165B
Machine Learning
Feedforward Network

Lei Li (leili@cs)
UCSB

Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and Mu Li & Alex Smola's 157 courses on Deep Learning, with modification

Announcement

- Instruction continue on zoom till Jan 31

Recap

- Logistic Regression for classification
  - single linear layer with Softmax output
- General framework to formulate a learning task is through empirical risk minimization (ERM)
- Minimizing cross-entropy is a realization of ERM
- Kullback-Leibler Divergence

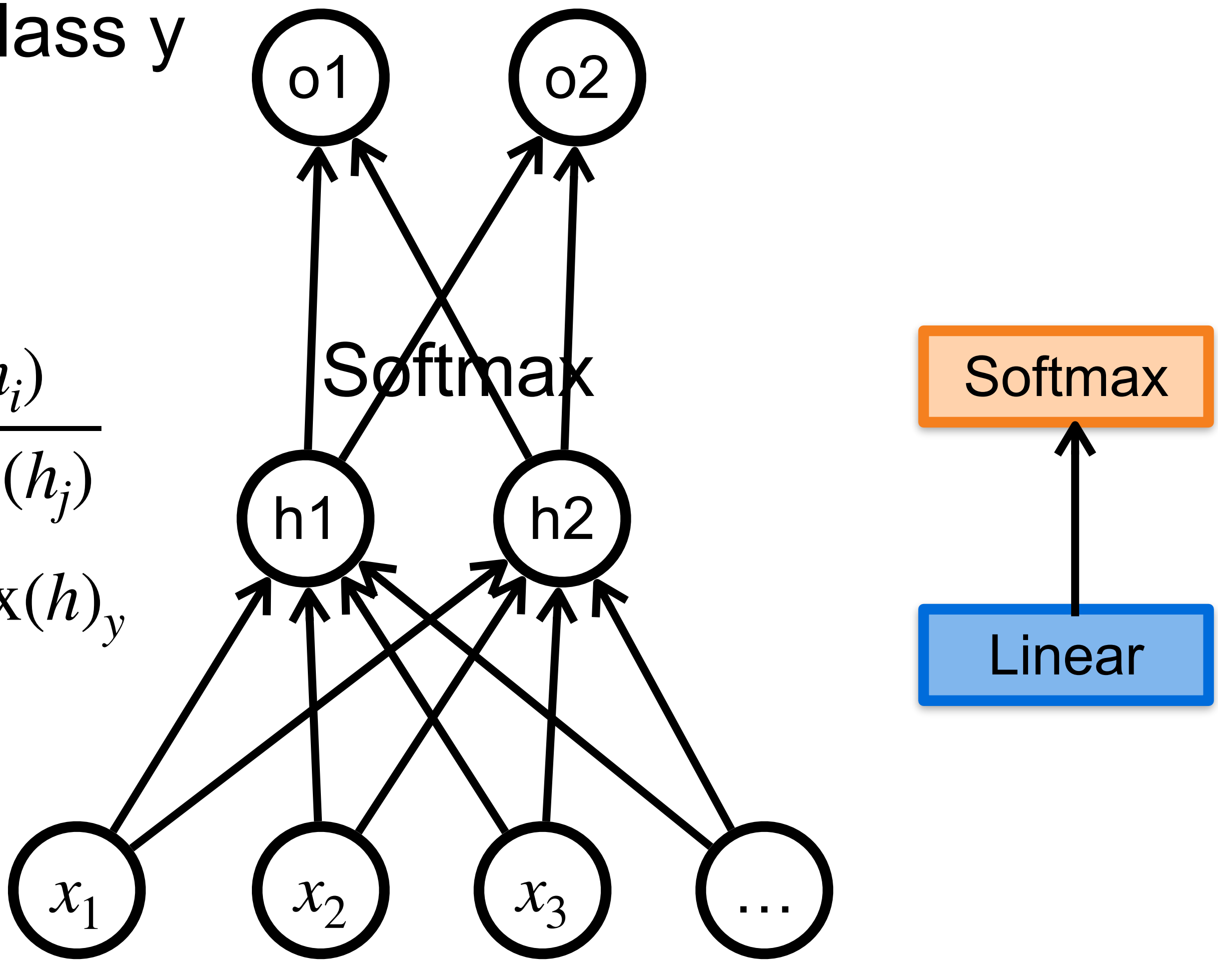
Logistic Regression

output: prob. of class y

$$h = \mathbf{W} \cdot \mathbf{x}$$

$$\text{softmax}(h)_i = \frac{\exp(h_i)}{\sum_j \exp(h_j)}$$

$$p(y \mid h) = \text{softmax}(h)_y$$

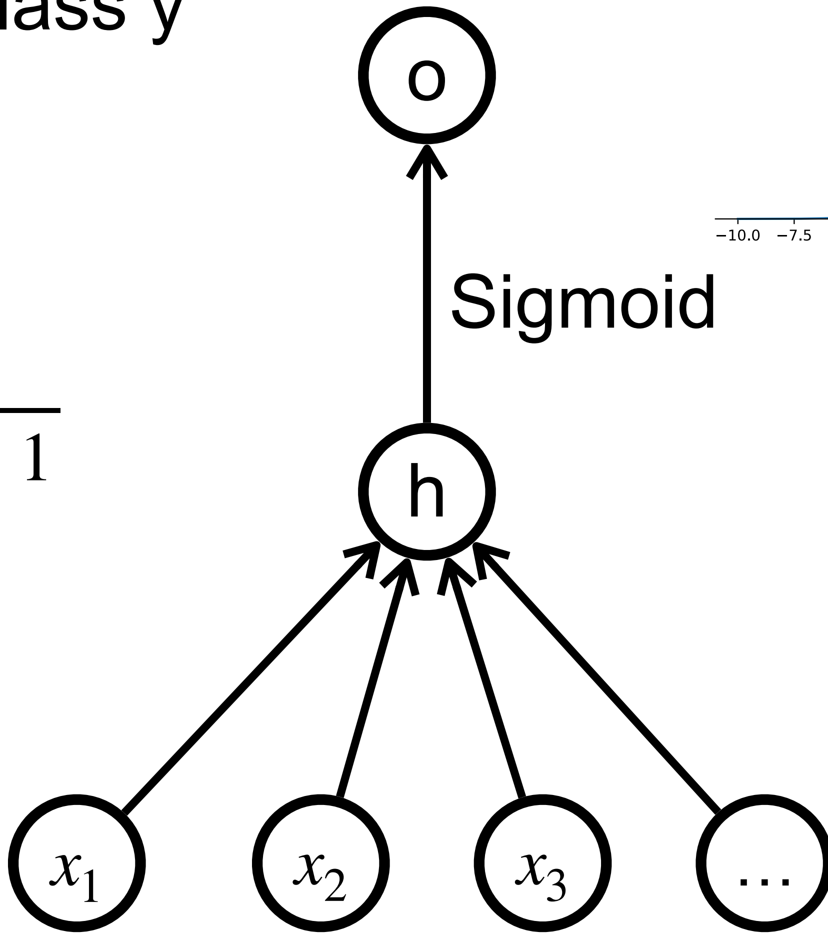


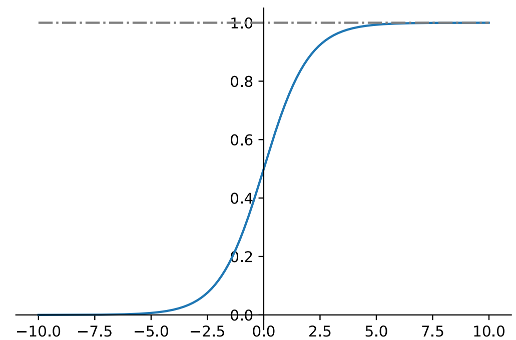
Logistic Regression for Binary Classification

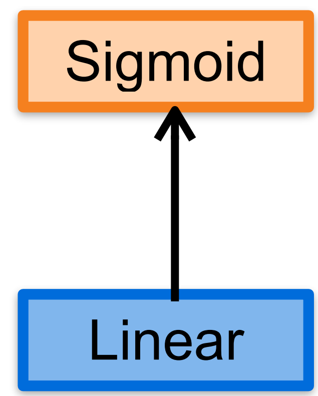
output: prob. of class y

$$h = \mathbf{w} \cdot \mathbf{x}$$

$$p(y \mid h) = \sigma(h) = \frac{e^h}{e^h + 1}$$







Cross-Entropy Loss for Classification

$$\min \mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, f(x_n)) = \frac{1}{N}\sum_{n=1}^{N} -\log f(x_n)_{y_n}$$

Kullback-Leibler Divergence

- "Distance" between distributions (e.g. truth & estimate)

Number of extra bits when using the wrong code

$$D[p\|q] = \int dp(x) \log \frac{p(x)}{q(x)} = \int dp(x) [-\log q(x)] - [-\log p(x)]$$

Inefficient bits

Optimal bits

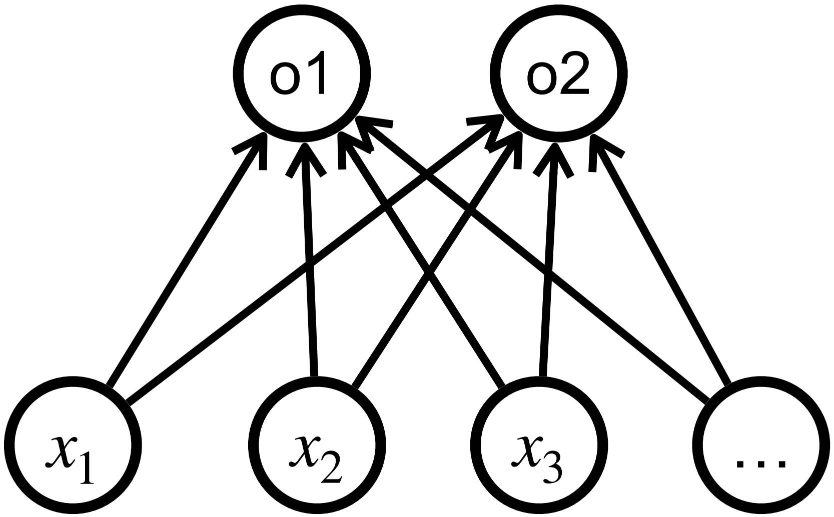
- Nonnegativity of KL Divergence

Jensen Inequality
log is concave

$$D[p\|p] = \int dp(x) \log \frac{p(x)}{p(x)} = 0$$

$$D[p\|q] = -\int dp(x) \log \frac{q(x)}{p(x)} \geq -\log \int dp(x) \frac{q(x)}{p(x)} = 0$$

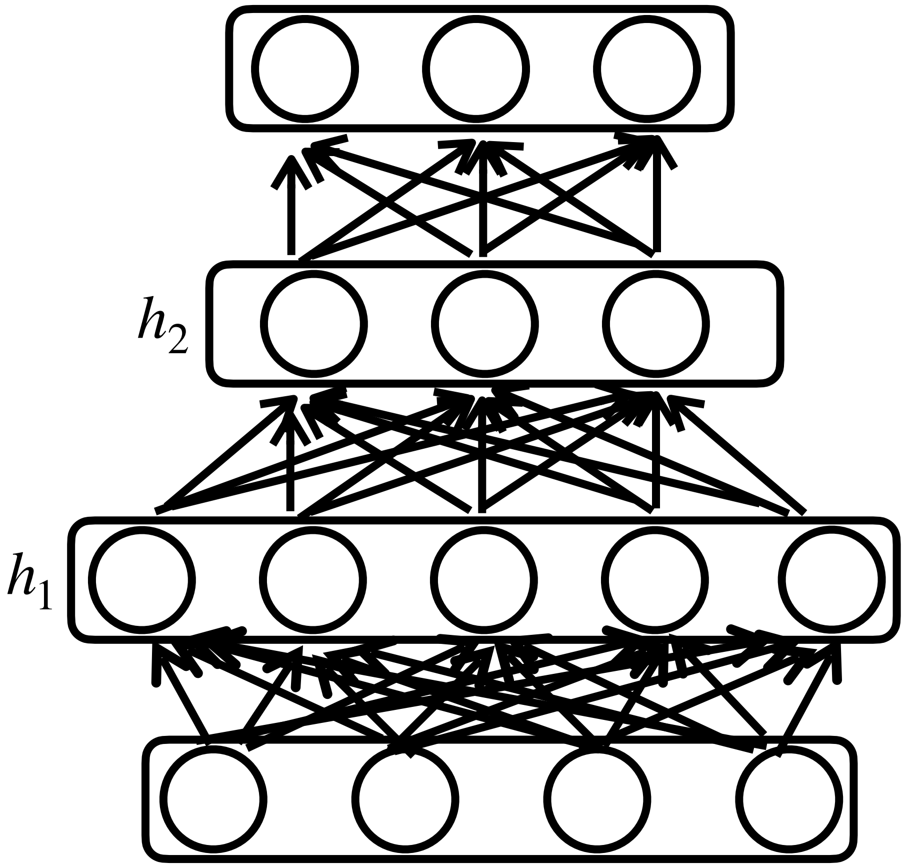
Limitation of Logistic Regression



- Single layer has limited capability
  - cannot learn XOR
- The decision boundary is linear
  - cannot learn a nonlinear decision boundary
  - why?

Feedforward Neural Net (FFN)

- also known as multilayer perceptron (MLP)
- Layers are connected sequentially
- Each layer has full-connection (each unit is connected to all units of next layer)
  - Linear project followed by
  - an element-wise nonlinear activation function
- There is no connection from output to input



Feedforward Neural Net (FFN)

- also known as multilayer perceptron (MLP)

$x \in \mathbb{R}^d$

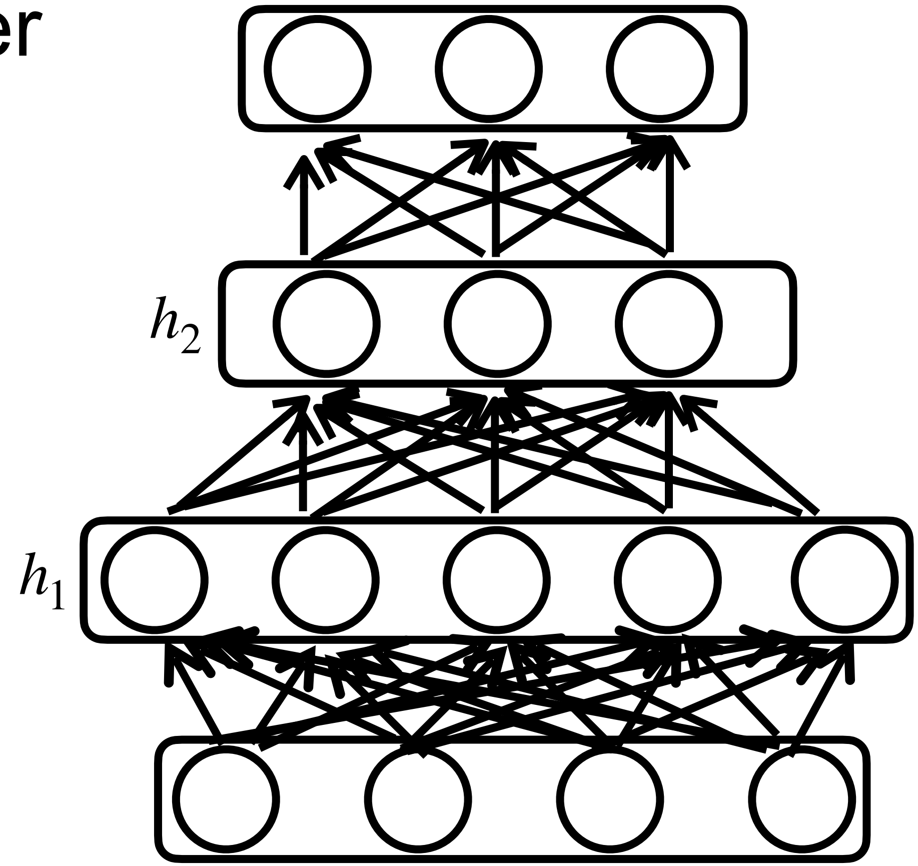
$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$

$h_l = \sigma(w_l \cdot h_{l-1} + b_l) \in \mathbb{R}^{d_l}$

$o = \text{Softmax}(w_L \cdot h_{L-1} + b_L)$

Parameters

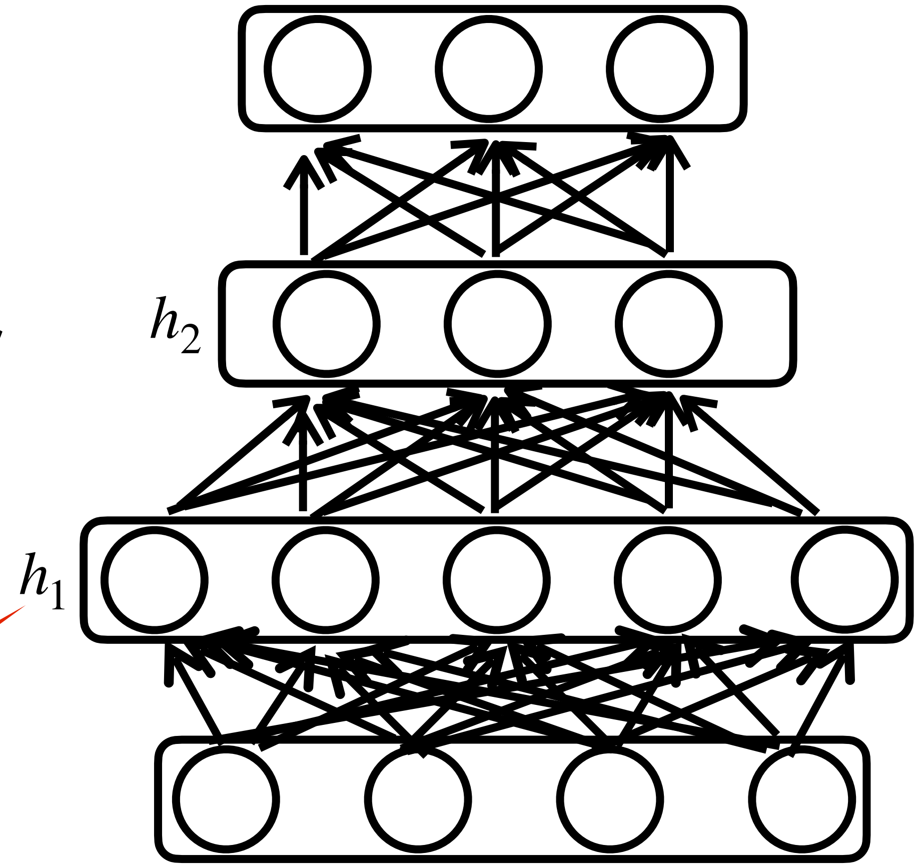
$\theta = \{w_1, b_1, w_2, b_2, \ldots\}$



Hidden layers

- $$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$$
  $$h_l = \sigma(w_l \cdot h_{l-1} + b_l) \in \mathbb{R}^{d_l}$$

σ is element-wise nonlinear activation function



Why do we need an a nonlinear

What-if Layer with no activation?

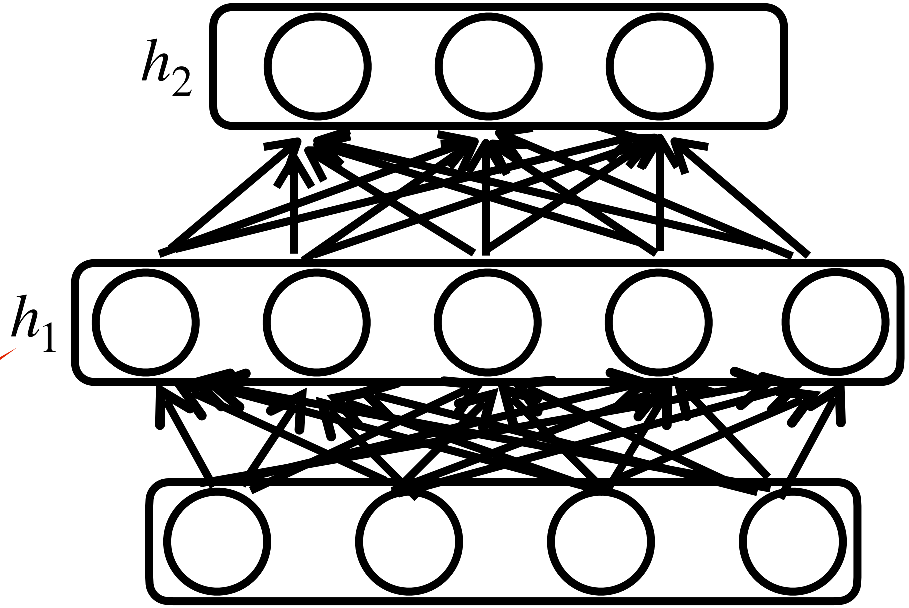
Linear ...

$$\mathbf{h}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{h}_2 = \mathbf{w}_2^T\mathbf{h}_1 + b_2$$

hence $h_2 = \mathbf{w}_2^\top\mathbf{W}_1\mathbf{x} + b'$

Why do we need an a nonlinear

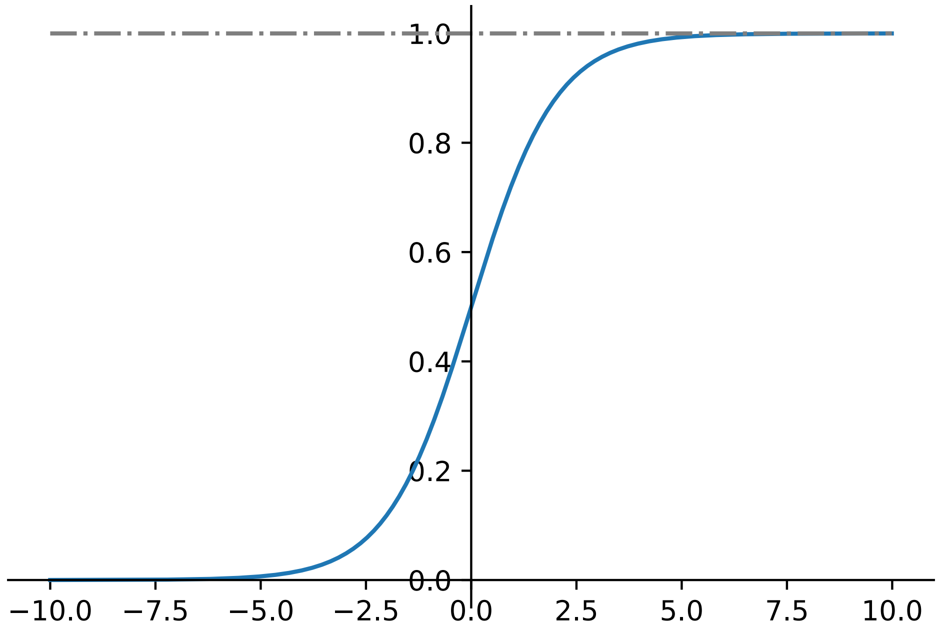


Sigmoid Activation

Map input into (0, 1), a soft version of

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

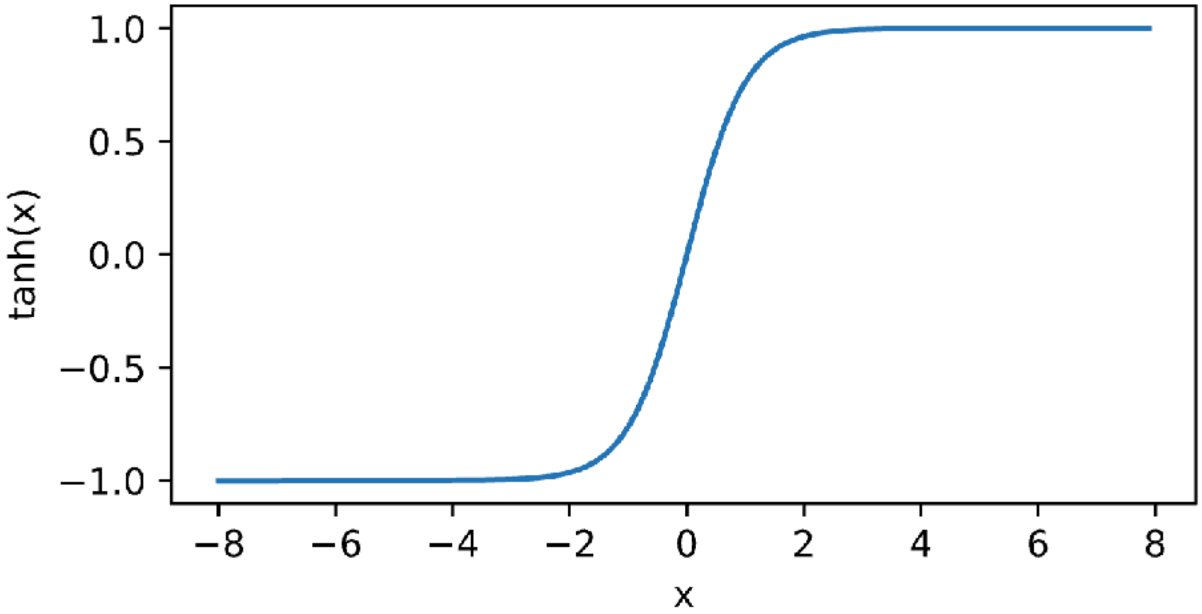
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Tanh Activation

Map inputs into (-1, 1)

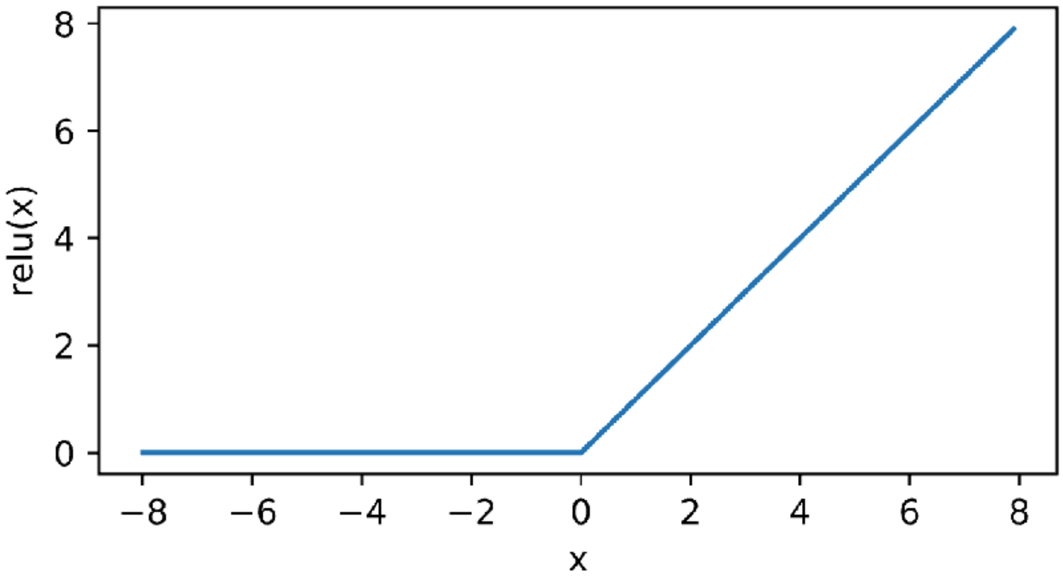
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$



ReLU Activation

ReLU: rectified linear unit

$$\text{ReLU}(x) = \max(x, 0)$$

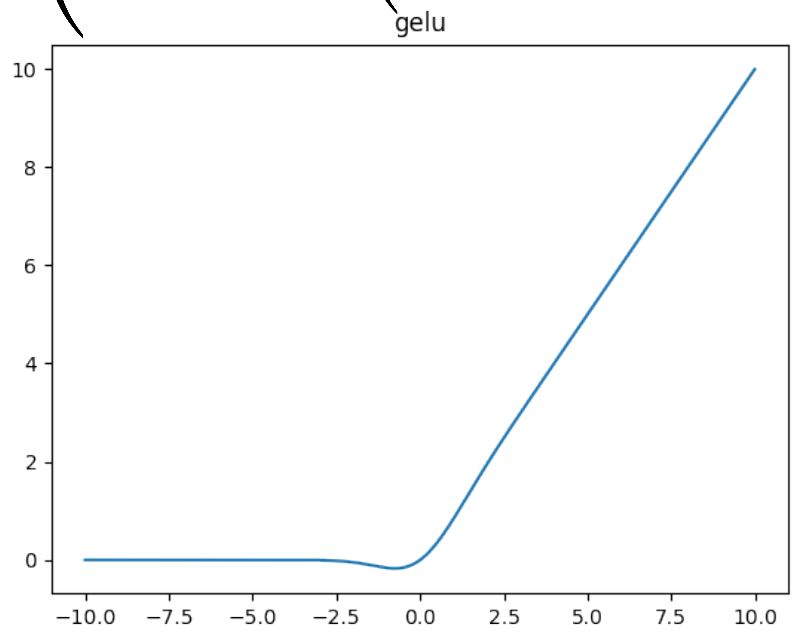


Gaussian Error Linear Units (GELU)

smoothed version of RELU

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right]$$

$$\text{GELU}(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}(x + 0.044715x^3)\right)\right)$$



Feedforward Network for Classification

Softmax as the final output layer.

$x \in \mathbb{R}^d$

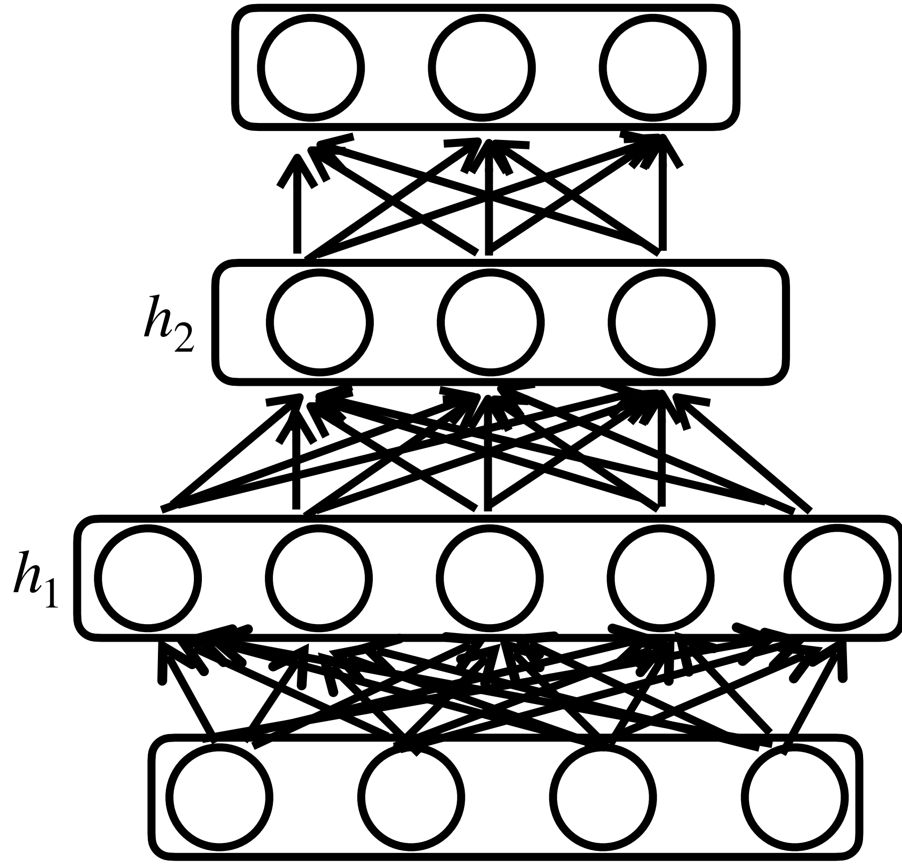
$h_1 = \sigma(w_1 \cdot x + b_1) \in \mathbb{R}^{d_1}$

$h_l = \sigma(w_l \cdot h_{l-1} + b_l) \in \mathbb{R}^{d_l}$

$o = \text{Softmax}(w_L \cdot h_{L-1} + b_L)$

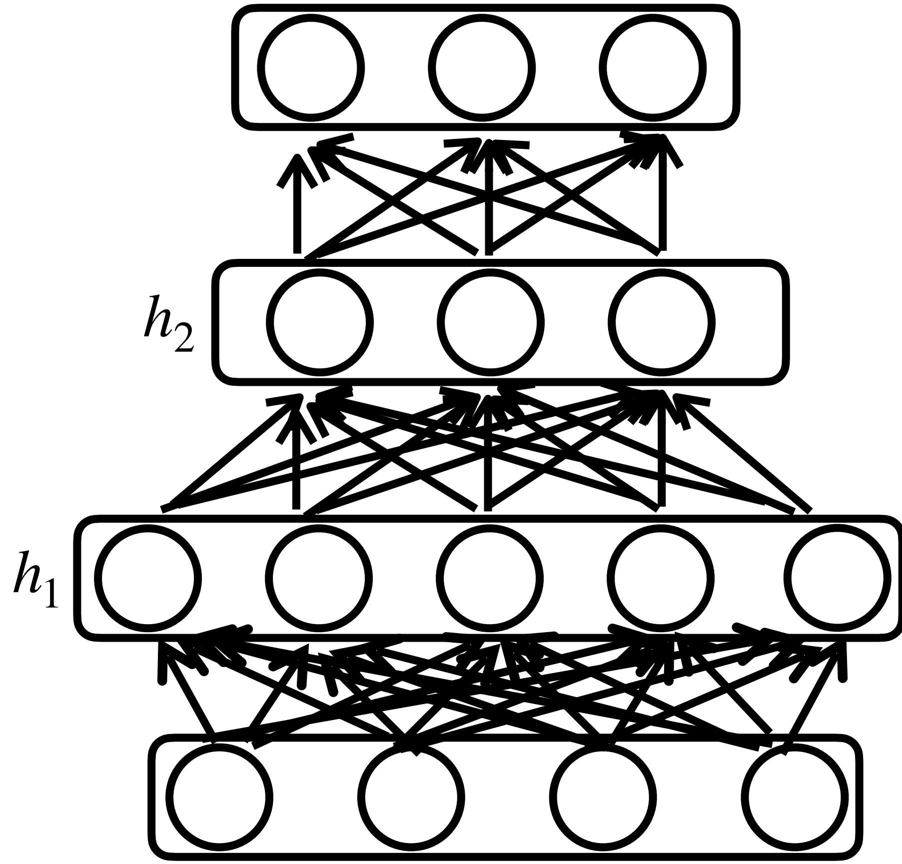
Parameters

$\theta = \{w_1, b_1, w_2, b_2, \ldots\}$



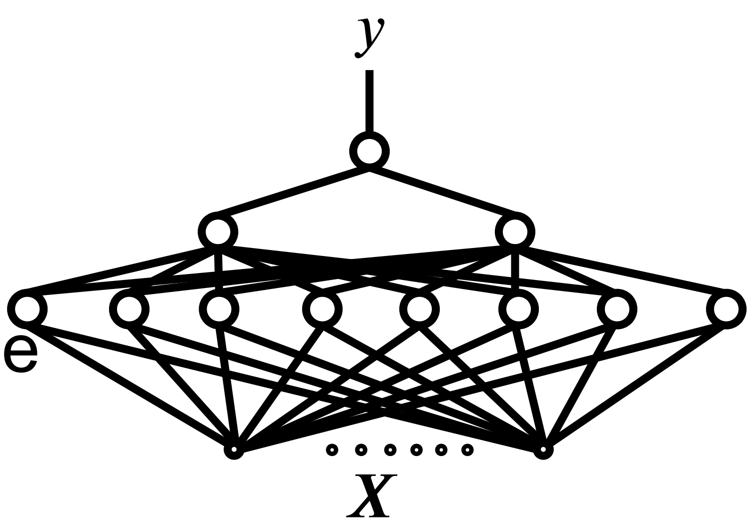
Hyperparameters for FFN

- Number of layers
- Number of hidden dimension for each layer



The Learning Problem

- Given a training set of input-output pairs $D = \{(x_n, y_n)\}_{n=1}^N$
  - x_n and y_n may both be vectors
- To find the model parameters such that the model produces the most accurate output for each training input
  - Or a close approximation of it
- Learning the parameter of a neural network is an instance!
  - The network architecture is given



Risk

- The expected risk is the average risk (loss) over the entire (x, y) data space

$$R(\theta) = E_{\langle x,y \rangle \in P}\left[\ell(y, f(x;\theta))\right] = \int \ell(y, f(x;\theta))dP(x,y)$$

The general learning framework: Empirical Risk Minimization (ERM)

- Ideally, we want to minimize the expected risk
  - but, unknown data distribution …
- Instead, given a training set of empirical data $D = \{(x_n, y_n)\}_{n=1}^{N}$
- Minimize the empirical risk over training data

$$\hat{\theta} \leftarrow \arg\min_{\theta} L(\theta) = \frac{1}{N} \sum_{n} \ell(y_n, f(x_n; \theta))$$

The general learning framework: Empirical Risk Minimization (ERM)

- Ideally, we want to minimize the expected

Note : Its really a measure of error, but using standard terminology, we will call it a "Loss"

Note 2: The empirical risk $L(\theta)$ is only an empirical approximation to the true risk $R(\theta) = E_{\langle x,y \rangle \in P}\left[\ell(y, f(x;\theta))\right]$, which is our ultimate optimization objective

Note 3: For a given training set the loss is only a function of θ

Loss function

- The empirical risk (loss) is determined by the loss function
- Ideal loss for classification: 0-1 loss

$$l(y, f(x)) = \begin{cases} 0 & \text{if } y = \arg\max_k f(x)_k \\ 1 & \text{otherwise} \end{cases}$$

- Cross entropy loss is one common loss for classification

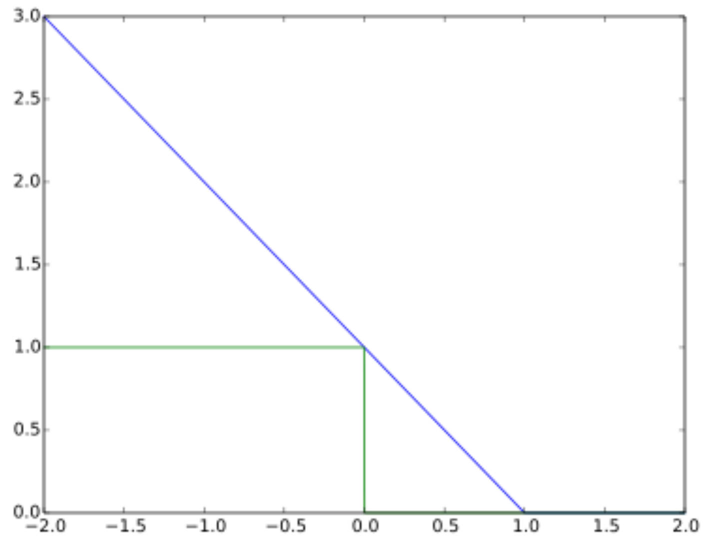
$$\min \mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} H(y_n, f(x_n)) = \frac{1}{N}\sum_{n=1}^{N} - y_n \cdot \log f(x_n)$$

Other Loss for Classification

- Hinge loss
  - Binary classification:

$$\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

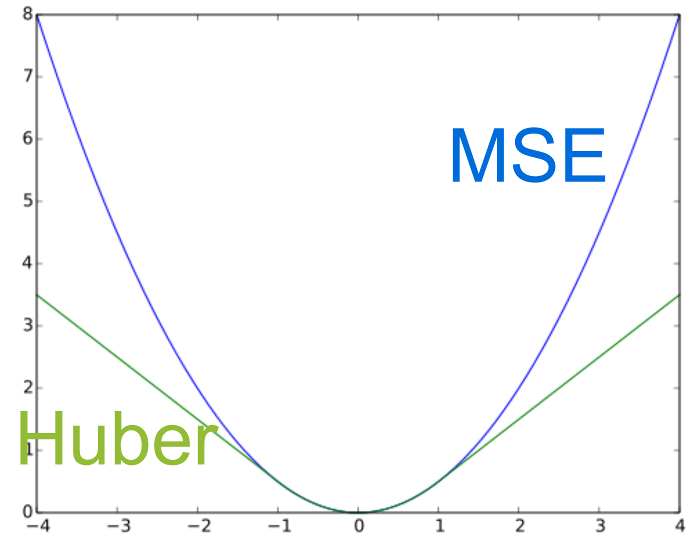
When ground-truth y is +1, prediction $\hat{y}$<0 lead to larger penalty



  - Multi-class

$$\ell(y, \hat{y}) = \sum_{k \neq y} \max(0, 1 - \hat{y}_y + \hat{y}_k)$$

Loss for Regression



- Continuous outcome

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(x_n))$$

- squared loss: $\ell(y,f) = \frac{1}{2}\|f - y\|_2^2$
- L1 loss: $\ell(y,f) = \frac{1}{2}|f - y|$
- Huber loss: $\ell(y,f) = \begin{cases} \frac{1}{2}\|f-y\|_2^2 & \text{if } \|f-y\|_2 \le \delta \\ \delta(|f-y| - \frac{\delta}{2}) & \text{otherwise} \end{cases}$

Recap

- General framework to formulate a learning task is through empirical risk minimization (ERM)
- Minimizing cross-entropy is a realization of ERM

Learning the Model

- Finding the parameter θ to minimize the empirical risk over training data $D = \{(x_n, y_n)\}_{n=1}^{N}$

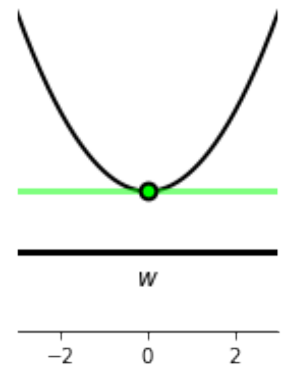
$$\hat{\theta} \leftarrow \arg\min_{\theta} L(\theta) = \frac{1}{N} \sum_{n} \ell(y_n, f(x_n; \theta))$$

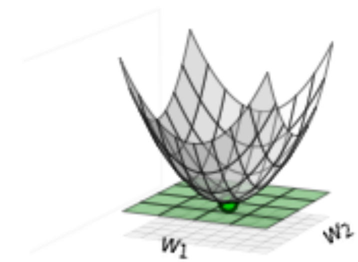
- This is an instance of function optimization problem
  - Many algorithms exist (following lectures)

Optimization

- Consider a generic function minimization problem

$$\min_x f(x) \text{ where } f: \mathbb{R}^d \to \mathbb{R}$$





- Optimality condition:

$$\nabla f|_x = 0, \text{ where i-th element of } \nabla f|_x \text{ is } \frac{\partial f}{\partial x_i}$$

- Linear regression has closed-form solution
- In general, no closed-form solution for the equation.

Generic Iterative Algorithm

- Consider a generic function minimization problem, where x is unknown variable

$$\min_x f(x) \text{ where } f: \mathbb{R}^d \to \mathbb{R}$$

- Iterative update algorithm

$$x_{t+1} \leftarrow x_t + \Delta$$

- so that $f(x_{t+1}) \ll f(x_t)$
- How to find Δ

Taylor approximation

- $f(x+\Delta x) = f(x) + \Delta x^T \nabla f|_x + \frac{1}{2}\Delta x^T \nabla^2 f|_x \Delta x + \cdots$
- Theorem: if f is twice-differentiable and has continuous derivatives around x, for any small-enough Δx, there is $f(x+\Delta x) = f(x) + \Delta x^T \nabla f|_x + \frac{1}{2}\Delta x^T \nabla^2 f|_z \Delta x$, where $\nabla^2 f|_z$ is the Hessian at z which lies on the line connecting x and $x + \Delta x$
- First-order and second-order Taylor approximation result in gradient descent and Newton's method

Gradient Descent

- $f(x_t + \Delta x) \approx f(x_t) + \Delta x^T \nabla f|_{x_t}$
- To make $\Delta x^T \nabla f|_{x_t}$ smallest
- $\Rightarrow \Delta x$ in the opposite direction of $\nabla f|_{x_t}$ i.e. $\Delta x = -\nabla f|_{x_t}$
- Update rule: $x_{t+1} = x_t - \eta \nabla f|_{x_t}$
- η is a hyper-parameter to control the learning rate

Gradient Descent Algorithm

```
learning rate eta.
1.set initial parameter θ ← θ_0
2.for epoch = 1 to maxEpoch or until
  converg:
3.  for each data (x, y) in D:
4.      compute error err(f(x; θ) - y)
5.      compute gradient g = ∂err(θ)/∂θ
6.      total_g += g
7.  update θ = θ - eta * total_g / N
```

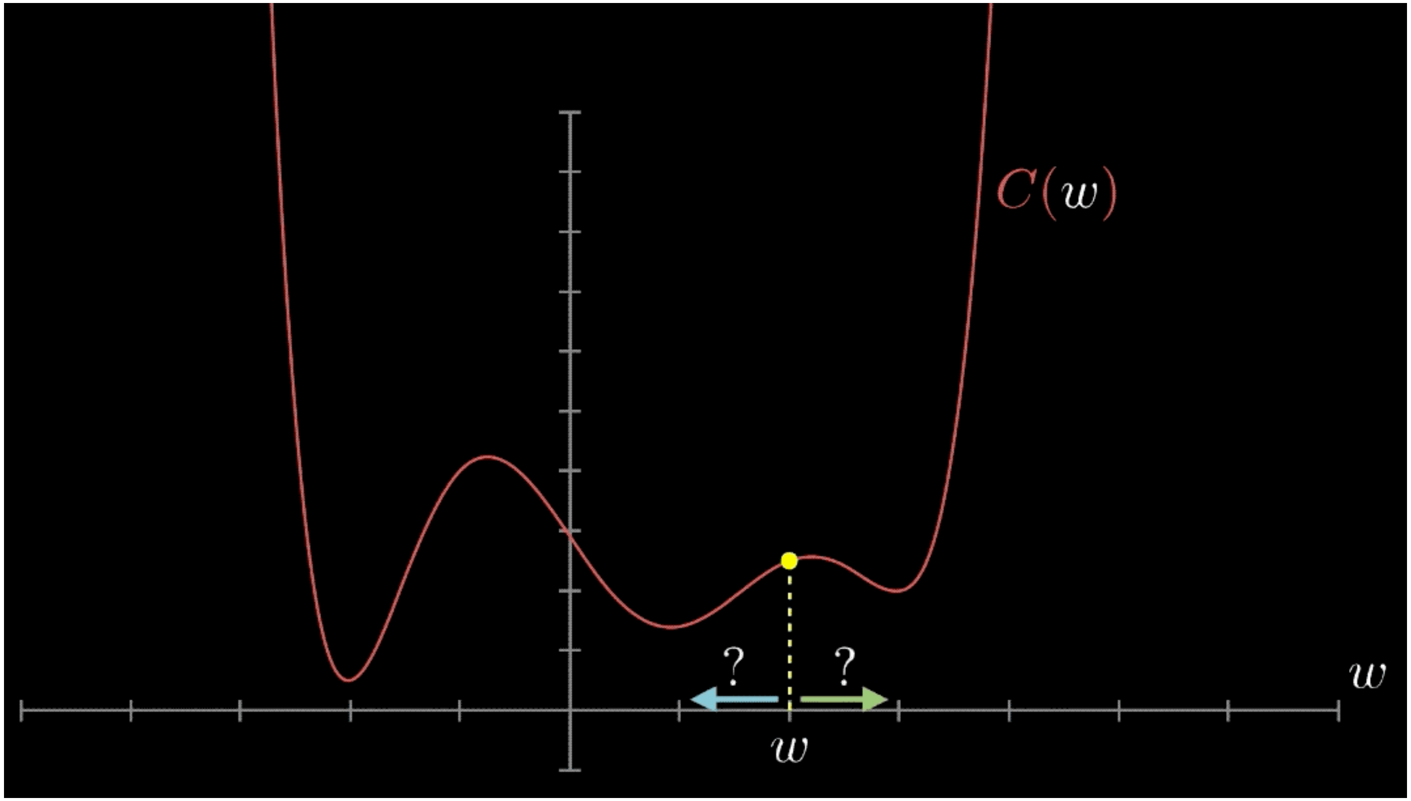
$\theta \leftarrow \theta_0$, $g = \frac{\partial \text{err}(\theta)}{\partial \theta}$

Understand GD

- Surrogate function

$$\tilde{f}(x_t) = f(x_t) + \Delta x^T \nabla f|_{x_t} + \frac{1}{2}\|\Delta x\|_2^2$$

GD: Illustration



[credit: gif from 3blue1brown]

Does gradient descent guarantee finding the optimal solution?

- Depends
- Convex and smooth function: yes!
- Non-convex? local optimal

Recap

- First-order optimality condition: gradient=0
- Gradient descent is an iterative algorithm to update the parameter towards the opposite direction of gradient

Next Up

- Gradient calculation using Back-propagation
- More on optimization
- Training/testing procedure
- Generalization problem
- Regularization tricks