Lecture 11:
Neural Networks

https://playground.tensorflow.org/

Kernel Methods

Kernel ridge regression

- Ridge regression with feature map $\phi(\cdot) \in \mathbb{R}^p$

$$\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \frac{1}{2}\|y - \phi(X)w\|_2^2 + \lambda \frac{1}{2}\|w\|_2^2$$

- $w = \sum_{i=1}^{n} \alpha_i \phi(x_i) = \phi(X)^T \alpha$
- Let $K \in \mathbb{R}^{n \times n}$ be the kernel matrix, where $K_{i,j} = \phi(x_i)^T \phi(x_j)$

$$\hat{\alpha} = \arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{2}\|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \frac{1}{2}\lambda \alpha^T K \alpha$$

Thus, $\hat{\alpha}_{\text{kernel}} = (\mathbf{K} + \lambda \mathbf{I}_{n \times n})^{-1}\mathbf{y}$

- Prediction $\hat{y}_{\text{new}} = \hat{w}^T \phi(x_{\text{new}}) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x_{\text{new}}) = \sum_{i=1}^{n} K(x_i, x_{\text{new}}) \alpha_i$

RBF kernel $k(x_i, x) = \exp\left\{-\frac{\|x_i - x\|_2^2}{2\sigma^2}\right\}$

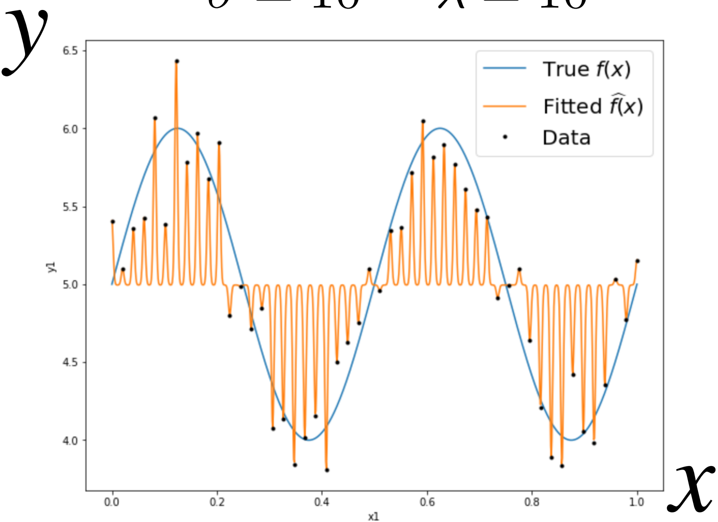
- $\mathcal{L}(\alpha) \;=\; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 \;+\; \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor, and the regularization coefficient λ also regularizes the predictor

bandwidth ↑ → underfit

regularization coefficient ↑ → underfit

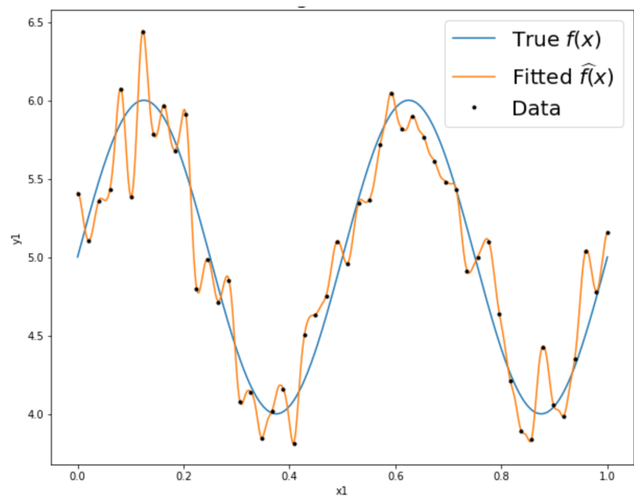
$\sigma = 10^{-3}\;\; \lambda = 10^{-4}$

y



x

$\sigma = 10^{-2}\;\; \lambda = 10^{-4}$



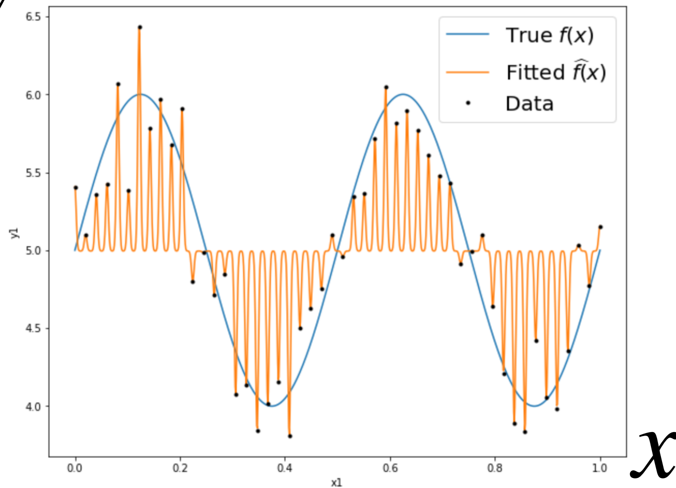
$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathcal{L}(\alpha) = \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor, and the regularization coefficient λ also regularizes the predictor

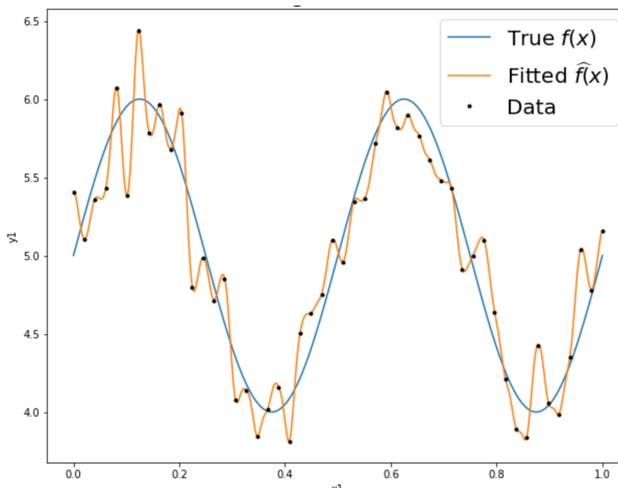
$\sigma = 10^{-3}$ $\lambda = 10^{-4}$

y

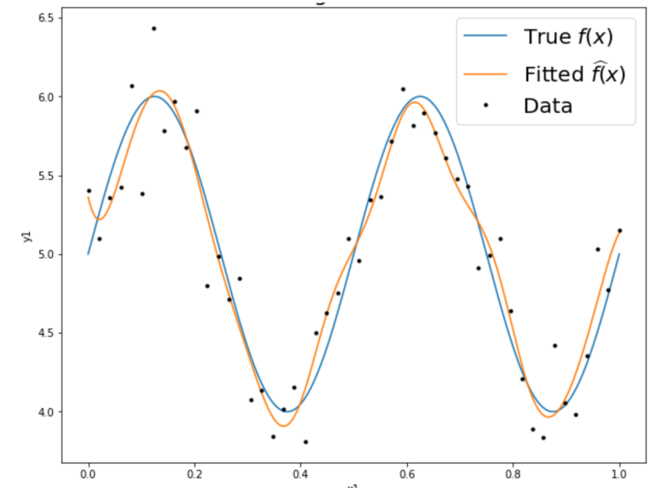


x

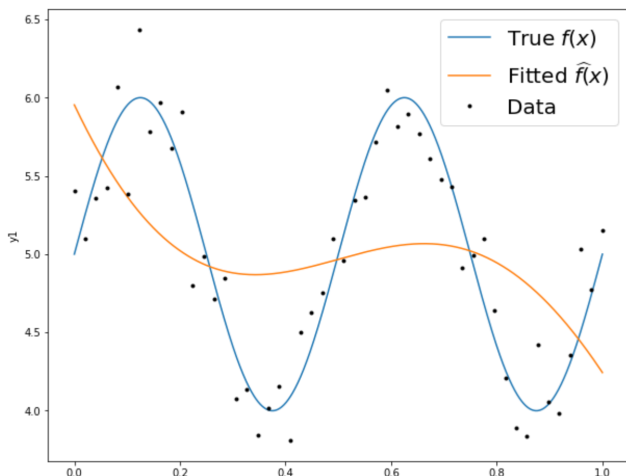
$\sigma = 10^{-2}$ $\lambda = 10^{-4}$



$\sigma = 10^{-1}$ $\lambda = 10^{-4}$



$\sigma = 10^{-0}$ $\lambda = 10^{-4}$



each kernel ↑ Interacts with neighboring points

* How many parameters? n : non-parametric method

$\|$

$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

param $\propto n$.

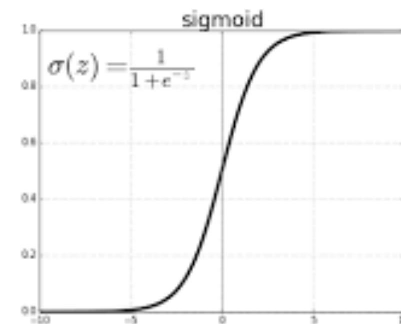
Neural Networks

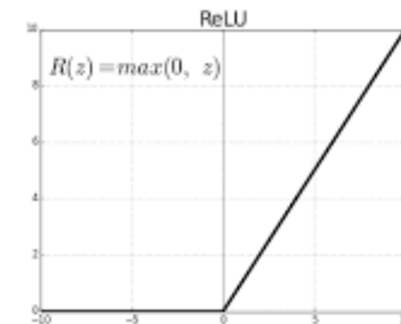
Single node of a neural network

- For a single node with input $x \in \mathbb{R}^d$, the node is defined by
  - **Parameter** $\theta \in \mathbb{R}^{d+1}$ (including the intercept/bias) (offset/y-intercept)

$$x = \begin{bmatrix} x_0 = 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}, \text{ such that } \theta^T x = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

  - **Activation function** $g : \mathbb{R} \to \mathbb{R}$ (non-linear; good for optimization)
    - A common choice is sigmoid function: $g(z) = \dfrac{1}{1 + e^{-z}}$
    - or Rectified Linear Unit (ReLU): $g(z) = \max\{0, z\}$





- The node performs $h_\theta(x) = g\left(\sum_{i=0}^{d} \theta_i x_i\right) = g(\theta^T x) = \dfrac{1}{1 + e^{-\theta^T x}}$

  (output of a single node)

Sequence of operations performed at a single node

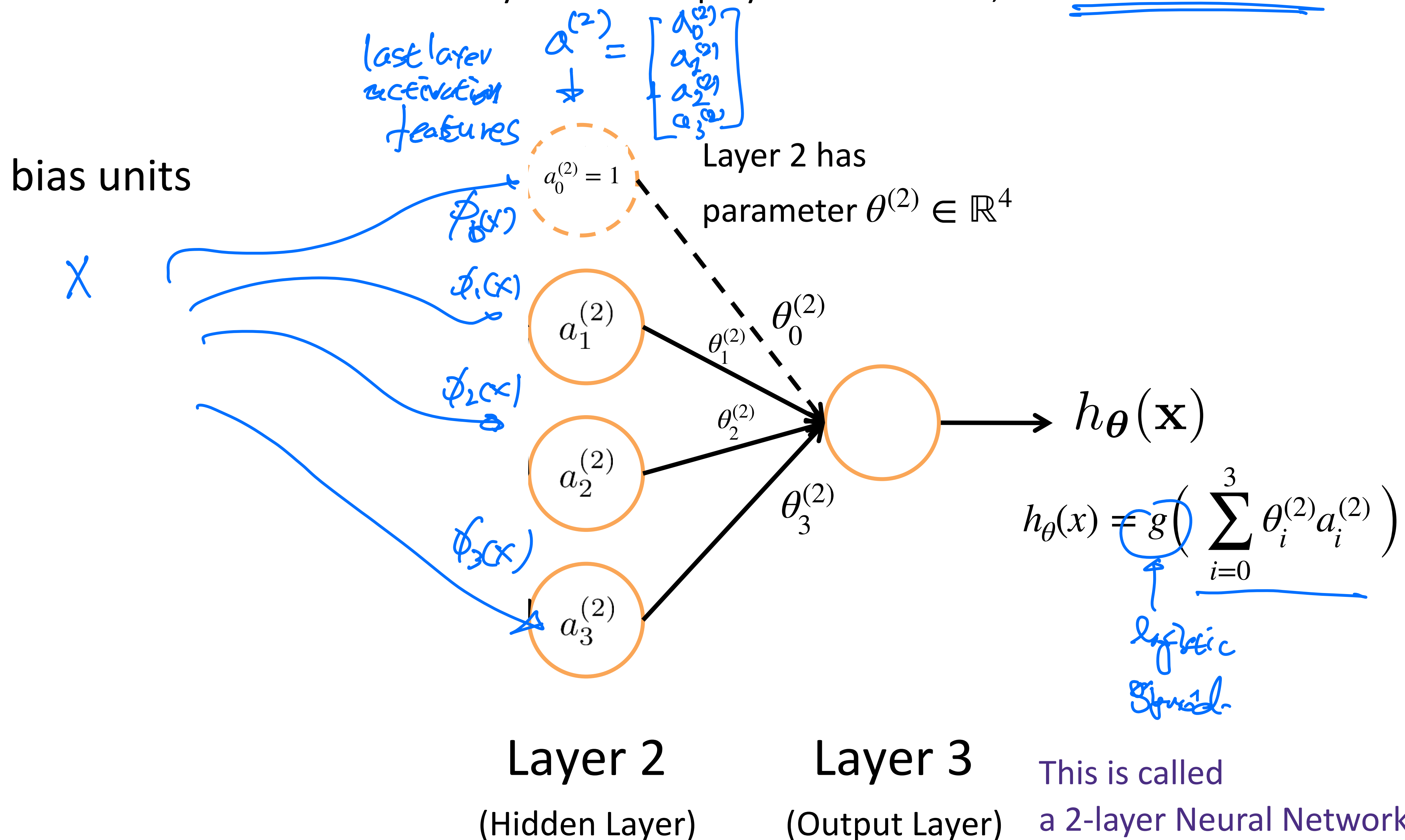
- For a single node with input $x \in \mathbb{R}^d$, the node is defined by
  - **Parameter** $\theta \in \mathbb{R}^{d+1}$ (including the intercept/bias)
  - **Activation function** $g : \mathbb{R} \rightarrow \mathbb{R}$
    - A common choice is sigmoid function: $g(z) = \dfrac{1}{1 + e^{-z}}$
- The node performs $h_\theta(x) = g\left(\sum_{i=0}^{d} \theta_i x_i\right) = g(\theta^T x) = \dfrac{1}{1 + e^{-\theta^T x}}$

$$x = \begin{bmatrix} x_0 = 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Neural Network composes simple functions to make complex functions

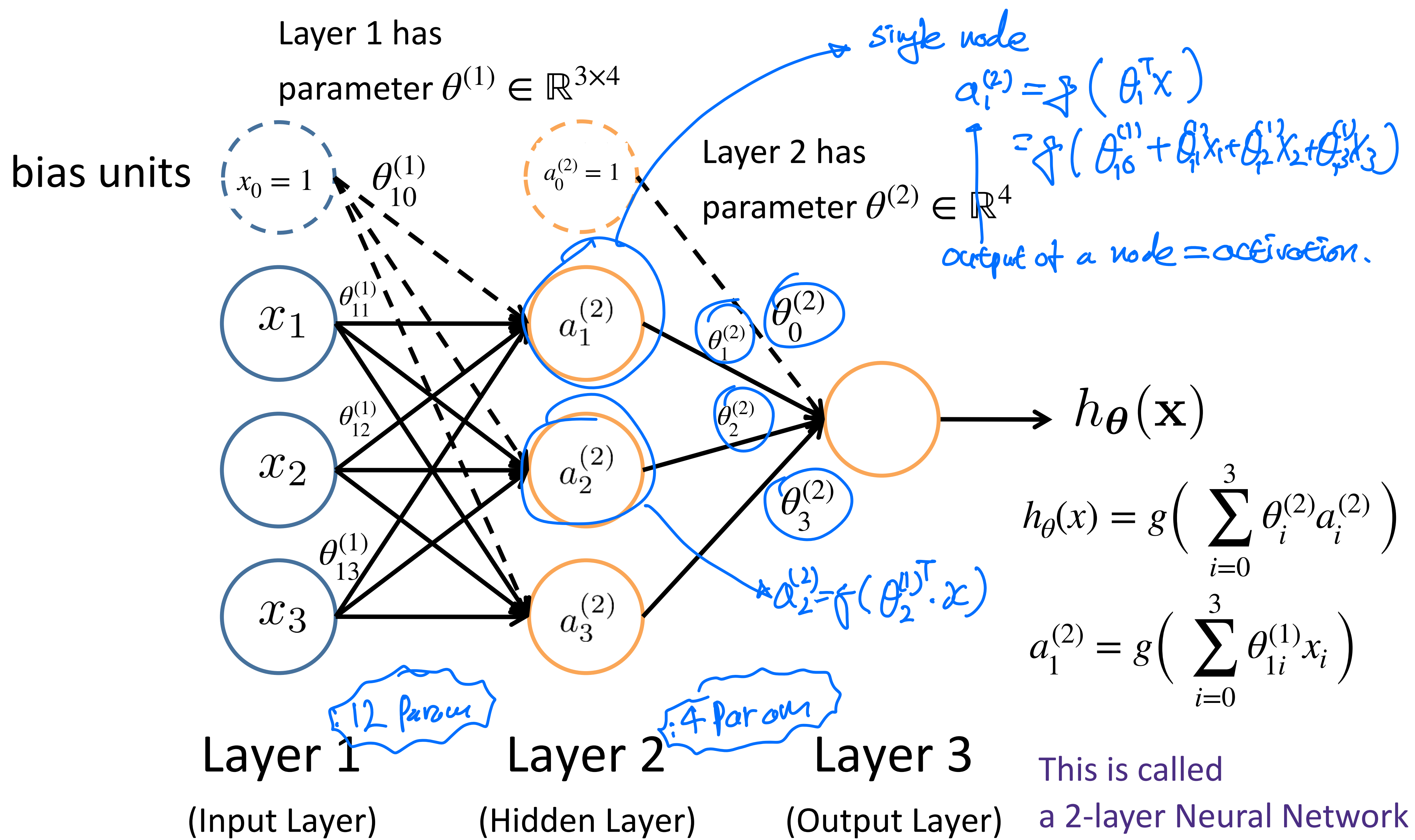
- Neural Network is a simple single node on features, in this example: $(a_0^{(2)}, a_1^{(2)}, a_3^{(2)}, a_3^{(2)})$
- These features are not manually chosen like polynomial features, but learned from data



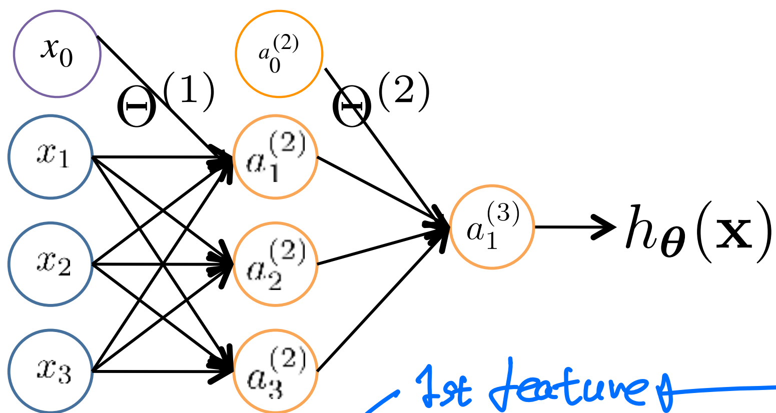
This is called
a 2-layer Neural Network

Neural Network composes simple functions to make complex functions

- Each layer performs simple operations
- Neural Network (with parameter $\theta = (\theta^{(1)}, \theta^{(2)})$) composes two layers



This is called
a 2-layer Neural Network



Deep Neural Network

Deep Learning

$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix stores parameters from layer j to layer $j + 1$

1st feature

Param

spread

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer j+1, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

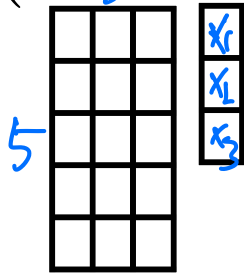
Multi-layer Neural Network - Binary Classification in $\{0,1\}$

L-th layer plays the role of features, but trained instead of pre-determined

This is a 5-dimensional vector

$$a^{(1)} = x$$

$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$



Scalar function g is applied coordinate-wise

⋮

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

⋮

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$

(Learned) feature representation

Logistic regression

$\Theta^{(1)}$ $\Theta^{(2)}$ $\Theta^{(L-1)}$ $\Theta^{(L)}$

$\mathbf{a}^{(1)}$ $\mathbf{a}^{(2)}$ $\mathbf{a}^{(3)}$ $\mathbf{a}^{(4)}$ $\mathbf{a}^{(5)}$

coordinate-wise application of sigmoid

x_1, x_2, x_3

width=5

depth = 4

5×5

1×4

features = last layer activations

Q. logistic Regression: $g(\theta^T x) = \hat{y}$

loss? $(x,y) \to l_\theta(x,y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$ } cross entropy.

Multi-layer Neural Network - Binary Classification in $\{0,1\}$

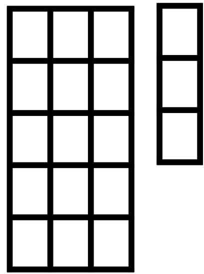
L-th layer plays the role of features, but trained instead of pre-determined

This is a 5-dimensional vector

$$a^{(1)} = x$$

$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

Scalar function g is applied coordinate-wise

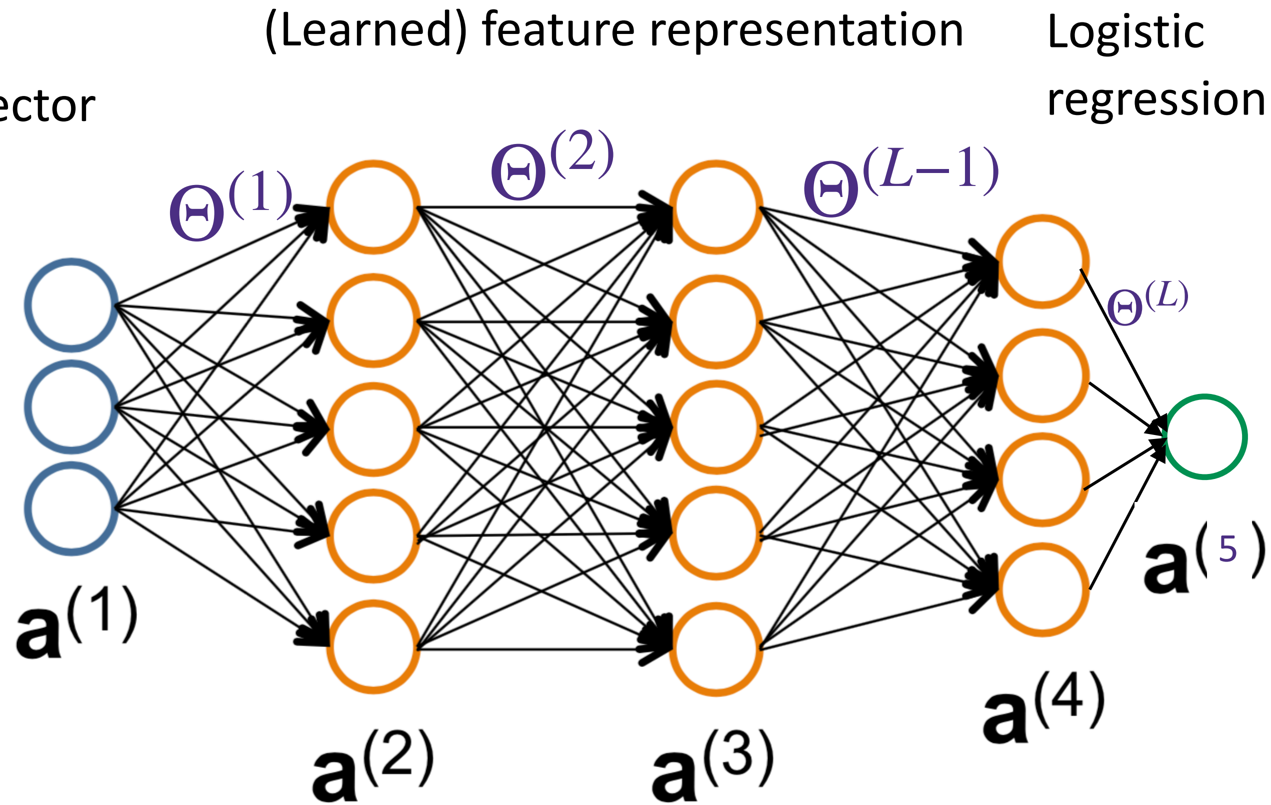


⋮

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

⋮

$$\widehat{y} = g(\Theta^{(L)} a^{(L)})$$



Cross entropy loss:

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Binary Logistic Regression with learned feature $a^{(4)}$

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$

ReLU

$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

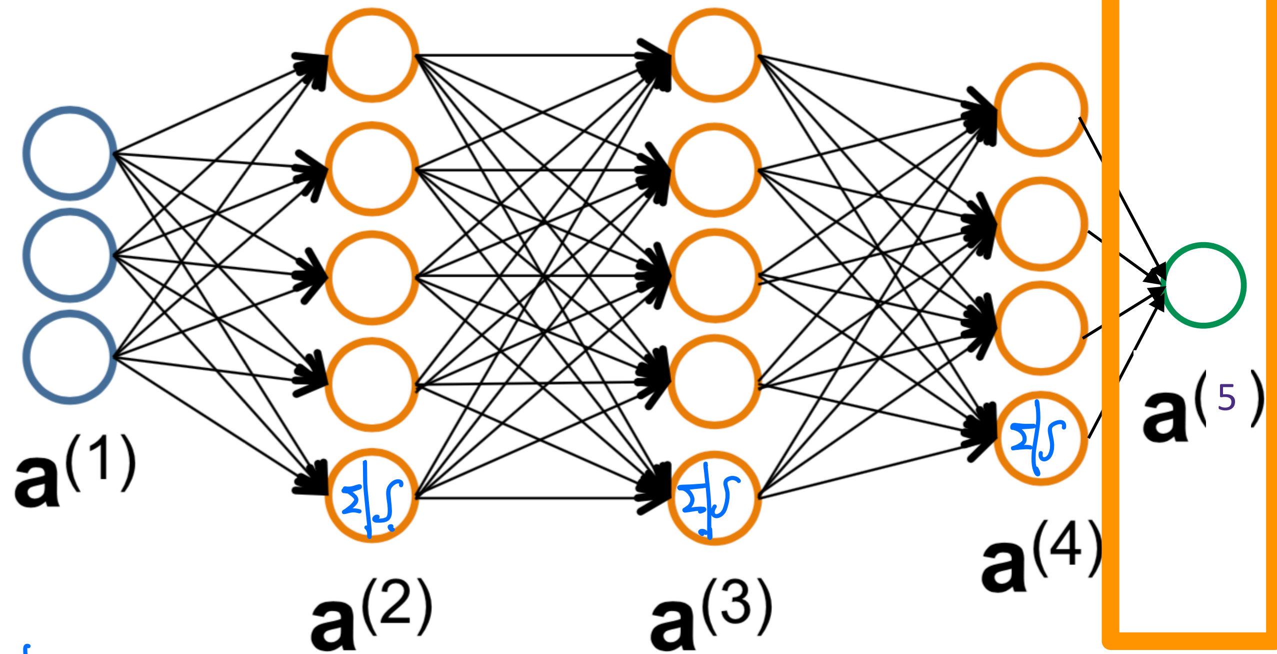
⋮

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

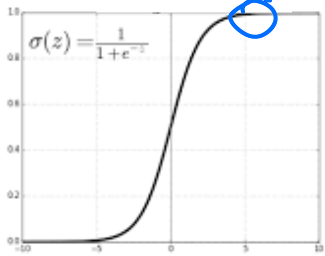
⋮

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$

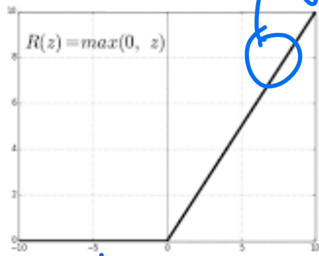
Sigmoid



Sigmoid



ReLU



gradient.

constant gradient

To fight gradient vanishing.

Cross entropy loss:

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$\sigma(z) = \max\{0, z\} = \text{ReLU}(z) \qquad g(z) = \frac{1}{1 + e^{-z}}$$

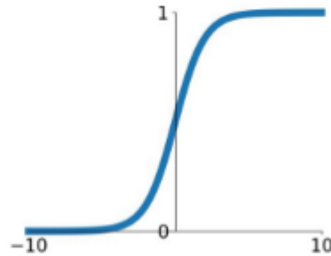
Binary Logistic Regression

Nonlinear activation function

- popular choices of activation function includes

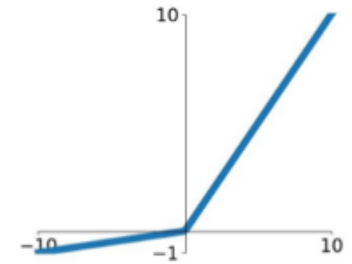
Sigmoid

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



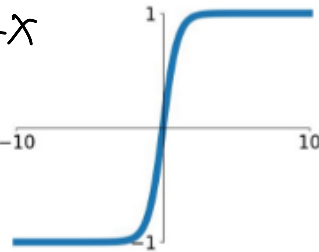
Leaky ReLU

$\max(0.1x, x)$



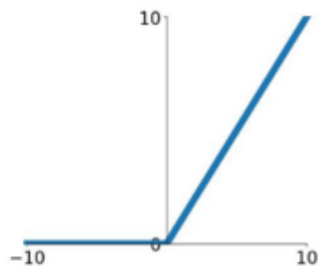
tanh

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



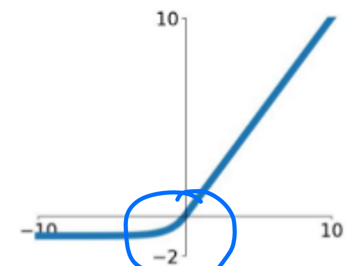
ReLU

$\max(0, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Smooth

- Why is ReLU better than Sigmoid?
- Why is ELU better than ReLU?

{ Convex : global minima / non-convex } { smooth : faster convergence / non-smooth }

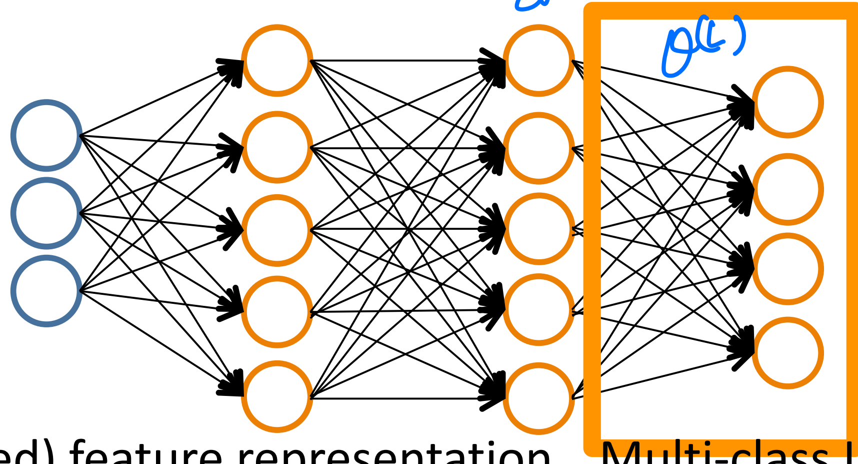
K-class Classification: multiple output units

Pedestrian

Car

Motorcycle

Truck



softmax($\theta^{(L)T} \cdot a^{(L-1)}$)
↑ features.

$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

Multi-class Logistic Regression

(Learned) feature representation   Multi-class Logistic regression

We want: ← one-hot encoding of the classes.

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when pedestrian   when car   when motorcycle   when truck

Multi-layer Neural Network - Regression

$$a^{(1)} = x$$

$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

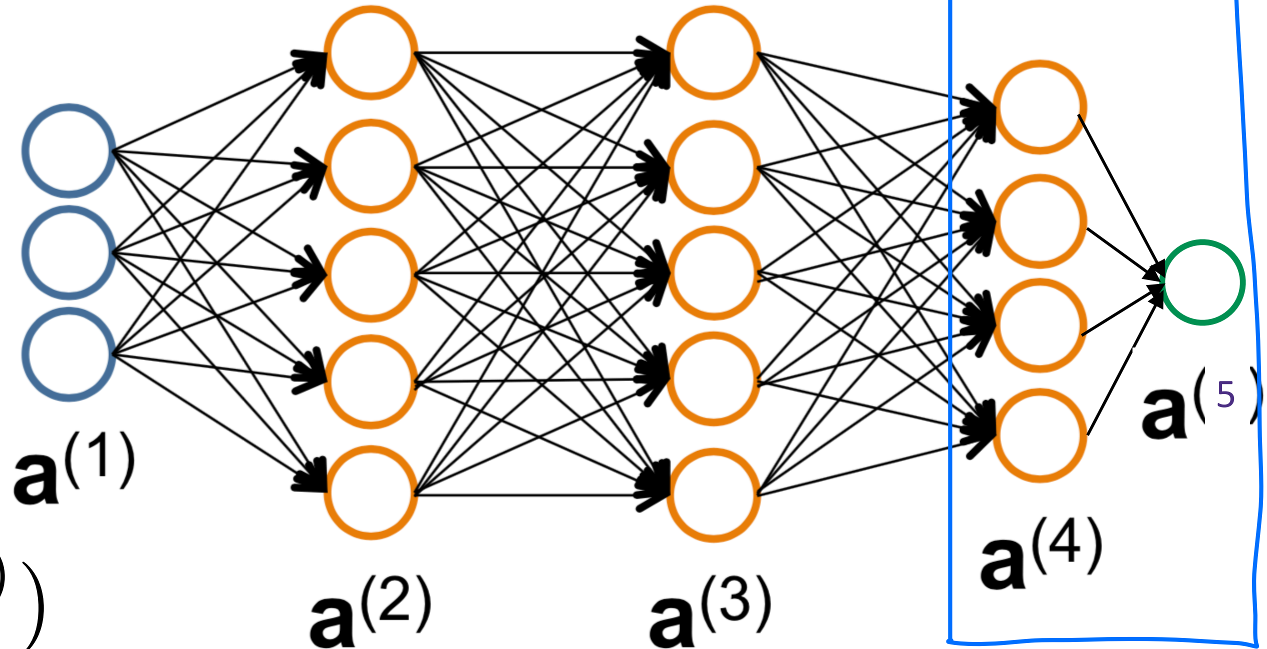
$$\vdots$$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

$$\vdots$$

$$\widehat{y} = \Theta^{(L)} a^{(L)}$$

Linear model



Square loss:

$$L(y, \widehat{y}) = (y - \widehat{y})^2$$

$$\sigma(z) = \max\{0, z\}$$

Training Neural Networks

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

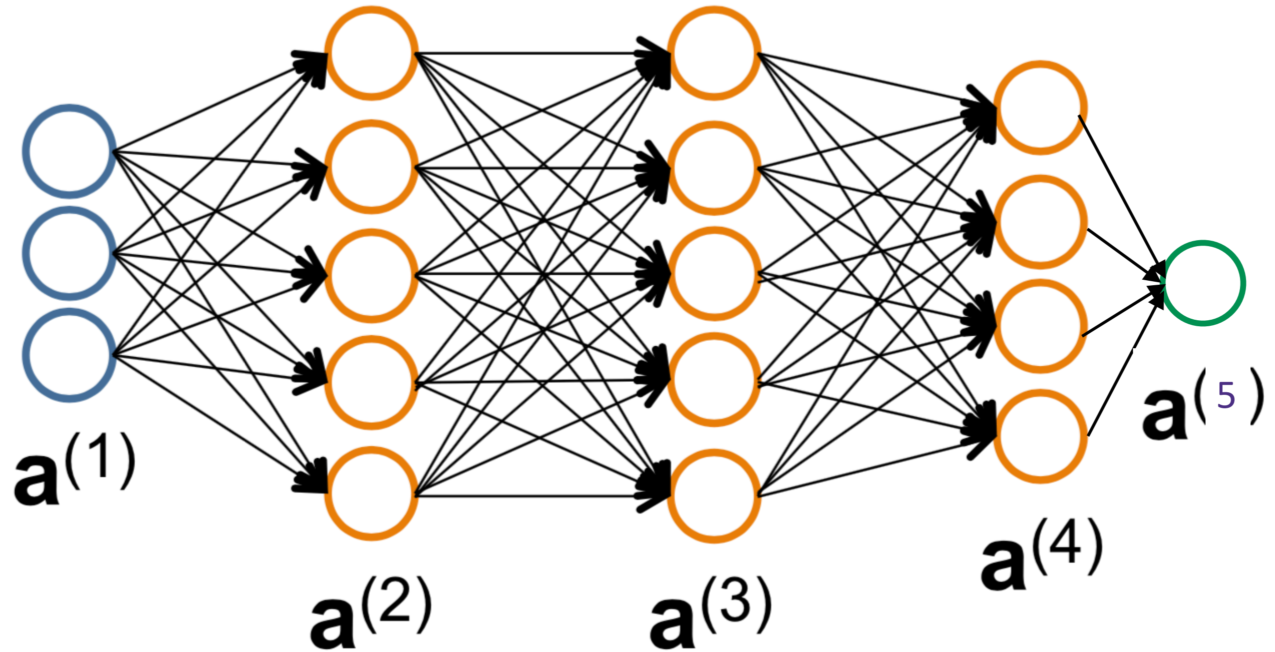
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \hat{y}) \quad \forall l$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \qquad \forall l$

Seems simple enough, what do packages like PyTorch, Tensorflow, Theano, Cafe, MxNet provide?

1. Automatic differentiation
    1. Given a NN, compute the gradient automatically
    2. Compute the gradient efficiently
2. Convenient libraries
    1. Set-up NN
    2. Choose algorithms (SGD,Adam,etc.) for Training
    3. Hyper-parameter Tuning
3. GPU support
    1. Linear algebraic operations

Common training issues

Neural networks are **non-convex**

- For large networks, **gradients** can **blow up** or **go to zero**. This can be helped by **batch-norm** or **ResNet** architecture

- **Stepsize** and **batchsize** have large impact on optimizing the training error *and* generalization performance

- Fancier alternatives to SGD (Adagrad, Adam, LAMB, etc.) can significantly improve training

- Overfitting is common and not undesirable: typical to achieve 100% training accuracy even if test accuracy is just 80%

- Making the network *bigger* may make training *faster!*

Back Propagation

Forward Propagation

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

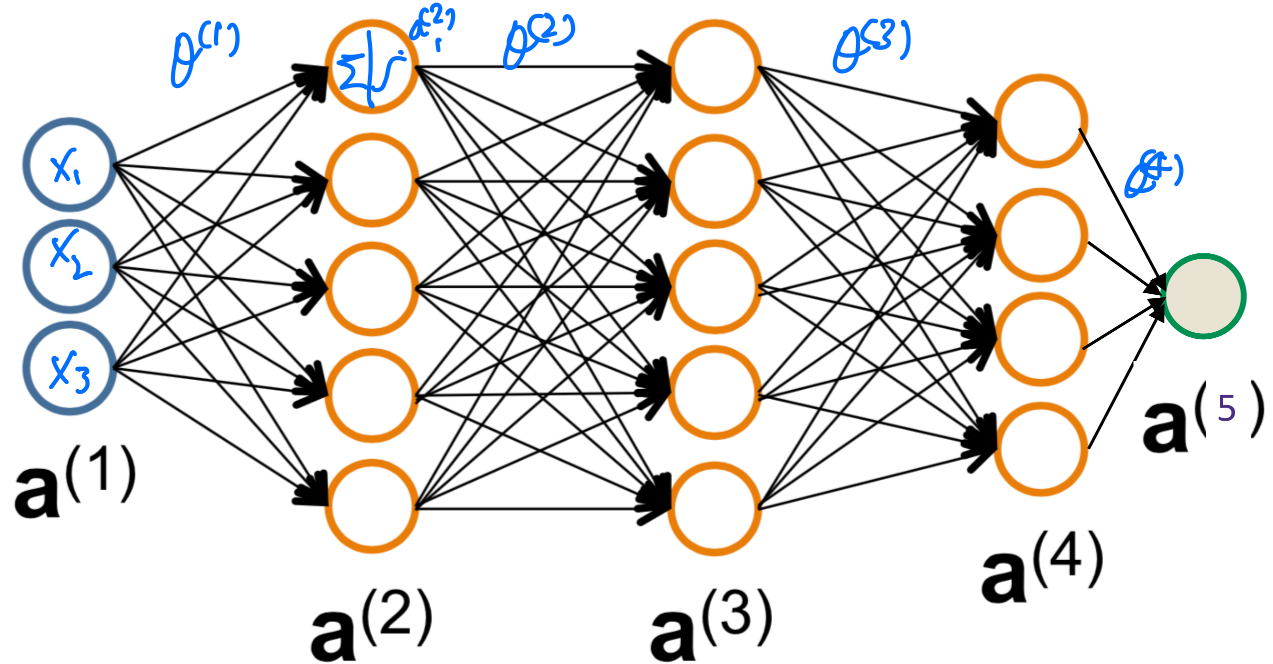
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

- We are not writing the intercept at each layer for simplicity
- To compute gradients, we first run forward pass to get the intermediate representations $\{a^{(2)}, \ldots, a^{(L)}\}$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Backprop

O(L) time to compute gradient for a single layer

- Parameters: $\Theta^{(1)} \in \mathbb{R}^{m\times d}, \Theta^{(2)}, \cdots \Theta^{(L-1)} \in \mathbb{R}^{m\times m}$
- Naive implementation takes $O(L^2)$ time, as each layer requires a full forward pass (with $O(L)$ operations) and some backward pass
- Backprop requires only $O(L)$ operations

Dynamic Programming to compute all gradients, efficiently.

$$a^{(1)} = x \in \mathbb{R}^d$$

$$z^{(2)} = \Theta^{(1)} a^{(1)} \in \mathbb{R}^m$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

Train by Stochastic Gradient Descent:

edge = parameter

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

Recursively computed in one backward pass

Computed in the forward pass

- Chain rule with $z_i^{(\ell+1)} = \Theta_{i,j}^{(\ell)} a_j^{(\ell)}$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$\delta_i^{(l+1)} \triangleq \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

Chain Rule

$$\delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \underbrace{\frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}}}_{\delta_k^{(\ell+1)}} \cdot \underbrace{\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}}_{\Theta_{k,i}^{(l)} \, g'(z_i^{(l)})}$$

$$z_k^{(\ell+1)} = \sum_{i=1}^{m} \Theta_{k,i}^{(l)} \, g(z_i^{(l)})$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \widehat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$= \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \, g'(z_i^{(l)})$$

$$= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

Computed in the forward pass

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$\delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

$$g'(z) = g(z)(1 - g(z))$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y,\widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y,\widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

- We can recursively compute all $\delta^{(\ell)}$'s in a single backward pass
- And compute all gradients via

$$\frac{\partial L(y,\widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y,\widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$L(y,\widehat{y}) = y\log(\widehat{y}) + (1-y)\log(1-\widehat{y})$$

$$g(z) = \frac{1}{1+e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y,\widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$a^{(L+1)} = g(z^{(L+1)})$$
$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}$$

(Last layer)

$$\delta_i^{(L+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[y \log(g(z^{(L+1)})) + (1-y)\log(1 - g(z^{(L+1)})) \right]$$

$$= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1-y}{1 - g(z^{(L+1)})} g'(z^{(L+1)})$$

$$= y - g(z^{(L+1)}) = y - a^{(L+1)}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1-y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

$$g'(z) = g(z)(1 - g(z))$$

Backprop

Recursive Algorithm!

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\delta^{(L+1)} = y - a^{(L+1)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backpropagation

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance (x_k, y_k) :

Set $\mathbf{a}^{(1)} = x_k$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Intercept do not have regularizer

Average loss + ℓ_2 regularizer

$$\frac{1}{n}\sum_{k=1}^{n} L(y_k, \hat{y}) + \lambda\|\Theta\|_2^2$$

Questions?

[Homework 3 Problem B1 on Perceptron]
[Homework 3 Problem A4 on PyTorch on simple NNs]
[Homework 3 Problem A5 on Simple NN with MNIST]
[Homework 4 Problem A3 on CNN and CIFAR]