CSED703R: Deep Learning for Visual Recognition (2017F)

Lecture 2: Neural Network Basics

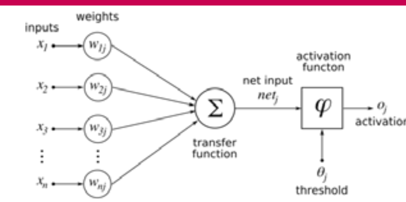
Bohyung Han

Computer Vision Lab.

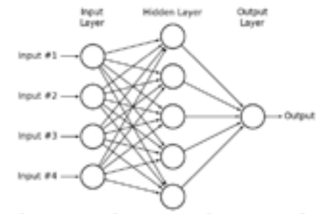
bhhan@postech.ac.kr

History of Neural Networks

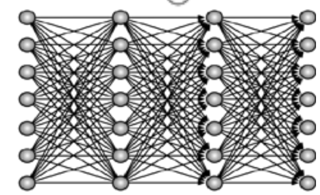
- First generation (since 1958):
  - Perceptrons



- Second generation (since 1986):
  - Multilayer perceptrons



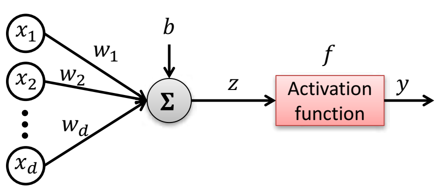
- Third generation (since 2006):
  - Deep learning



Perceptron: Single-Layer Neural Net

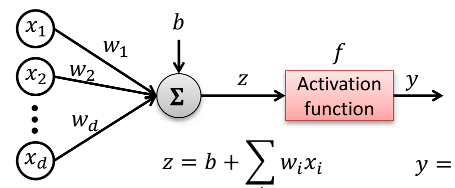
- Framework
  - Input: $\boldsymbol{x} = (x_1, x_2, \dots, x_d)^{\mathrm{T}}$
  - Output: y
  - Model: weight vector $\boldsymbol{w} = (w_1, w_2, \dots, w_d)^{\mathrm{T}}$ and bias b

$$y = f(z) = f\left(\sum_i w_i x_i + b\right) = f(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)$$



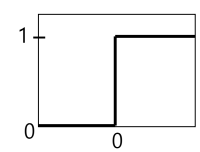
Perceptron: Single-Layer Neural Net

- Activation function



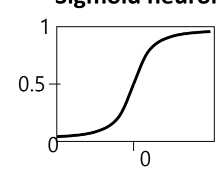
$$z = b + \sum_i w_i x_i \qquad y = f(z)$$

Binary threshold neuron



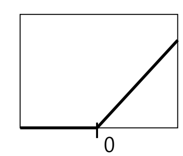
$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

Sigmoid neuron



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

Rectified linear neuron



$$y = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$

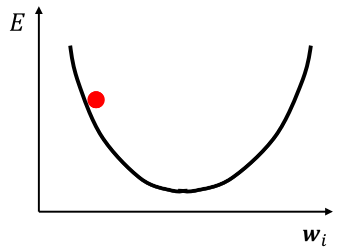
Training Perceptrons

- Gradient descent
  - Training data: $(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \cdots, (\boldsymbol{x}^N, y^N)$
  - Goal: Finding the optimal parameters $\boldsymbol{w}$, which minimize

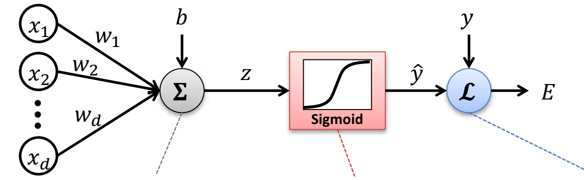
$$E = \frac{1}{2}\sum_n (y^n - \hat{y}^n)^2 \qquad \text{where } \hat{y}^n = f(\boldsymbol{x}^n; \boldsymbol{w})$$

  - Error backpropagation: iteratively updating the model parameters to decrease E as

$$w_i(t+1) = w_i(t) - \epsilon \frac{\partial E}{\partial w_i}$$



Gradient Computation



$$z = b + \sum_i w_i x_i \qquad \hat{y} = \frac{1}{1+e^{-z}} \qquad E = \frac{1}{2}\sum_n (y^n - \hat{y}^n)^2$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{d\hat{y}}{dz} = \hat{y}(1-\hat{y}) \qquad \frac{\partial E}{\partial \hat{y}^n} = -(y^n - \hat{y}^n)$$

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial \hat{y}^n}{\partial w_i}\frac{\partial E}{\partial \hat{y}^n} = \sum_n \frac{\partial z^n}{\partial w_i}\frac{d\hat{y}^n}{dz^n}\frac{\partial E}{\partial \hat{y}^n} = -\sum_n x_i^n \hat{y}^n(1-\hat{y}^n)(y^n - \hat{y}^n)$$

Training Perceptrons

for $t = 1, \ldots, T$

$\hat{y}^n = f(\boldsymbol{x}^n; \boldsymbol{w}_t) \quad (n = 1, \ldots, N)$

$\frac{\partial E}{\partial w_i} = -\sum_n x_i^n \hat{y}^n(1-\hat{y}^n)(y^n - \hat{y}^n) \quad (i = 1, \ldots, d)$

$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{\Delta w}$

endfor

an epoch

- Problems in the standard gradient descent method
  - There are sometimes a lot of training data.
  - Many epochs (iterations) are typically required for optimization.
  - Computing gradients in each epoch takes too much time.

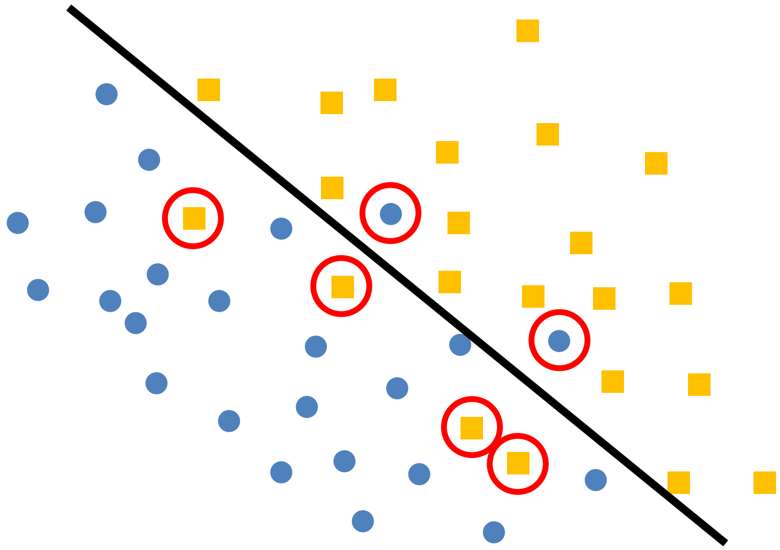
It typically requires a lot of training time!

Characteristics of Single Layer Perceptrons

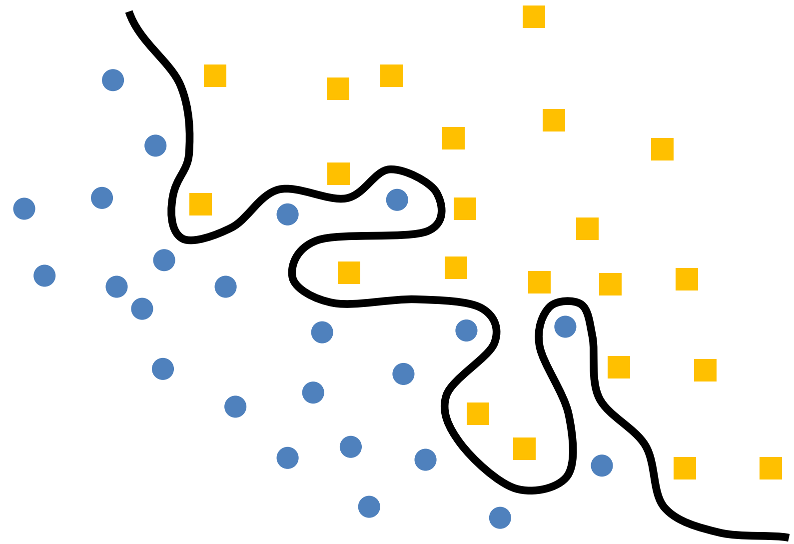
- Pros
  - Single layer: Inference is fast and network is easy to optimize.
  - Intuitive classifier: Feature vectors and classification scores have intuitive connections.

- Cons
  - Linear classifier: less powerful than nonlinear ones
  - No feature learning
  - Training time: It takes much time to compute gradients in each epoch when the size of training data is large.

Linear Classifier

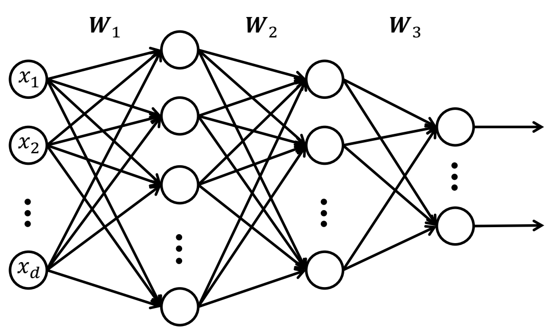


Nonlinear Classifier

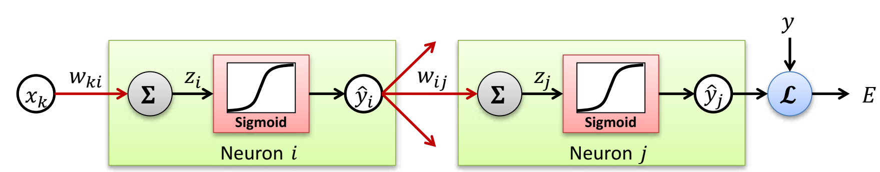


Multilayer Perceptron

- Advantages
  - Nonlinear classification: More complex decision boundary can be defined using multiple layers.
  - Typically achieves better performance



Multi-Layer: Backpropagation



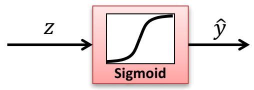
$$\frac{\partial E}{\partial z_j} = \frac{d\hat{y}_j}{dz_j}\frac{\partial E}{\partial \hat{y}_j}$$

$$\frac{\partial E}{\partial \hat{y}_i} = \sum_j \frac{dz_j}{d\hat{y}_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{d\hat{y}_j}{dz_j}\frac{\partial E}{\partial \hat{y}_j}$$

$$\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}}\frac{d\hat{y}_i^n}{dz_i^n}\frac{\partial E}{\partial \hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}}\frac{d\hat{y}_i^n}{dz_i^n}\sum_j w_{ij}\frac{d\hat{y}_j^n}{dz_j^n}\frac{\partial E}{\partial \hat{y}_j^n}$$

Issues in Deep Neural Networks

- Large amount of training time
  - There are sometimes a lot of training data.
  - Many iterations (epochs) are typically required for optimization.
  - Computing gradients in each iteration takes too much time.
- Overfitting
  - Learned function may be too much optimized to be generalized.
- Vanishing gradient problem



$$\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial E}{\partial \hat{y}_i^n} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \boxed{\frac{d\hat{y}_i^n}{dz_i^n}} \sum_j w_{ij} \boxed{\frac{d\hat{y}_j^n}{dz_j^n}} \frac{\partial E}{\partial \hat{y}_j^n}$$

  - Gradients in the lower layers are typically extremely small.
  - Optimizing multi-layer neural networks takes huge amount of time.

Stochastic Gradient Descent (SGD)

- Update weights for each sample

$$E = \frac{1}{2}(y^n - \hat{y}^n)^2 \qquad \boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) - \epsilon \frac{\partial E^n}{\partial \boldsymbol{w}_i}$$

+ Fast, online
− Sensitive to noise

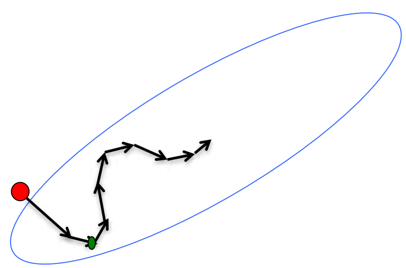
- Minibatch SGD: Update weights for a small set of samples

$$E = \frac{1}{2}\sum_{n \in B}(y^n - \hat{y}^n)^2 \qquad \boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) - \epsilon \frac{\partial E^B}{\partial \boldsymbol{w}_i}$$

+ Fast, online
+ Robust to noise

Momentum

- Remember the previous direction



+ Converge faster
+ Avoid oscillation

$$v_i(t) = \alpha v_i(t-1) - \epsilon \frac{\partial E}{\partial w_i}(t)$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \boldsymbol{v}(t)$$

Weight Decay

- Penalize the size of the weights

$$C = E + \frac{1}{2}\sum_i w_i^2$$

$$w_i(t+1) = w_i(t) - \epsilon \frac{\partial C}{\partial w_i} = w_i(t) - \epsilon \frac{\partial E}{\partial w_i} - \lambda w_i$$

+ Improve generalization a lot!