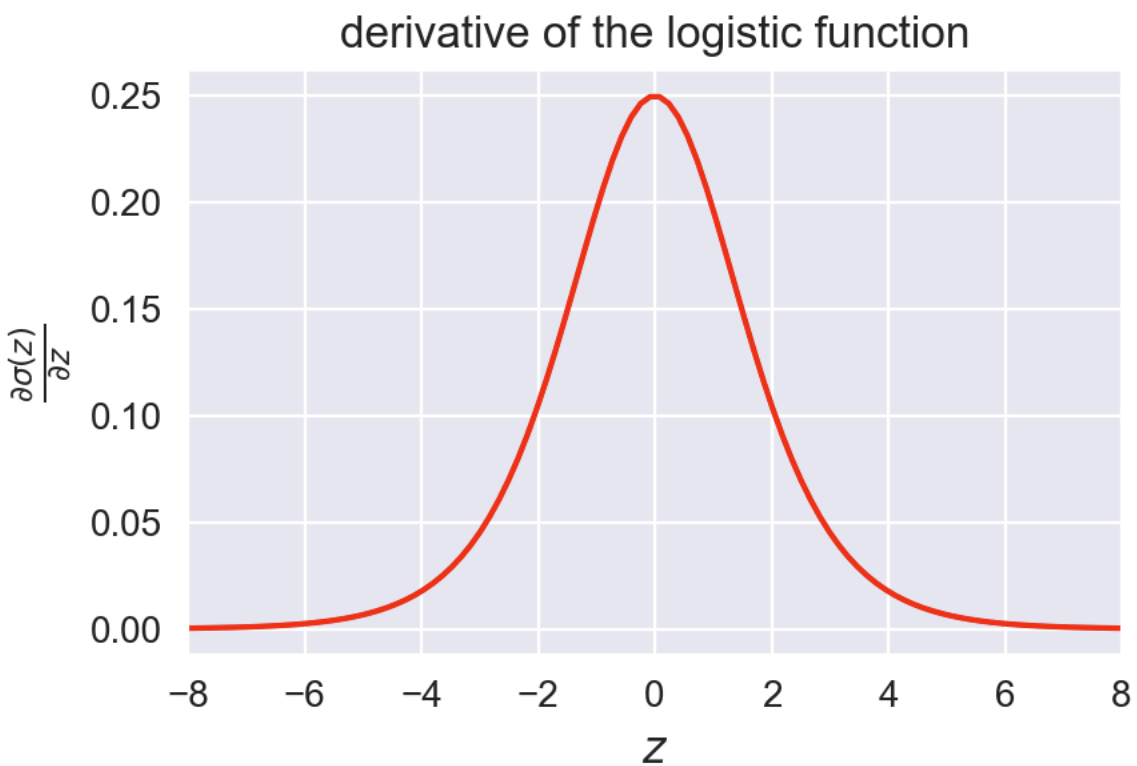
$$\frac{\partial y}{\partial z} = \frac{1}{1+e^{-z}} \frac{e^{-z}}{1+e^{-z}} = \sigma(z) \cdot (1 - \sigma(z)) = y(1-y)$$

This derivative is implemented as `logistic_derivative(z)` and is plotted below.

```
def logistic_derivative(z):
    """Derivative of the logistic function."""
    return logistic(z) * (1 - logistic(z))
```

```python
def logistic_derivative(z):
    """Derivative of the logistic function."""
    return logistic(z) * (1 - logistic(z))
```

```python
# Plot the derivative of the logistic function
```



Cross-entropy loss function for the logistic function

Having set up our notation, $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can use cross entropy to get a measure of dissimilarity between p and q:

$$H(p,q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

Cross-entropy ~ Log likelihood

The neural network model will be optimized by maximizing the likelihood that a given set of parameters θ of the model can result in a prediction of the correct class of each input sample. The parameters θ transform each input sample i into an input to the logistic function z_i. The likelihood maximization can be written as:

$$\underset{\theta}{\operatorname{argmax}}\ \mathcal{L}(\theta|t,z) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} \mathcal{L}(\theta|t_i,z_i)$$

$$\underset{\theta}{\operatorname{argmax}}\ \mathcal{L}(\theta|t,z) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} \mathcal{L}(\theta|t_i,z_i)$$

The likelihood $\mathcal{L}(\theta|t,z)$ can be rewritten as the joint probability of generating t and z given the parameters θ: $P(t,z|\theta)$. Since $P(A,B) = P(A|B)P(B)$ this can be written as:

$$P(t,z|\theta) = P(t|z,\theta)P(z|\theta)$$

Since we are not interested in the probability of z we can reduce this to: $\mathcal{L}(\theta|t,z) = P(t|z,\theta) = \prod_{i=1}^{n} P(t_i|z_i,\theta)$. Since t_i is a Bernoulli variable , and the probability $P(t|z) = y$ is fixed for a given θ we can rewrite this as:

$$\begin{aligned} P(t|z) &= \prod_{i=1}^{n} P(t_i = 1|z_i)^{t_i} \cdot (1 - P(t_i = 1|z_i))^{1-t_i} \\ &= \prod_{i=1}^{n} y_i^{t_i} \cdot (1-y_i)^{1-t_i} \end{aligned}$$

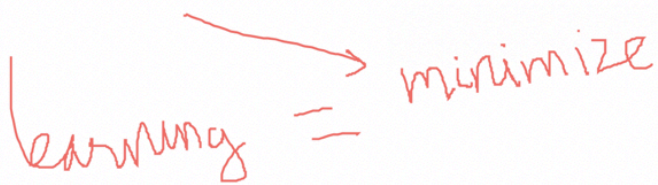
Since the logarithmic function is a monotone increasing function we can optimize the log-likelihood function $\underset{\theta}{\operatorname{argmax}} \log \mathcal{L}(\theta|t,z)$. This maximum will be the same as the maximum from the regular likelihood function. The benefit of using the log-likelihood is that it can prevent numerical underflow when the probabilities are low. The log-likelihood function can be written as:

$$\log \mathcal{L}(\theta|t,z) = \log \prod_{i=1}^{n} y_i^{t_i} \cdot (1-y_i)^{1-t_i}$$

$$= \sum_{i=1}^{n} t_i \log(y_i) + (1-t_i) \log(1-y_i)$$

Having set up our notation, $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can use cross entropy to get a measure of dissimilarity between p and q:

$$H(p,q) = -\sum_{i} p_i \log q_i = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$



Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\frac{\partial}{\partial\theta}J(\theta) = -\frac{\partial}{\partial\theta}\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\,log(h_\theta(x^{(i)})) + (1-y^{(i)})\,log(1-h_\theta(x^{(i)}))\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\,\frac{\partial}{\partial\theta}log(h_\theta(x^{(i)})) + (1-y^{(i)})\,\frac{\partial}{\partial\theta}log(1-h_\theta(x^{(i)}))\right)$$

$$\frac{d}{dz}log(z) = \frac{1}{z}$$

$$\frac{d}{dz}h(z) = h(z)(1-h(z))$$

$$\frac{\partial log(h(z))}{\partial\theta} = \frac{\partial log\,h(z)}{\partial z}\cdot\frac{\partial z}{\partial\theta}$$

$$= \frac{\partial log\,h(z)}{\partial h(z)}\cdot\frac{\partial h(z)}{\partial z}\cdot\frac{\partial z}{\partial\theta}$$

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

$$\frac{\partial y}{\partial z} = \frac{1}{1+e^{-z}}\,\frac{e^{-z}}{1+e^{-z}} = \sigma(z)\cdot(1-\sigma(z)) = y(1-y)$$

$$\frac{\partial}{\partial\theta}J(\theta) = -\frac{\partial}{\partial\theta}\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\,log(h_\theta(x^{(i)})) + (1-y^{(i)})\,log(1-h_\theta(x^{(i)}))\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\,\frac{\partial}{\partial\theta}log(h_\theta(x^{(i)})) + (1-y^{(i)})\,\frac{\partial}{\partial\theta}log(1-h_\theta(x^{(i)}))\right)$$

$$\frac{d}{dz}log(z) = \frac{1}{z}$$

$$\frac{d}{dz}h(z) = h(z)(1-h(z))$$

$$\frac{\partial log(h(z))}{\partial\theta} = \frac{\partial log\,h(z)}{\partial z}\cdot\frac{\partial z}{\partial\theta}$$

$$= \frac{\partial log\,h(z)}{\partial h(z)}\cdot\frac{\partial h(z)}{\partial z}\cdot\frac{\partial z}{\partial\theta}$$

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \frac{x_j^{(i)}}{h(x^{(i)})} h(x^{(i)})(1 - h(x^{(i)})) + (1 - y^{(i)}) \frac{x_j^{(i)}}{1 - h(x^{(i)})} (-h(x^{(i)})(1 - h(x^{(i)}))) \\
&= -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} (1 - h(x^{(i)})) - (1 - y^{(i)}) h(x^{(i)})) x_j^{(i)} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - y^{(i)} h(x^{(i)}) - h(x^{(i)}) + y^{(i)} h(x^{(i)}) \right) x_j^{(i)} \\
&= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) x_j^{(i)}
\end{aligned}
$$

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$