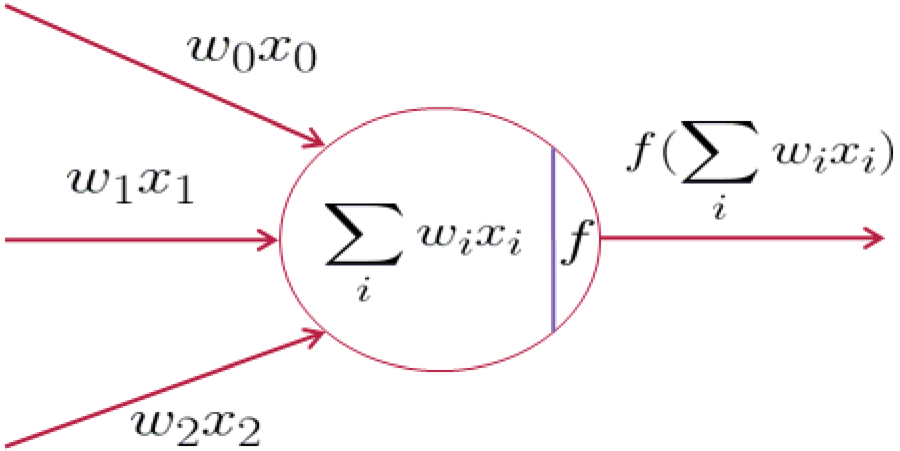
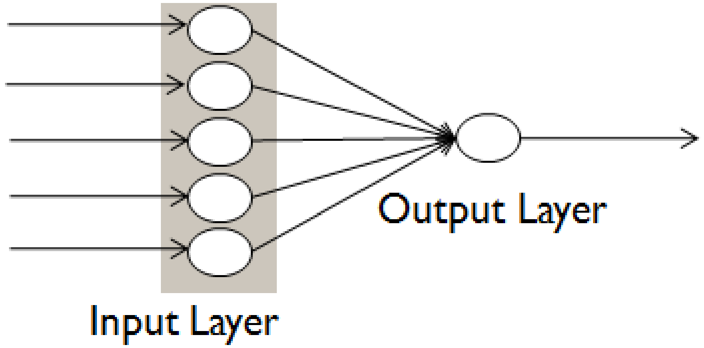
Backpropagation and Training

C. V. Jawahar

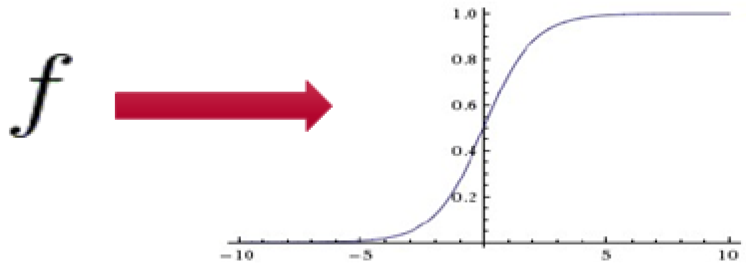
IIIT Hyderabad

Neuron, Perceptron and MLP

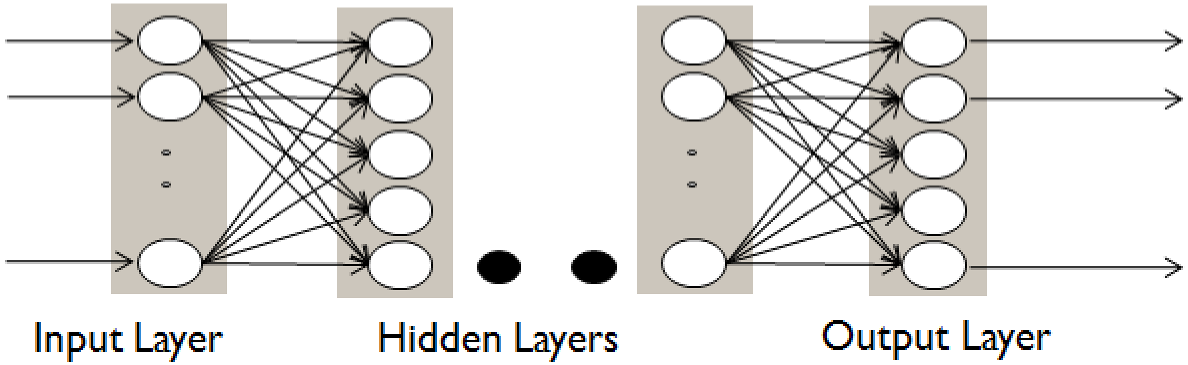




Perceptron

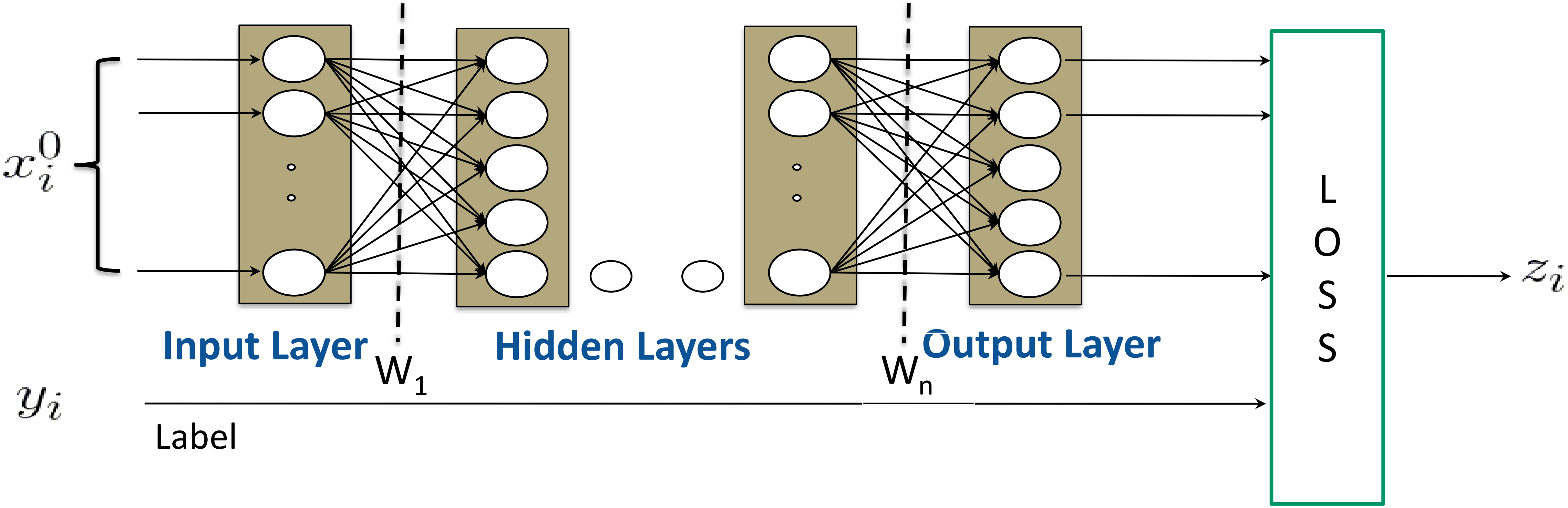


E.g. Sigmoid Activation Function



Multi Layer Perceptron

Loss or Objective



Objective: Find out the best parameters which will minimize the loss.

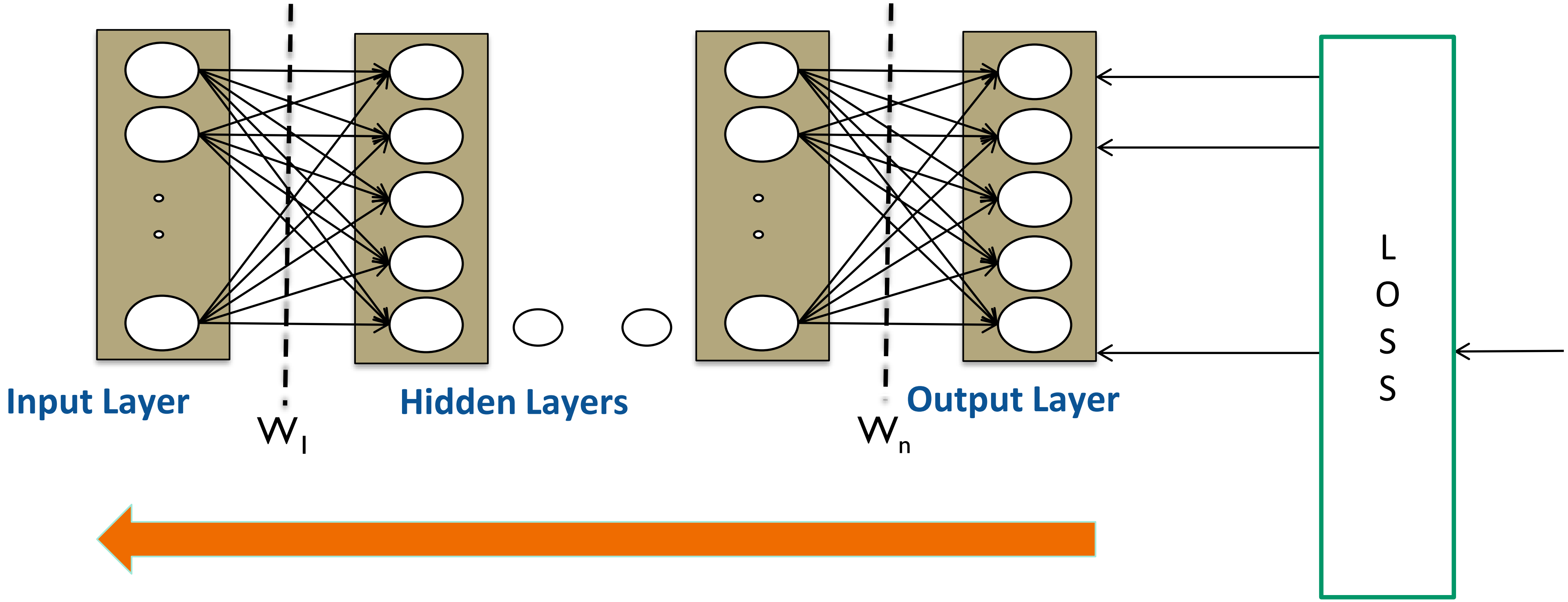
$$W^* = arg \min_W \sum_{i=1}^{N} L(x_i^n, y_i; W)$$

Weight vector

$$z_i = \frac{1}{2} \parallel x_i^n - y_i \parallel_2^2$$

E.g. Squared Loss

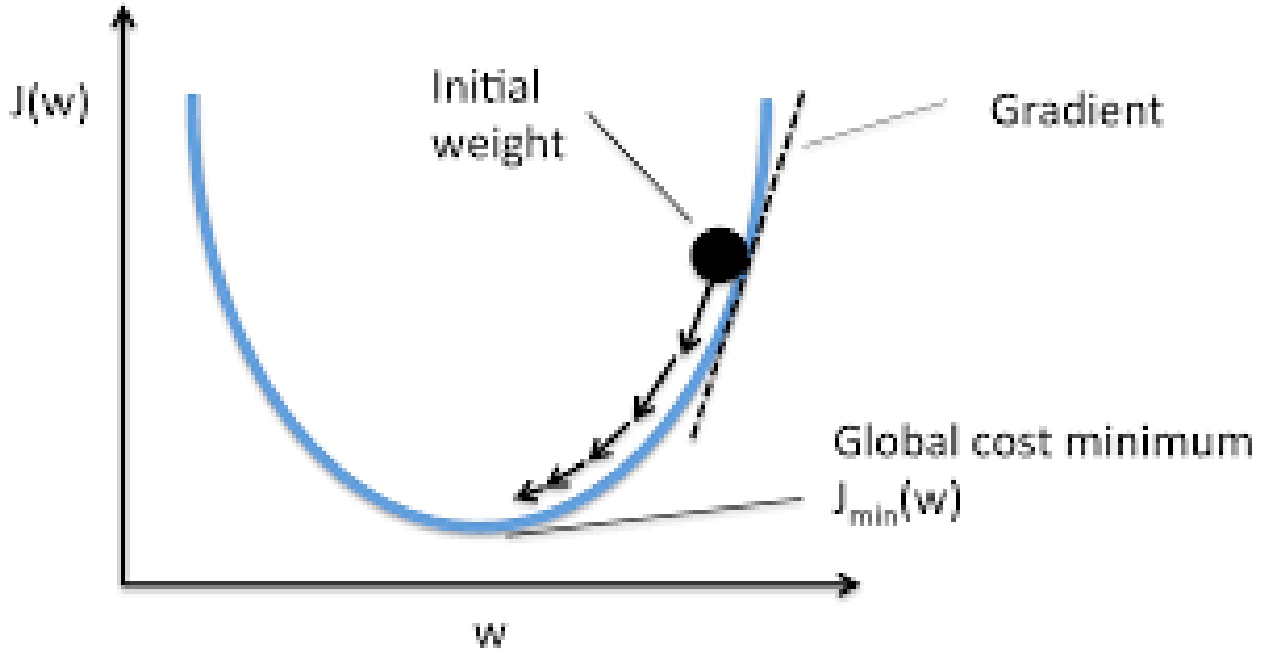
Back Propagation



Solution: Iteratively update W along the direction where loss decreases.

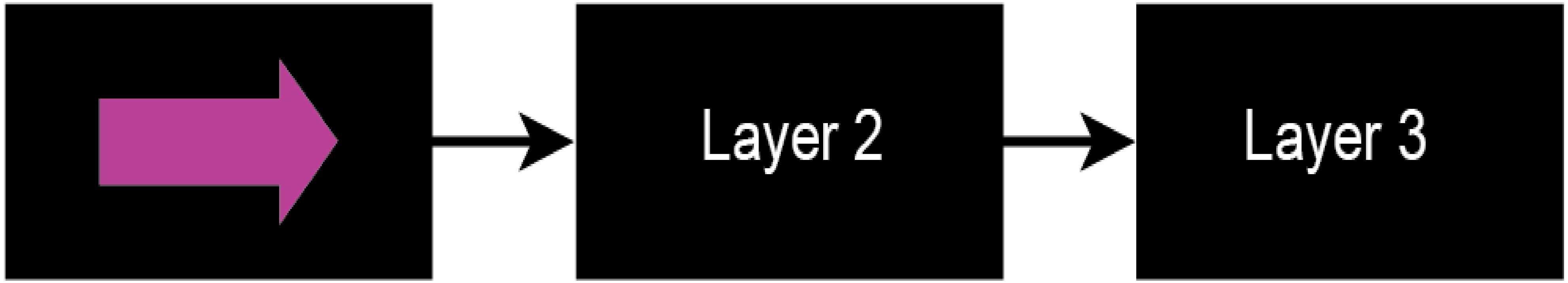
Each layer's weights are updated based on the derivative of its output w.r.t. input and weights

Gradient Descent



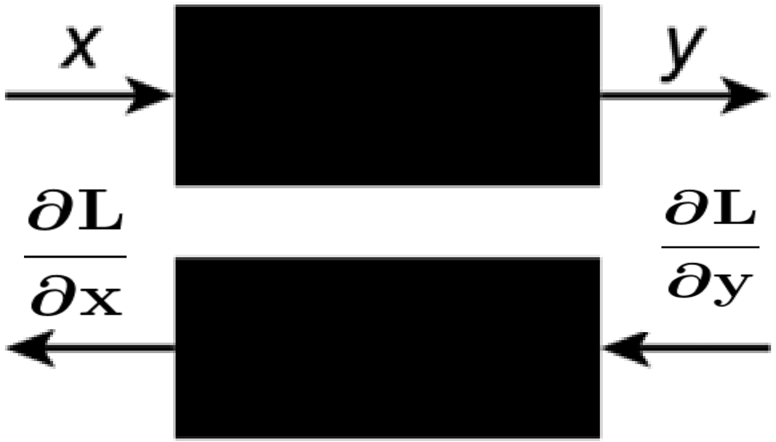
Neural Network Training

- Step 1: Compute loss for samples (on mini-batch) [F-Pass]
- Step 2: Compute gradients w.r.t parameters and update [B-Pass]



$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{L}}{\partial \mathbf{W}}$$

Chain Rule



Given y(x) and $\frac{\partial \mathbf{L}}{\partial \mathbf{y}}$

What is $\frac{\partial \mathbf{L}}{\partial \mathbf{x}}$ *?*

$$\frac{\partial \mathbf{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

Chain Rule

Given $\frac{\partial y}{\partial x}$ **and** $\frac{\partial L}{\partial y}$

What is $\frac{\partial L}{\partial x}$ **?**

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

For each block/parameters, we only need to find $\frac{\partial y}{\partial x}$

Key Computation: Forward-Propagation

W is the parameter, say the weights within the box.

Key Computation: Backward-Propagation

$$\frac{\partial L}{\partial x}$$

$$\left\{\frac{\partial Z}{\partial x}, \frac{\partial Z}{\partial W}\right\}$$

$$\frac{\partial L}{\partial Z}$$

$$\frac{\partial L}{\partial W}$$

$$\frac{\partial \mathrm{L}}{\partial \mathrm{x}} = \frac{\partial \mathrm{L}}{\partial \mathrm{Z}} \frac{\partial \mathrm{Z}}{\partial \mathrm{x}}$$

$$\frac{\partial \mathrm{L}}{\partial \mathrm{W}} = \frac{\partial \mathrm{L}}{\partial \mathrm{Z}} \frac{\partial \mathrm{Z}}{\partial \mathrm{W}}$$

$$\mathrm{W} \leftarrow \mathrm{W} - \eta \frac{\partial \mathrm{L}}{\partial \mathrm{W}}$$

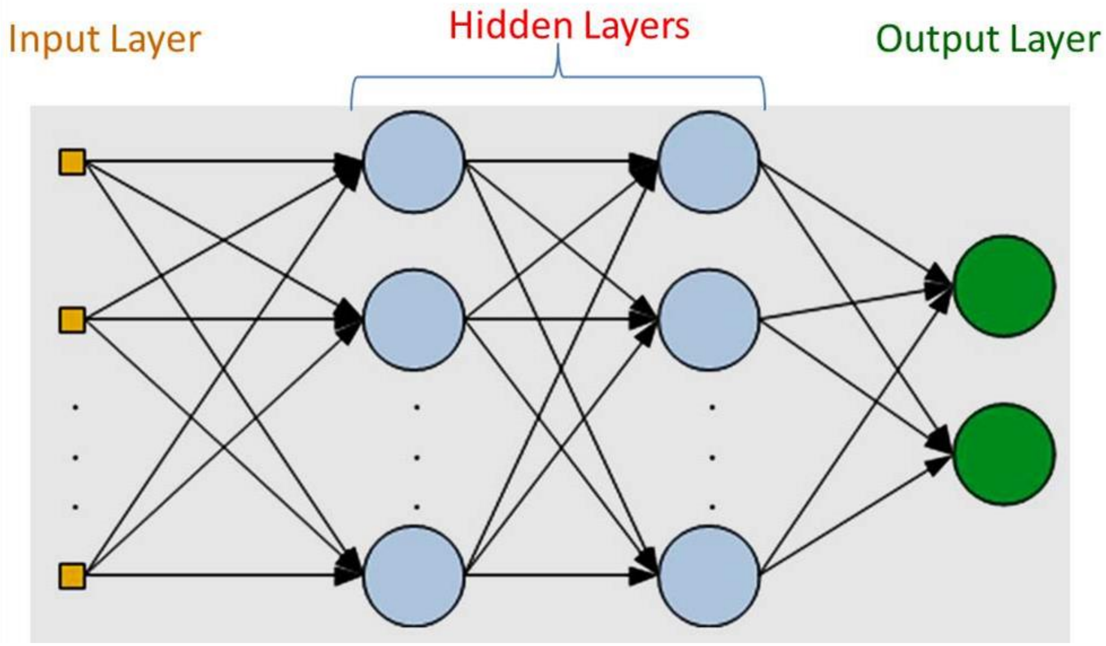
Back Propagation for MLP

- Two computational blocks/steps

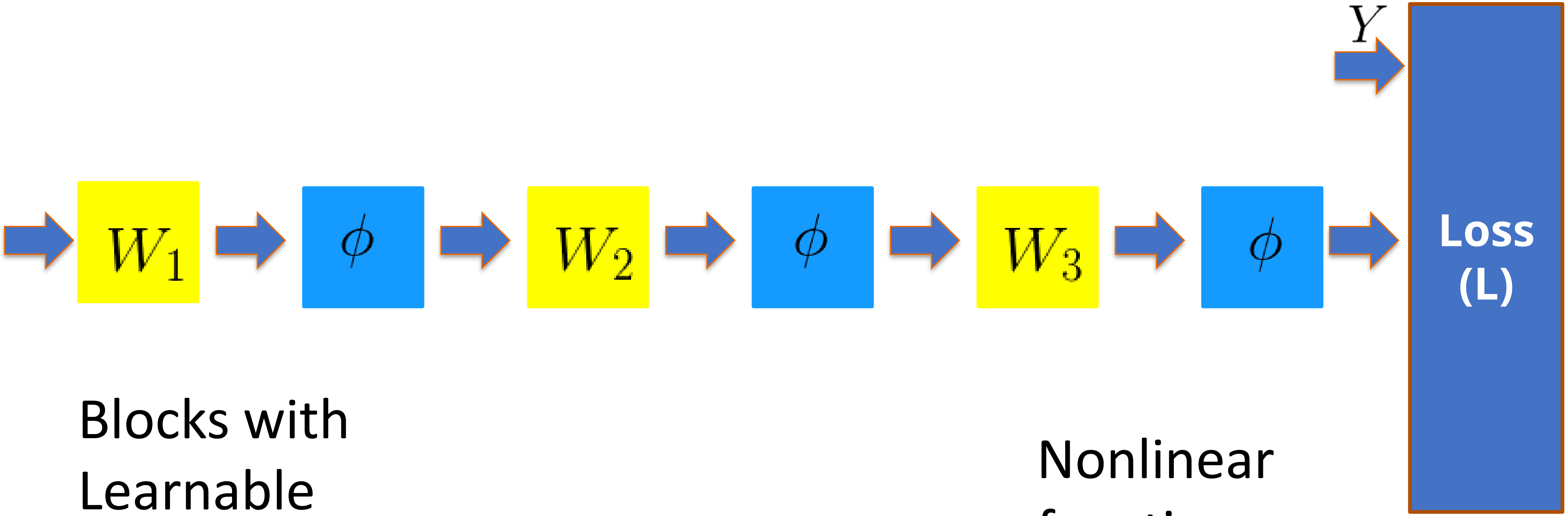
$$y = Wx$$

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

- In either case we can compute easily. $\frac{\partial y}{\partial x}$



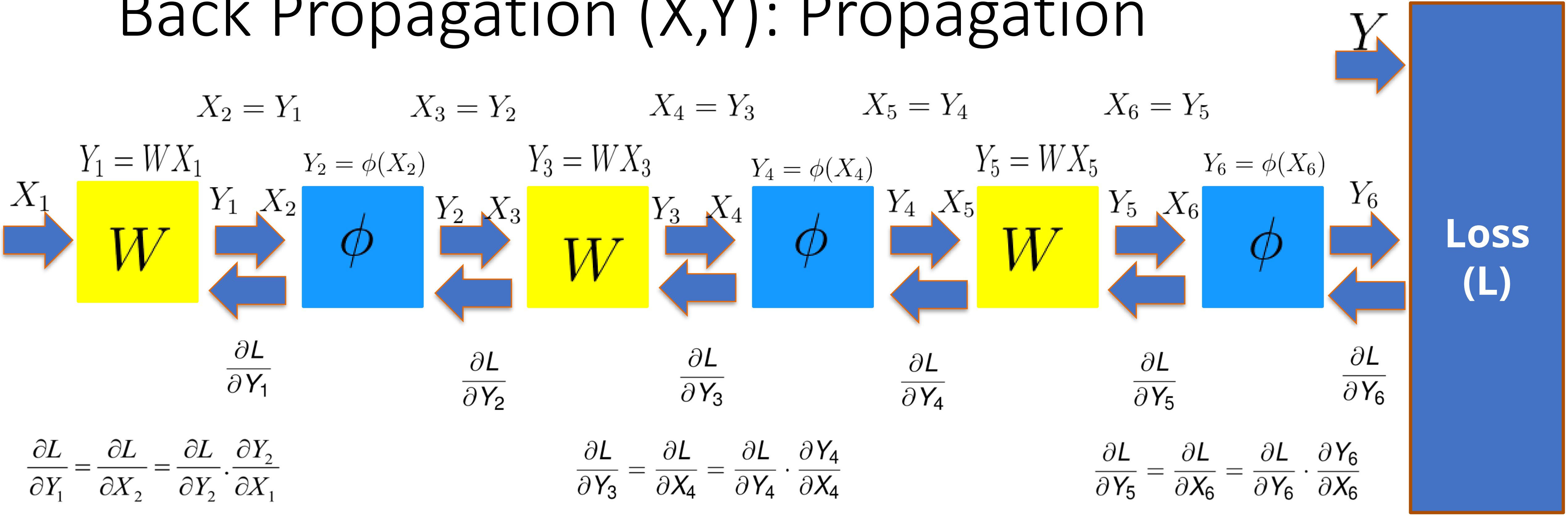
A simpler view point



Blocks with Learnable parameters Matrix Multiplication

Nonlinear functions (often non learnable)

Back Propagation (X,Y): Propagation



$$\frac{\partial L}{\partial Y_1} = \frac{\partial L}{\partial X_2} = \frac{\partial L}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial X_1}$$

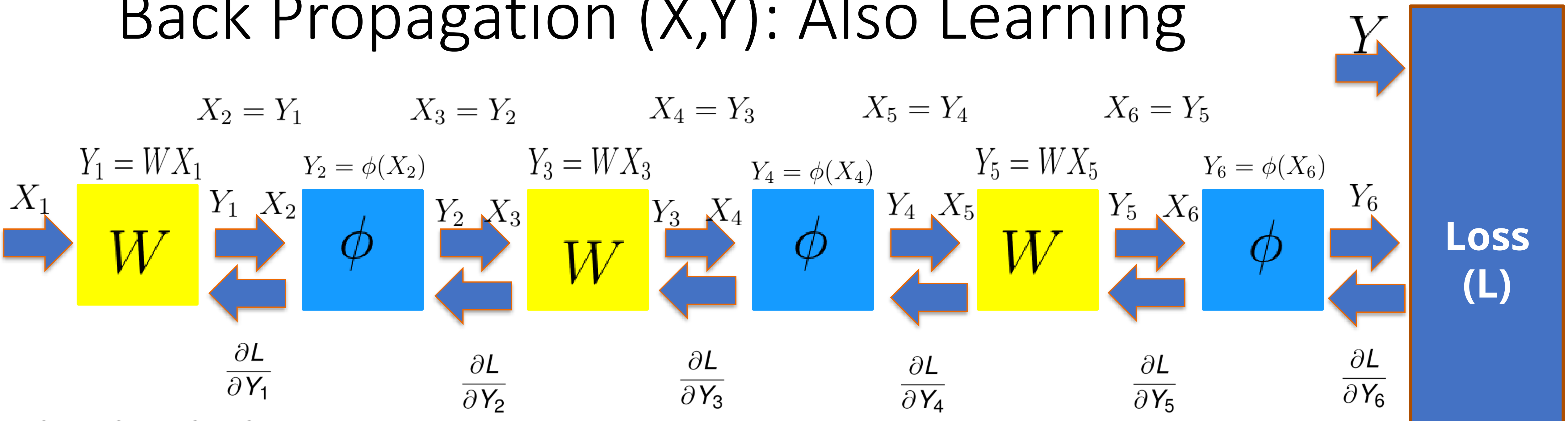
$$\frac{\partial L}{\partial Y_3} = \frac{\partial L}{\partial X_4} = \frac{\partial L}{\partial Y_4} \cdot \frac{\partial Y_4}{\partial X_4}$$

$$\frac{\partial L}{\partial Y_5} = \frac{\partial L}{\partial X_6} = \frac{\partial L}{\partial Y_6} \cdot \frac{\partial Y_6}{\partial X_6}$$

$$\frac{\partial L}{\partial Y_4} = \frac{\partial L}{\partial X_5} = \frac{\partial L}{\partial Y_5} \cdot \frac{\partial Y_5}{\partial X_5}$$

$$\frac{\partial L}{\partial Y_2} = \frac{\partial L}{\partial X_3} = \frac{\partial L}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial X_3}$$

Back Propagation (X,Y): Also Learning



$$\frac{\partial L}{\partial Y_1} = \frac{\partial L}{\partial X_2} = \frac{\partial L}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial X_2}$$

$$\frac{\partial L}{\partial Y_3} = \frac{\partial L}{\partial X_4} = \frac{\partial L}{\partial Y_4} \cdot \frac{\partial Y_4}{\partial X_4}$$

$$\frac{\partial L}{\partial Y_5} = \frac{\partial L}{\partial X_6} = \frac{\partial L}{\partial Y_6} \cdot \frac{\partial Y_6}{\partial X_6}$$

$$\frac{\partial L}{\partial Y_2} = \frac{\partial L}{\partial X_3} = \frac{\partial L}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial X_3}$$

$$\frac{\partial L}{\partial Y_4} = \frac{\partial L}{\partial X_5} = \frac{\partial L}{\partial Y_5} \cdot \frac{\partial Y_5}{\partial X_5}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y_1} \cdot \frac{\partial Y_1}{\partial W}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial W}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y_5} \cdot \frac{\partial Y_5}{\partial W}$$

$$W^{n+1} = W^n - \eta \frac{dL}{dW}$$

Summary

- Step 0:
  - Initialize the Network, weights
- Step 1:
  - Do forward pass for a batch of randomly selected samples.
  - Predict outputs with the existing weights.
- Step 2:
  - Compute Loss for the set of samples.

Summary

- Step 3:
  - Update all the weights using gradient descent.

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{L}}{\partial \mathbf{W}}$$

- Step 4:
  - Repeat all steps till the Loss is less than a threshold.

Loss Functions

- **For Classification**
  - **Binary Cross Entropy**
  - **Hinge Loss**
- **For Regression**
  - **Mean Square Error / Quadratic Loss / L2 Loss**
  - **Mean Absolute Error / L1 Loss**

$$L = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right)$$

$$L = \max\left(0, 1 - y * f(x)\right)$$

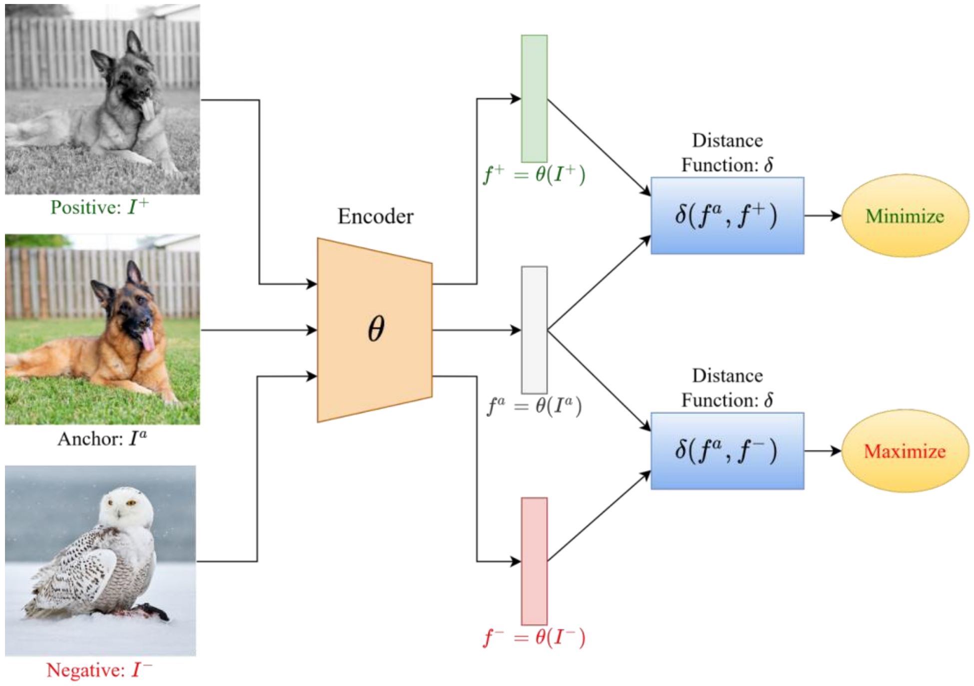
$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

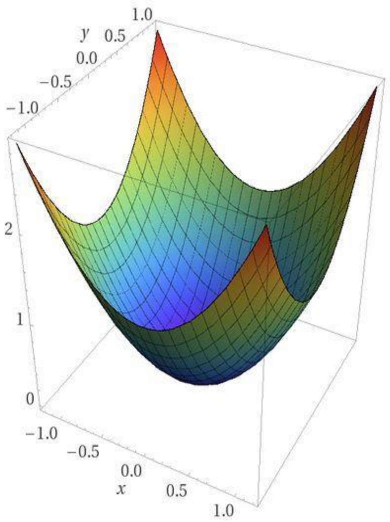
$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

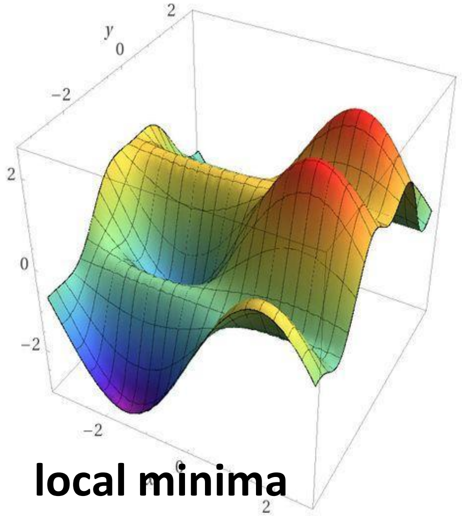
Loss Functions

- For Comparing Structured Data
  - Strings
  - Graphs
  - Trees
- For Comparing Multiple Samples
  - Contrastive Learning
  - Siamese Learning
  - Triplet Loss



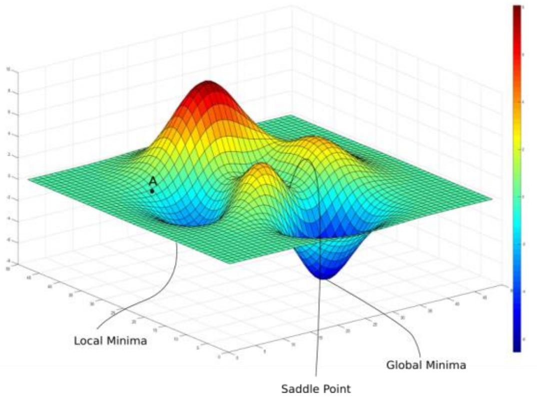
Why Challenging?

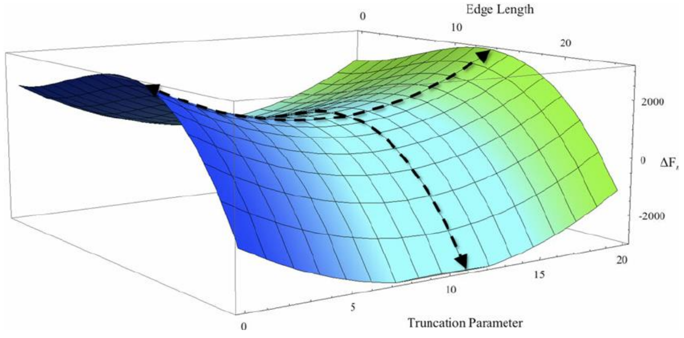




local minima

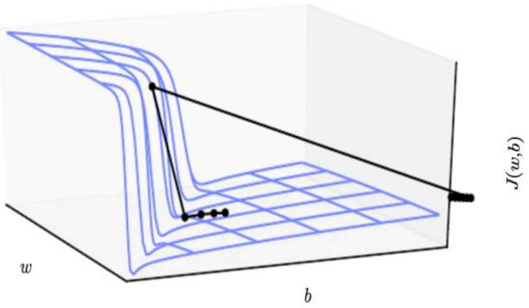
$$\nabla J = 0$$

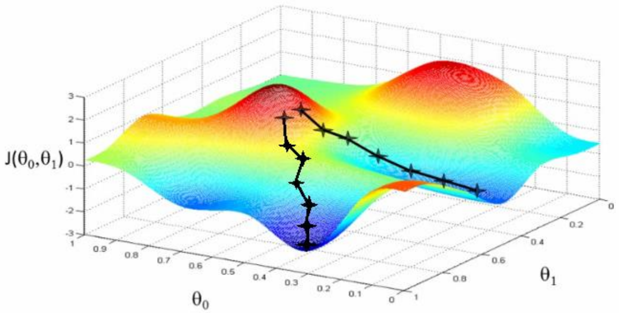




Saddle Point

Cliffs





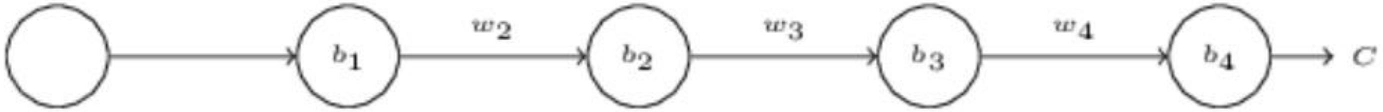
Saddle Point

Vanishing and Exploding Gradients

The Vanishing Gradient Problem:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

$$\left|\left|\frac{\delta h_i}{\delta h_{i-1}}\right|\right|_2 > 1$$



Solution: Gradient clipping

$$\left|\left|\frac{\delta h_i}{\delta h_{i-1}}\right|\right|_2 < 1$$

When gradient is too small, the repetitive multiplication results in vanishing gradient

When gradient is too high, the repetitive multiplication results in exploding gradient

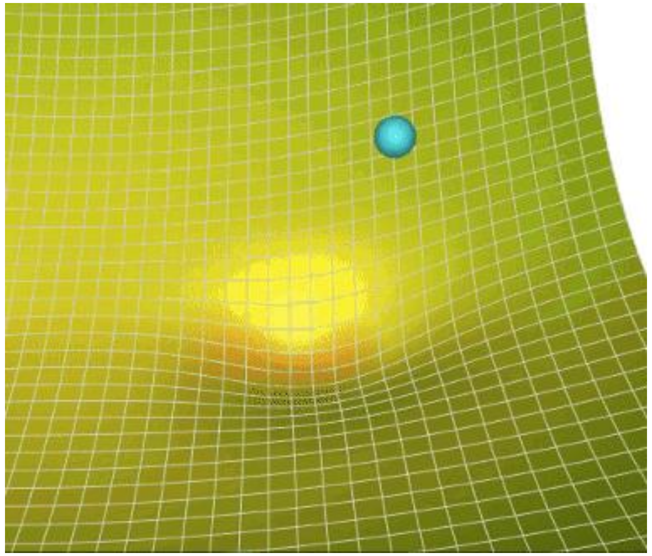
Image Credit:machinelearningjourney.com

Solution: Better Estimate of the Gradient

Ideal optimizer:

$$w_{t+1} = w_t - \alpha * \frac{\delta L}{\delta w_t}$$

- Finds minimum fast and reliably well
- Doesn't get stuck in local minima, saddle points, or plateau region



Vanilla Gradient Descent: One step for the entire dataset

Stochastic Gradient Descent: One step for each stochastically chosen sample

Mini-batch Gradient Descent: One step for each mini-batch of samples chosen stochastically

Best of both worlds!

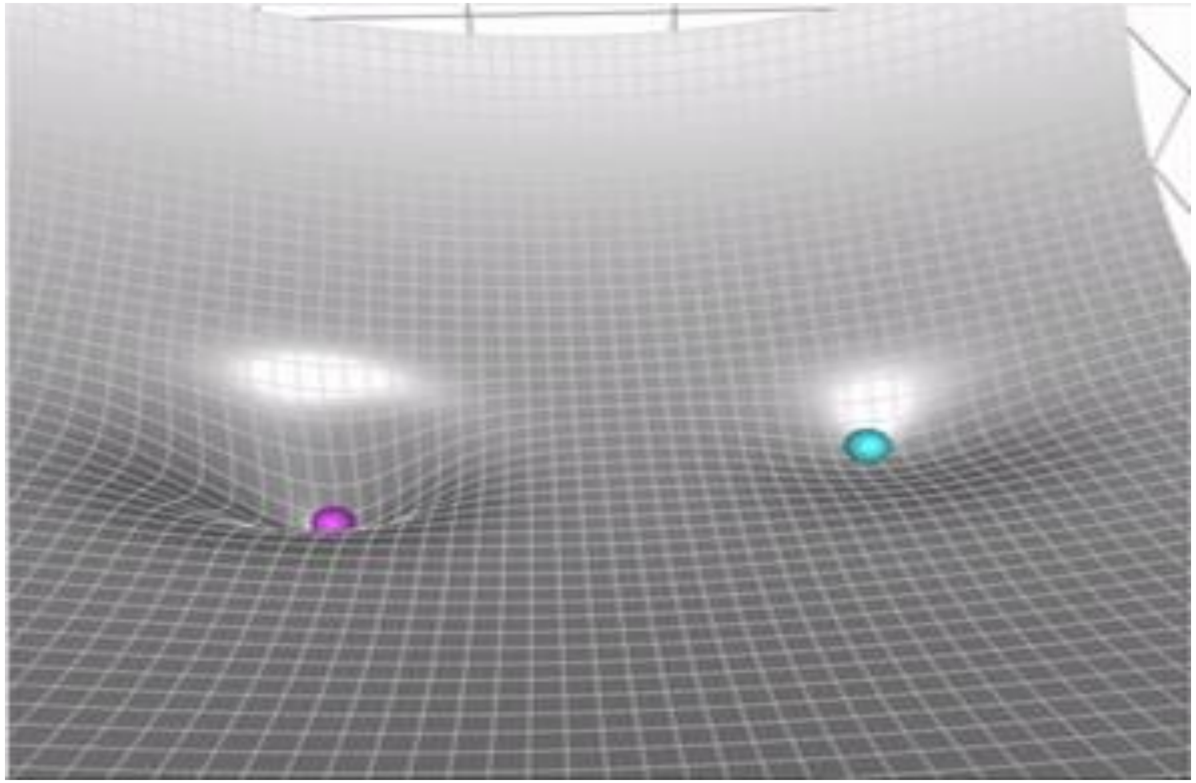
```
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0)
```

Image Credit: towrdsdatascience.com

Solution: Better Update Equation

Gradient Descent with Momentum

Intuitively, this helps us to come out of local minima



$$v_t = \underbrace{\boxed{\gamma\, v_{t-1}}}_{\text{Momentum}} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

```python
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

Image Credit: towrdsdatascience.com

Solution: Better Optimizers

Other Variants:

- RMSprop
- Adagrad
- **Adam**
- Adadelta
- etc.

```
import torch.optim as optim

optimizer = optim.RMSprop(model.parameters(), lr=0.01, alpha=0.99)
```

```
import torch.optim as optim

optimizer = optim.Adagrad(model.parameters(), lr=0.01)
```

```
import torch.optim as optim

optimizer = optim.Adam(model.parameters(), lr=0.01, betas=(0.9, 0.999))
```

```
import torch.optim as optim

optimizer = optim.Adadelta(model.parameters(), lr=0.01, rho=0.9)
```

Simplified View: Variable Learning Rates for Features/Dimensions

Solution: Better Weight Initialization

- All zero initialization
  - Initializing all the weights with zeros leads the neurons to learn the same features during training.

- Random initialization
  - Gaussian
  - Xavier
    - uniform
    - normal
  - Kaiming
    - uniform
    - normal

```python
import torch

w = torch.empty(3, 5)
torch.nn.init.kaiming_uniform_(w, mode='fan_in', nonlinearity='relu')
```

Regularization of the Network

- Input Level
  - Data Augmentation

- Activation level
  - Dropout
  - Dropconnect

- Feature statistics level
  - Batch normalization
  - Layer normalization
  - Group normalization

- Decision level
  - Ensemble

- Constraining network weights
  - $\ell_1$norm, ℓ_2 norm

- Terminating early based of the performance on validation set
  - Early stopping

Summary

- Data Normalization
- Data Augmentation
- Weight Initialization
- Optimization Algorithms
- Regularizer
- Batch Norm
- Dropout

- Better Learning/Optimization
- Better Generalization/Regularization
- Better Loss Functions
- BP as a General Algorithm
- Abstraction of layers and ease in defining layers
- Good implementations

Thank you!

Questions?