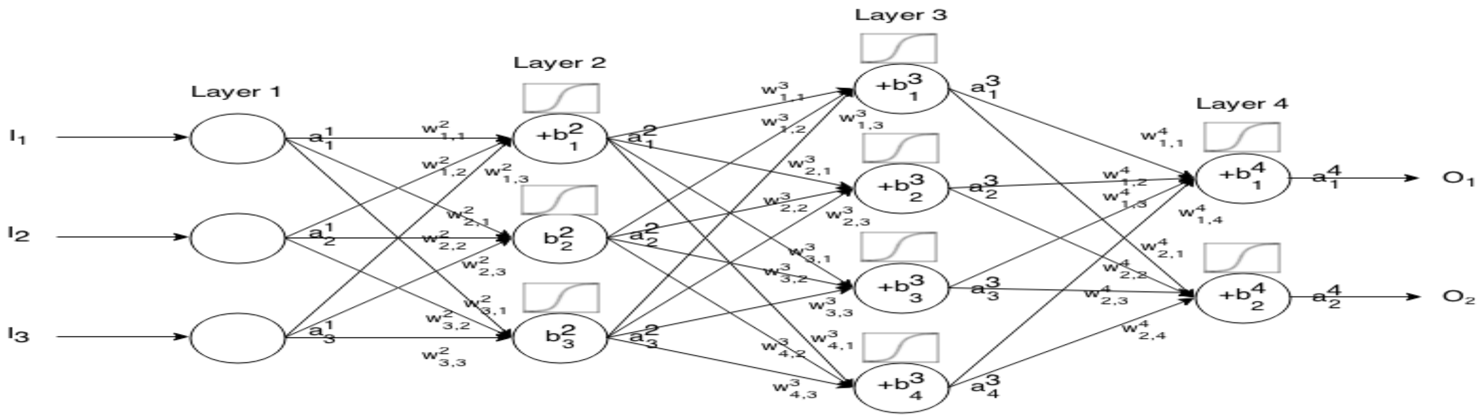
Artificial Intelligence and Machine Learning. 6CS012.

Lecture-5:Multi layer Neural Network.

Siman Giri.

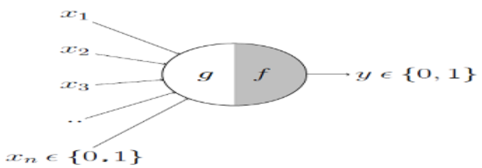
A. Motivation for Multi Layer-Neural Network.

What are Multi-layer Neural Network?

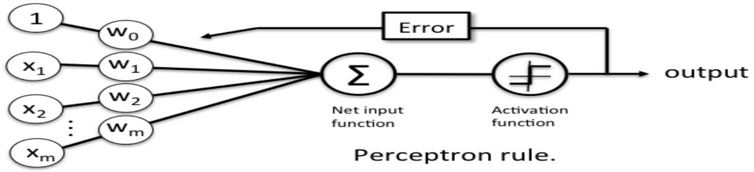


Story So Far….

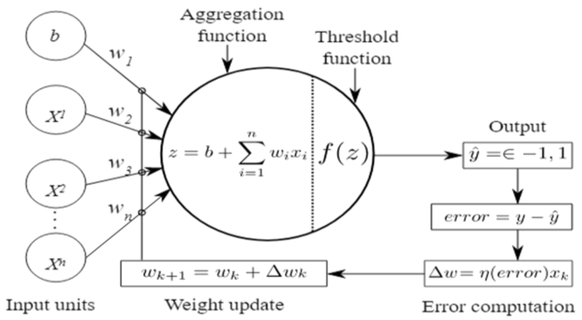
- MCP was first simple computational Model that emulates Human Neuron behavior.



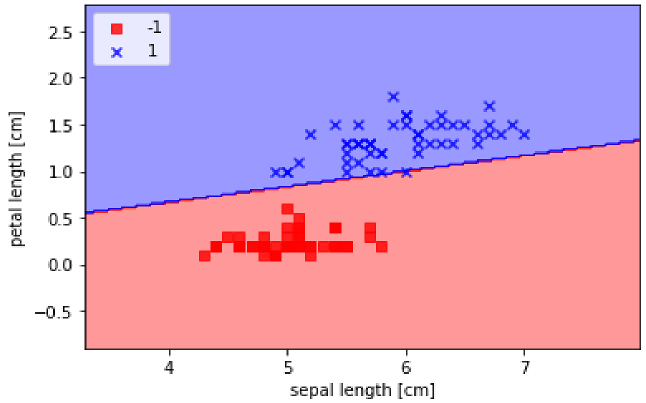
- Perceptron were computationally better representations of Human Neuron.

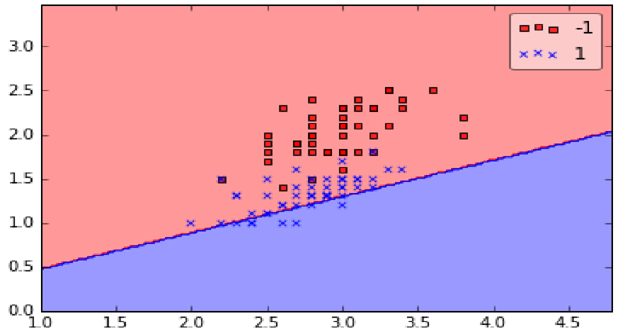
- 

l.



- Linear Separability-Decision Boundry.

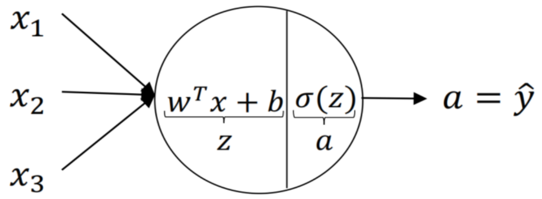




- Multi-Layer of Perceptron and Theory of Universal Approximations.

What are Multi-layer Neural Networks?

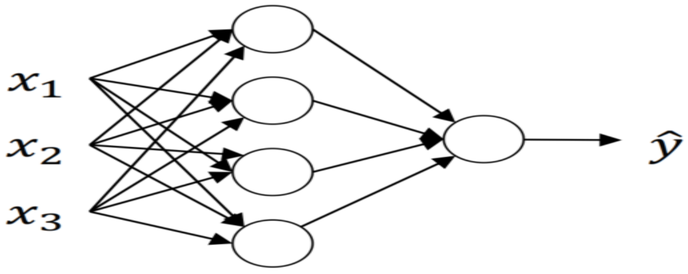
Single Perceptron



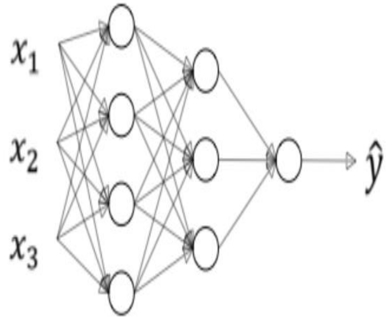
$$z = w^T x + b$$

$$a = \sigma(z)$$

One-Hidden Layer

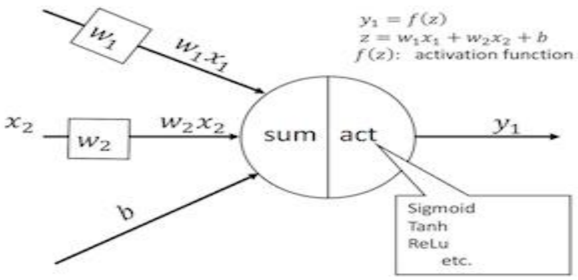


Two-Hidden Layer



1. Activation Functions.

1.1 What are Activation Functions?



- Activation Functions introduce **non-linearity** to the **output** of neurons.
- The activation function does the **non-linear transformation to the input**, making it **capable to learn and perform more complex tasks**.
- The activation function should be **differentiable** or the concept of updating weights (**Backpropagation**)fails, which is the core idea of deep learning.

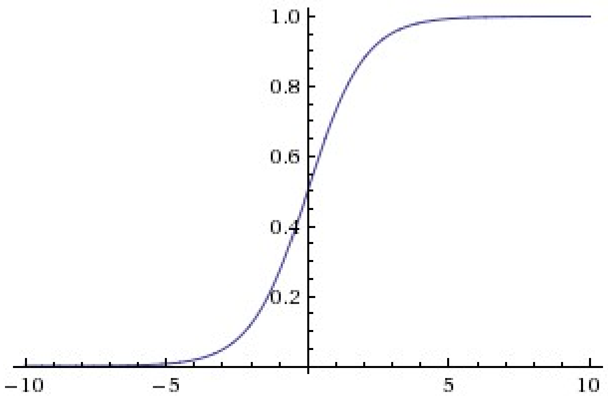
1.2 Some Common Non Linear Activation Functions.

Sigmoid

- Mathematically;

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

- Squashes real numbers to range between [0,1].



Pros and Cons.

- the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used.
  - Logistic sigmoid can cause a neural network to get "stuck" during training. This is because, if a strongly-negative input is provided to the logistic sigmoid, it outputs a value, which is very near to zero.
  - Because of this behavior, updating weights will be slow and they are less regularly updated. In simple terms, weights which are updated through back-propagation will be quite slow.

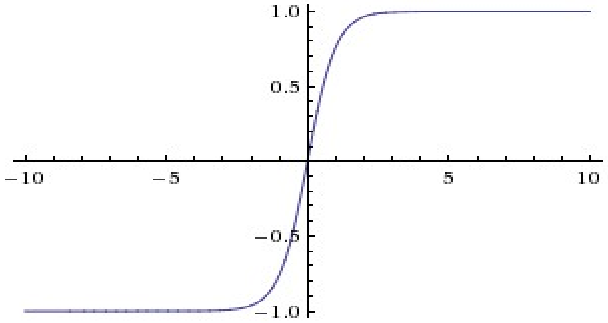
1.2 Some Common Non Linear Activation Functions.

Tanh(Hyperbolic Tangent Function)

- Mathematically;

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

- Squashes real numbers to range between $[-1,1]$.



- Scaled version of sigmoid neuron.

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

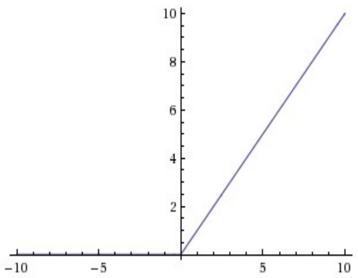
Pros and Cons.

- Outperforms the sigmoid activation functions.
- Zero centered output.

1.2 Some Common Non Linear Activation Functions.

ReLU

- It computes the function f(x)=max(0,x).



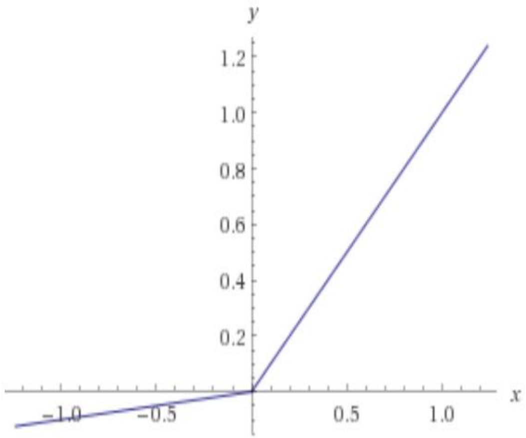
- Pros:
  - It was found to greatly accelerate the convergence of (stochastic) gradient descent compared to the sigmoid/tanh functions.
  - Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- Cons:
  - ReLU units can be fragile during training and can “die”.
    - What if, the current weights put the ReLU on the left flat side while it optimally should be on the right side for this particular input ?
    - The gradient is 0 and so the weight will not be updated, not even a tiny bit, so where is "learning" in this case?

Leaky ReLU

- Leaky ReLUs are one attempt to fix the “dying ReLU” problem.
- That is, the function computes:

$$f(x)=1(x<0)(\alpha x)+1(x>=0)(x)$$

- where α is a small constant.



1.3 What activation should I use?

- Never Use Sigmoid.
- Try tanh, but expect it to work worse than ReLU.
- While using ReLU/Leaky ReLU, be sensible while picking the learning rates.
- If possible, monitor the dead neuron in a network.

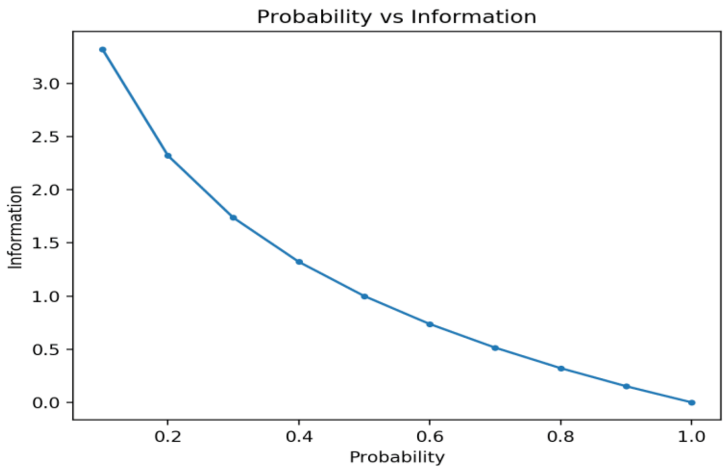
2. Error/Loss Function.

2.1 Cross Entropy Loss.

- CEL measures the distance between two probability distributions.

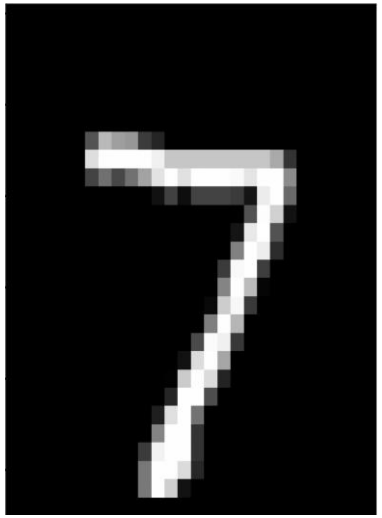
$$H(P^*|\,P) = -\sum_i P^*(i)\ \log P(i)$$

- In information theory, information are quantified using the level of surprise any event can cause.
  - h(x) = -log(p(x))
  - Low Probability Event: High Information (surprising).
  - High Probability Event: Low Information (unsurprising).



Where are We?
1.Activation Functions.
2.Error/Loss Functions.
  2.1 Cross Entropy Loss.

2.2 Problem Setup-Classification.



$input\ image - x_i$

$\boldsymbol{P^*(y|x_i)}$

$P(0|x_i)$
$P(1|x_i)$
$P(2|x_i)$
$P(3|x_i)$
$P(4|x_i)$
$P(5|x_i)$
$P(6|x_i)$
$P(7|x_i)$
$P(8|x_i)$
$P(9|x_i)$

$\mathbf{P(y|x_i;\theta)}$

$$D_{KL}(\underbrace{\mathbf{P^*(y|x_i)}}_{\text{TRUE CLASS DISTIRBUTION}}||\underbrace{\boldsymbol{P(y|x_i;\theta)}}_{\text{PREDICTED CLASS DISTIRBUTION}})$$

$$D_{KL}(P^*||P) = \sum_y P^*(y|x_i)\log\frac{P^*(y|x_i)}{P(y|x_i;\theta)}$$

$$= \sum_y P^*(y|x_i)\left[\log P^*(y|x_i) - \log P(y|x_i;\theta)\right]$$

$$= \sum_y \overbrace{P^*(y|x_i)\log P^*(y|x_i)}^{\text{DOESN'T DEPEND ON }\theta} - \sum_y P^*(y|x_i)\log P(y|x_i;\theta)$$

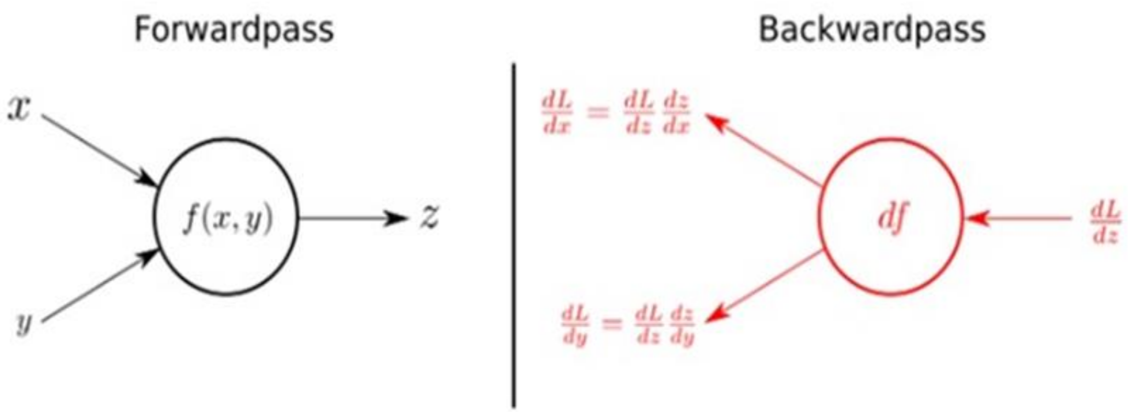
$$\underset{\theta}{\operatorname{argmin}}\, D_{KL}(P^*||P) \equiv \underset{\theta}{\operatorname{argmin}} -\sum_y P^*(y|x_i)\log P(y|x_i;\theta)$$

3. Calculation of Gradient.

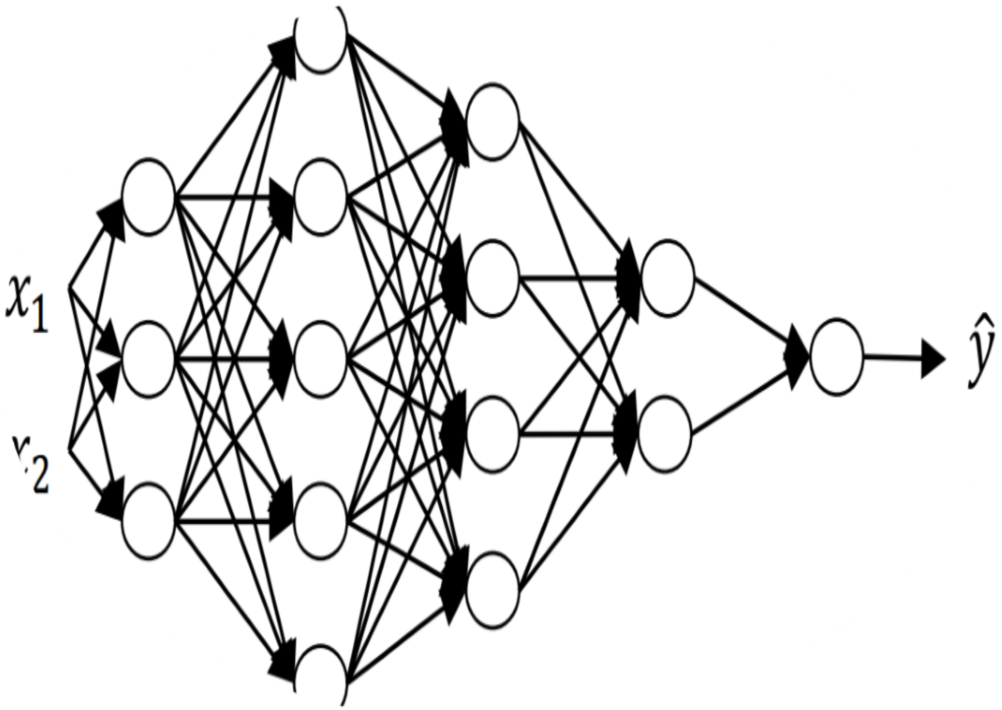
How do we Learn Weights?

3.1 Forward and Backward Propagations.

- The weights in Multi layer networks are learned with the combinations of forward and backward propagations.
- a network forward propagates activation to produce an output and it backward propagates error to determine weight changes



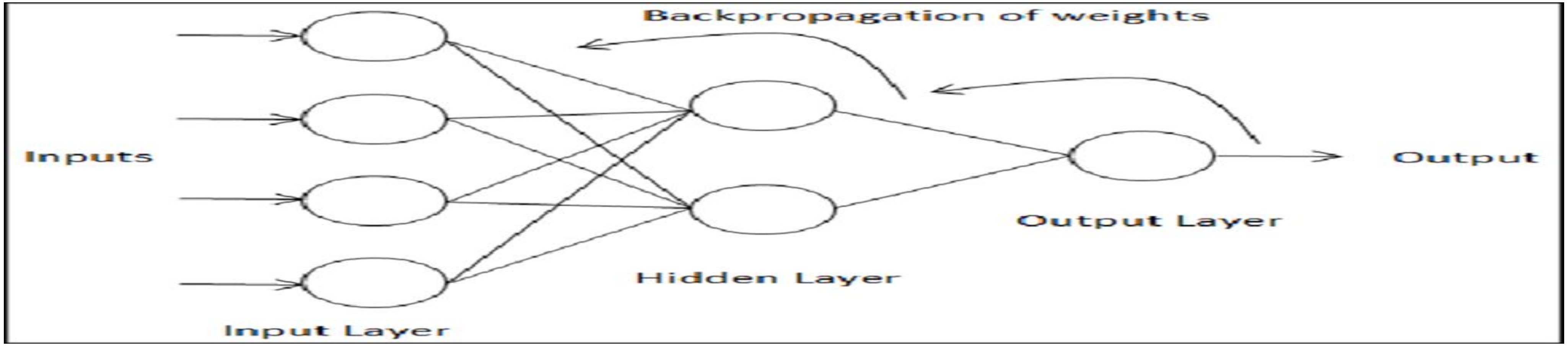
3.1 Feed Forward Calculations.



$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}\left(Z^{[L]}\right) = \hat{Y}
\end{aligned}
$$

3.2 Back-Propagation.

- Backpropagation is a technique used by deep layer networks to find the error of the network.
- The error is calculated by comparing an expected output with a predicted output, this algorithm then propagated these errors backward to update weights and biases.

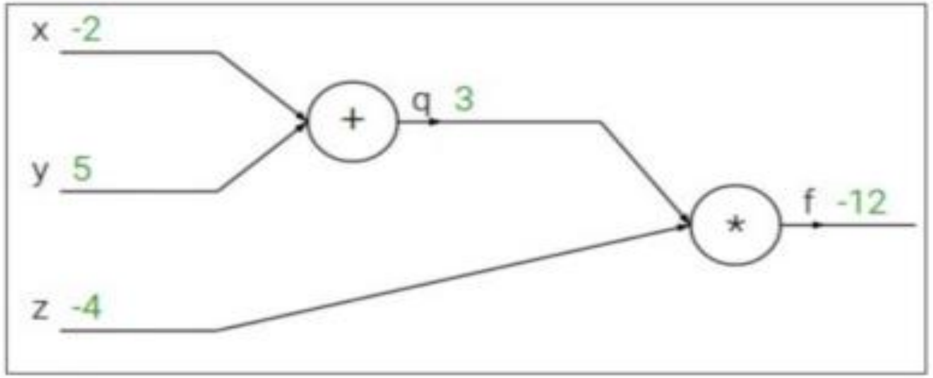


Backpropagation Intuition and Computational Graph.

Computational Graph -1

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Computational Graph -2

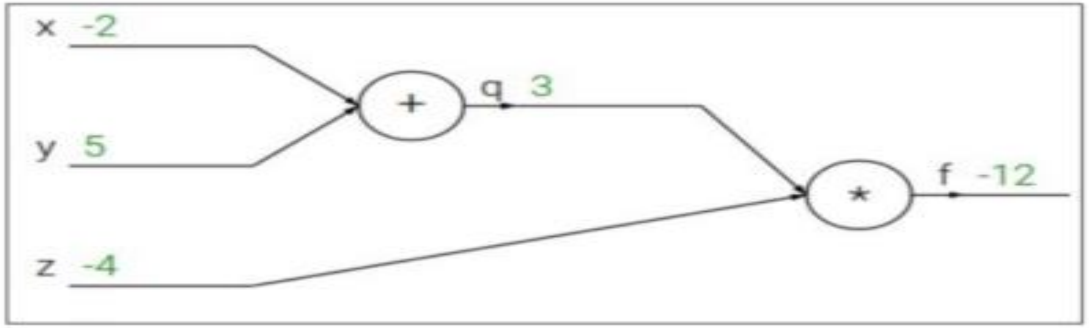
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -3

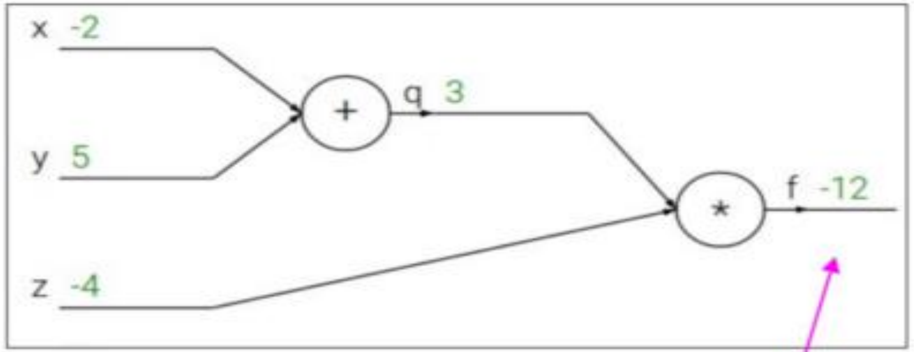
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial f}$$

Computational Graph -4

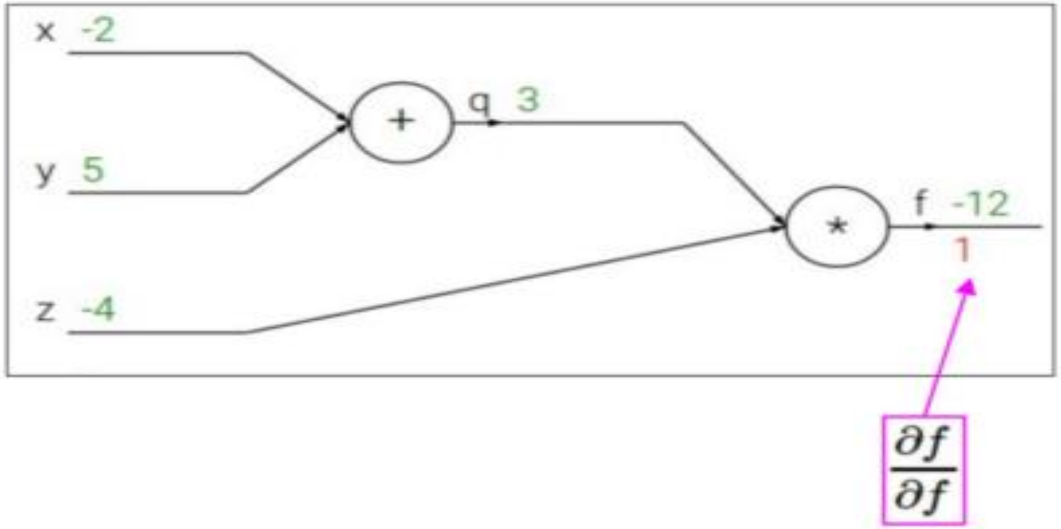
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -5

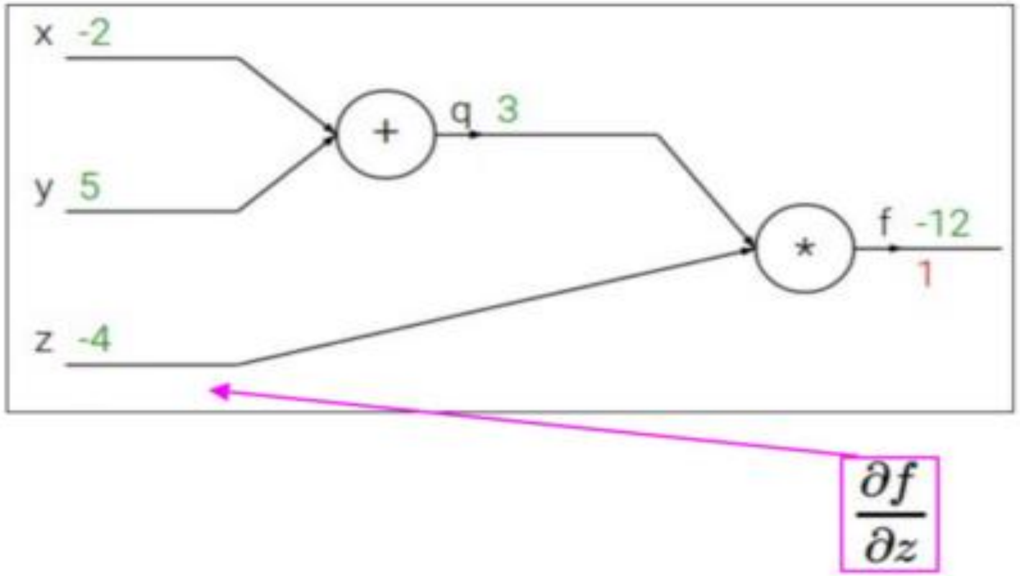
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -6

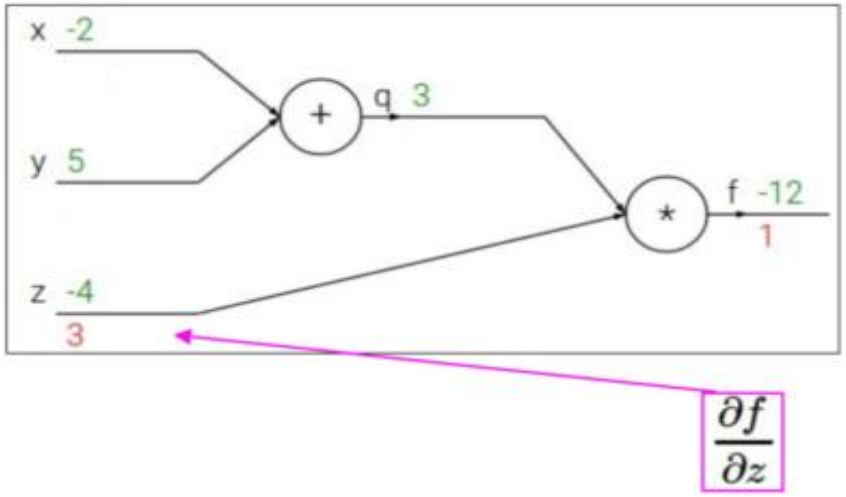
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -7

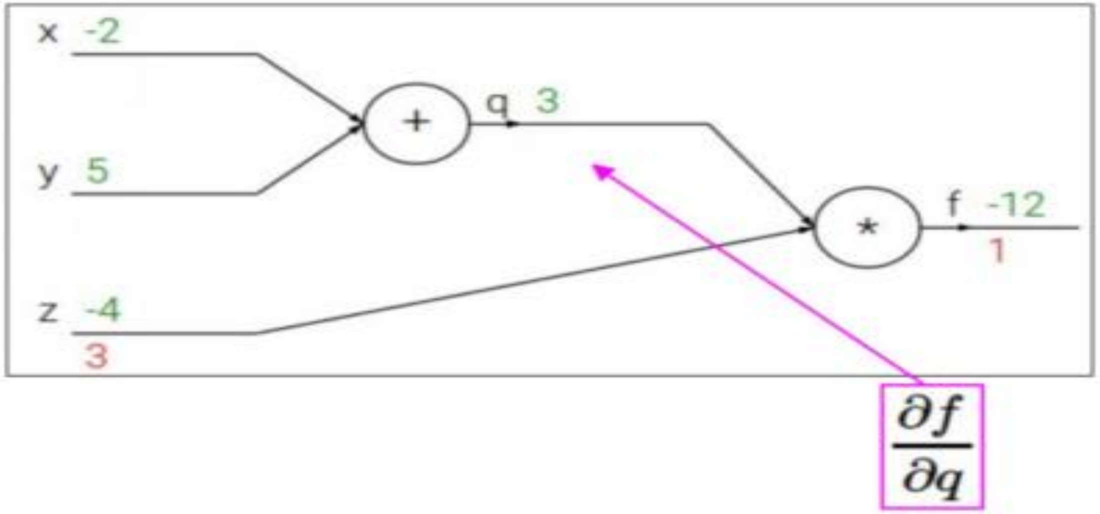
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -8

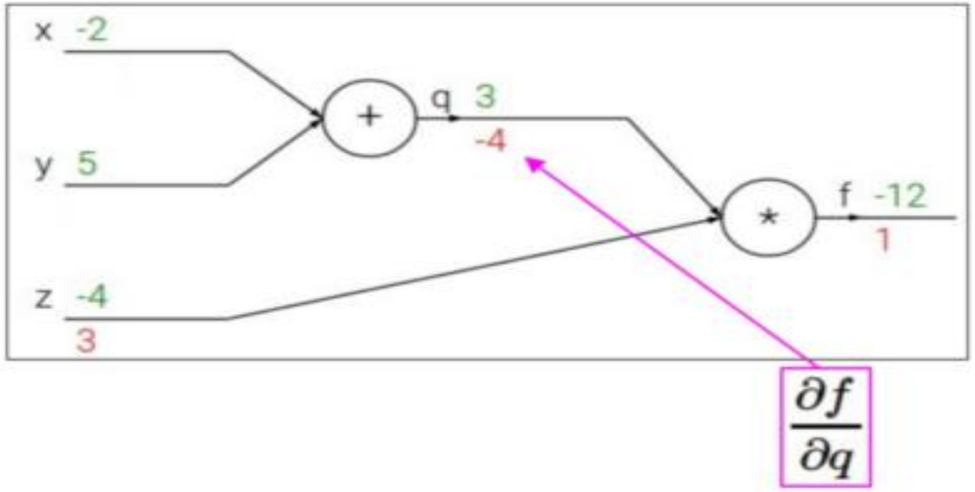
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -9

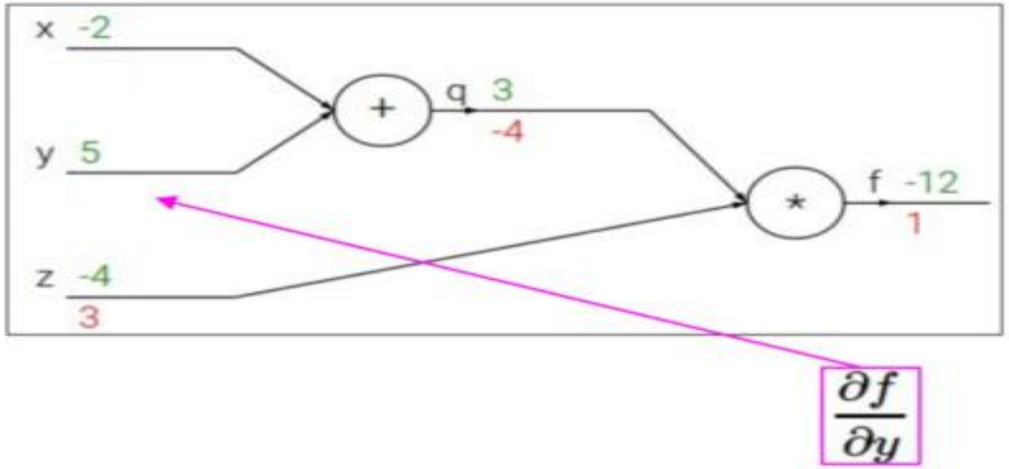
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -10

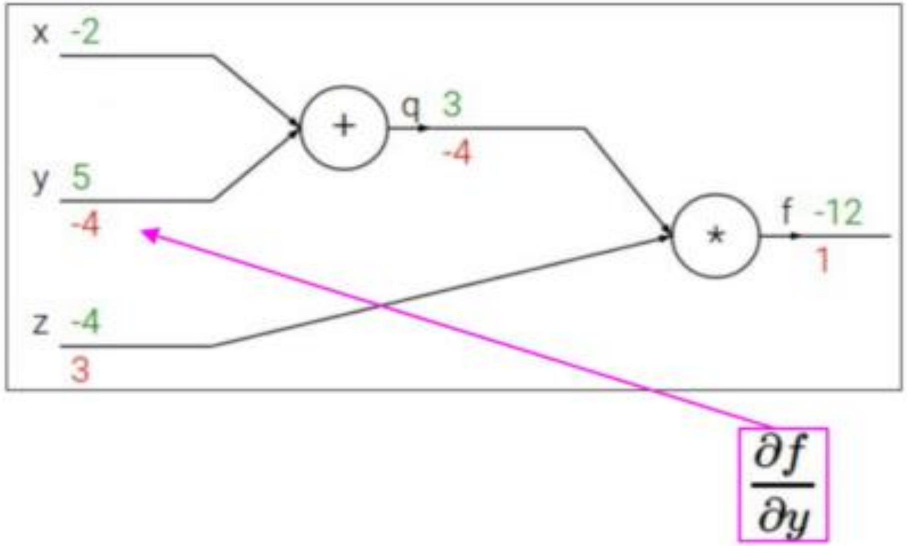
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Computational Graph -11

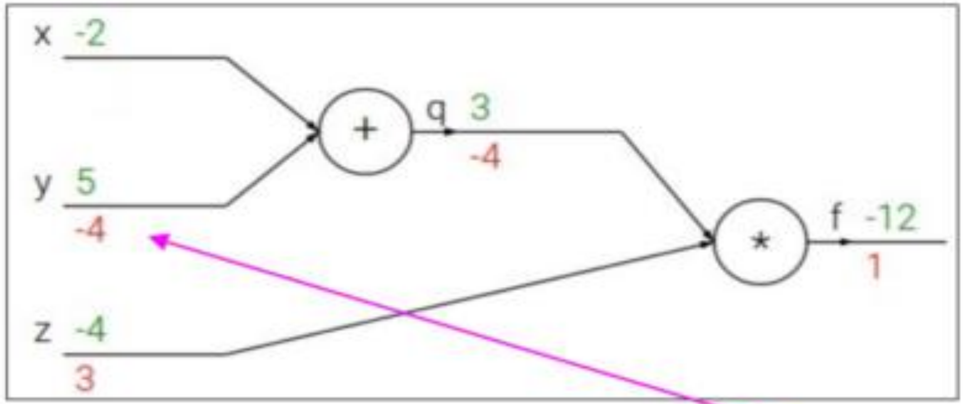
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial y}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}}$$

Computational Graph -12

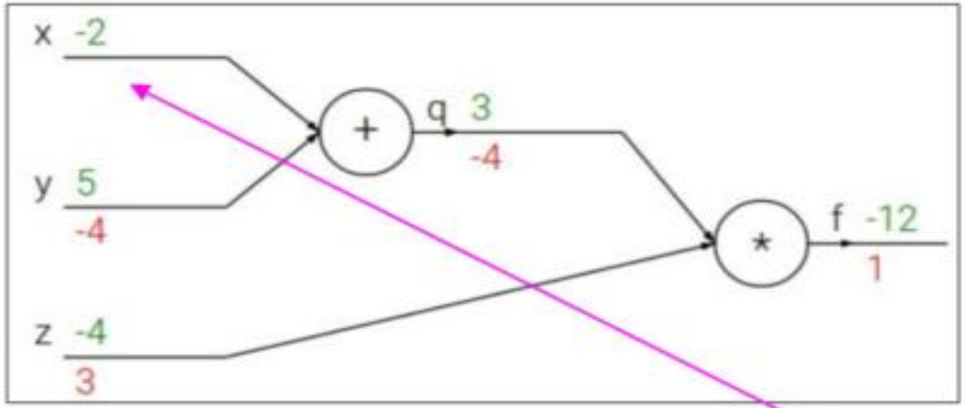
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\frac{\partial f}{\partial x}$

Computational Graph -13

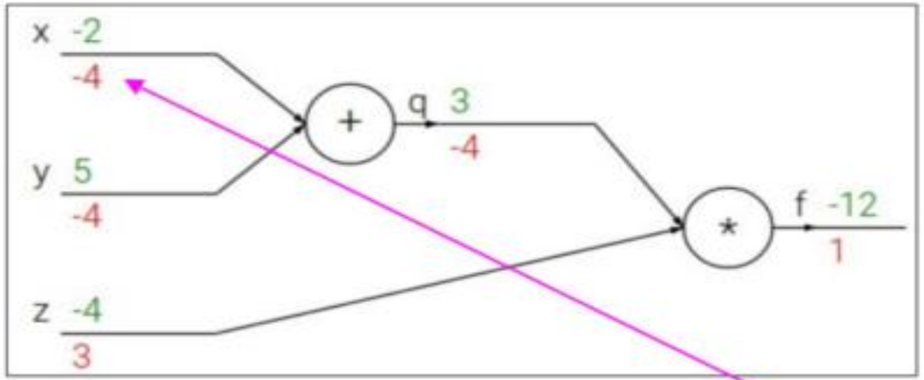
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



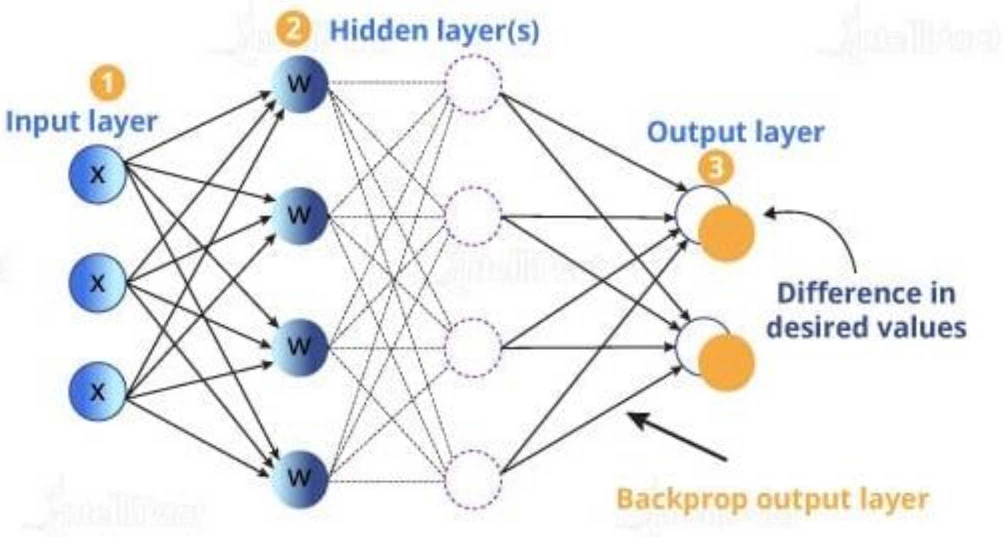
$$\boxed{\frac{\partial f}{\partial x}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}}$$

Backpropagation Requirements

- Backprop Requires following three things:
  1. Dataset: in the pairs of input-output i.e. (x_i, y_i).
  2. A feed-forward neural network;
  3. An Error Function, E which defines the error between the desired output and calculated output.



Backpropagation Algorithm.

Notations:
The subscript k denotes the output layer.
The subscript j denotes the hidden layer.
The subscript i denotes the input layer

Algorithm 2 Backpropagation algorithm for feedforward networks.

Require: a set of training examples D, learning rate α.

1. Create a feed-forward network with n_{in} inputs, n_h hidden units, and n_o output units.
2. Initialize all network weights to *small* random numbers.
3. **Repeat** until the termination condition is met:

   For each training example $(\mathbf{x}, \mathbf{t}) \in D$ **do:**

   Propagate the input $\mathbf{x}$ forward through the network, i.e.:

   1. Input $\mathbf{x}$ to the network and compute the output a_k of units in the output layer.

   Backpropagate the errors through the network:

   1. **For** each network output unit k, calculate its error term δ_k:

   $$\delta_k \leftarrow -a_k(1 - a_k)(t_k - a_k) \tag{3}$$

   2. **For** each hidden unit h, calculate its error term δ_h:

   $$\delta_h \leftarrow a_h(1 - a_h) \sum_k \delta_k w_{kh} \tag{4}$$

   3. Update each network weight w_{ji}

   $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

   where

   $$\Delta w_{ji} = -\alpha \delta_j x_{ji}$$

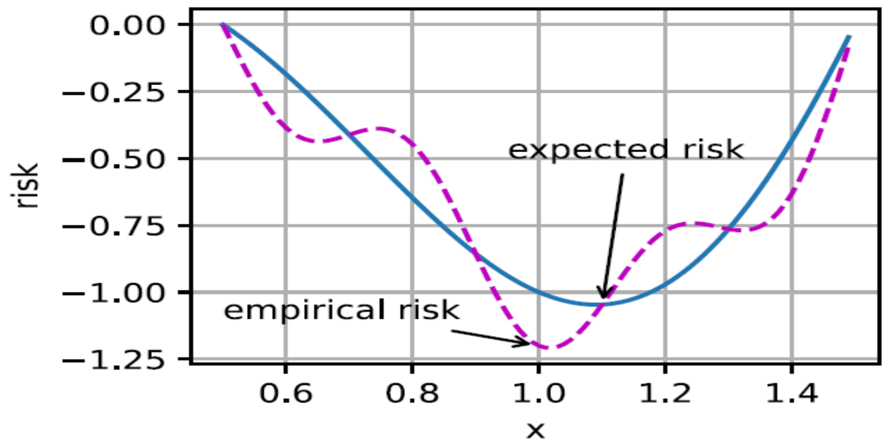
In Tutorial.

- Pen paper calculation on Forward and Backward Propagation.

4.Optimization in Deep Learning.

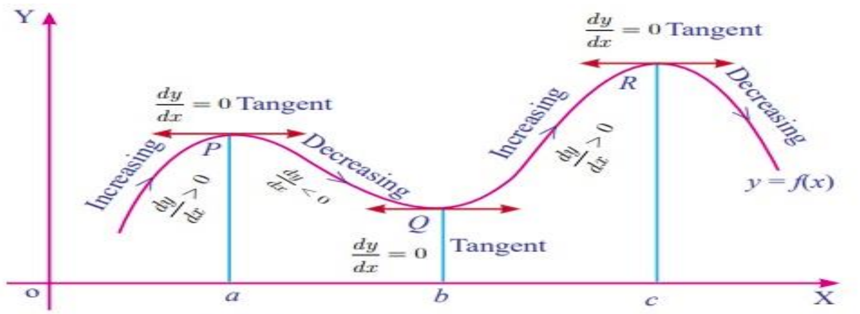
4.1 Optimization.

- Optimization and machine learning have related, but somewhat different goals
  - Goal in optimization: minimize an objective function
    - For a set of training examples, reduce the training error
  - Goal in ML: find a suitable model, to predict on data examples
    - For a set of testing examples, reduce the generalization error
- For a given empirical function *g* (dashed purple curve), optimization algorithms attempt to find the point of minimum empirical risk
- The expected function *f* (blue curve) is obtained given a limited amount of training data examples
- ML algorithms attempt to find the point of minimum expected risk, based on minimizing the error on a set of testing examples
  - Which may be at a different location than the minimum of the training examples
  - And which may not be minimal in a formal sense



4.2 Stationary points.

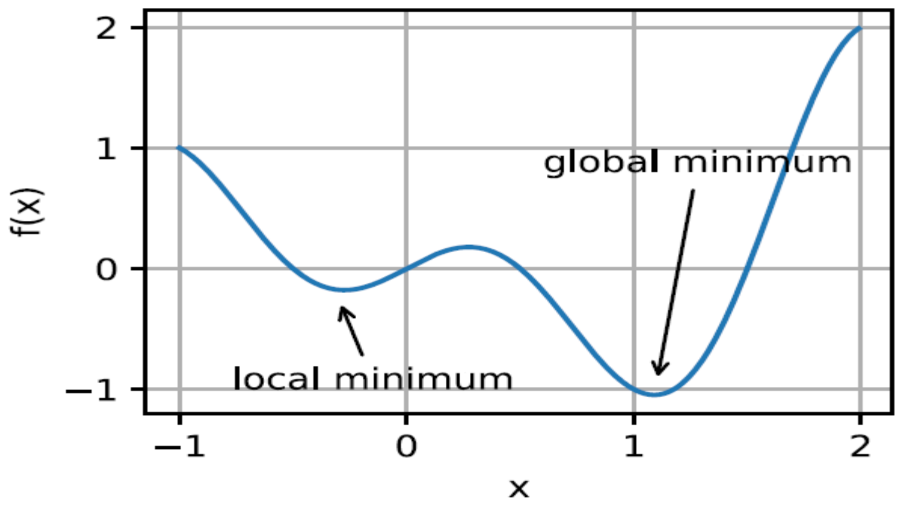
- ***Stationary points*** (or critical points) of a differentiable function $f(x)$ of one variable are the points where the derivative of the function is zero, i.e., $f'(x) = 0$
- The stationary points can be:
  - ***Minimum***, a point where the derivative changes from negative to positive
  - ***Maximum***, a point where the derivative changes from positive to negative
  - ***Saddle point***, derivative is either positive or negative on both sides of the point
- The minimum and maximum points are collectively known as extremum points
- The nature of stationary points can be determined based on the second derivative of $f(x)$ at the point
  - If $f''(x) > 0$, the point is a minimum
  - If $f''(x) < 0$, the point is a maximum
  - If $f''(x) = 0$, inconclusive, the point can be a saddle point, but it may not
- The same concept also applies to gradients of multivariate functions



P, Q, R are called Stationary Points

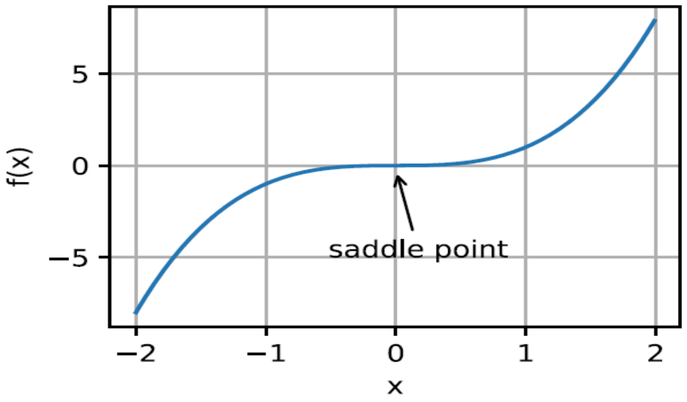
4.3 Local Minima.

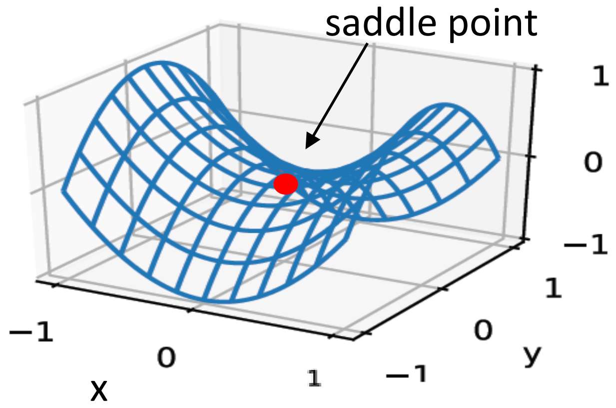
- Among the challenges in optimization of model's parameters in ML involve local minima, saddle points, vanishing gradients
- For an objective function $f(x)$, if the value at a point x is the minimum of the objective function over the entire domain of x, then it is the ***global minimum***
- If the value of $f(x)$ at x is smaller than the values of the objective function at any other points in the vicinity of x, then it is the ***local minimum***
  - The objective functions in ML usually have many local minima
    - When the solution of the optimization algorithm is near the local minimum, the gradient of the loss function approaches or becomes zero (vanishing gradients)
    - Therefore, the obtained solution in the final iteration can be a local minimum, rather than the global minimum



4.4 Saddle Points.

- The gradient of a function $f(x)$ at a saddle point is 0, but the point is not a minimum or maximum point
  - The optimization algorithms may stall at saddle points, without reaching a minima
- Note also that the point of a function at which the sign of the curvature changes is called an inflection point
  - An inflection point ($f''(x) = 0$) can also be a saddle point, but it does not have to be
- For the 2D function (right figure), the saddle point is at (0,0)
  - The point looks like a saddle, and gives the minimum with respect to *x*, and the maximum with respect to *y*





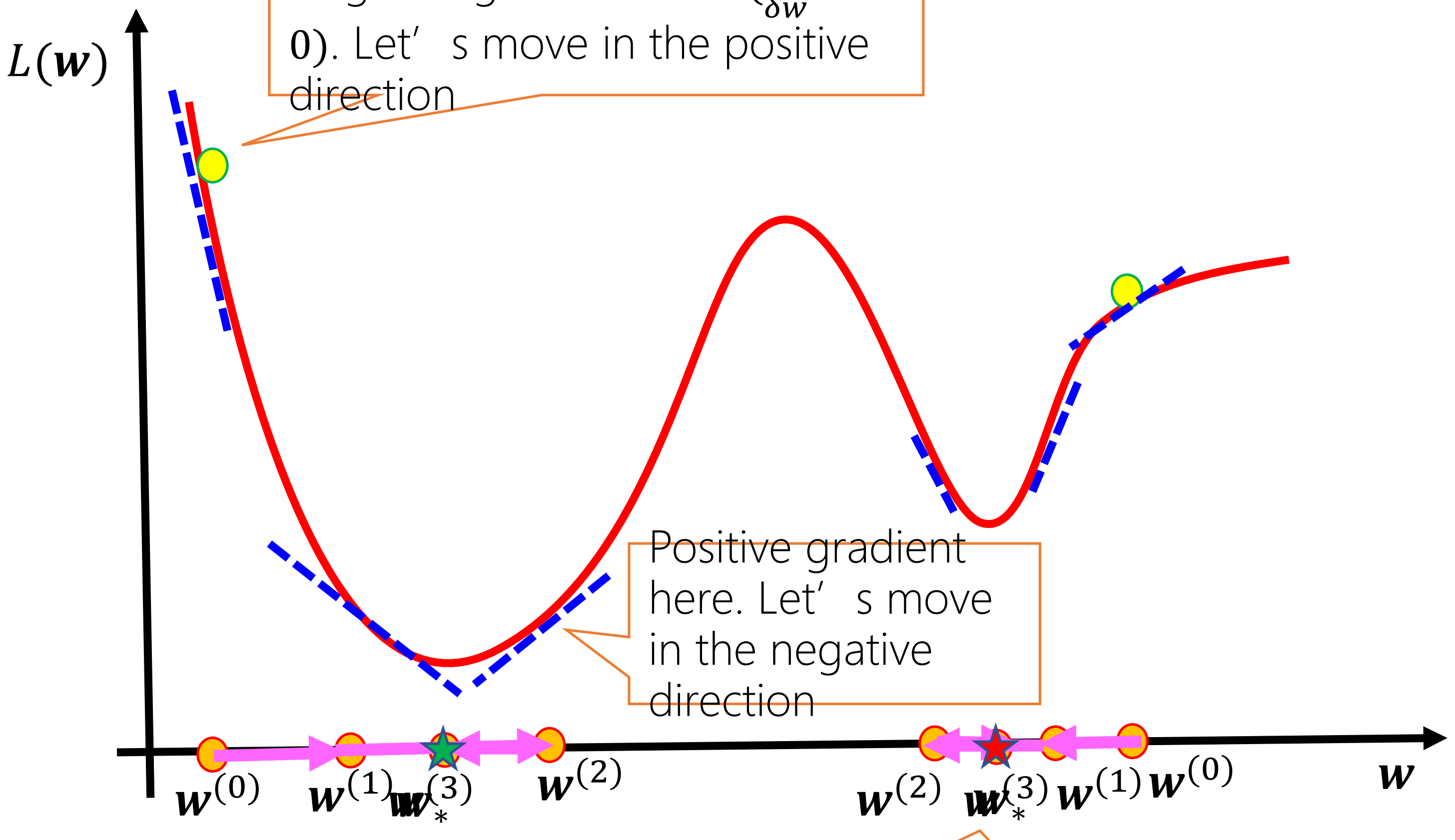
5. Optimization via Gradient Descent.

5.1 Pseudo-code Gradient Descent.

- Choose a starting point.(Initialization)
- Calculate the gradient at this point.
- Make a scaled step in the opposite direction to the gradient.
- Repeat until stopping criteria is met, which can be:
  - Maximum number of iterations reached.
  - Step size is smaller than tolerance.
- Input Parameters:
  - Starting point: 0 or random initialization.
  - Gradient function: Cost function to be minimized.
  - Learning Rate: scaling factor for step sizes.
  - Max. Iterations: number of iterations the algorithm must run.

5.2 Gradient Descent: An Illustration

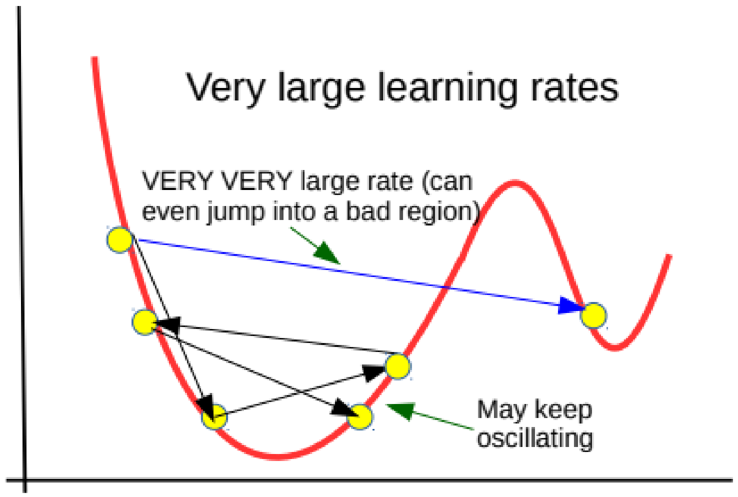
Negative gradient here ($\frac{\delta L}{\delta w} <$ 0). Let' s move in the positive direction

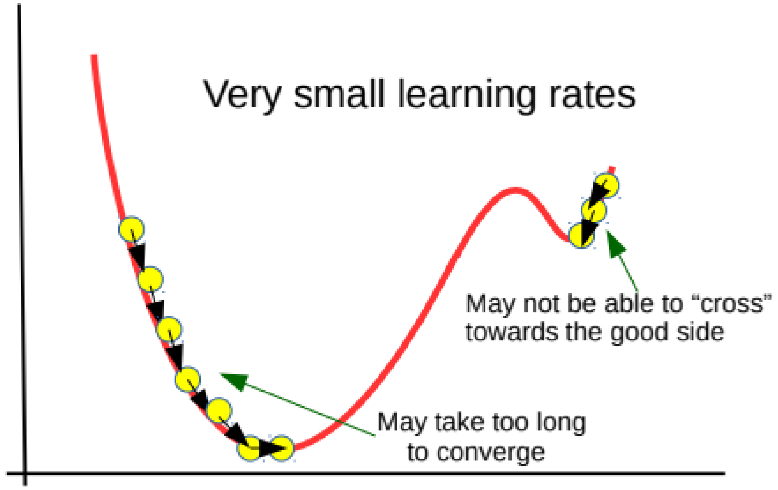


Positive gradient here. Let' s move in the negative direction

Stuck at a local minima

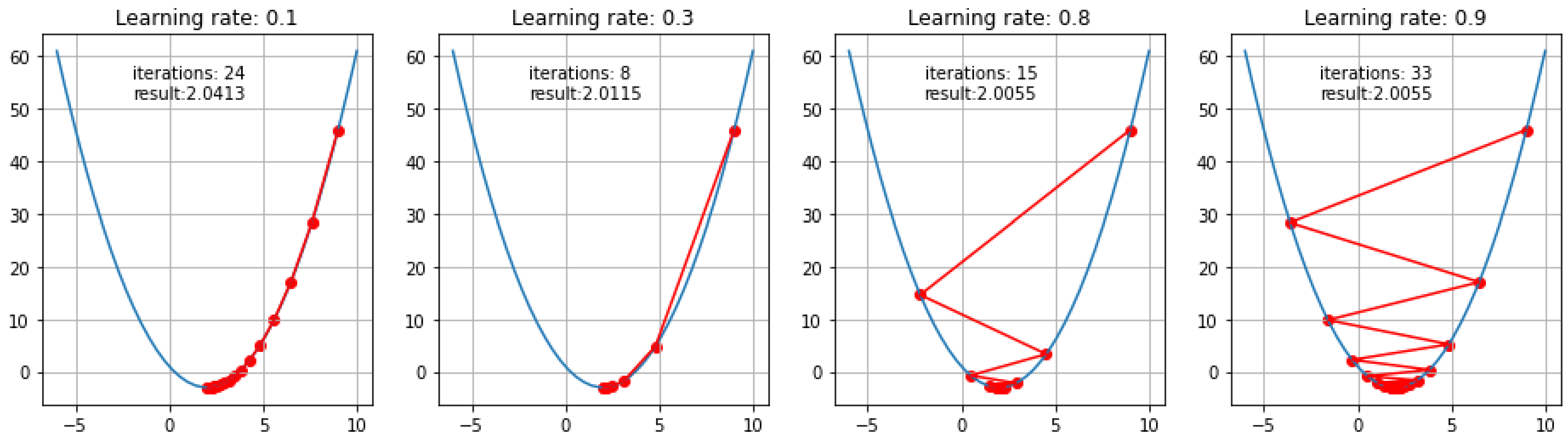
Good initialization is very important





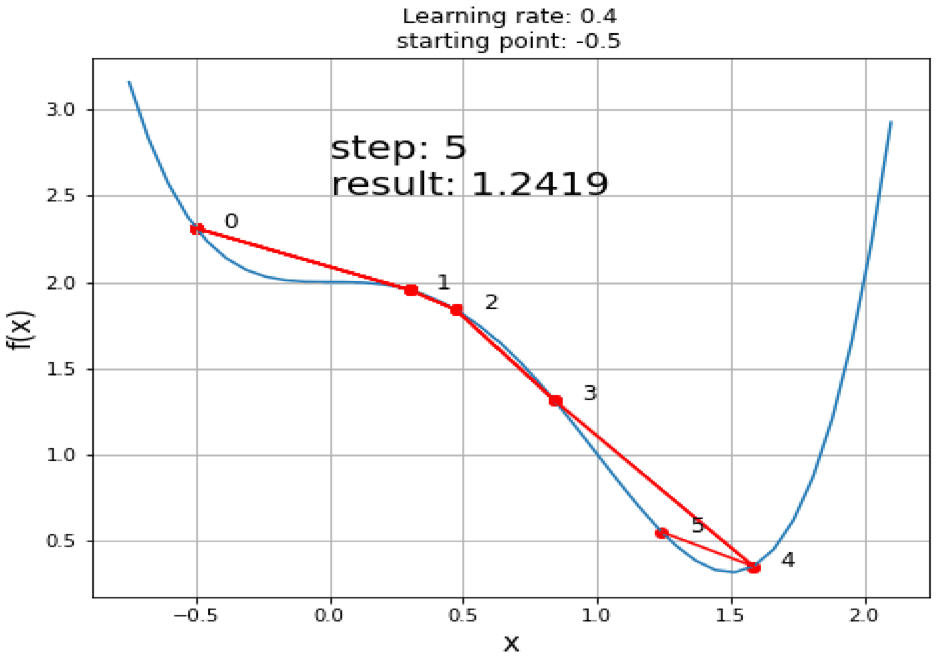
5.3 Learning Rate and Convergence.

- High Learning Rate does not Guarantees speedy Convergence.



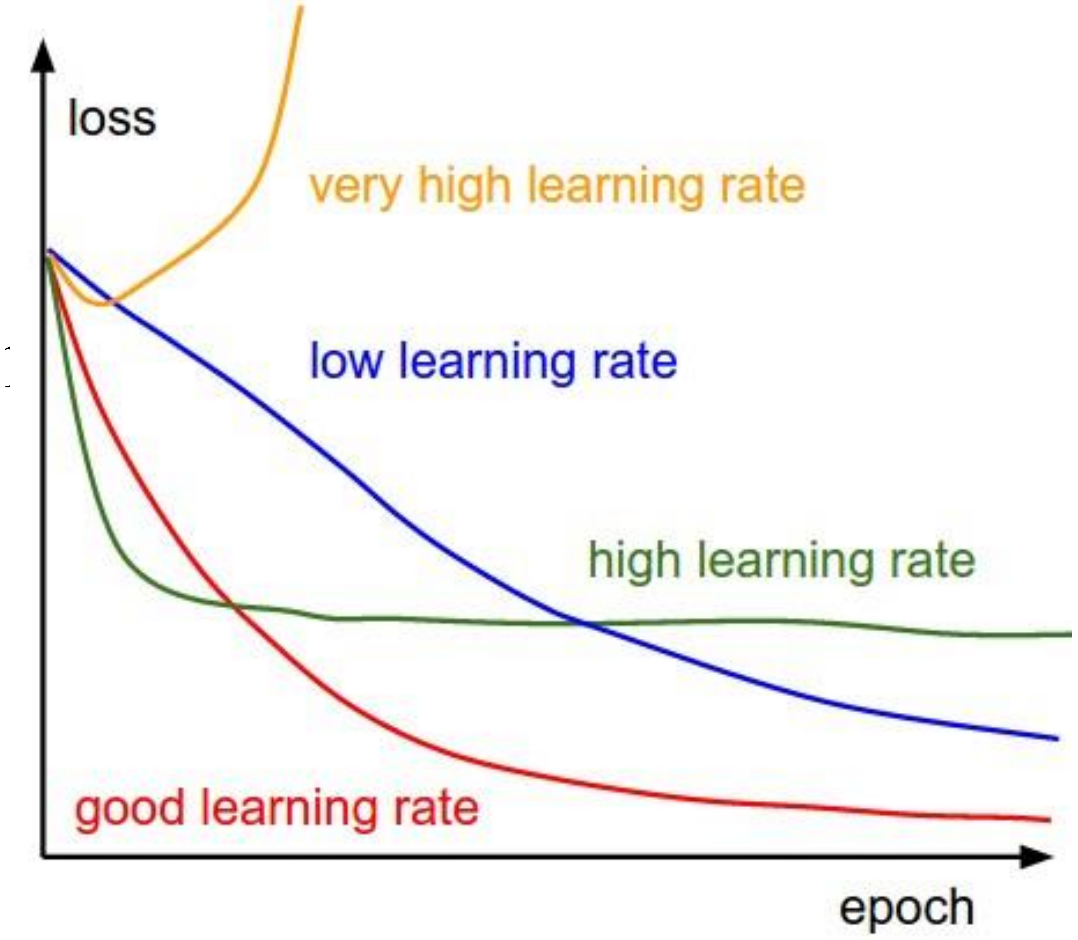
5.4 Starting point and Convergence.

- Convergence speed also depends on our starting point.



5.5 Gradient descent in practice: Learning rate

- Automatic convergence test
- α too small: slow convergence
- α too large: may not converge
- To choose α, try 0.001, … 0.01, …, 0.1, … ,



6. Variants of Gradient Decent.

6.1 Gradient descent variants.

- Based on how much data we use to compute the gradient of the objective function, there are three main variants of gradient descent:
  - Batch Gradient Descent.
  - Stochastic Gradient Descent.
  - Mini-Batch Gradient Descent.
- More data may mean more accuracy of the parameter update, but it also means more time needed to reach convergence, thus the **trade off** is necessity.

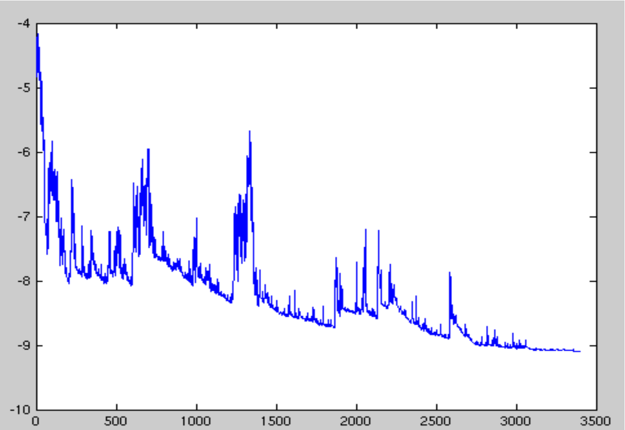
6.2 Batch Gradient Descents.

- Also known as vanilla gradient descent, computes the gradient of the cost function w.r.t to the parameters θ for the entire training set:
  - **Update rule:**
    - $\boldsymbol{\theta = \theta - \alpha \nabla_{\theta} J(\theta)}$.
- It can be very slow and is intractable for datasets that do not fit the memory.

6.3 Stochastic Gradient Descent.

- In contrast SGD performs a parameter update for each training example i.e. x^i and label y^i.
  - **Update Rule:**
  - $\boldsymbol{\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^i; y^i)}$.
- It is usually much faster compared to BGD, and also can be used to learn online.
- SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in image:
- It can enable it to jump to new and potential better local minima.
- It can also ultimately complicates the convergence to the exact

minimum.



6.4 Mini Batch Gradient Descent.

- It is the mixture of BGD and SGD i.e. it updates the parameter for every mini batch of n training examples:
  - **Update Rule:**
  - $\boldsymbol{\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}}$.
- Merits:
  - Reduces the variance of the parameter updates, which can lead to more stable convergence;
  - Efficient computation for large models.
- Common mini-batch size range between 50 and 256.

6.5 Challenges

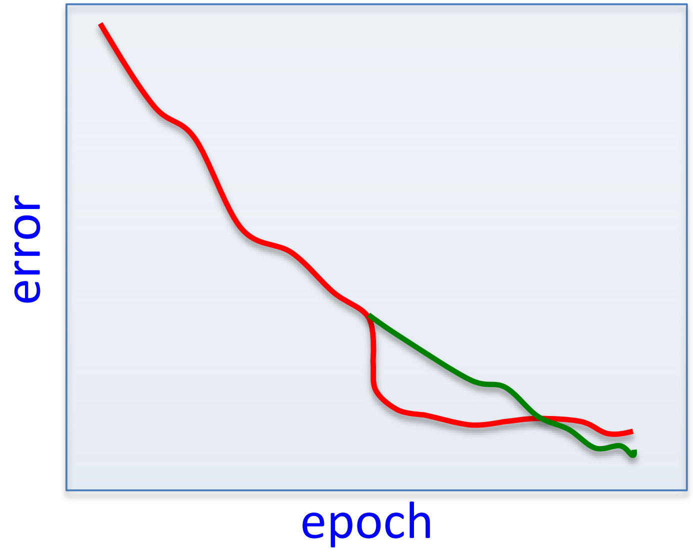
- Same learning rate applies to all parameter updates.
- Getting trapped into suboptimal local minima or saddle points.
- Choosing a Proper Learning Rate.
  - Annealing:
    - Learning rate schedules or reducing the learning rate according to a pre-defined schedule or it becomes smaller then a pre-set threshold.
    - Schedules and threshold values have to be pre-defined, thus may not be able to adapt to a dataset’s characteristics.

6.6 Basic Mini-Batch Gradient Descent.

- Guess an **initial learning rate**.
  - If the **error** keeps getting worse or **oscillates** wildly, **reduce** the **learning rate**.
  - If the **error** is falling **fairly** consistently but slowly, increase the **learning rate**.
- Write a simple program to automate this way of **adjusting** the **learning rate**.
- Towards the end of mini-batch learning it nearly always helps to **turn down** the **learning rate**.
  - This removes **fluctuations** in the **final weights** caused by the **variations between mini-batches**.
- Turn down the **learning rate** when the **error stops decreasing**.
  - Use the error on a separate validation set

6.7 Turning Down the learning rate.

- Turning down the learning rate reduces the random fluctuations in the error due to the different gradients on different mini-batches.
  - So we get a quick win.
  - But then we get slower learning.
- Don’t turn down the learning rate too soon!



Today's Lesson…

- Activations Function.
- Error Function in Classification with Neural Network.
- Learn the weights:
- Forward Pass.
- Backward Pass.
- Optimization-Gradient Descent and its Variants.

Next Up!!!!

- You are given a training set with 1M labeled points. When you train a shallow neural net with one fully-connected feed-forward hidden layer on this data you obtain 86% accuracy on test data. When you train a deeper neural net as in which consist of a convolutional layer, pooling layer, and three fully-connected feed-forward layers on the same data you obtain 91% accuracy on the same test set.
- What is the source of this improvement?

At the end…….

- Questions?

- **Reminder!!!!**
  - **Assignment-I is out.**
  - **Date to submit-26 March.**