(Artificial) Neural Networks: Training

Industrial AI

Prof. Seungchul Lee

Training Neural Networks: Optimization

- Learning or estimating weights and biases of multi-layer perceptron from training data

- 3 key components
  - objective function $f(\cdot)$
  - decision variable or unknown ω
  - constraints $g(\cdot)$

- In mathematical expression

$$\min_{\omega} \quad f(\omega)$$

Training Neural Networks: Loss Function

- Measures error between target values and predictions

$$\min_{\omega} \sum_{i=1}^{m} \ell\left(h_{\omega}\left(x^{(i)}\right), y^{(i)}\right)$$

- Example
  - Squared loss (for regression):

$$\frac{1}{m} \sum_{i=1}^{m} \left(h_{\omega}\left(x^{(i)}\right) - y^{(i)}\right)^{2}$$

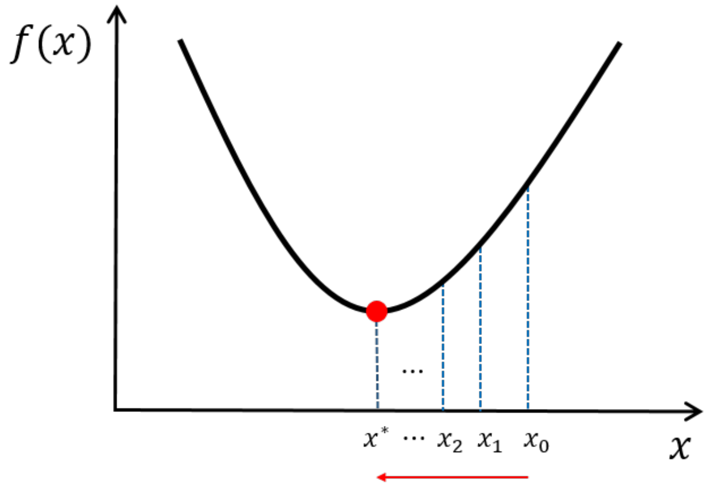
  - Cross entropy (for classification):

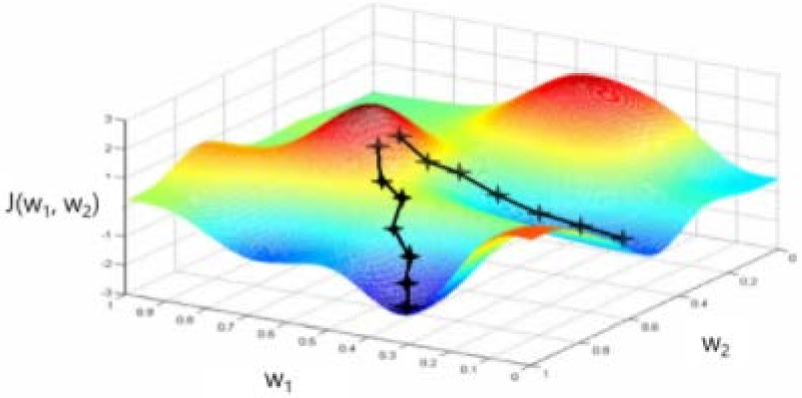
$$-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_{\omega}\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\omega}\left(x^{(i)}\right)\right)$$

Training Neural Networks: Gradient Descent

- Negative gradients points directly downhill of the cost function
- We can decrease the cost by moving in the direction of the negative gradient (α is a learning rate)

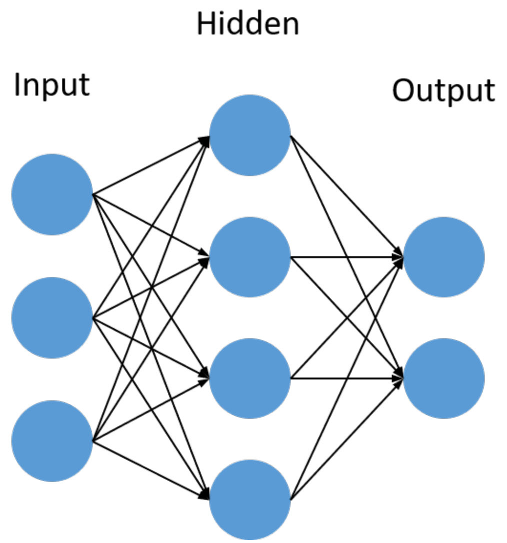
$$\omega \Leftarrow \omega - \alpha \nabla_\omega \ell \left(h_\omega \left(x^{(i)} \right), y^{(i)} \right)$$





Gradients in ANN

- Learning weights and biases from data using gradient descent
- $\frac{\partial \ell}{\partial \omega}$: too many computations are required for all ω
- Structural constraint of NN:
  - Composition of functions
  - Chain rule
  - Dynamic programming

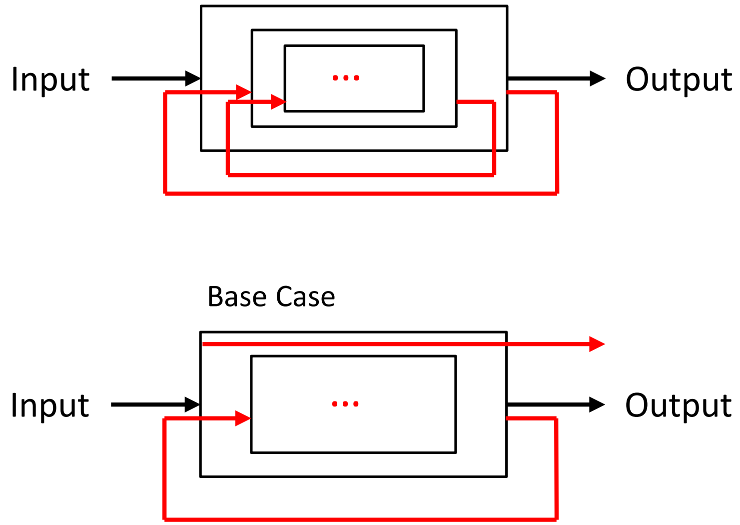


$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

Dynamic Programming

Recursive Algorithm

- One of the central ideas of computer science
- Depends on solutions to smaller instances of the same problem (= sub-problem)
- Function to call itself (it is impossible in the real world)
- Factorial example
  - $n! = n \cdot (n-1) \cdots 2 \cdot 1$



Dynamic Programming

- Dynamic Programming: general, powerful algorithm design technique

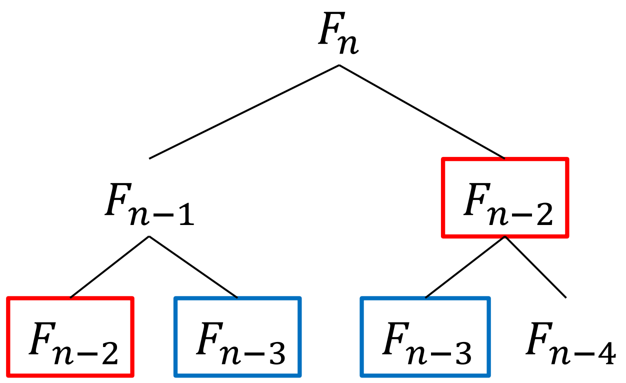
- Fibonacci numbers:

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Naïve Recursive Algorithm

fib(n) :
  if $n \le 2$: $f = 1$
  else : $f = \text{fib}(n-1) + \text{fib}(n-2)$
  return f

- It works. Is it good?



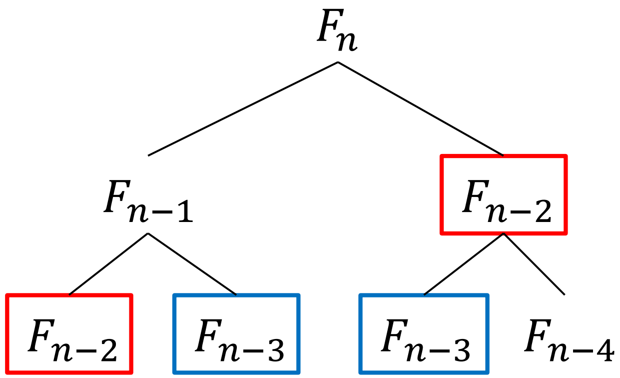
Memorized Recursive Algorithm

```
memo  = [ ]
    fib(n) :
            if n in memo : return memo[n]

            if n ≤ 2 :  f = 1
            else :  f = fib(n − 1) + fib(n − 2)

            memo[n] = f
            return f
```

- Benefit?
  - fib(n) only recurses the first time it's called



Dynamic Programming Algorithm

- Memorize (remember) & re-use solutions to subproblems that helps solve the problem

- DP $\approx$ recursion + memorization

Backpropagation

Training Neural Networks: Optimization

- Learning or estimating weights and biases of multi-layer perceptron from training data

- 3 key components
  - objective function $f(\cdot)$
  - decision variable or unknown ω
  - constraints $g(\cdot)$

- In mathematical expression

$$\min_{\omega} \quad f(\omega)$$

Training Neural Networks: Loss Function

- Measures error between target values and predictions

$$\min_{\omega} \sum_{i=1}^{m} \ell\left(h_\omega\left(x^{(i)}\right), y^{(i)}\right)$$

- Example
  - Squared loss (for regression):

$$\frac{1}{m}\sum_{i=1}^{m}\left(h_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2$$

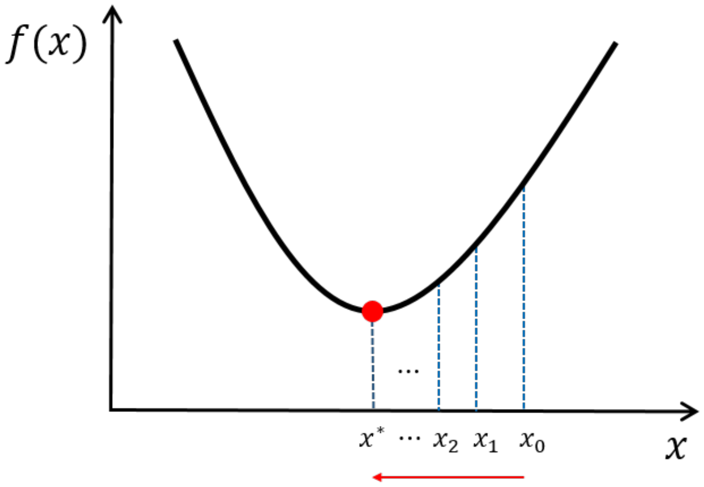
  - Cross entropy (for classification):

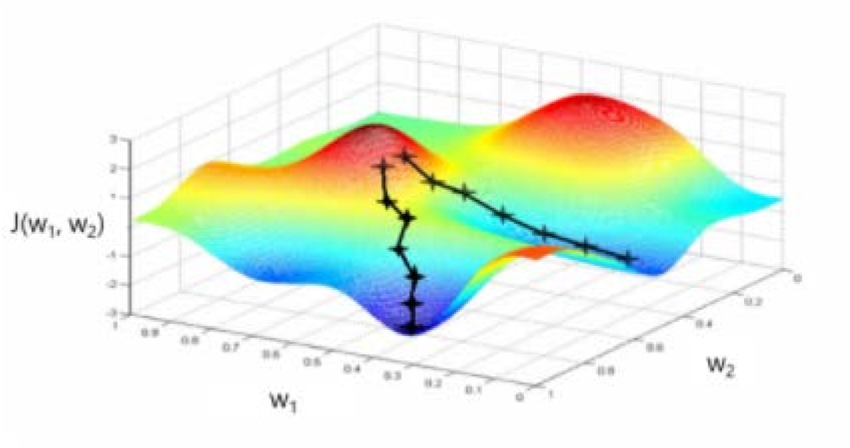
$$-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h_\omega\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\omega\left(x^{(i)}\right)\right)$$

Training Neural Networks: Gradient Descent

- Negative gradients points directly downhill of the cost function
- We can decrease the cost by moving in the direction of the negative gradient (α is a learning rate)

$$\omega \Leftarrow \omega - \alpha \nabla_{\omega} \ell \left(h_{\omega} \left(x^{(i)} \right), y^{(i)} \right)$$

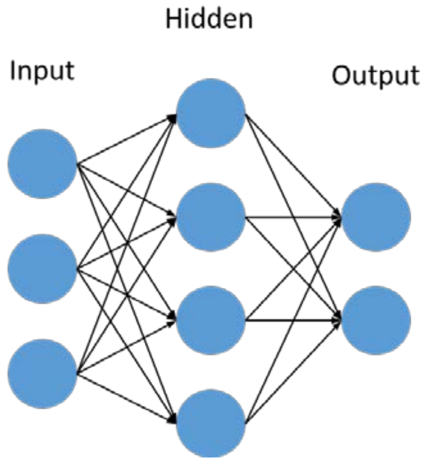




Gradients in ANN

- Learning weights and biases from data using gradient descent
- $\frac{\partial \ell}{\partial \omega}$: too many computations are required for all ω

- Structural constraint of NN:
  - Composition of functions
  - Chain rule
  - Dynamic programming



$$x \longrightarrow \boxed{\hat{y} = f_{\omega_1,\cdots,\omega_k}(x)} \longrightarrow y$$

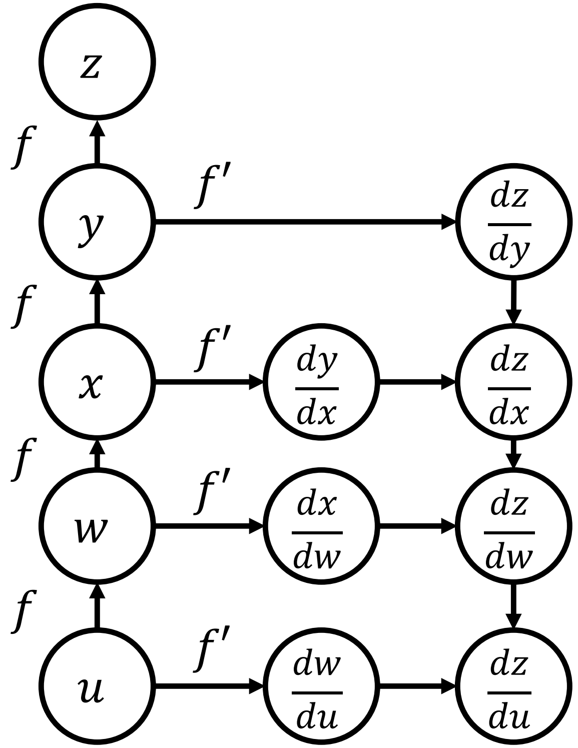
Training Neural Networks: Backpropagation Learning

- Forward propagation
  - the initial information propagates up to the hidden units at each layer and finally produces output

- Backpropagation
  - allows the information from the cost to flow backwards through the network in order to compute the gradients

Backpropagation

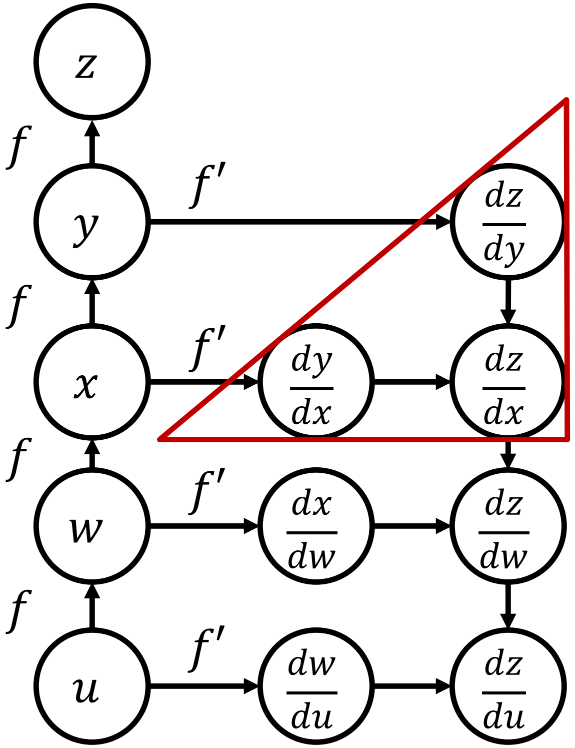
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

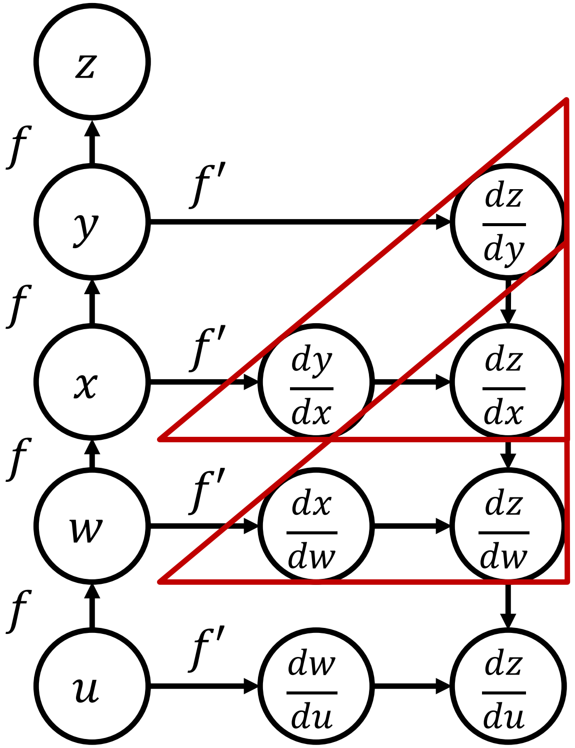
- Chain Rule
  – Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  – Update weights recursively

Backpropagation

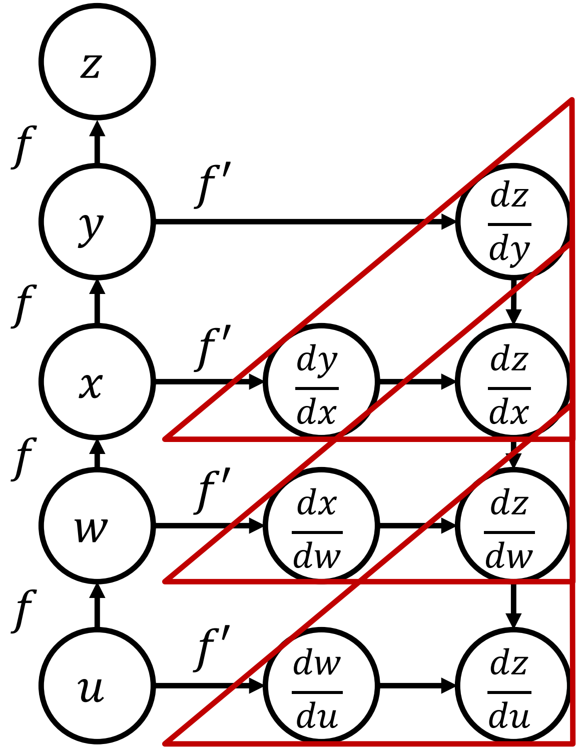
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f(g(x))' = f'(g(x))g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

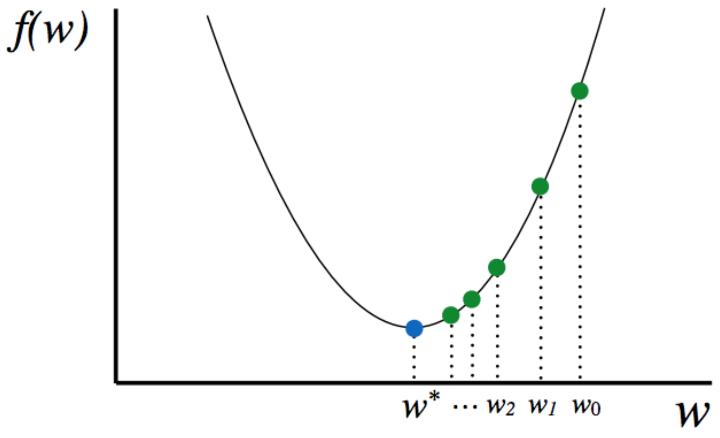
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$
- Backpropagation
  - Update weights recursively with memory



Training Neural Networks with TensorFlow

- Optimization procedure

Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until stopping criterion is satisfied



- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools → We will use the TensorFlow

Core Foundation Review

The Perceptron	Neural Networks	Training in Practice
• Structural building blocks • Nonlinear activation functions	• Stacking Perceptrons to form neural networks • Optimization through backpropagation	• Adaptive learning • Batching • Regularization