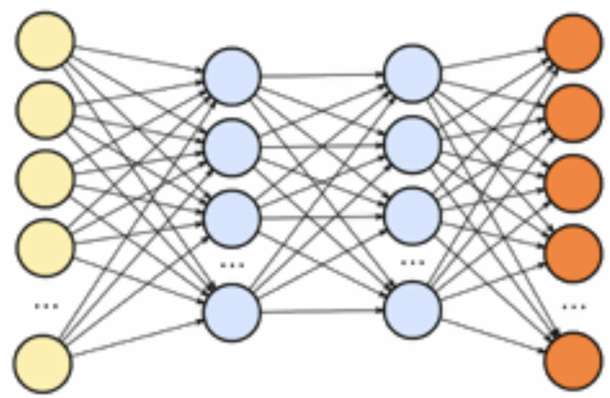
Deep Learning

CSE 543/599I

Simon Du



CSE543/599I: Deep Learning

Instructor: Simon Du

Teaching Assistant: Prashant Ranagarajan, Yuhao Wang

Course Website (contains all logistic information): https://courses.cs.washington.edu/courses/cse543/22sp/

Piazza: https://piazza.com/washington/spring2022/cse543/home

Announcements: Canvas

Homework: Canvas

CSE543/599I: Deep Learning

What this class is:

- **Fundamentals of DL:** Neural network architecture, approximation properties, optimization, architecture, generalization, generative models, representation learning
- **Preparation for further learning / research:** the field is fast-moving, you will be able to apply the fundamentals and teach yourself the latest

What this class is not:

- **An easy course:** mathematically easy
- **A survey course:** laundry list of algorithms
- **An application course**: implementation of different architectures on different datasets

Prerequisites

- Working knowledge of:
  - Linear algebra
  - Vector calculus
  - Probability and statistics
  - Algorithms
  - Machine leanring (CSE 446/546)
- Mathematical maturity
- "Can I learn these topics concurrently?"

Lecture

- Time: Tuesday and Thursday 9:00 - 10:20AM
- CSE2 G10
- Slides + handwritten notes (e.g., proofs)
- Please ask questions
- Some lectures will be on Zoom
- Recording on Canvas
- Tentative schedule on course website

Homework (40%)

- 2 homework (20%+20%)
  - Each contains both theoretical questions and will have programming
  - Related to course materials
  - Collaboration okay but must write who you collaborated with. You must write, submit, and understand your answers and code
  - Submit on Canvas
  - Must be **typed**
  - **Two** late days
  - Tentative timeline:
    - HW 1 due: 4/22
    - HW 2 due: 5/6

Course Project (60%)

- Group of 1 - 2.
- Topic: literature review (state-of-the-art) or original research.
- Some potential topics are in listed on Canvas. OK to do a project on listed.
- You can work on a project related to your research.
- Proposal (due: 4/8): **5%**
  - Format: NeurIPS Latex format, ~1 - 1.5 pages
- Presentations on (5/31 and 6/2 on Zoom): **20%**
- Final report (due: 6/10): **35%**
  - Format: NeurIPS Latex format, ~8 pages
- Submit on Canvas

Possible Topics

- Approximation properties
- Advanced optimization methods
- Optimization theory for deep learning
- Generalization theory for deep learning
- Deep reinforcement learning
- Implicit regularization
- Meta-learning algorithm / theory
- Robustness
- Lottery ticket hypothesis
- Deep learning application
- …

Communication Chanels

- **Announcements**
  - Canvas
- **questions about class, homework help**
  - Piazza
  - Office hours:
    - Simon Du: Tu 10:30 - 11:30 AM (in person Gates 312 and Zoom)
    - Prashant Ranagarajan
    - Yuhao Wan
- **Regrade requests / Personal concerns:**
  - Email to instructor or TAs

Addcodes

- Email: Elle Brown (elleab@cs.washington.edu) for addcodes

Topic 1: Review (Today)

- ML Review: training, generalization
- Neural network basics: fully-connected neural network, gradient descent

Topic 2: Approximation Theory

- Why neural networks can express the (regression, classification, …) function you want?
- Construction of such desired neural networks
- Universal approximation theorem

Topic 3: Optimization

- Review: Back-propagation
- Auto-differentiation
- Advanced optimizers: momentum (Nesterov acceleration), adaptive method (AdaGrad, Adam)
- Techniques for improving optimization
- Theory: global convergence of gradient of over-parameterized neural networks
- Neural Tangent Kernel

Topic 4: Architecture

- Convolutional neural network
- Recurrent neural network
- Attention-based neural network
- General framework

Topic 5: Generalization

- Measures of generalization
- Double descent
- Techniques for improving generalization
- Generalization theory beyond VC-dimension
- Implicit regularization

Topic 6: Unsupervised learning

- Explicit models
- Generative adversarial network
- Sampling

Topic 7: Representation Learning

- Transfer learning
- Domain adaptation
- Meta-learning
- Theory

NETFLIX
98% Match 2017 TV

Spotify
Discover Weekly
FOLLOWING

amazon prime

ML uses past data to make predictions

Supervised Learning Process

Collect a **dataset**

$\{(x_i, y_i)\}_{i=1}^{n} \overset{i.i.d.}{\sim} D$

x_i : input $\in \mathbb{R}^d$, image,

$y_i \in \{0, \dots, k\}$ classification

$y_i \in \mathbb{R}$ regression

Decide on a **model**

$f: \mathbb{R}^d \to \mathbb{R}$

$f \in \mathcal{F}$: function class

(1) linear

(2) kernel

(3) tree

(3) NN

Find the function which fits the data best

Choose a loss function $\ell(f(x), y) \to \mathbb{R}$

$(f(x) - y)^2$

logistic

Pick the function which minimizes loss on data

$$\hat{f} \leftarrow \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \lambda R(f)$$

$\lambda \in \mathbb{R}^+$

f: linear θ

$\|\theta\|_2^2$

Use function to make prediction on new examples

x_{new}

prediction: $\hat{f}(x_{new}) \approx y_{new}$

Framework

Fix $f \in \mathcal{F}$

Goal: Test Error

$$L_{te}(f) = \mathbb{E}_{(x,y)\sim D}\left[\ell(f(x),y)\right]$$

$$L_{tr}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

$$L_{te}(f) = L_{tr}(f) + L_{te}(f) - L_{tr}(f)$$

$$= \min_{\tilde{f} \in \mathcal{F}} L_{tr}(\tilde{f})$$ approximation error

$$+ L_{tr}(f) - \min_{\tilde{f} \in \mathcal{F}} L_{tr}(\tilde{f})$$ opt error

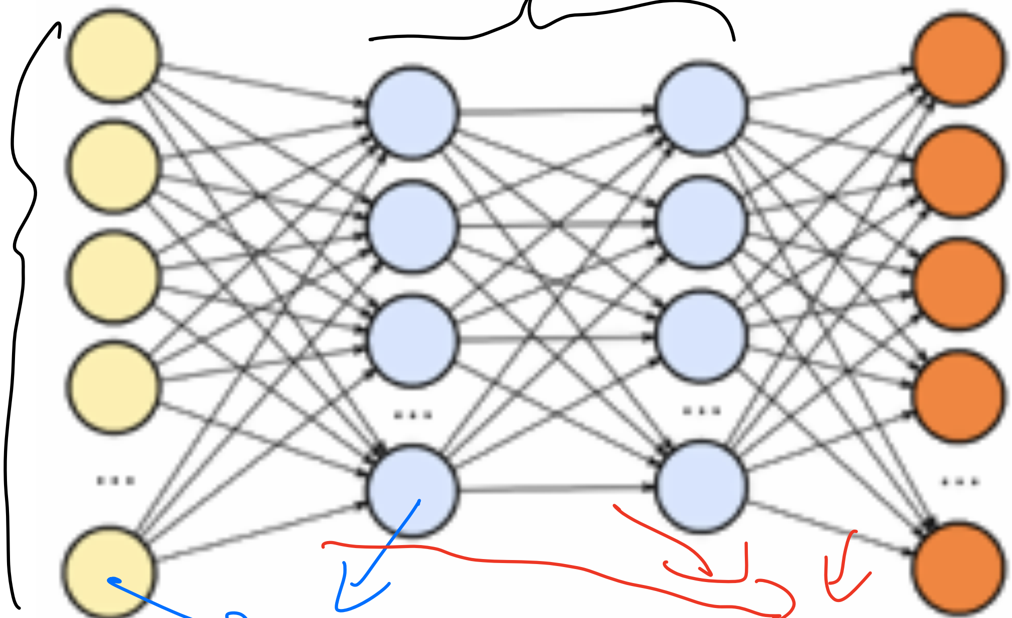
$$+ L_{te}(f) - L_{tr}(f)$$ generalization error

Neural Networks

intermediate layers

input

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



output

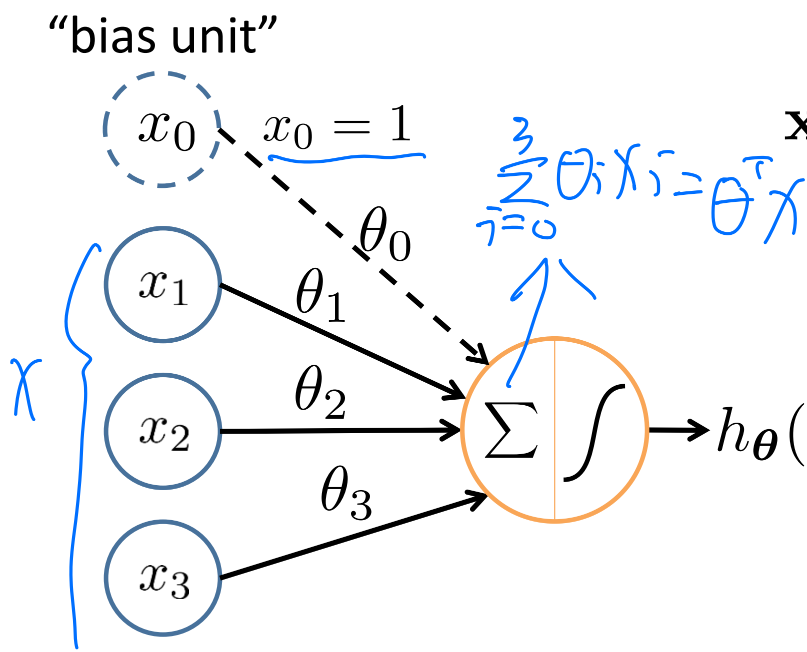
node/neuron/unit

link:

each node:
1) input
2) activation function
3) output

link:
- maps output of neurons to the input of neurons in the next layer
- each link has weight $\in \mathbb{R}$

Single Node



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

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

Binary Logistic Regression

$\mathbb{R} \to \mathbb{R}$

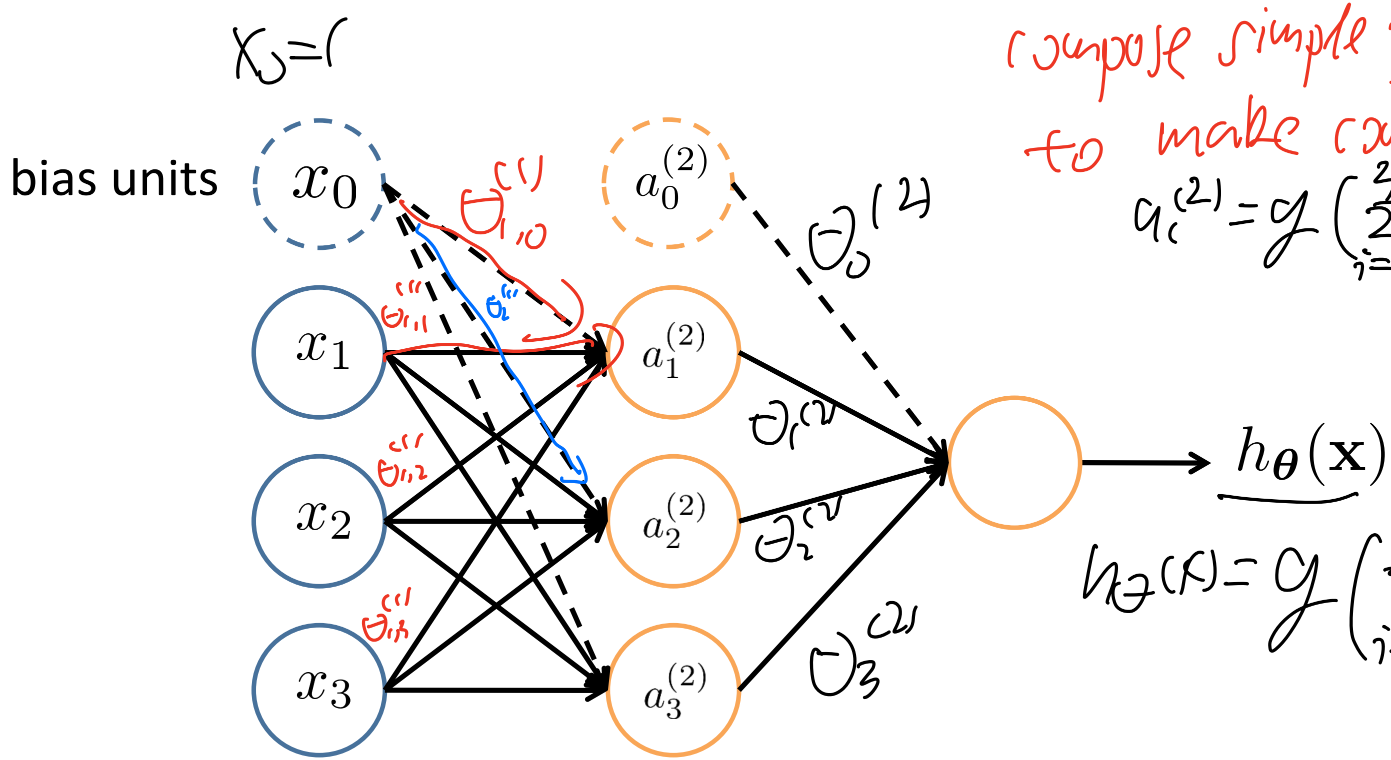
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

other: ReLU

Neural Network

Key idea: compose simple func to make complex func

$a_1^{(2)} = g\left(\sum_{i=0}^{3} \Theta_{1,i}^{(1)} x_i\right)$

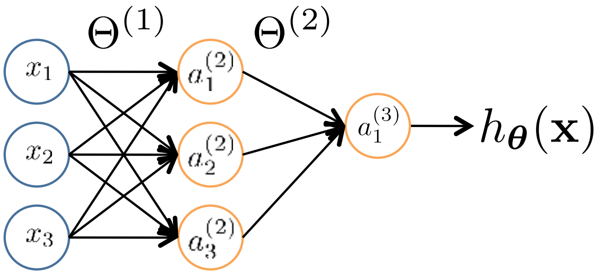


$h_\theta(x) = g\left(\sum_{i=0}^{3} \Theta_i^{(2)} a_i^{(2)}\right)$

Layer 1
(Input Layer)

Layer 2
(Hidden Layer)

Layer 3
(Output Layer)



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

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

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

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

pointwise

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

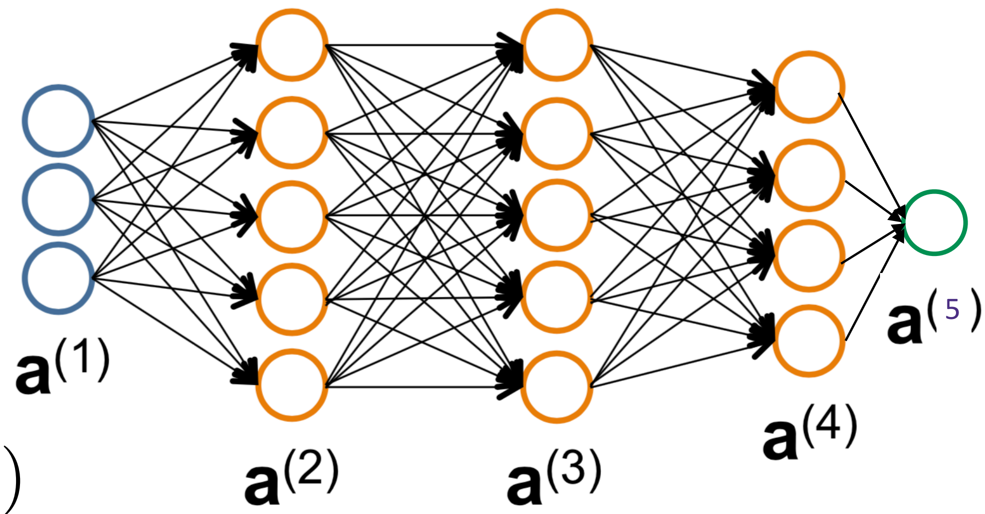
$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

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

$$\vdots$$

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



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

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

Binary Logistic Regression

Multi-layer Neural Network - Binary Classification

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

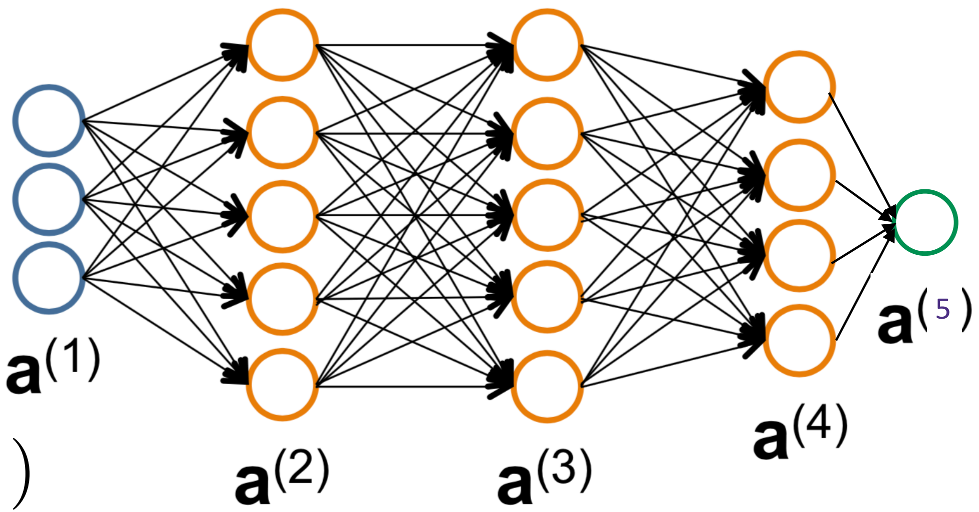
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

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

$$\vdots$$

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



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

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

Binary
Logistic
Regression

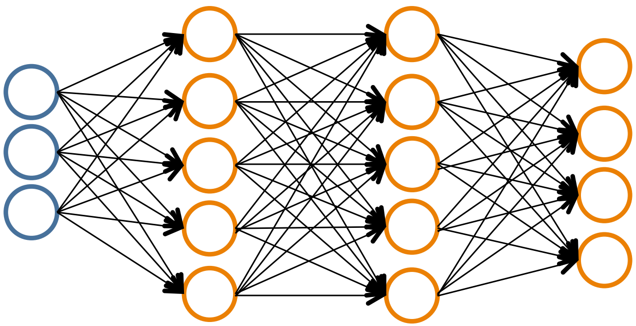
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



cross-entropy

$\ell(h_\theta(x), y) = \sum_{k=1}^{K} -\log [h_\theta(x)]_k \cdot y_k$

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

Multi-class
Logistic
Regression

We want:

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

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

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

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

Multi-layer Neural Network - Regression

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

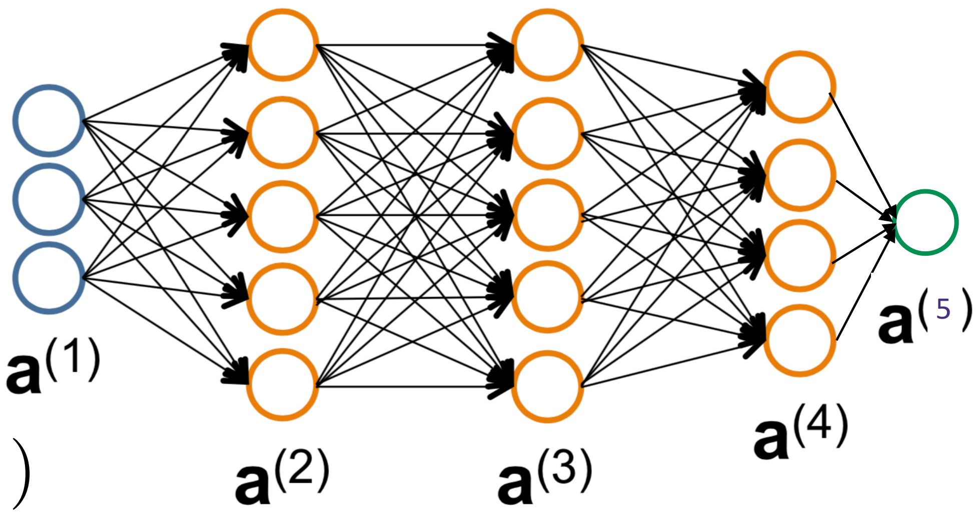
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$\vdots$

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

$\vdots$

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



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

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

Regression

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

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

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

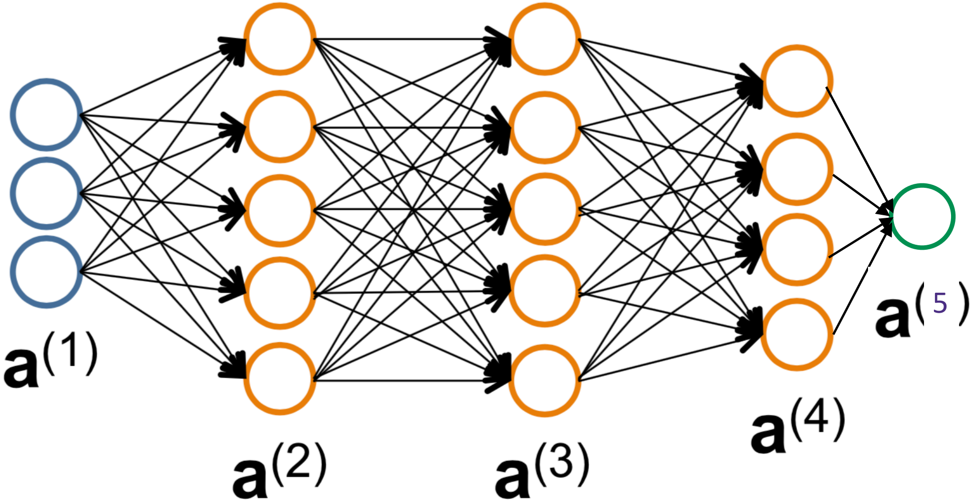
$$\vdots$$

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

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

$$\vdots$$

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



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

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

η : step size, learning rate

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

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

Seems simple enough, why are packages like PyTorch, Tensorflow, Theano, Cafe, MxNet synonymous with deep learning?

1. Automatic differentiation
2. Convenient libraries
   ① set up NN
   ② training
3. GPU support
   ① linear algebra operations
   ② pointwise operations

Gradient Descent:

Seems simple enough,
Theano, Cafe, MxNet s

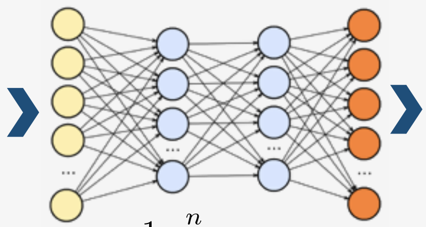
1. Automatic differ

2. Convenient libra

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

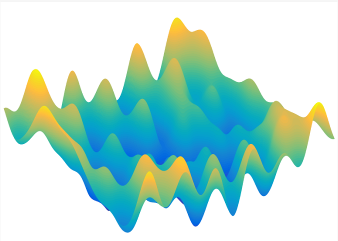
```
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

Optimization Error: Theory and Practice

x  $f(\theta, x)$

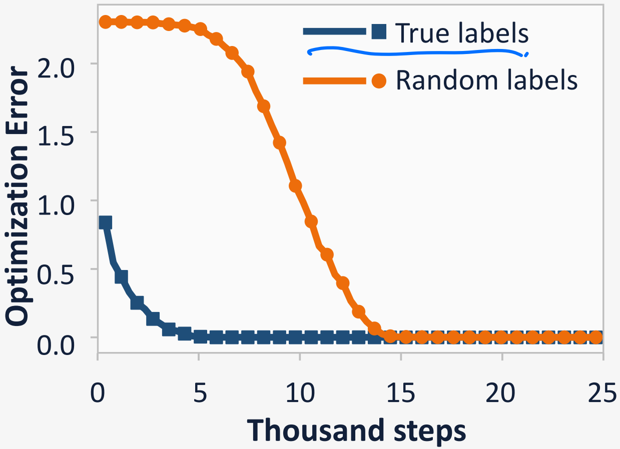
$$L(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(\theta, x_i) - y_i)^2$$

Theory: Non-convex. NP-hard [Blum and Rivest 88]



Practice: gradient descent

$$\theta(t+1) \leftarrow \theta(t) - \eta \frac{\partial L(\theta(t))}{\partial \theta(t)}$$



Optimization error -> **0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017
Understanding DL Requires Rethinking Generalization

Over-parameterization

CIFAR - 10	n: 50K
Inception	1.6M
Alexnet	1.4M
MP 1x512	1.2M
ImageNet	**n: 1.2M**
Inception V4	43M
Alexnet	61M
Resnet-152	60M
VGG-19	143M
AmoebaNet	600M

Why large neural NN has 0 error?

Q: Why there exists such an NN ?

approx

Q: Why does **gradient descent** find such a neural network?

opt

Over-parameterization => Overfit?

Generalization Error Bound:

$$\text{Generalization Error} \leq \sqrt{\frac{\text{Complexity Measure}}{n}}$$

Complexity Measure: # of parameters

Over-parameterization: # of parameters >> n