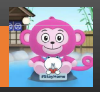
Chapter 10 ANNs
Artificial Neural Networks

Associate Professor Yachai Limpiyakorn Ph.D.

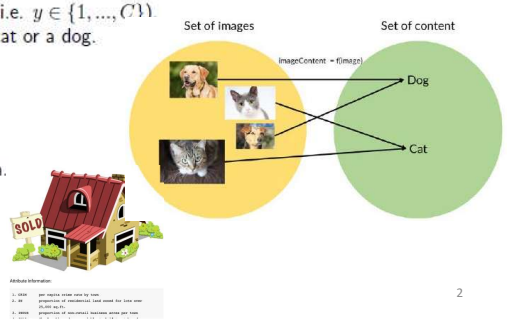


Neural Network (NN)

- Consuming time & resources during learning phase
- Black box model (not human understandable)
- Classification or Regression

Classification
Output y belongs to one of C predetermined classes (i.e. $y \in \{1, ..., C\}$).
For example, determine whether the picture shows a cat or a dog.

Regression
Output y is continuous (i.e. $y \in \mathbb{R}$).
For example, predict the price of a house given its area.



Approximating functions

- Think about neural networks as function approximators
  y= f(x) or *y=f(X) ;* where input vector *X*
- Given dataset as below, the function want to approximate is

X_1	X_2	X_3	y
0	1	0	0
1	0	0	1
1	1	1	1
0	1	1	0

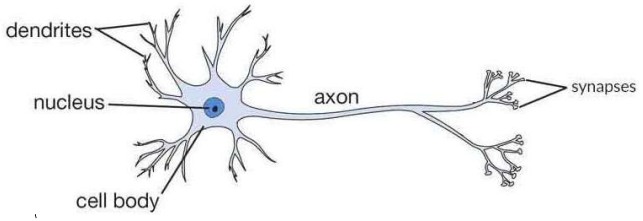
$$f(X) = X_1$$

- Functions expressed by neural networks can become very complex, and it is not possible to write down the function.

  imageContent = f(image) ; where image stored as matrix of numbers

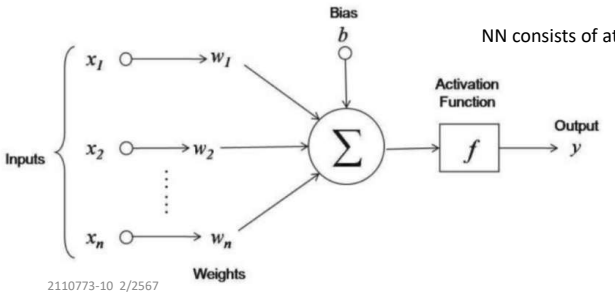
- A neural network, if it is big enough, can approximate any function.

Biological Neuron



Perceptron

NN consists of atomic units called *Perceptron* ≈ human *Neuron*



Binary Activation Function

Bipolar Activation Function

$$o(x_1,x_2,\ldots,x_n)=\begin{cases}1 & if\ w_1x_1+w_2x_2+\cdots+w_nx_n>\theta\\ -1 & if\ w_1x_1+w_2x_2+\cdots+w_nx_n<\theta\end{cases}$$ (สูตรที่1)

จากฟังก์ชันในสูตรที่ 1 เราจัดรูปใหม่โดยย้าย θ ไปรวมกับผลรวมเชิงเส้นแล้วแทน $-\theta$ ด้วย w_0 ($-w_0$ คือ ค่าขีดแบ่ง θ) เราจะได้ฟังก์ชันของเอาท์พุตต่อไปนี้

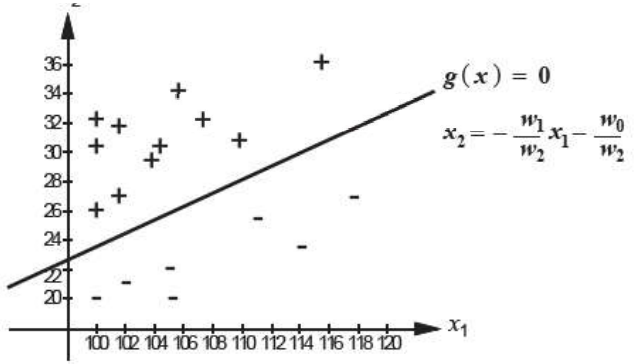
$$o(x_1,x_2,\ldots,x_n)=\begin{cases}1 & if\ w_0+w_1x_1+w_2x_2+\cdots+w_nx_n>0\\ -1 & if\ w_0+w_1x_1+w_2x_2+\cdots+w_nx_n<0\end{cases}$$ (สูตรที่2)

กำหนดให้ $g(\vec{x})=\sum_{i=0}^{n}w_ix_i=\vec{w}\cdot\vec{x}$

โดยที่ $\vec{x}$ แทนเวกเตอร์อินพุต เราสามารถเขียนฟังก์ชันของเอาท์พุตได้ใหม่ดังนี้

$$o(x_1,x_2,\ldots,x_n)=\begin{cases}1 & if\ g(\vec{x})>0\\ -1 & if\ g(\vec{x})<0\end{cases}$$ (สูตรที่3)

เพอร์เซปตรอนเป็นระนาบตัดสินใจหลายมิติ (hyperplane decision surface) ในกรณีที่มีอินพุต 2 ตัว (ไม่รวม x_0) เราจะได้ $g(\vec{x})=w_0+w_1x_1+w_2x_2$ ซึ่งถ้าเราให้ $g(\vec{x})=0$ จะได้ว่า $w_0+w_1x_1+w_2x_2=0$ ซึ่งแทนสมการเส้นตรงในระนาบสองมิติ



Perceptron

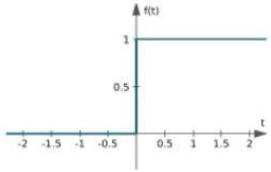
Perceptron algorithm is one of the oldest methods for binary classification.

Decision rule

$$\hat{y}=f(\boldsymbol{w}^T\boldsymbol{x}+w_0)$$

where f is the step function defined as:

$$f(t)=\begin{cases}1 \text{ if } t>0,\\ 0 \text{ else.}\end{cases}$$

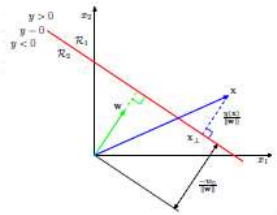


Bipolar Activation Function

Binary Activation Function

Hyperplane as a decision boundary

For a 2 class problem ($\mathcal{C}=\{0,1\}$) we can try to separate points from the two classes by a hyperplane.



A hyperplane be defined by a normal vector w and an offset w_0.

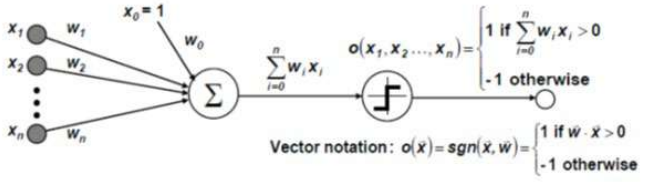
$$\boldsymbol{w}^T\boldsymbol{x}+w_0\begin{cases}=0 & \text{if } \boldsymbol{x} \text{ on the plane}\\ >0 & \text{if } \boldsymbol{x} \text{ on normal's side}\\ <0 & \text{else}\end{cases}$$

$$f_{\boldsymbol{w}}(\boldsymbol{x}_i)=w_0+w_1x_{i1}+w_2x_{i2}+\ldots+w_Dx_{iD}$$

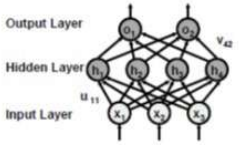
Hyperplanes are computationally very convenient: easy to evaluate.

A data set $\mathcal{D}=\{(\boldsymbol{x}_i,y_i)\}$ is linearly separable if there exists a hyperplane for which all $\boldsymbol{x}_i$ with $y_i=0$ are on one and all $\boldsymbol{x}_i$ with $y_i=1$ on the other side.

Perceptron vs. Multi-Layer Perceptron (MLP)



- Single Neuron Model
  - Linear Threshold Unit (LTU) inputs to unit: defined as linear combination
  - Output of unit: threshold (activation) function on net input (threshold $\theta = w_0$)
- Neural Networks
  - Neuron is modeled using a unit connected by weighted links w_i to other units
  - Multi-Layer Perceptron (MLP): future lecture



Perceptron Training

- เริ่มจากการสุ่มค่าน้ำหนัก W_i
- สอนเพอร์เซปตรอนกับทุกตัวอย่างที่สอนทีละตัว โดย
  - คำนวณผลรวมเชิงเส้นของอินพุตทุกตัว
  - คำนวณหาค่าเอาท์พุตจากฟังก์ชันกระตุ้น
  - คำนวณหาค่าความผิดพลาดระหว่างเอาท์พุตที่ได้จากฟังก์ชันกระตุ้นและเอาท์พุตที่แท้จริง
  - แก้ไขน้ำหนักเมื่อเพอร์เซปตรอนแยกตัวอย่างผิดพลาด
- วนทำซ้ำกับตัวอย่างที่สอน จนกระทั่งเพอร์เซปตรอนแยกตัวอย่างได้ถูกต้องทั้งหมด

Weight Update

$$W_i \leftarrow W_i + \Delta W_i$$

โดยที่ $$\Delta W_i = \eta (t-o) x_i$$

เมื่อ t เป็นเอาท์พุตเป้าหมาย หรือผลลัพธ์ที่ถูกต้อง

o เป็นเอาท์พุตที่แท้จริง หรือผลลัพธ์ที่ได้จากเพอร์เซปตรอน

(t-o) คือ ค่าผิดพลาด

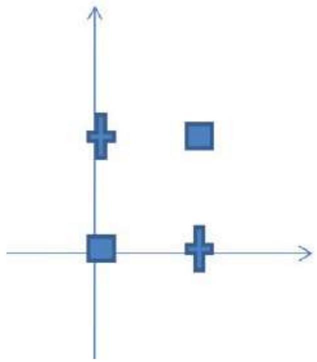
η เป็นค่าที่แสดงอัตราการเรียนรู้ (learning rate) เป็นค่าคงที่บวกจำนวนน้อยๆ

- กรณี t=1, o=-1 (ใช้ฟังก์ชันกระตุ้นสองขั้ว) หมายความว่า เพอร์เซปตรอนให้ผลรวมเชิงเส้นน้อยเกินไปและน้อยกว่า 0 น้ำหนักจึงต้องถูกปรับให้สามารถเพิ่มค่า $\sum w_i x_i$ เพื่อที่จะทำให้เพอร์เซปตรอนให้ผลลัพธ์ค่า 1 กล่าวคือ
  - W_i ของ x_i ที่เป็นค่าบวกจะถูกปรับเพิ่มขึ้น
  - W_i ของ x_i ที่เป็นค่าลบจะถูกปรับลดลง
- ในทางตรงกันข้าม เมื่อ t=-1, o=1 เพื่อให้การปรับเป็นไปในทิศทางที่ถูกต้อง
  - W_i ของ x_i ที่เป็นค่าลบจะถูกปรับเพิ่มขึ้น
  - W_i ของ x_i ที่เป็นค่าบวกจะถูกปรับลดลง

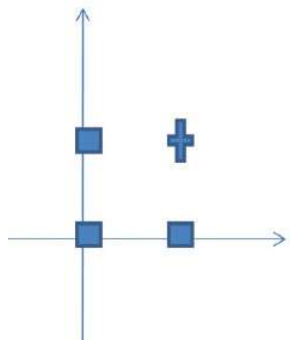
Choice of Learning Rate

Learning rate size	Advantages/disadvantages
Smaller learning rate	Converges slower but more accurate results
Larger learning rate	Less accurate, but converges faster

Linearly Nonseparable XOR Function



Linearly Separable AND Function



Perceptron Learning of Function AND Using Binary Activation Function

Bias Input x0 = +1 Learning Rate (η) = 0.7

Input X1	Input X2	1.0*W0	X1*W1	X2*W2	Net Sum Input	Target Output (t)	Actual Output (o)	η(t-o)	Weight Values		
									ΔW0	ΔW1	ΔW2
									0.1	0.1	0.1
0	0	0.1	0	0	0.1	0	1	-0.7	-0.7	0	0
0	1	0.1	0	0.1	0.2	0	1	-0.7	-0.7	0	-0.7
1	0	0.1	0.1	0	0.2	0	1	-0.7	-0.7	-0.7	0
1	1	0.1	0.1	0.1	0.3	1	1	0	0	0	0
Update W_i									-2	-0.6	-0.6
0	0	-2	0	0	-2	0	0	0	0	0	0
0	1	-2	0	-0.6	-2.6	0	0	0	0	0	0
1	0	-2	-0.6	0	-2.6	0	0	0	0	0	0
1	1	-2	-0.6	-0.6	-3.2	1	0	0.7	0.7	0.7	0.7
Update W_i									-1.3	0.1	0.1
0	0	-1.3	0	0	-1.3	0	0	0	0	0	0
0	1	-1.3	0	0.1	-1.2	0	0	0	0	0	0
1	0	-1.3	0.1	0	-1.2	0	0	0	0	0	0
1	1	-1.3	0.1	0.1	-1.1	1	0	0.7	0.7	0.7	0.7
Update W_i									-0.6	0.8	0.8
0	0	-0.6	0	0	-0.6	0	0	0	0	0	0
0	1	-0.6	0	0.8	0.2	0	1	-0.7	-0.7	0	-0.7
1	0	-0.6	0.8	0	0.2	0	1	-0.7	-0.7	-0.7	0
1	1	-0.6	0.8	0.8	1	1	1	0	0	0	0
Update W_i									-2	0.1	0.1
0	0	-2	0	0	-2	0	0	0	0	0	0
0	1	-2	0	0.1	-1.9	0	0	0	0	0	0
1	0	-2	0.1	0	-1.9	0	0	0	0	0	0
1	1	-2	0.1	0.1	-1.8	1	0	0.7	0.7	0.7	0.7
Update W_i									-1.3	0.8	0.8
0	0	-1.3	0	0	-1.3	0	0	0	0	0	0
0	1	-1.3	0	0.8	-0.5	0	0	0	0	0	0
1	0	-1.3	0.8	0	-0.5	0	0	0	0	0	0
1	1	-1.3	0.8	0.8	0.3	1	1	0	0	0	0
Update W_i									-1.3	0.8	0.8

Perceptron Learning of Function XOR Using Binary Activation Function

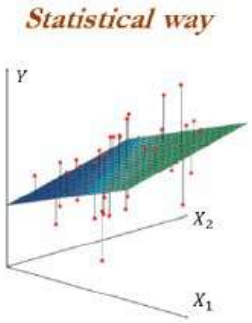
Bias Input x0 = +1 Learning Rate (η) = 0.7

Input X1	Input X2	1.0*W0	X1*W1	X2*W2	Net Sum Input	Target Output (t)	Actual Output (o)	η(t-o)	Weight Values		
									ΔW0	ΔW1	ΔW2
									0.1	0.1	0.1
0	0	0.1	0	0	0.1	0	1	-0.7	-0.7	0	0
0	1	0.1	0	0.1	0.2	1	1	0	0	0	0
1	0	0.1	0.1	0	0.2	1	1	0	0	0	0
1	1	0.1	0.1	0.1	0.3	0	1	-0.7	-0.7	-0.7	-0.7
Update W_i									-1.3	-0.6	-0.6
0	0	-1.3	0	0	-1.3	0	0	0	0	0	0
0	1	-1.3	0	-0.6	-1.9	1	0	0.7	0.7	0	0.7
1	0	-1.3	-0.6	0	-1.9	1	0	0.7	0.7	0.7	0
1	1	-1.3	-0.6	-0.6	-2.5	0	0	0	0	0	0
Update W_i									0.1	0.1	0.1
0	0	0.1	0	0	0.1	0	1	-0.7	-0.7	0	0
0	1	0.1	0	0.1	0.2	1	1	0	0	0	0
1	0	0.1	0.1	0	0.2	1	1	0	0	0	0
1	1	0.1	0.1	0.1	0.3	0	1	-0.7	-0.7	-0.7	-0.7
Update W_i									-1.3	-0.6	-0.6
0	0	-1.3	0	0	-1.3	0	0	0	0	0	0
0	1	-1.3	0	-0.6	-1.9	1	0	0.7	0.7	0	0.7
1	0	-1.3	-0.6	0	-1.9	1	0	0.7	0.7	0.7	0
1	1	-1.3	-0.6	-0.6	-2.5	0	0	0	0	0	0
Update W_i									0.1	0.1	0.1
0	0	0.1	0	0	0.1	0	1	-0.7	-0.7	0	0
0	1	0.1	0	0.1	0.2	1	1	0	0	0	0
1	0	0.1	0.1	0	0.2	1	1	0	0	0	0
1	1	0.1	0.1	0.1	0.3	0	1	-0.7	-0.7	-0.7	-0.7
Update W_i									-1.3	-0.6	-0.6
0	0	-1.3	0	0	-1.3	0	0	0	0	0	0
0	1	-1.3	0	-0.6	-1.9	1	0	0.7	0.7	0	0.7
1	0	-1.3	-0.6	0	-1.9	1	0	0.7	0.7	0.7	0
1	1	-1.3	-0.6	-0.6	-2.5	0	0	0	0	0	0
Update W_i									0.1	0.1	0.1

Example Network for XOR

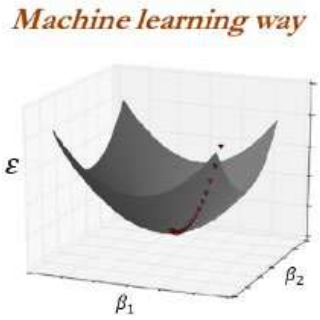
Statistical Modeling vs. Machine Learning

- For a quick view, in statistical modeling, linear regression with two independent variables is trying to fit the best plane with the least errors, whereas when constructing a model, machine learning tries to minimize the error between the predictions and the expected outcomes (ground truth).
- That error comes from the **loss function**.
- Optimization algorithms are the heart of machine learning algorithms.
- What exactly ML algorithms optimize?
- Machine learning utilizes **optimization methods** for tuning all the parameters of various algorithms.



$$\hat{Y} = \beta_1 * X_1 + \beta_2 * X_2$$

$$\varepsilon = (Y - (\beta_1 * X_1 + \beta_2 * X_2))^2$$



Loss Function

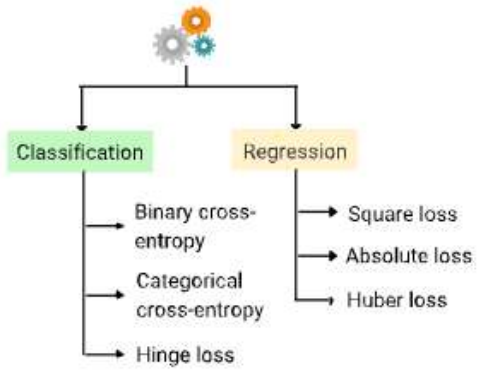
Output of loss function called ***Loss*** which is a measure of how well the model can predict the outcome

A high value of ***Loss*** means the model performs very poorly, while a low value is preferrable.

Selection of the proper loss function is critical for training an accurate model.

Gradient descent is a famous optimizer algorithm to help finding the optimum values of parameters faster.

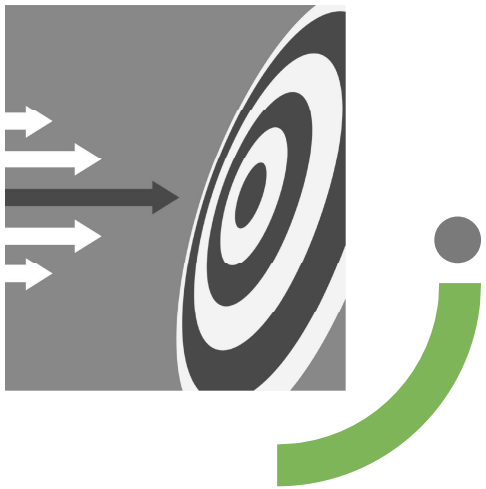
Famous Loss Function

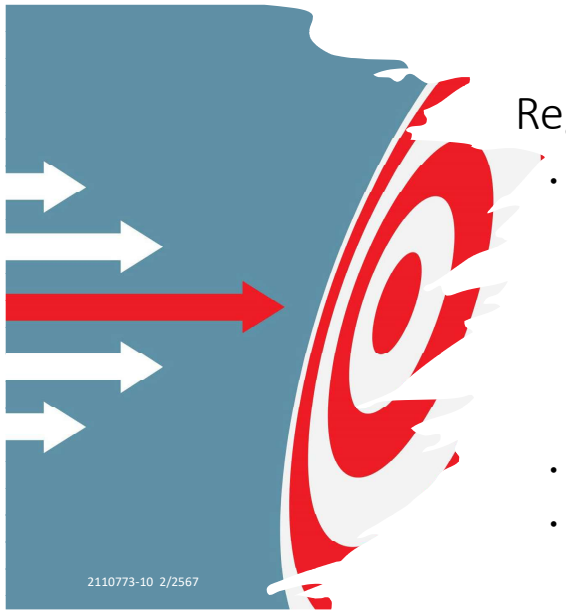


https://www.enjoyalgorithms.com/blog/loss-and-cost-functions-in-machine-learning

Loss Function vs. Cost Function

- Loss function is associated with every training example.
- Cost function is the average value of the loss function over all training samples.
- Usually try to optimize cost function rather than loss function





Regression Loss Function

- In regression tasks, we try to predict the continuous target variables. Suppose we are trying to fit the function f using machine learning on the training data X = [X1, X2, ... , Xn] so that f(x) fits Y = [Y1, Y2, ... , Yn]. But this function f can not be perfect, and there will be errors in the fitting.
- Absolute Error
- Square Error

Absolute Error / L1 Loss

L1 norm loss or **absolute loss function** is not smooth at the target, resulting in algorithms not converging well.

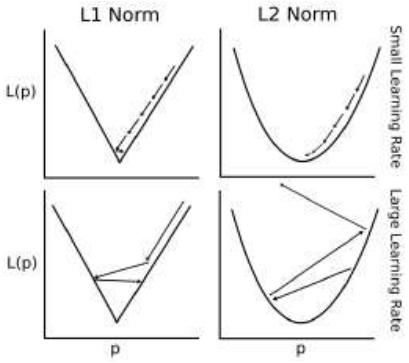
$$Loss_i = |Y_i - f(x_i)|$$

Widely used in industries, especially when training data is more prone to outliers

L1 loss is more robust to outliers than L2, or when difference is higher, L1 is more stable than L2

The corresponding cost function is Mean Absolute Error (MAE)

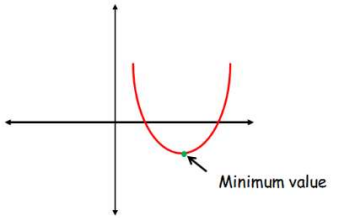
$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$



Square Error / L2 Loss

- L2 norm loss or **Euclidean loss function** is a quadratic equation that only has a global minimum and no local minima.
- One of the most favorable loss functions as it is very curved near the target and algorithms can converge to the target closer to zero.
- L2 loss is more stable than L 1 loss, especially when the difference between prediction and actual is smaller.
- However, the squaring part magnifies the error if the model makes very bad prediction.

$$Loss_i = (Y_i - f(x_i))^2$$



The corresponding cost function is Mean Squared Error (MSE) which is less robust to outlier presence.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Classification Loss Function

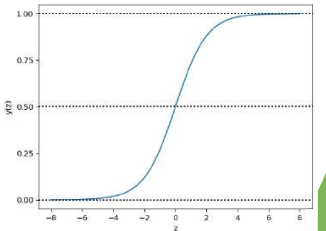
- Binary Cross-Entropy (two classes e.g. yes/no, true/false, cat/dog)
- Categorical Cross-Entropy (multiple classes)

Log Loss (Binary Cross-Entropy)

$$Loss_i = -(Y * log(P) + (1 - Y) * log(1 - P)) = -log(P), \quad \text{if } Y = 1$$
$$= -log(1 - P), \quad \text{if } Y = 0$$

- Use for measuring how well a model's predicted probabilities align with true outcomes
- Inputs:
  - Y : **true** label (1 or 0) for a given data point
  - P : predicted probability (between 0 and 1) that the data point belongs to class 1
- Sigmoid function is used to calculate P, where z represents the input parameters to model.
- If Y=1, we want P (predicted prob. of being in class 1) as close to 1
- If Y=0, we want P as close to 0 i.e. (1-P) is close to 1
- The corresponding cost function

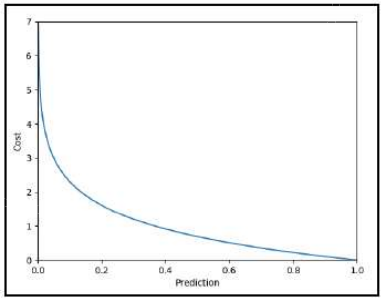
$$Cost = \frac{\Sigma_i^n Loss_i}{n}$$



$$P = \frac{1}{1 + e^{-z}}$$

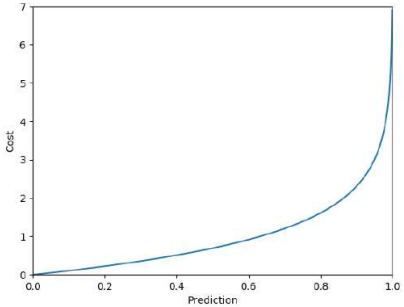
Consider loss of a **single** training sample

$$j(w) = -y^{(i)} log(\hat{y}(x^{(i)})) - (1 - y^{(i)}) log(1 - \hat{y}(x^{(i)}))$$
$$= \begin{cases} -log(\hat{y}(x^{(i)})), & if\ y^{(i)} = 1 \\ -log(1 - \hat{y}(x^{(i)})), & if\ y^{(i)} = 0 \end{cases}$$



If *y(i)*=1, when it predicts correctly (positive class in 100% probability), the sample cost function is *0*

if *y(i)=0*, when it predicts correctly (negative class in 100% probability), the sample cost function is *0*



Categorical Cross-Entropy

$$J(w) = -\sum_{i=1}^{n} [y_i \times log(\phi(z_i))]$$

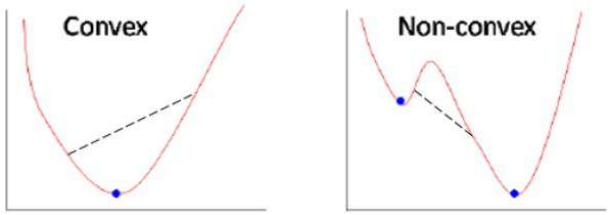
Here, $\phi(z)$ is the *softmax* function

$$softmax\ \ \phi_i(z) = \frac{e_i^z}{\sum_j e_j^z} = \frac{e_i^{(w \times x + b)}}{\sum_j e_j^{(w \times x + b)}}$$

- Instead of sigmoid function, *softmax* function is used to produce the probabilities for each class so that the probabilities vector adds up to *1*.
- At the time of inference, the class with the highest *softmax* value becomes the output or predicted class.
- The loss function, as discussed earlier, can be minimized by the optimizers, such as gradient descent.

Convex and Non-convex Function

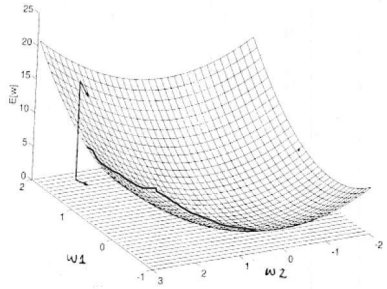
- Before stepping into gradient descent, it is important to know whether the function is convex or non-convex due to the fact that in convex functions, the local optimum is also the global optimum, whereas for non-convex functions, the local optimum does not guarantee the global optimum.



Delta Rule

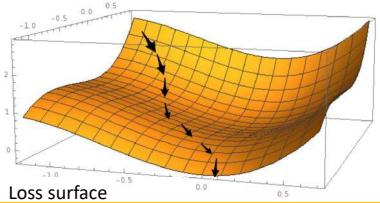
- การเรียนรู้จะลู่เข้าสู่ระนาบหลายมิติที่ให้ค่าผิดพลาดน้อยสุด
- ใช้หลักการ ***gradient descent*** โดยการหาอนุพันธ์ทางคณิตศาสตร์
- จำเป็นต้องใช้ฟังก์ชันกระตุ้นที่หาอนุพันธ์ได้
- Linear Function $o(\vec{x}) = \vec{w} \cdot \vec{x} = \sum w_i x_i$
- นิยามฟังก์ชันค่าผิดพลาด (error function)

$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$



Gradient Descent Learning Method

- Following the steepest gradient down a surface which represents the error function as it tries to find the minimum of the error function and by doing so decrease the error. The *learning rate* determines the size of steps taken to reach the minimum



Loss surface

Weight update with Delta Rule:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{โดยที่} \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d \in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d \in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d \in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D}(t_d - o_d)(-x_{id})$$

$$\Delta w_i = \eta \sum_{d \in D}(t_d - o_d)x_{id}$$

Full Batch Gradient Descent

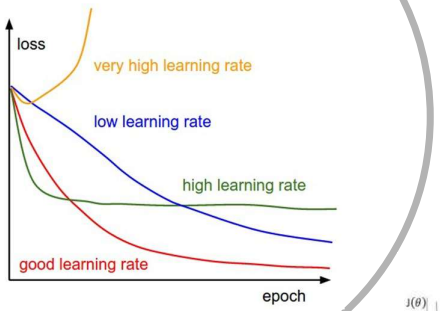
(all training observations considered in each and every iteration)

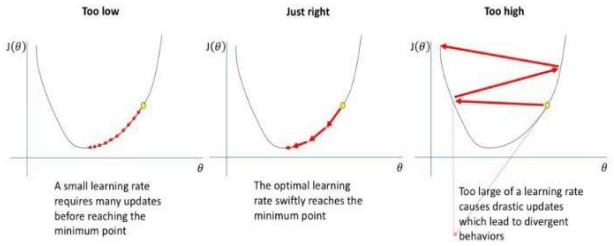
- takes a lot of memory and slow
- not practical, to have all the huge observations to update the weights.
- provide the best way of updating parameters with less noise at the expense of huge computation.

Stochastic Gradient Descent

(update weights by taking one observation per iteration)

- randomness may help move out of local minimums
- provide the quickest way of traversing weights; however, a lot of noise is involved while converging.





Gradient Descent

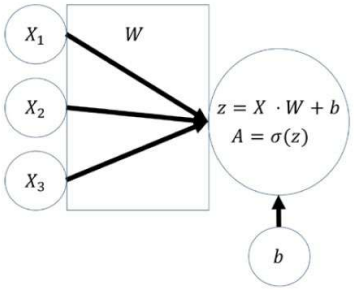
- Gradient descent requires that the loss function has a derivative with respect to the parameters we want to optimize.
- Here, gradient descent can only optimize the parameters, weights, and biases
- What it cannot do is optimize how many layers our model has or which activation functions it should use, since there is no way to compute the gradient with respect to model topology.
- These settings, which cannot be optimized by gradient descent, are called **hyperparameters** and are usually set by humans.

Logistic Regression

- Logistic regression is a probabilistic classifier, similar to Naive Bayes
- Simplest NN for *binary classification*
- Applications e.g. predict likelihood of homeowner defaulting mortgage
- Take all kinds of values into account when trying to predict the likelihood of someone defaulting on their payment, such as the debtor's salary, whether they have a car, the security of their job, and so on.
- But the likelihood will always be a value between zero and one. Even the worst debtor ever cannot have a default likelihood above 100%, and the best cannot go below 0%

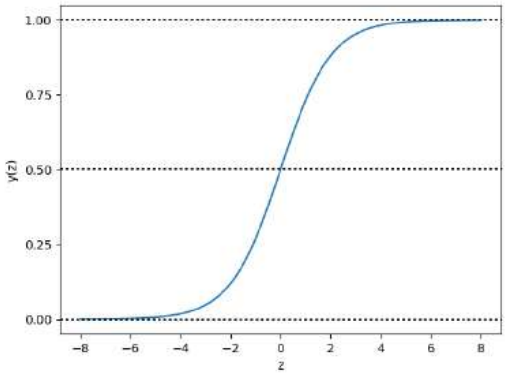
Logistic Regressor

- To compute the output of the regressor, first do a linear step.
- Compute the dot product of the input, X, and the weights, W which is a vector of 3 weights (different line thickness)
- Take the sum, then add the bias, b.
- Afterward, we do a nonlinear step.



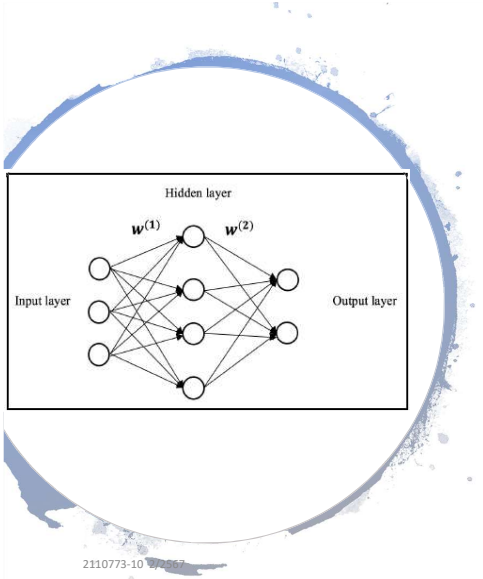
- In the nonlinear step, run the linear intermediate product, *z*, through an **activation function** (sigmoid function). The sigmoid function squishes the input values to outputs between zero and one

$$y(z) = \frac{1}{1 + exp(-z)}$$



Logistic function

- more commonly referred to as sigmoid function
- S-shaped curve, all inputs are transformed into range from 0 to 1.
- for positive inputs, a greater value results in an output closer to 1
- for negative inputs, a smaller value generates an output closer to 0
- when the input is 0, the output is the midpoint, 0.5



Neural Networks 1

- A neural network is composed of three layers, the **Input layer**, **Hidden layer**, and **Output layer**
- Nodes simulate neurons in a biological brain
- Edges connecting two adjacent layers are considered as synapses in a biological brain, which transmit signal from one neuron in a layer to another neuron in the next layer
- Edges are parameterized by the weights **W** of the model

Neural Networks 2

Input layer represents the input features *x*, each node is a predictive feature *x*

Output layer represents the target variable(s)

In multiclass classification, the output layer consists of n nodes (number of possible classes), the value of each node is the probability of predicting that class

In binary classification, the output layer contains only one node

Hidden layer can be considered a composition of latent information extracted from the previous layer

Learning with a neural network with two or more hidden layers is called Deep Learning (DL). More hidden layers enable learning more about the underneath relationship between the input data and target.

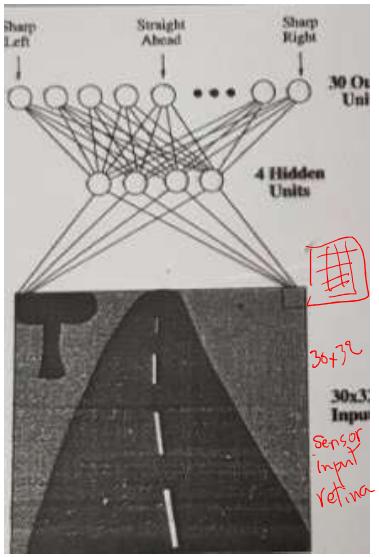
Applications of Neural Networks

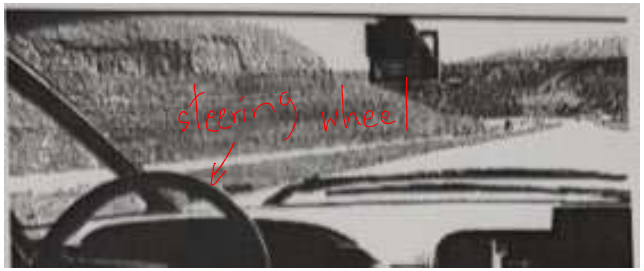
Images and videos: To identify an object in an image or to classify whether it is a dog or a cat

Text processing (NLP): Deep-learning-based chatbot and so on

Speech: Recognize speech

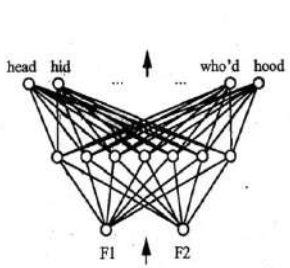
Structured data processing: Building highly powerful models to obtain a non-linear decision boundary

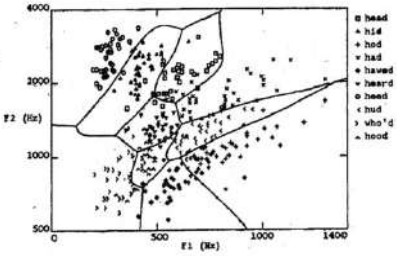




Example ANN Application

เน็ตเวิร์กหลายชั้นและแบ็คพรอพาเกชัน (Multilayer Network and Backpropagation)





การจำแนกประเภทเสียงของคำ “h_d”

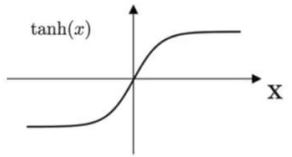
Neural networks with one or more hidden layers between the input and output layer should be able to learn more about the underneath relationship between the input data and target. With the more number of hidden layers are being added to the neural network, more complex decision boundaries are being created to classify different categories.

Building Neural Networks

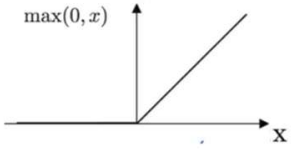
- **Activation function:** Choosing an activation function plays a major role in aggregating signals into the output signal to be propagated to the other neurons of the network.
- **Network architecture or topology:** This represents the number of layers required and the number of neurons in each layer. More layers and neurons will create a highly non-linear decision boundary, whereas if we reduce the architecture, the model will be less flexible and more robust.
- **Training optimization algorithm:** The selection of an optimization algorithm plays a critical role as well, in order to converge quickly and accurately to the best optimal solutions.

Activation Function

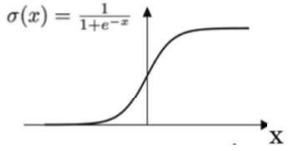
Hyper Tangent Function



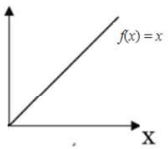
ReLU Function



Sigmoid Function



Identity Function



Activation Function defines output of a neuron

- In **Binary Step Function**, if the value of Y is above a certain value known as the threshold, the output is True(or activated), and if it's less than the threshold, then the output is false (or not activated).
- **Identity function**, also known **Linear function** is used in linear regression problems, where it always provides a derivative as 1 due to the function used being f(x) = x.
- **Sigmoid Function** called S-shaped functions. Two types of sigmoid/ logistic functions:
  - **Binary Sigmoid Function** $\sigma(x) = 1 / (1+e^{-x})$ takes a real number and squashes it into the range [0, 1]. Sigmoid makes calculating derivatives easy and easy to interpret.
  - **Bipolar Sigmoid Function** also known as Hyperbolic Tangent Function or tanh. Tanh squashes the input real number into the range [-1, 1]. The output is zero-centered. Tanh non-linearity is scaled sigmoid neuron $tanh(x) = 2\sigma(2x) - 1$.
- **ReLu** or Rectified Linear unit. It output 0 for negative values of x. It provides faster convergence property.

Neural Networks 3

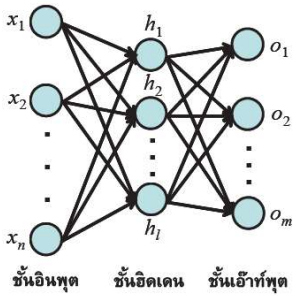
How can we obtain the optimal weights W = {W(1), W(2)} of the model?

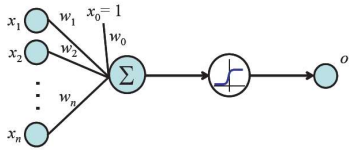
Similar to logistic regression, all weights are learned using gradient descent with the goal of minimizing the loss function J(W).

The difference is that the gradients ΔW are computed through backpropagation.

Multi-layer Feed Forward (MLFF)

- Each layer is *fully connected* with subsequent layer.
- The output of each artificial neuron in a layer is an input to every artificial neuron in the next layer towards the output.
- The hidden layer can be considered a composition of latent information extracted from the previous layer.
- There can be more than one hidden layer. Learning with a neural network with two or more hidden layers is called Deep Learning.
- *Logistic regression* is a feed-forward neural network with no hidden layer where the output layer connects directly with the input.





Backpropagation Neural Network Learning

For each input entry in the training data set:

- feed input data in (feed forward)
- check output against desired value and feed back error (back-propagate)

Where back-propagation:

- calculate error gradients
- update weights

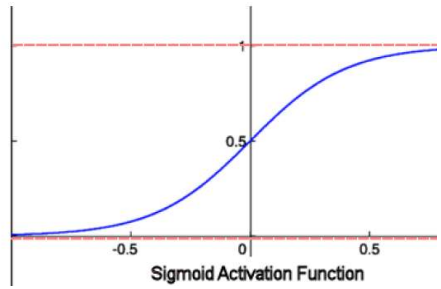
Stopping conditions:

- desired accuracy , elapsed epochs

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

$$\frac{d\sigma(x)}{dx} = \sigma(x) \times (1 - \sigma(x))$$

nction has the following graph :



- Calculate error gradient for each output neuron k

$$\delta_k = y_k(1-y_k)(d_k - y_k)$$
where y_k is the value at output neuron k
and d_k is the desired value at output neuron k

- Calculate error gradient for each hidden neuron j

$$\delta_j = y_j(1-y_j)\sum_{k=1}^{n} w_{jk}\ \delta_k$$

$$w_{ij} = w_{ij} + \Delta w_{ij} \text{ and } w_{jk} = w_{jk} + \Delta w_{jk}$$

$$\text{where } \Delta w_{ij}(t) = \alpha.inputNeuron_i.\delta_j$$
$$\text{and } \Delta w_{jk}(t) = \alpha.hiddenNeuron_j.\delta_k$$

$\alpha - learning\ rate$
$\delta - error\ gradient$

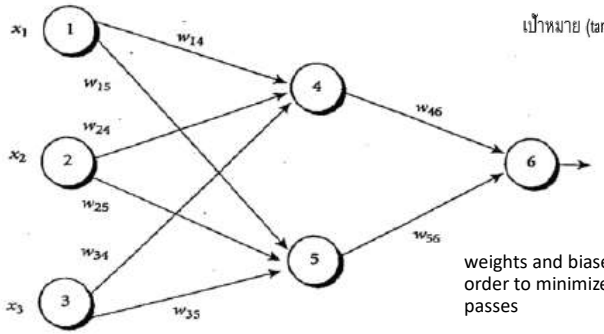
Weight Update

$$w_i \leftarrow w_i + \Delta w_i \text{ โดยที่ } \Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$
$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$
$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$
$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$
$$\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d) - x_{id}$$

ตัวอย่างการปรับค่าน้ำหนักด้วย Backpropagation

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1



เป้าหมาย (target output) เท่ากับ 1 และใช้อัตราการเรียนรู้ η=0.9

weights and biases are randomly initiated, and updated in order to minimize the errors during forward and backward passes

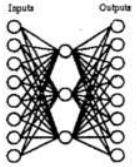
Unit j	Net input, I_j	Output, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

Weight or bias	New value
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

Hidden Layer

- ข้อดีของ Backpropagation คือ ความสามารถในการค้นพบการแทนสารสนเทศที่เป็นประโยชน์ไว้ในชั้นซ่อน (hidden layer) ซึ่งสามารถดักจับคุณสมบัติของอินพุตที่มีผลหรือเกี่ยวข้องมากที่สุดในการเรียนรู้ฟังก์ชันเป้าหมาย
- ตัวอย่างการเรียนรู้ฟังก์ชันเป้าหมาย f(X) = X หรือ Identity function ซึ่งใช้โครงสร้าง MLFF (8 x 3 x 8) ประกอบด้วยโหนดในชั้นอินพุต 8 โหนด รับข้อมูลเข้าที่มีค่า 0, 1 จำนวน 8 แบบที่แตกต่างกัน และแสดงค่าผลลัพธ์ที่ตรงกันในชั้นเอาต์พุต



A target function:

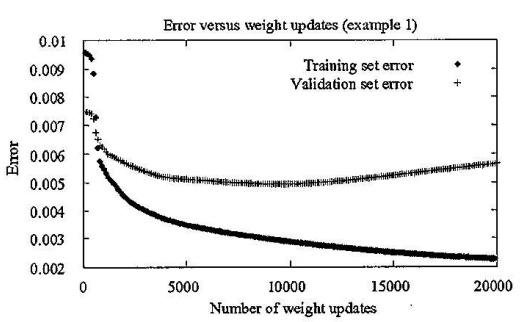
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

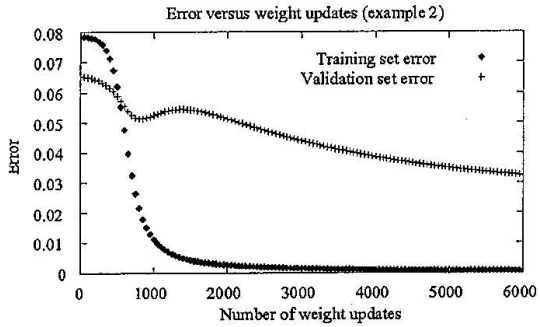
ผลลัพธ์การเรียนรู้ของ

Identity function

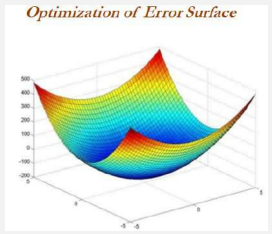
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Training Overfit





Optimization of Neural Networks



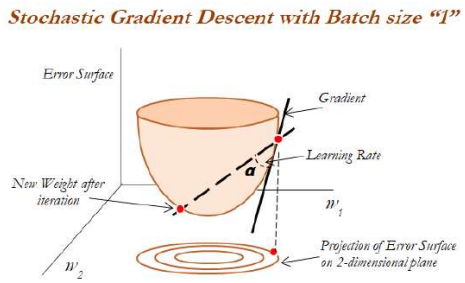
Various techniques for optimizing weights:

- Stochastic gradient descent (SGD)
- Momentum
- Nesterov accelerated gradient (NAG)
- Adaptive gradient (Adagrad)
- Adadelta
- RMSprop
- Adaptive moment estimation (Adam)
- Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

In practice, **Adam** is a good default choice.
Try L-BFGS if you cannot afford full batch updates.

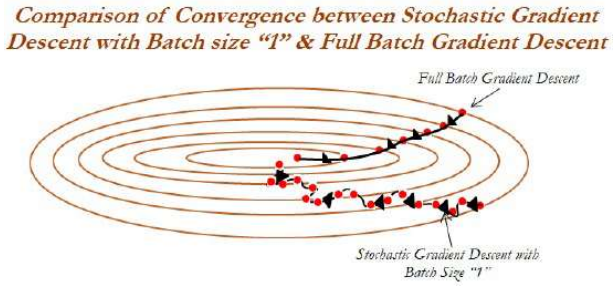
Stochastic Gradient Descent - SGD

- Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameter $\theta \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function with regard to the parameters. The learning rate determines the size of the steps taken to reach the minimum:
  - Batch gradient descent (all training observations utilized in each iteration)
  - SGD (one observation per iteration)
  - Mini batch gradient descent (size of about 50 training observations for each iteration)



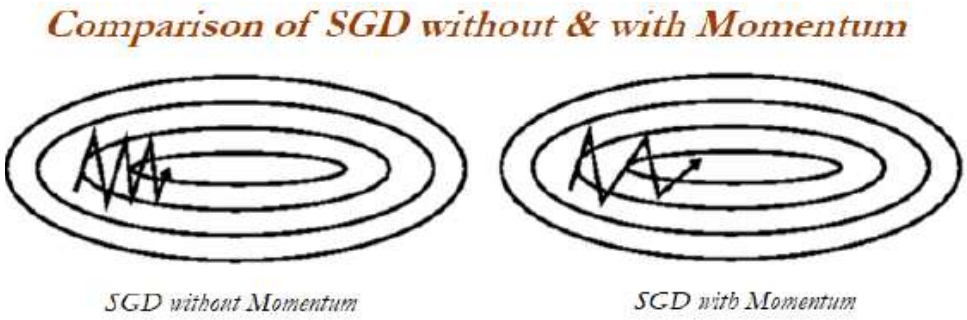
Projection of error surface on 2-dim plane

- Full batch updates, are smoother due to the consideration of all the observations. Whereas, SGD have wiggly convergence characteristics due to the reason of using *1* observation for each update.



Momentum

The momentum term increases for dimensions whose gradients point in the same direction and reduces updates for dimensions whose gradients change direction. As a result, we gain faster convergence and reduced oscillations



Hyperparameter optimization

Getting the last out of your NN, you need to tune:

- number of hidden layers (1, 2, 3, ...)
- number of hidden units (50, 100, 200, ...)
- type of activation function (sigmoid, tanh, rectifier, ...)
- learning method (adam, adadelta, rprop, ...)
- learning rate
- data preprocessing
- ...

Grid Search for hyperparameter tuning will loop through predefined hyperparameters and fit the model on the training set. For example, GridSearchCV in Sklearn library