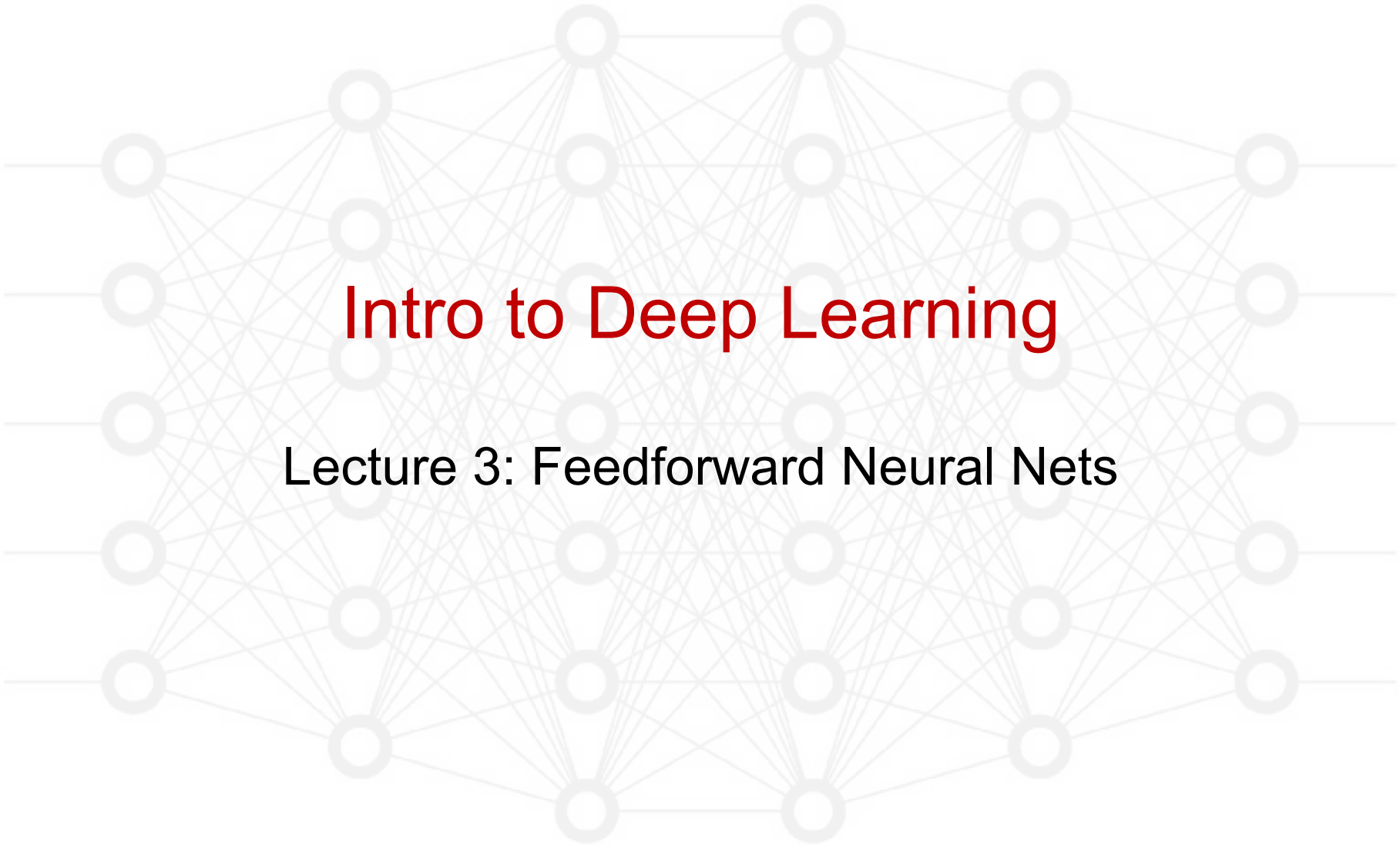


Intro to Deep Learning

Lecture 3: Feedforward Neural Nets

Schedule

- Midterm:
  - Time: February 9th, 9:30 — 10:50 am
  - Location: Materials Sci and Engineering Room 103 (in class)
- Proposal Due: February 13th
  - No late submission

EE/CS 228: Introduction to Deep Learning

Winter 2026

Lecture Times: 9:30 am – 10:50 am on Mondays and Wednesdays
Location: Materials Science and Engineering Room 103

Instructor: Yinglun Zhu
Contact Info: yzhu@ucr.edu
Office hour: By appointment

TA: Bowen Zuo (bzuo002@ucr.edu)
Office Hours:

4:00 pm – 5:00 pm on Thursdays via Zoom
1:00 pm – 2:00 pm on Fridays via Zoom
Zoom link: https://ucr.zoom.us/j/8090428745

Linear regression

1. Data: $\left(x_i, y_i\right)_{i=1}^n$, $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$
2. Model: Linear model $f(x; w) = x^\top w$
3. Squared Loss: $\ell(f(x), y) = \frac{1}{2}\left(f(x) - y\right)^2$
4. Optimize

$$\widehat{w} = \arg\min_{w \in \mathbb{R}^d} \widehat{\mathscr{L}}(w) \quad \text{where} \quad \widehat{\mathscr{L}}(w) = \frac{1}{2n} \sum_{i=1}^n (x_i^\top w - y_i)^2$$

Linear regression

- Let us move to matrix notation

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \quad n \times d \text{ matrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \quad n \times 1 \text{ vector}$$

- The loss function becomes

$$\widehat{\mathscr{L}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (x_i^\top w - y_i)^2 = \frac{1}{2n} \left\| y - Xw \right\|_2^2$$

Linear regression

Optimization becomes

$$w^\star = \arg\min_{w \in \mathbb{R}^d} \frac{1}{2} \left\| y - Xw \right\|_2^2$$

Solution can be found by differentiation.

Convex analysis

- Set the gradient to zero: $\nabla \mathscr{L}(w) = 0$

$$\nabla \mathscr{L}(w) = X^\top X w - X^\top y$$

- Solution $w^\star$ satisfies: $X^\top X w^\star = X^\top y$

The normal equation

One vs All updates

Further algorithmic consideration

- Using full gradient (all n examples)

$$w_{t+1} = w_t - \eta\left(X^\top X w_t - X^\top y\right)$$

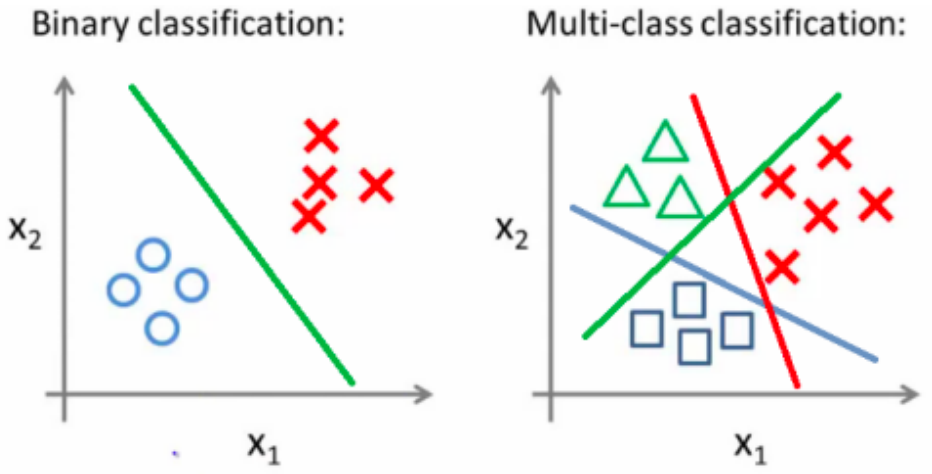
- Using one example at a time
  - for $1 \le t \le n$

$$w_{t+1} = w_t - \eta x_t\left(x_t^\top w_t - y_t\right)$$

Relates to online/streaming data and stochastic gradient descent

Multiclass Classification

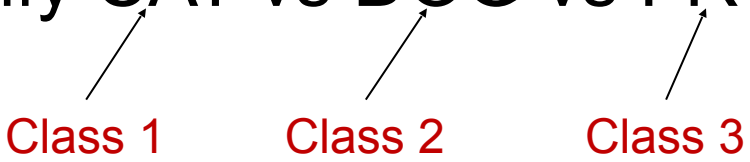
- So far, we assumed label is a scalar
- Modern ML models can classify 100s of objects



- How do we handle multiple classes?

One-hot encoding

- For a K-class problem, use K dimensional labels
- Ex: Classify CAT vs DOG vs FROG.

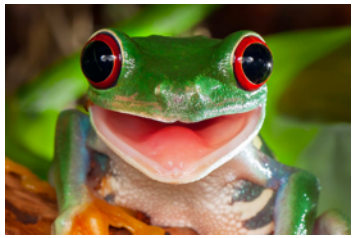


  - CAT receives label $[1\,0\,0]$
  - DOG receives label $[0\,1\,0]$
  - FROG receives label $[0\,0\,1]$

Example Dataset

Inputs x_i

Labels y_i



$[0\,0\,1]$

$[0\,1\,0]$

$[1\,0\,0]$

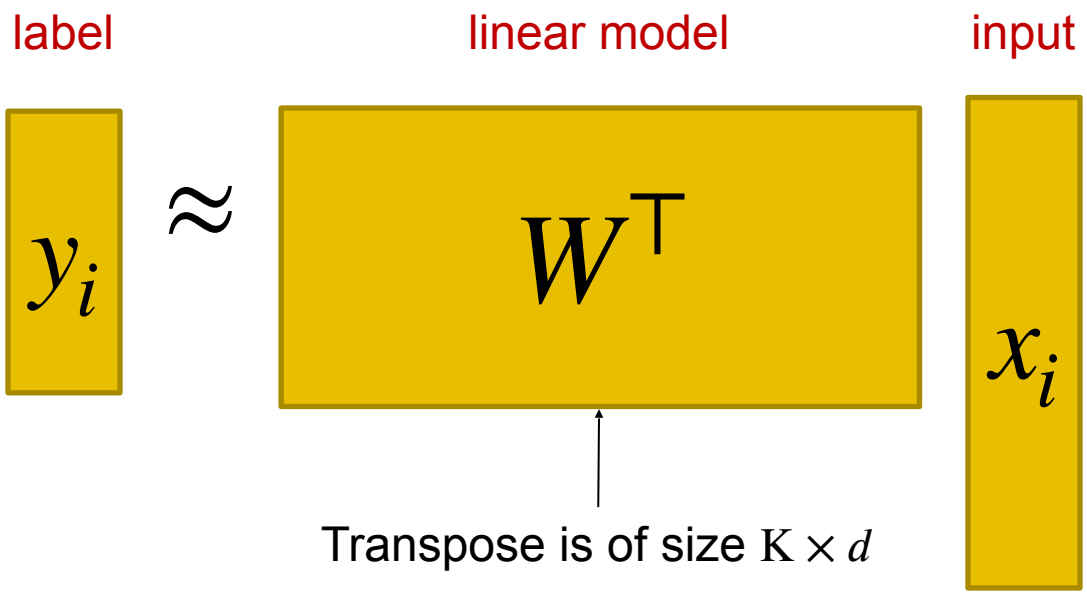
$[0\,1\,0]$

Multiclass Classifier

- Suppose $x \in \mathbb{R}^d$, $y \in \mathbb{R}^K$
- Need a ML model $f : \mathbb{R}^d \to \mathbb{R}^K$
- What does this mean for linear model?

Multiclass Linear Model

- Multiclass linear model
  - a matrix W rather than vector w
  - dimensions $d \times K$



Multiclass Optimization

- Back to linear regression with squared-loss

$$\widehat{\mathscr{L}}(W) = \frac{1}{2}\sum_{i=1}^{n} \left\| y_i - W^\top x_i \right\|_2^2$$

The L_2 distance between label vector and prediction

Multiclass Optimization

- Back to linear regression with squared-loss

$$\widehat{\mathscr{L}}(W) = \frac{1}{2}\sum_{i=1}^{n} \left\| y_i - W^\top x_i \right\|_2^2$$

$$= \frac{1}{2}\left\| Y - XW \right\|_F^2$$

$$\left\| Z \right\|_F^2 = \sum_{i,j} Z_{i,j}^2$$

Multiclass Optimization

- Back to linear regression with squared-loss

$$\widehat{\mathscr{L}}(W) = \frac{1}{2}\sum_{i=1}^{n} \left\| y_i - W^\top x_i \right\|_2^2 = \frac{1}{2}\left\| Y - XW \right\|_F^2$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \quad n \times d \text{ matrix}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \quad n \times K \text{ matrix}$$

Optimizing with Gradient Descent

- Gradient is given by

$$\nabla \mathcal{L}(W) = X^\top XW - X^\top Y$$

- Plugging in the gradient formula

$$W_{t+1} = W_t - \eta\left(X^\top XW - X^\top Y\right)$$

- Computational cost of each iteration

$$O(K \times nd)$$

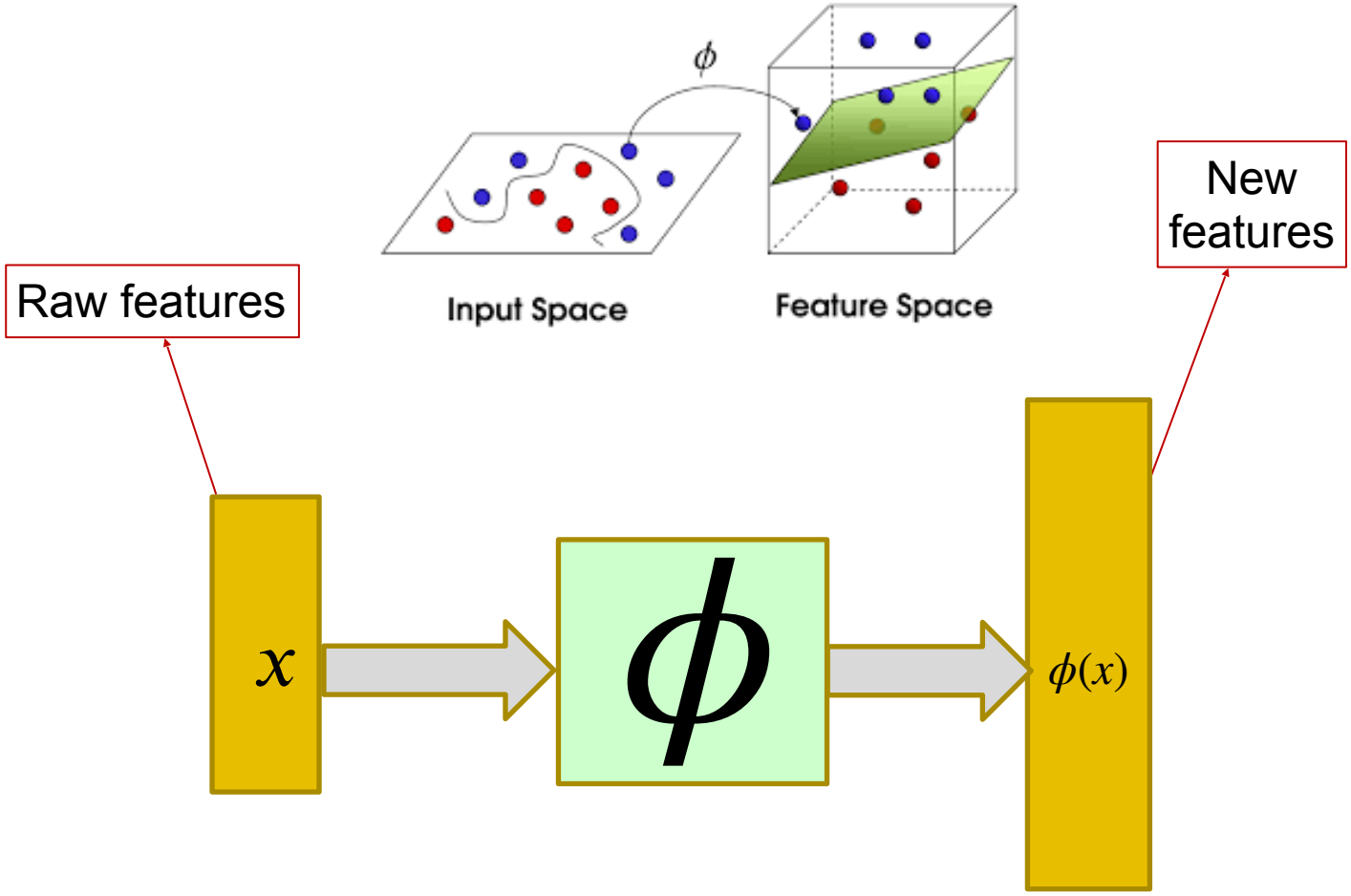
- The equations remain (almost) the same! 😀

- We have talked about linear models.

- What about models whose output is non-linear with respect to the raw features?

- We have talked about linear models in previous lectures.

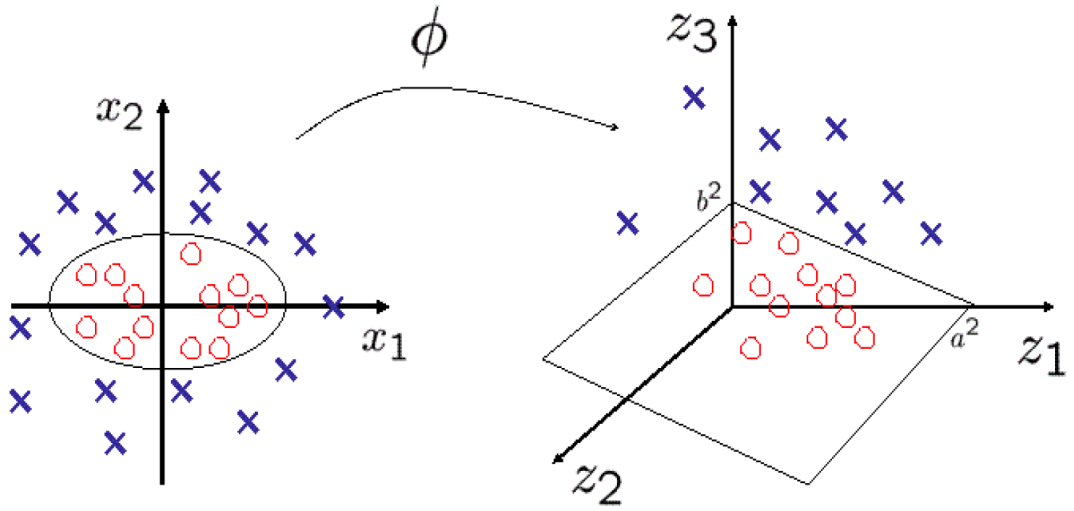
- What about models whose output is non-linear with respect to the raw features?



ϕ is some kind of non-linear transformation, which can be either pre-fixed or learned.

Features: part of the model

Nonlinear model

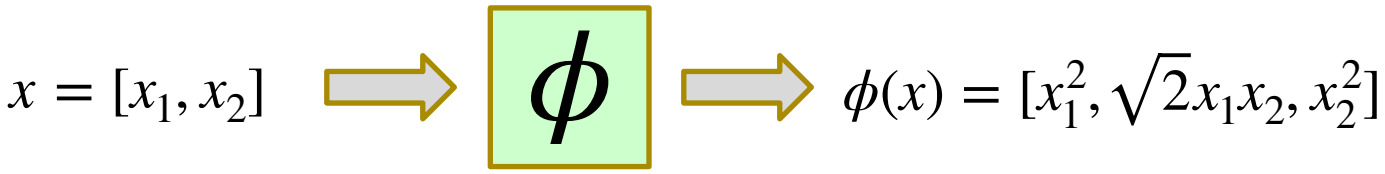


build hypothesis

$y = w^T \phi(x)$

Linear model

- Example for pre-fixed ϕ: Kernel methods



Linearity is preserved in ϕ

- Example for learned ϕ: Neural networks

No need to select ϕ ahead of time

Regression with Non-linear Mapping

1. Data: $\left(x_i, y_i\right)_{i=1}^n$, $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$
2. Non-linear mapping $\phi : \mathbb{R}^d \to \mathbb{R}^D$
3. Model: $f(x; w) = \phi(x)^\top w, w \in \mathbb{R}^D$
4. Squared Loss: $\ell(f(x), y) = \frac{1}{2}\left(f(x) - y\right)^2$
5. Optimize

$$\widehat{w} = \arg\min_{w \in \mathbb{R}^d} \widehat{\mathscr{L}}(w) \quad \text{where} \quad \widehat{\mathscr{L}}(w) = \frac{1}{2}\sum_{i=1}^n (\phi(x_i)^\top w - y_i)^2$$

Regression with Non-linear Mapping

- Let us move to matrix notation

$$\Phi = \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \phi(x_3) \\ \vdots \\ \phi(x_n) \end{bmatrix} \quad n \times D \text{ matrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \quad n \times 1 \text{ vector}$$

- The loss function becomes

$$\widehat{\mathscr{L}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (\phi(x_i)^\top w - y_i)^2 = \frac{1}{2n} \left\| y - \Phi w \right\|_2^2$$

Regression with Non-linear Mapping

- Recall that we can write

  $\widehat{w} = \Phi^{\dagger} y$, where $\Phi^{\dagger}$ is the pseudo-inverse

- Assume $D > n$ and rows of Φ are linearly independent:

$$\widehat{w} = \Phi^{\dagger} y = \Phi^{\top}\left(\Phi\Phi^{\top}\right)^{-1} y$$

If assumptions are don't hold, add regularization!

Regression with Non-linear Mapping

- Assume $D > n$ and rows of Φ are linearly independent:

$$\widehat{w} = \Phi^{\dagger} y = \Phi^{\top}\left(\Phi\Phi^{\top}\right)^{-1} y$$

$O(n^3 + nD)$

- Future prediction:

$$\hat{f}(x) = \phi(x)^{\top}\widehat{w} = \phi(x)^{\top}\Phi^{\top}\left(\Phi\Phi^{\top}\right)^{-1} y$$

$O(D)$

What if D is very large?

Kernel Trick

- Fortunately, it's usually easy to compute the kernel function $k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

$$k(u, v) = \big(\langle u, v \rangle\big)^2$$

- Suppose $\phi(u) = [u_1^2, \sqrt{2}u_1u_2, u_2^2]$:

$$\begin{aligned} \overset{O(D)}{\langle \phi(u), \phi(v) \rangle} &= u_1^2v_1^2 + 2u_1u_2v_1v_2 + v_1^2v_2^2 \\ &= \big(u_1v_1 + u_2v_2\big)^2 \\ &= \big(\langle u, v \rangle\big)^2 \qquad O(d) \\ &= k(u, v) \end{aligned}$$

Kernel Trick

- Denote $\alpha = \left(\Phi\Phi^\top\right)^{-1} y \in \mathbb{R}^n$:

$$\begin{aligned}
\hat{f}(x) &= \phi(x)^\top \Phi^\top \left(\Phi\Phi^\top\right)^{-1} y \\
&= \phi(x)^\top \Phi^\top \alpha \\
&= \sum_{i=1}^{n} \alpha_i \langle \phi(x), \phi(x_i) \rangle \\
&= \sum_{i=1}^{n} \alpha_i \, k(x, x_i)
\end{aligned}$$

no dependence on $O(D)$

$$\sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_j, x_i))^2$$

Popular Kernels

- Polynomial of degree q:

$$k(u,v) = \big(\langle u,v\rangle\big)^q \qquad D = O(d^q)$$

- Polynomial of degree up to q:

$$k(u,v) = \big(\langle u,v\rangle + 1\big)^q \qquad D = O(d^q)$$

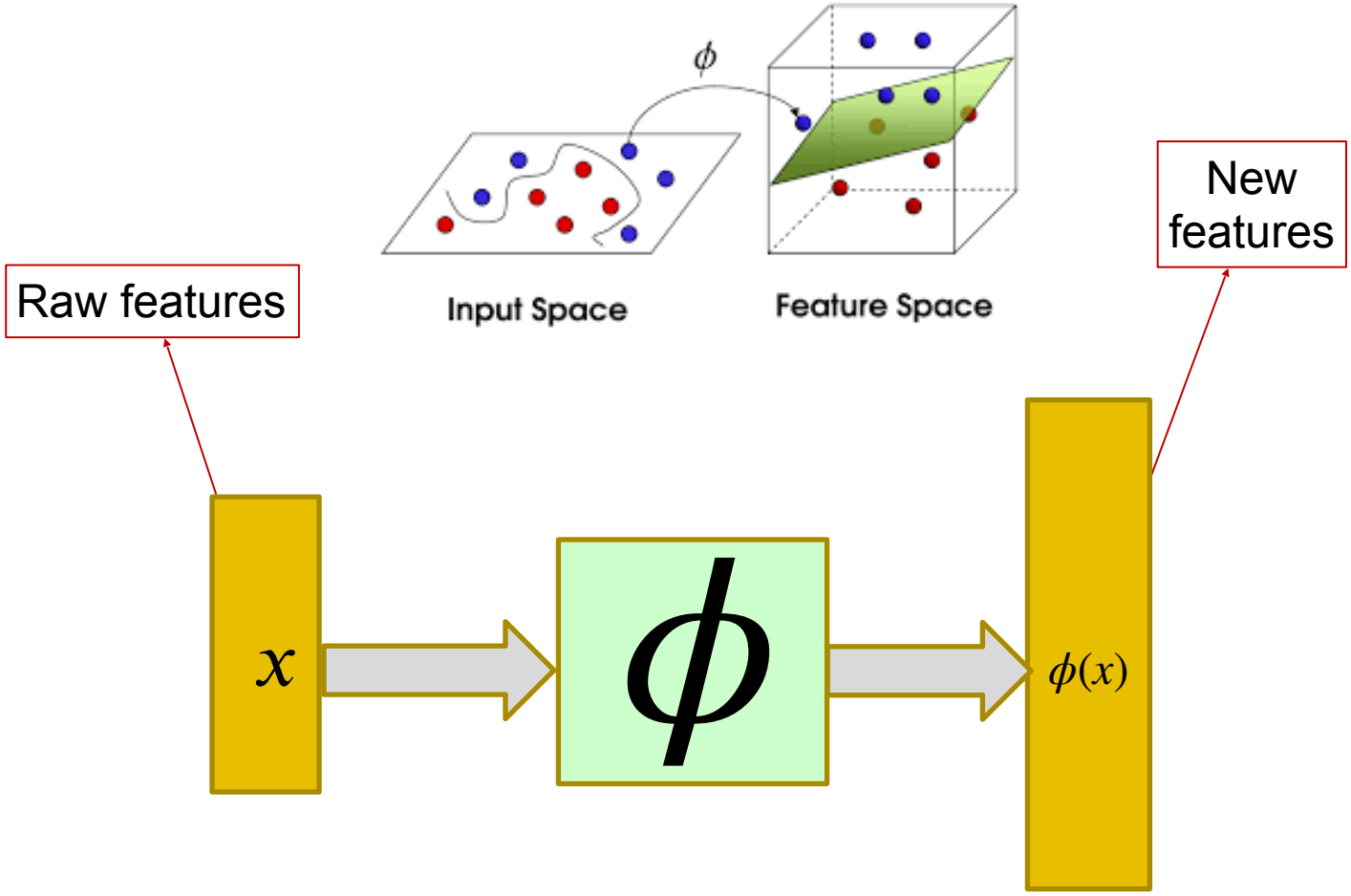
- Gaussian kernel (with parameter σ):

$$k(u,v) = \exp\left(-\frac{\|u-v\|_2^2}{2\sigma^2}\right)$$

$D = \infty$

No closed-form for ϕ

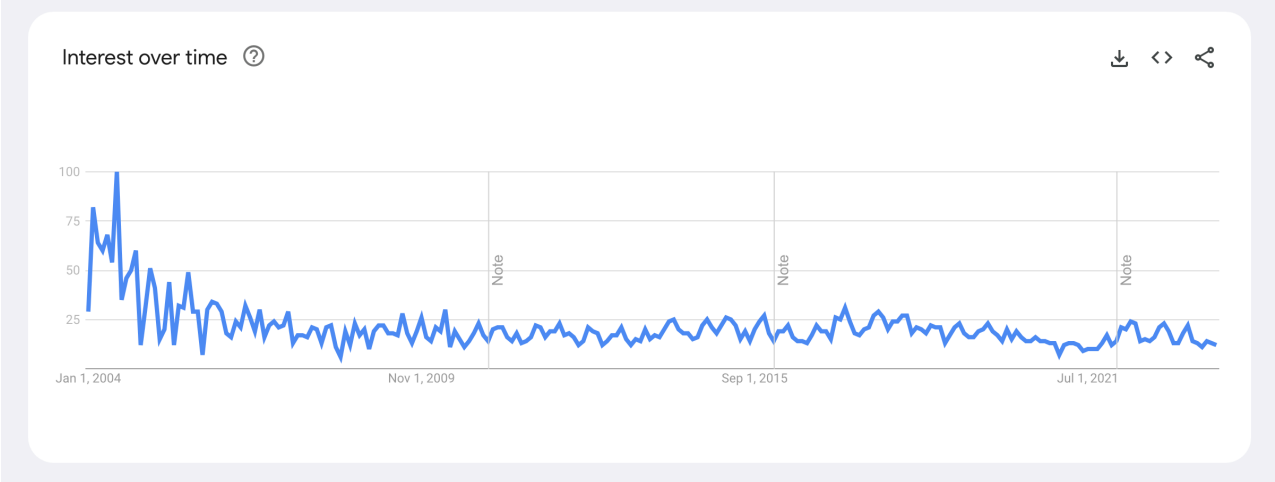
Break

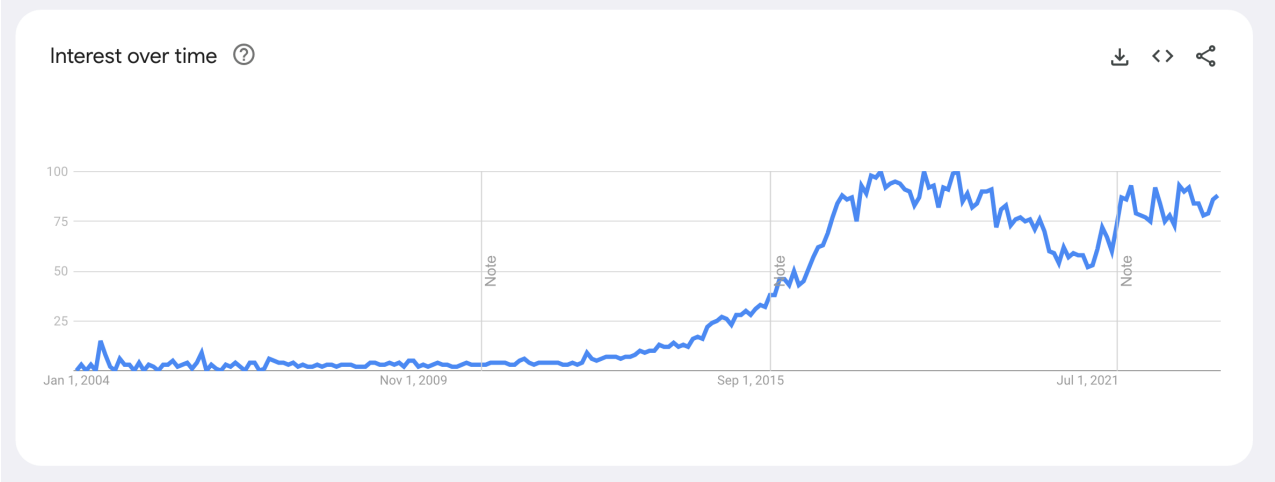


ϕ is some kind of non-linear transformation, which can be either pre-fixed or learned.

Comparison of Two Approaches

Google trend: support vector machine

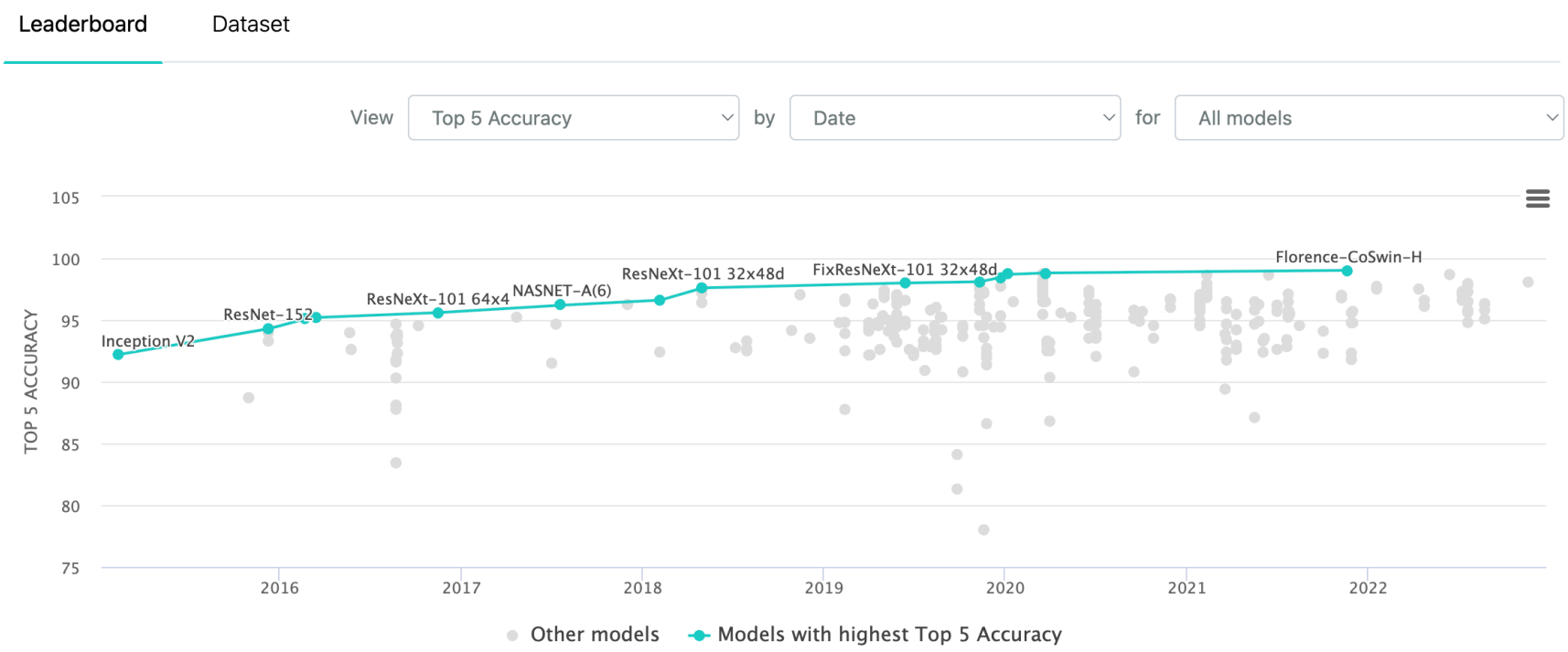




Google trend: deep learning

Comparison of Two Approaches

Image Classification on ImageNet



All presented results are obtained with learned ϕ

This Lecture: Representation Learning

- Why don't we also learn $\phi(x)$?

x

Learn $\phi(x)$

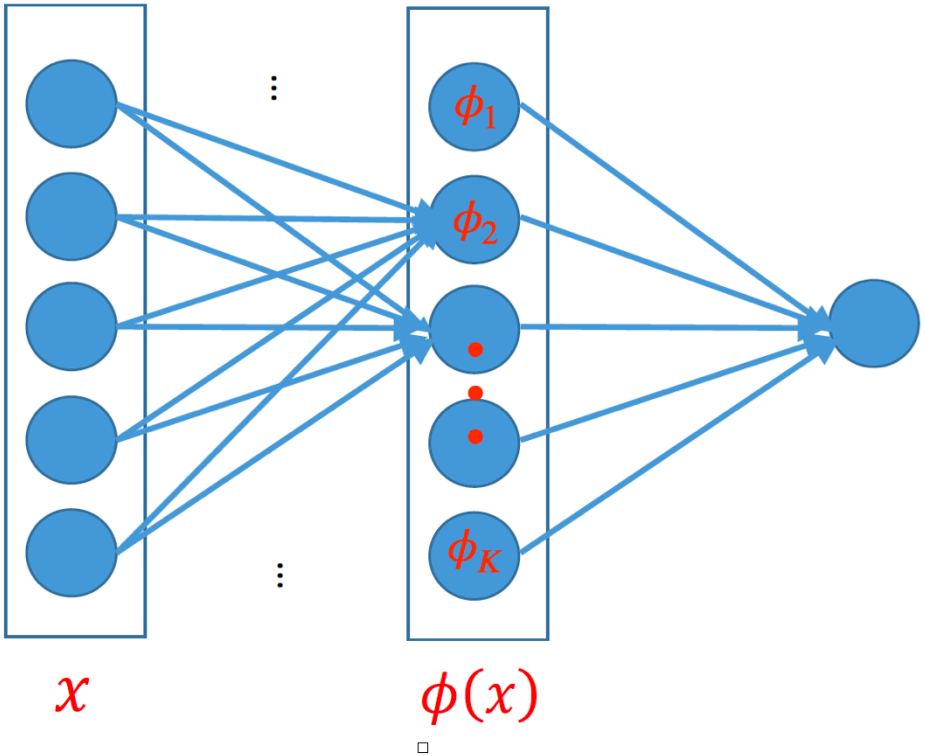
$\phi(x)$

Learn w

$y = w^T \phi(x)$

We will learn ϕ via feedforward neural networks.

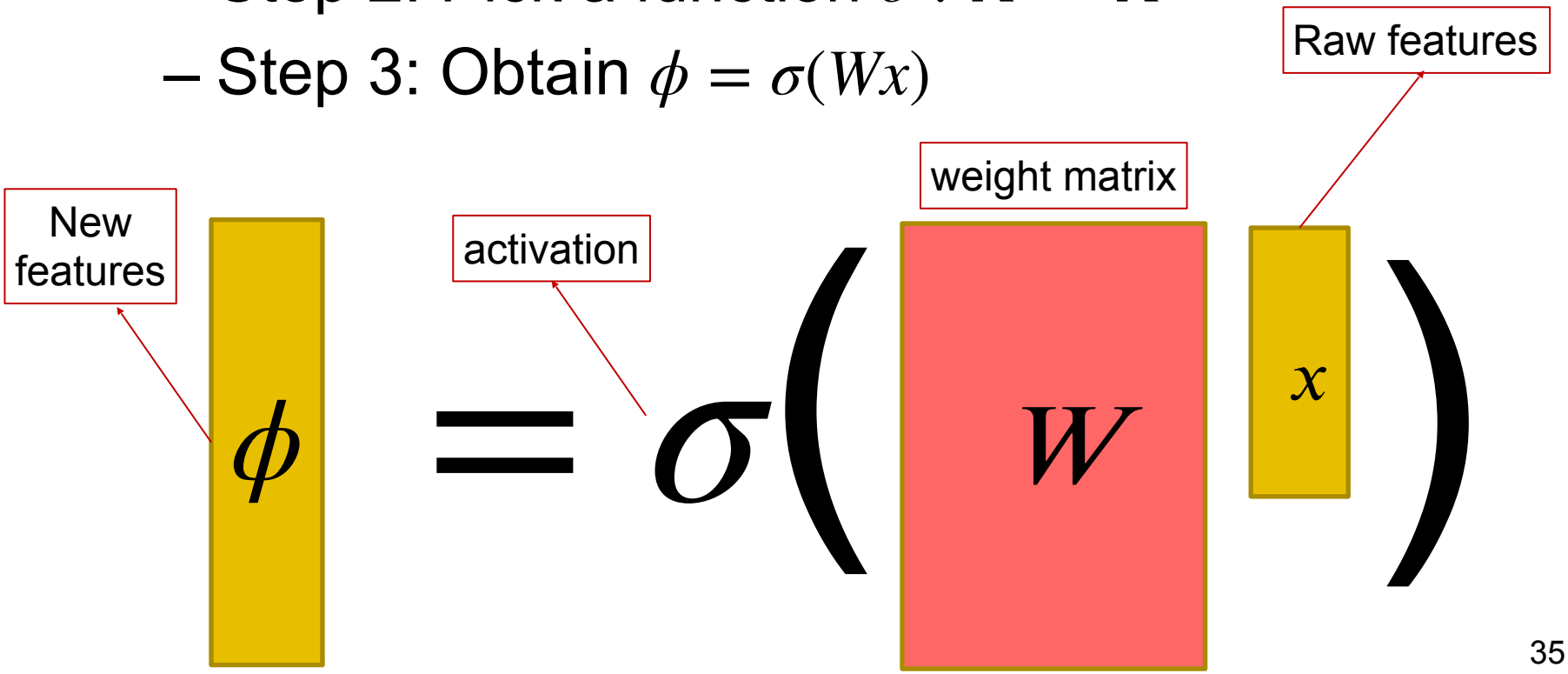
View each entry $\phi_i(x)$ as a feature to learn

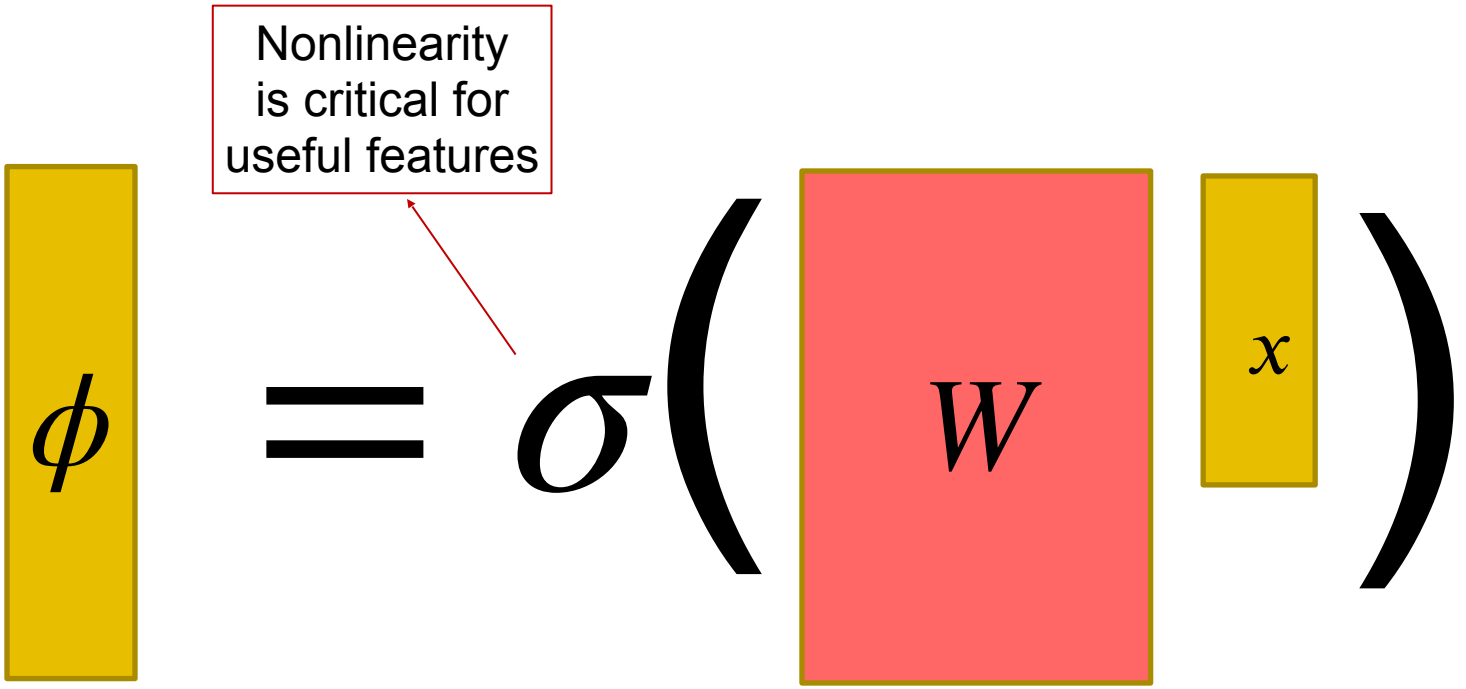


$$y = w^T \phi(x)$$

Learn features

- Starting point: Raw input features x
  - Step 1: Pick a matrix W
  - Step 2: Pick a function $\sigma : \mathbb{R} \to \mathbb{R}$
  - Step 3: Obtain $\phi = \sigma(Wx)$

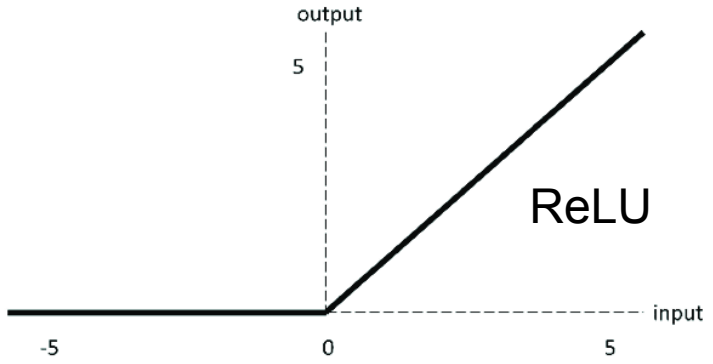




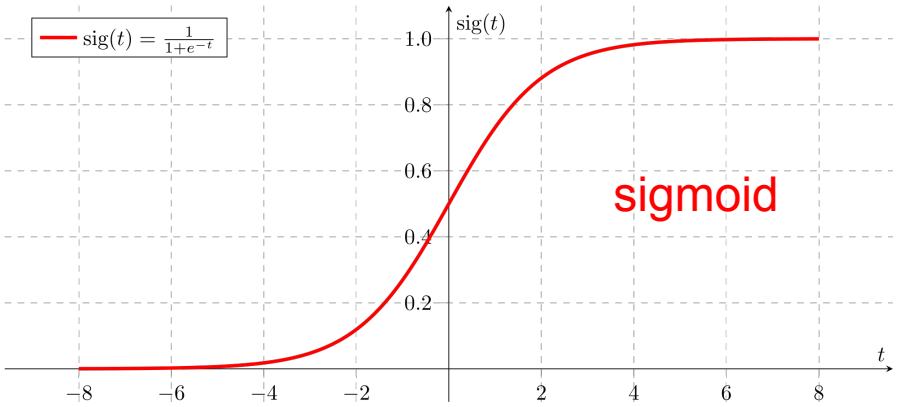
- **Activation** σ applies entry-wise i.e.,
  - i-**th feature** $\phi_i = \sigma(w_i^\top x)$ where w_i is the i-th row of W

Example Activation Functions

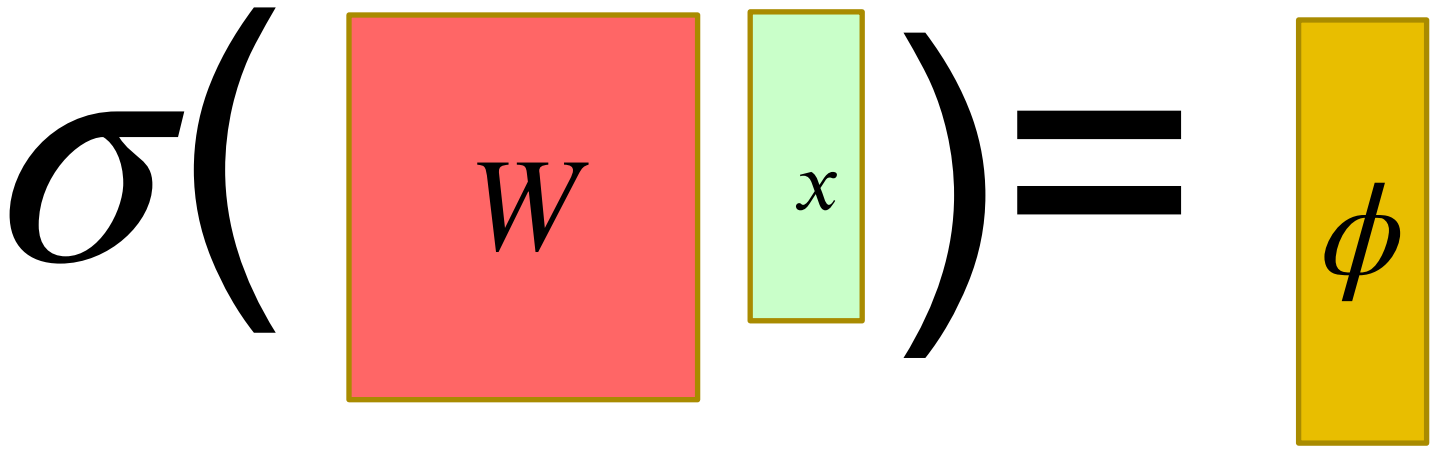
- ReLU: $\sigma(x) = \max\{x, 0\}$



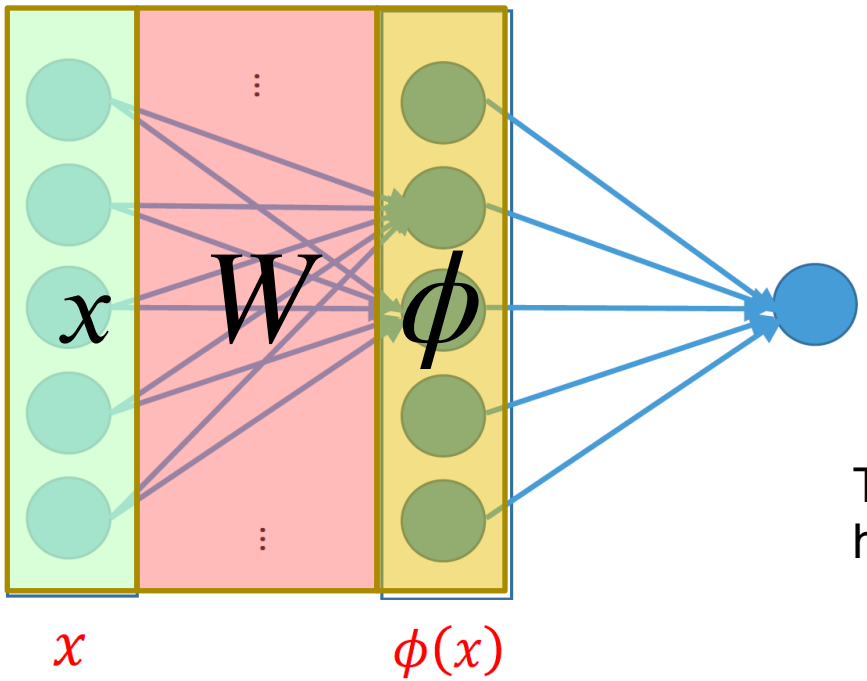
- Sigmoid: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$



Visualization



W is a learned weight matrix

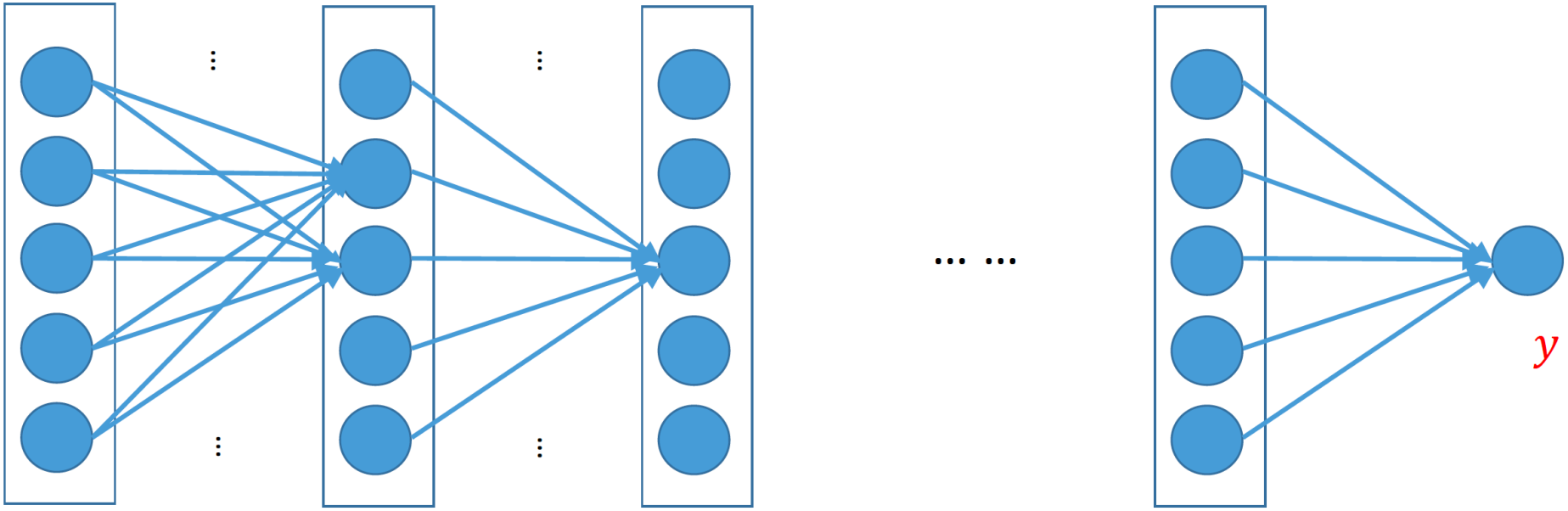


$y = w^T \phi(x)$

The output doesn't have to be a scalar

Deep Neural Networks

Repeatedly apply matrix multiplication and activation



The output doesn't have to be a scalar

Deep Neural Networks

Final linear layer

What does a deep net look like?

1. Pick weight matrices $W_1, W_2, \cdots, W_L, w$
2. Pick activation function σ
3. The neural net f predicts

$$f(x) = w^\top \sigma\Big(W_L \sigma\Big(W_{L-1} \cdots \sigma\Big(W_2 \sigma\big(W_1 x \big)\Big)\Big)\Big)$$

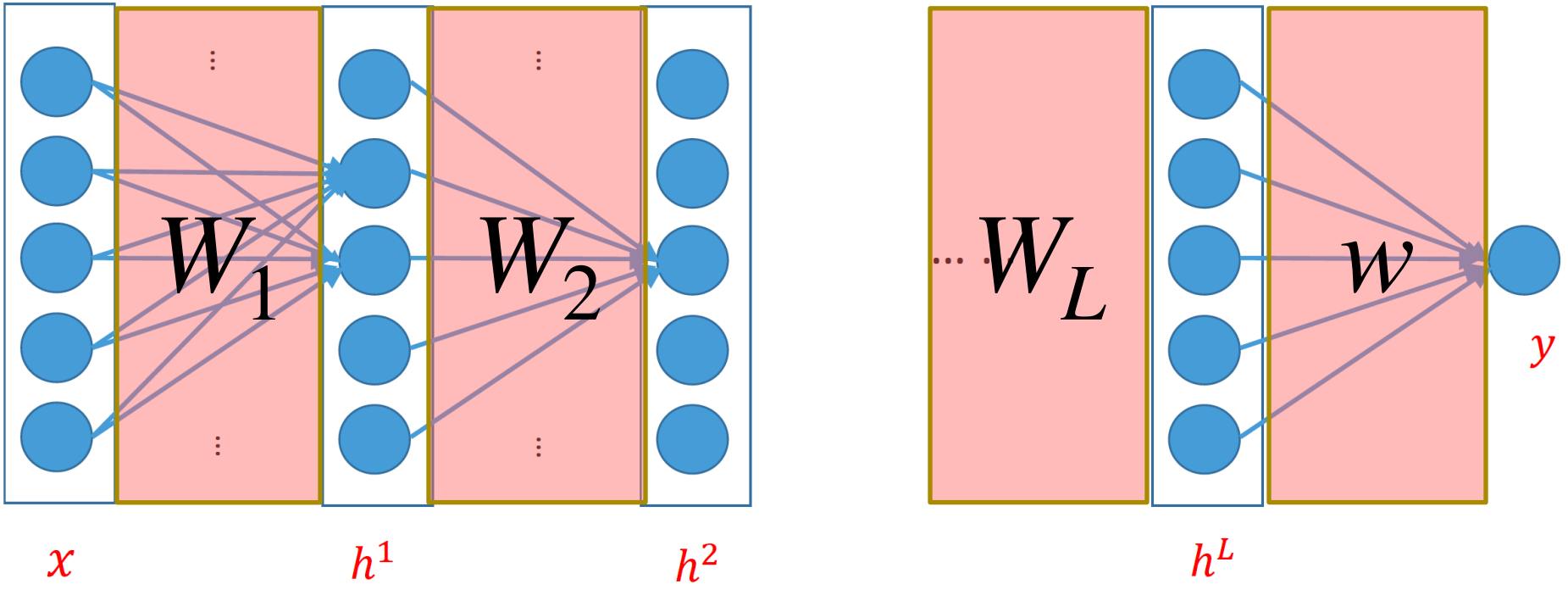
h_1

h_2

h_L

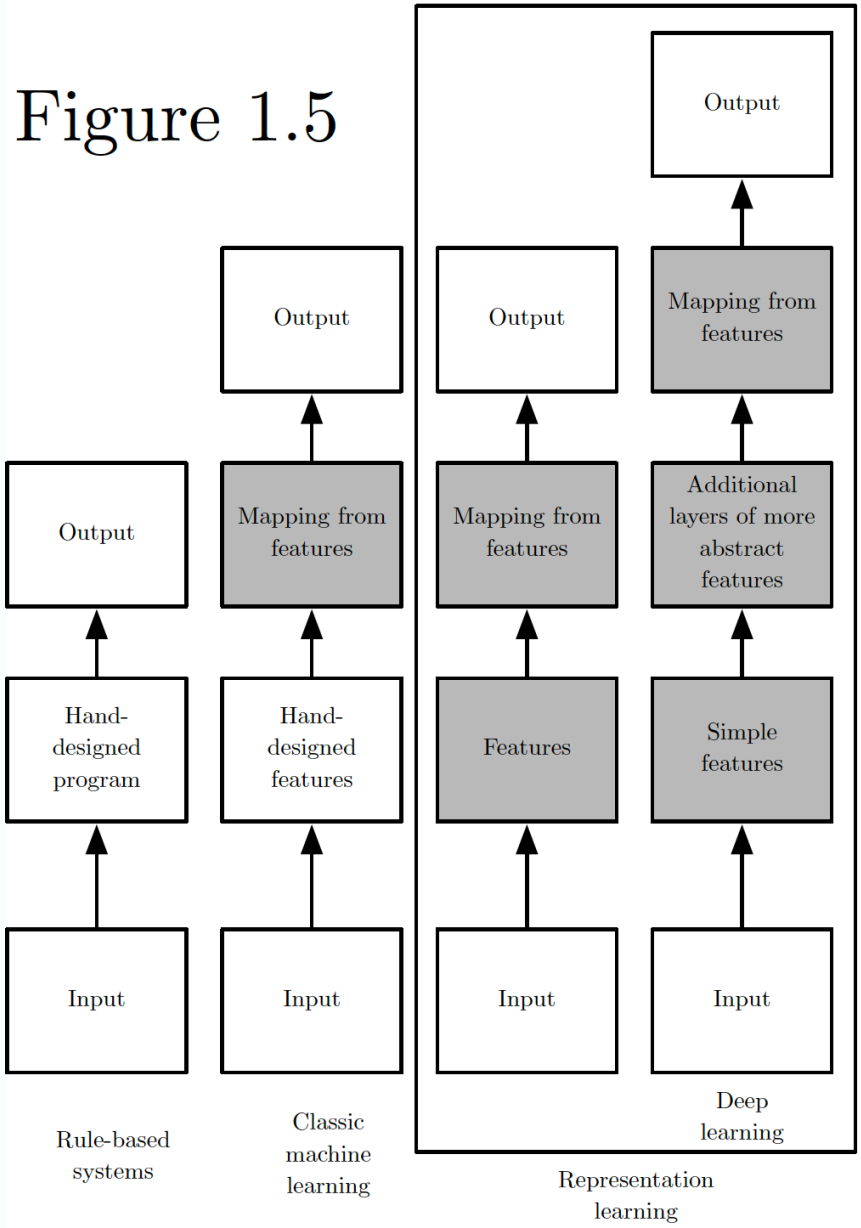
Deep Neural Networks

Repeatedly apply matrix multiplication and activation

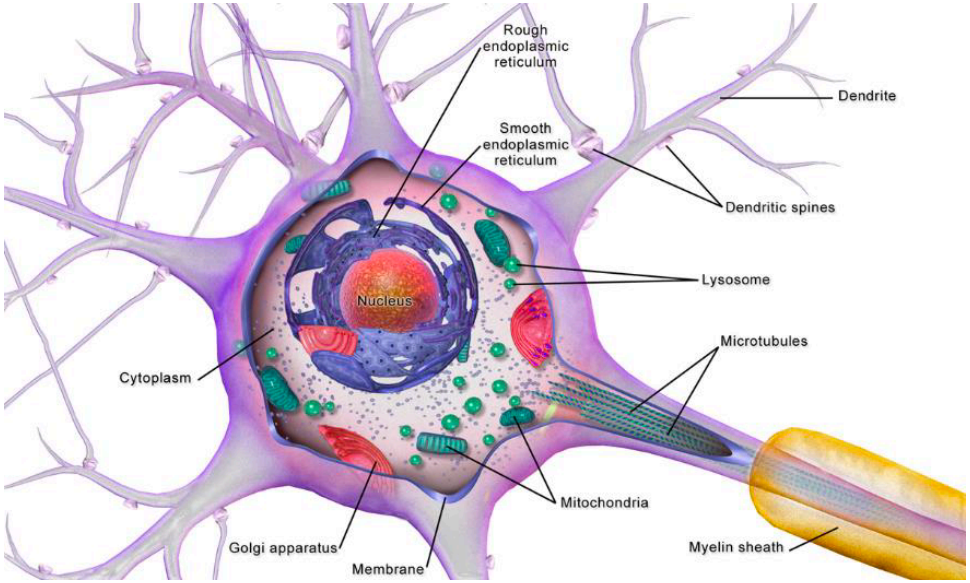


$W_1, W_2, \cdots, W_L, w$ are learned weight matrices/vectors

Figure 1.5


Output
Output
Output
Mapping from features
Output
Mapping from features
Mapping from features
Additional layers of more abstract features
Hand-designed program
Hand-designed features
Features
Simple features
Input
Input
Input
Input
Rule-based systems
Classic machine learning
Deep learning
Representation learning

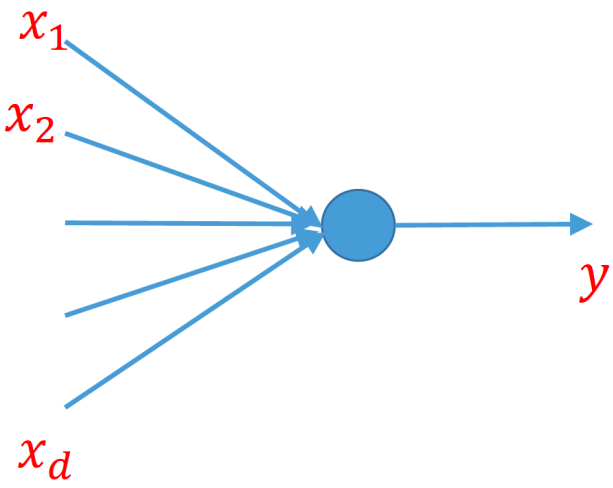
Why “Neural” Network?



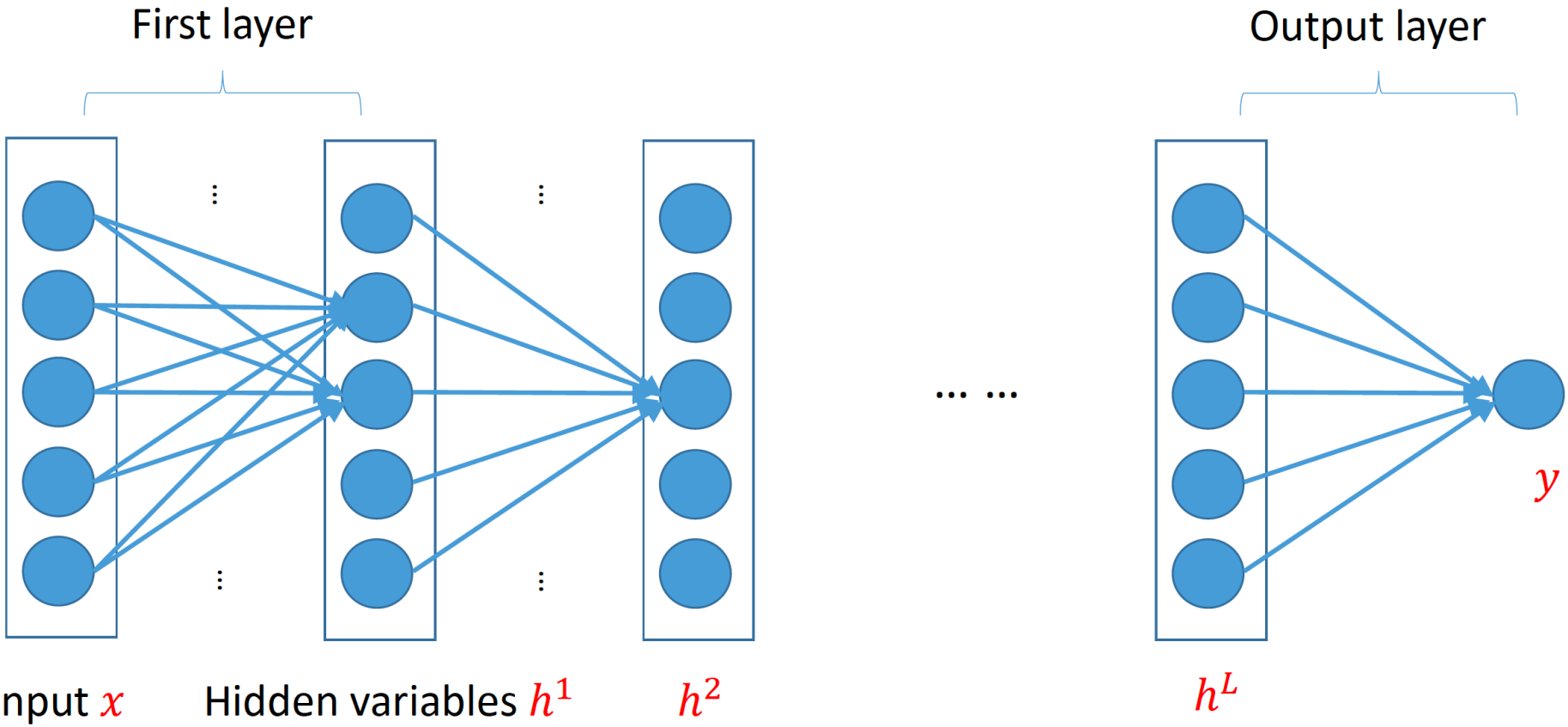
Neurons

Motivation: Abstract Neuron Model

- Looking at each activation function σ individually…
- $\sigma(w^\top x - b)$ resembles a neuron in the sense that it takes input from multiple sources and outputs its own activation
- If $\sigma(x) = \max\{x, 0\}$, the "neuron" is activated iff a certain threshold is reached, i.e., $w^\top x \geq b$.



Deep Neural Network Jargon



Input

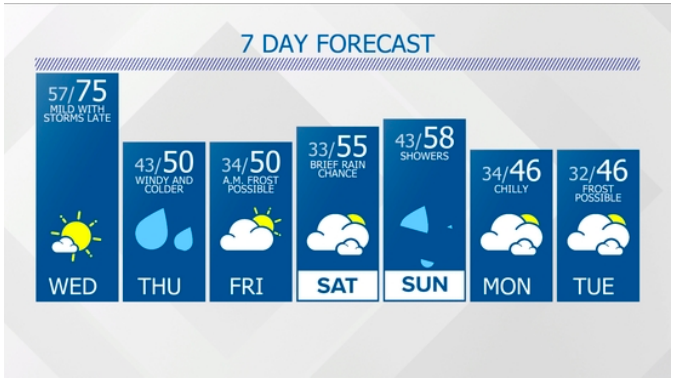
- Represented as a vector

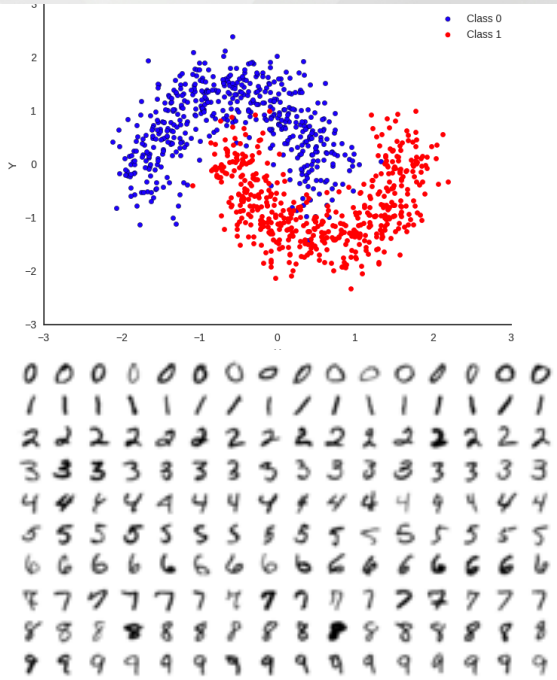
- Often useful to standardize
  - Subtract mean
  - Scale features to have unit variance

Expand

Output Layer

- Depends on the task
  - Regression
  - Binary classification
  - Multiclass classification





The Learning Problem

1. Data: $\left(x_i, y_i\right)_{i=1}^{n},\ x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^K$
2. Model: neural networks with trainable weight matrices $W_1, W_2, \cdots, W_L, W_{\text{out}}$
3. Loss function depending on the task

The rest of this lecture

4. Optimization

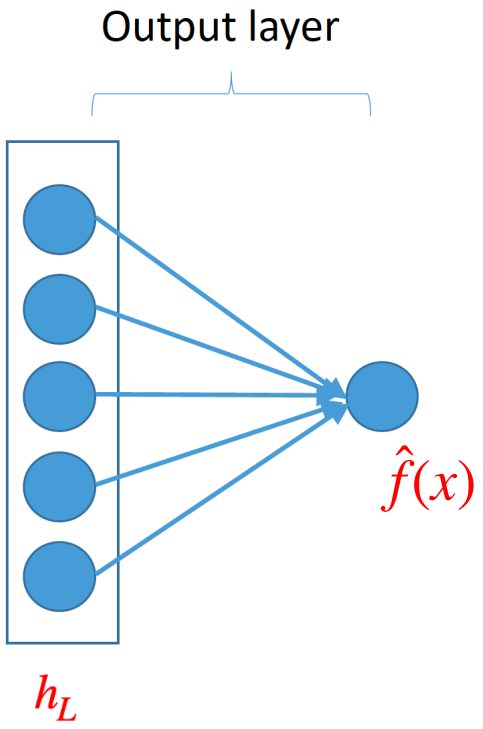
Next lecture

Regression

Regression output: $\hat{f}(x) = w^\top h_L + b$

Loss function: Square loss

$\ell(f(x), y) = \big(f(x) - y\big)^2$



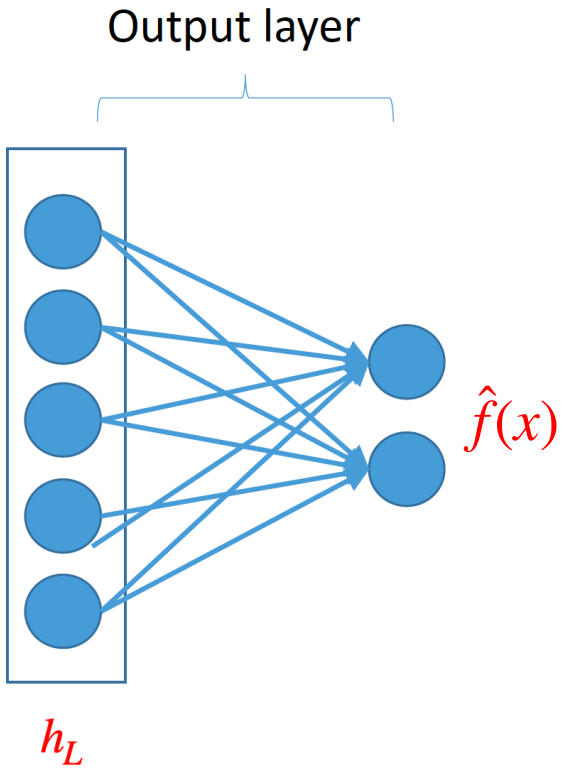
Vector-valued Regression

- Multi-dimensional regression: $\hat{f}(x) = W^\top h_L + b$

- Linear operation:
  - no nonlinearity
  - just matrix multiplication

- Loss function: Vector-valued square loss

$$\ell(f(x), y) = \left\| f(x) - y \right\|_2^2$$



Binary Classification

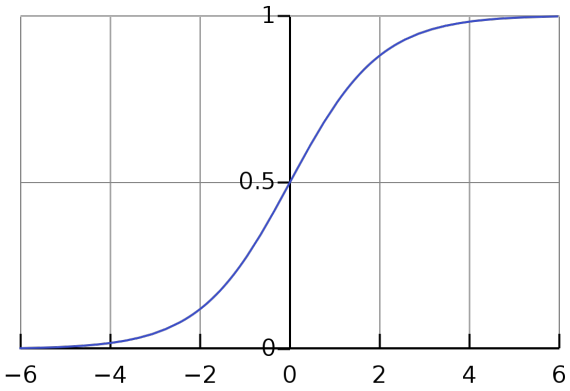
- We often want model to output a probability

$$\hat{f}(x) = \mathbb{P}(y = 1)$$

- Converting a prediction to probability

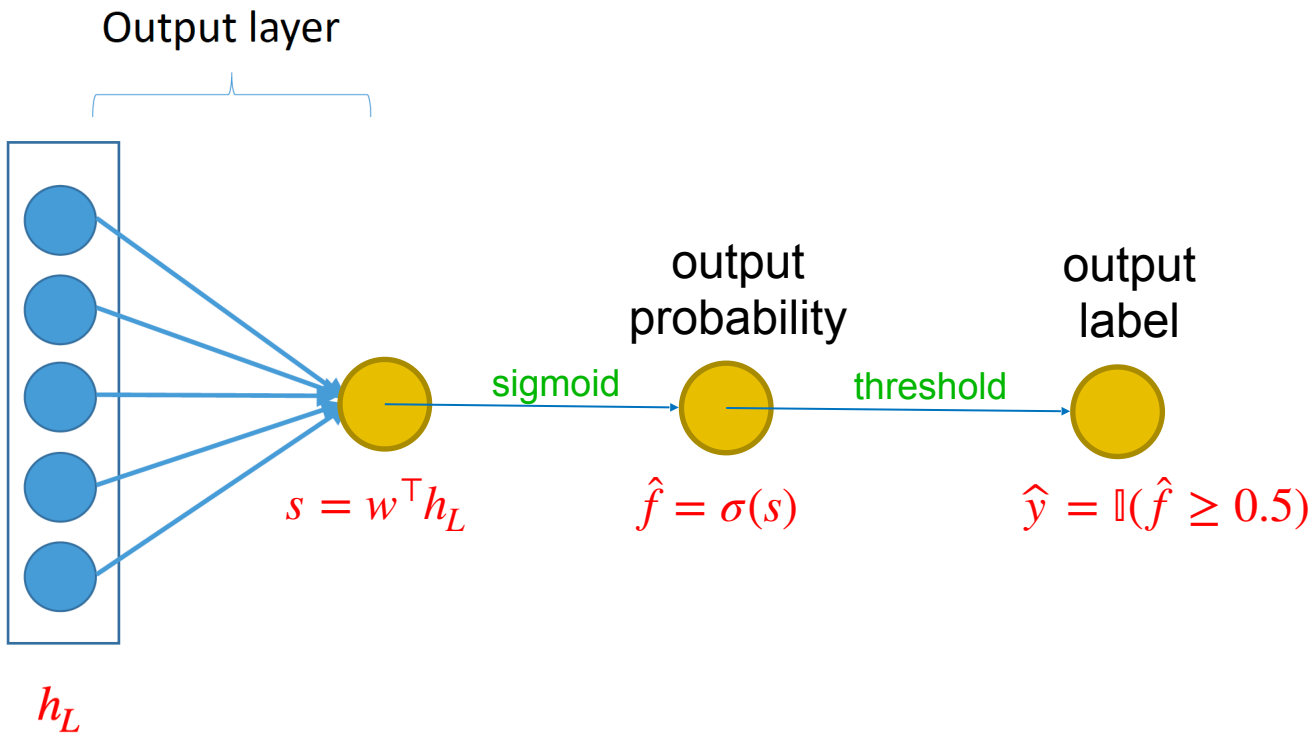
$$\text{Score } s(x) \mapsto \text{Probability } \hat{f}(x)$$

- Use sigmoid function $\sigma : \mathbb{R} \to (0,1)$

$s(x) \to$  $\to f(x)$

Binary Classification

- Final feature representation h_L
- Linear score function $s(x) = w^\top x + b$
- Probability $\hat{f}(x) = \sigma(s(h_L)) = \dfrac{1}{1 + e^{-s(h_L)}}$


Output layer
output probability
output label
sigmoid
threshold
$s = w^\top h_L$
$\hat{f} = \sigma(s)$
$\hat{y} = \mathbb{I}(\hat{f} \geq 0.5)$
h_L

Example task: Is this a dog?

x	$s(x)$	$\hat{f}(x)$	$\hat{y}$	
	1.5	$0.82 \approx \frac{1}{1+e^{-1.5}}$	1	✅
	-2.0	0.076	0	❌
	-0.2	0.45	0	❌

Binary Classification

- Final feature representation h_L
- Linear score function $s(x) = w^\top x + b$
- Probability $\hat{f}(x) = \sigma(s(h_L)) = \dfrac{1}{1 + e^{-s(h_L)}}$
- Loss function: Log loss

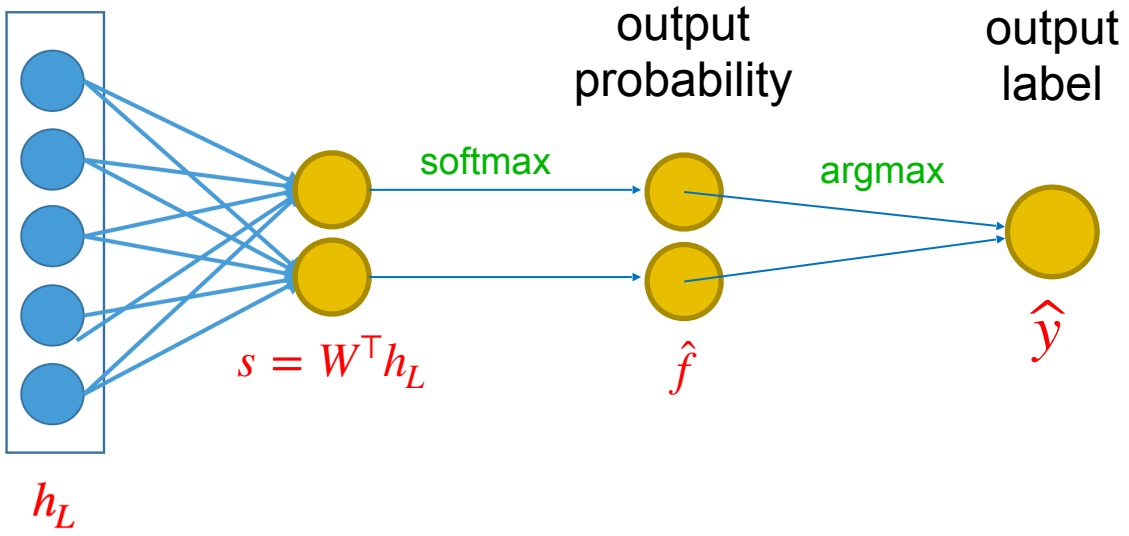
$$\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y)\log(1 - \hat{y})$$

Measure the cross-entropy between dist. $[1 - y, y]$ and dist. $[1 - \hat{y}, \hat{y}]$

Multiclass Classification

- $\hat{f} = \text{softmax}(s)$ where $s = W^\top h_L + b$

- Intuition: multi-class logistic regression on feature h_L



Softmax Function

- Generalizes the sigmoid function
- Converts a vector to a probability distribution
- Let $\hat{f} = \text{softmax}(s)$. Then

$$\hat{f}_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Softmax Function

- Let $\hat{f} = \text{softmax}(s)$. Then

$$\hat{f}_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

- Properties
  - $\hat{f}_i \geq 0$
  - $\sum_i f_i = 1$
- Intuition: probability distribution over classes

$$\hat{f}_i(x) = \mathbb{P}(y = i \,|\, x)$$

Dog (0) vs Frog (1) vs Hog (2)

x	$\hat{f}$	$\hat{y}$	
	$[0.5,\ 0.2,\ 0.3]$	0	✅
	$[0.1,\ 0.6,\ 0.3]$	1	✅
	$[0.45,\ 0.15\ 0.4]$	0	❌

Multiclass Classification

- $\hat{f} = \text{softmax}(s)$ where $s = W^\top h_L + b$

- Intuition: multi-class logistic regression on feature h_L

- Loss function: Cross-entropy loss

$$\ell\left(y, \hat{f}(x)\right) = -\sum_{i \in [C]} y_i \log(p_i), \quad \text{where} \quad p_i = \frac{e^{\hat{f}_i(x)}}{\sum_{j=1}^{n} e^{\hat{f}_j(x)}}$$

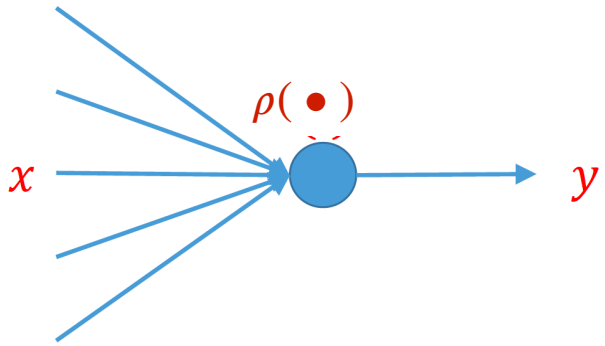
Measure the cross-entropy between dist. y and dist. $\hat{f}$

Activation Choice

Typical activations

- Sigmoid $\sigma(z) = \dfrac{1}{1+e^{-z}}$
- $\tanh(z) = 2\sigma(2z) - 1$
- $\mathrm{ReLU}(z) = \max(z,0)$

$$y = \rho\left(w^T x\right)$$



Vanishing Gradient

Sigmoid saturates and gradient vanishes...

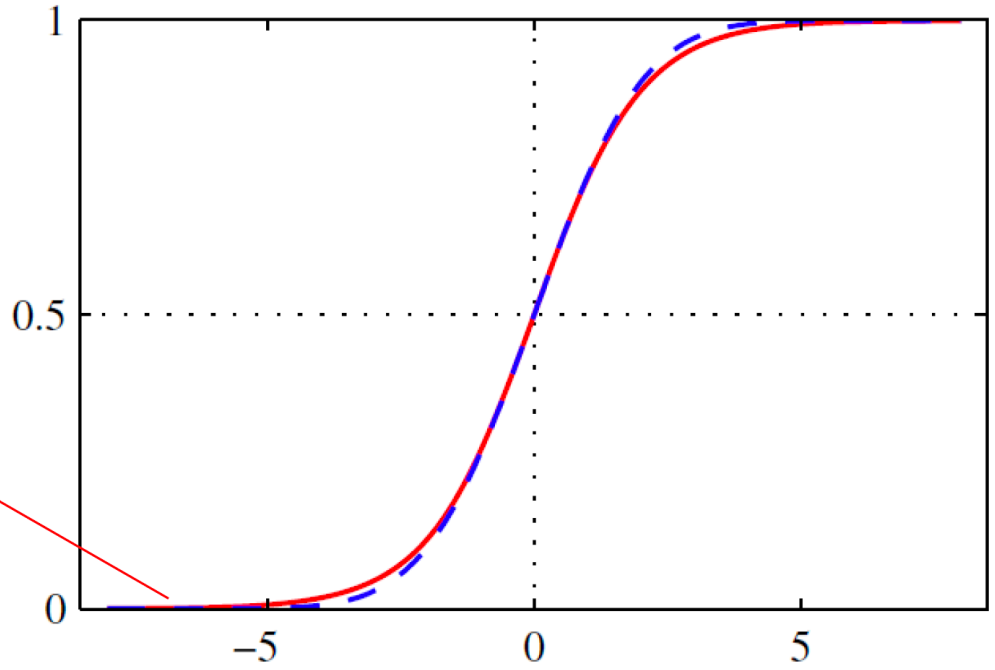


Figure borrowed from *Pattern Recognition and Machine Learning*, Bishop

Sigmoid to ReLU

- Activation function ReLU (rectified linear unit)
  - $\mathrm{ReLU}(z) = \max\{z, 0\}$

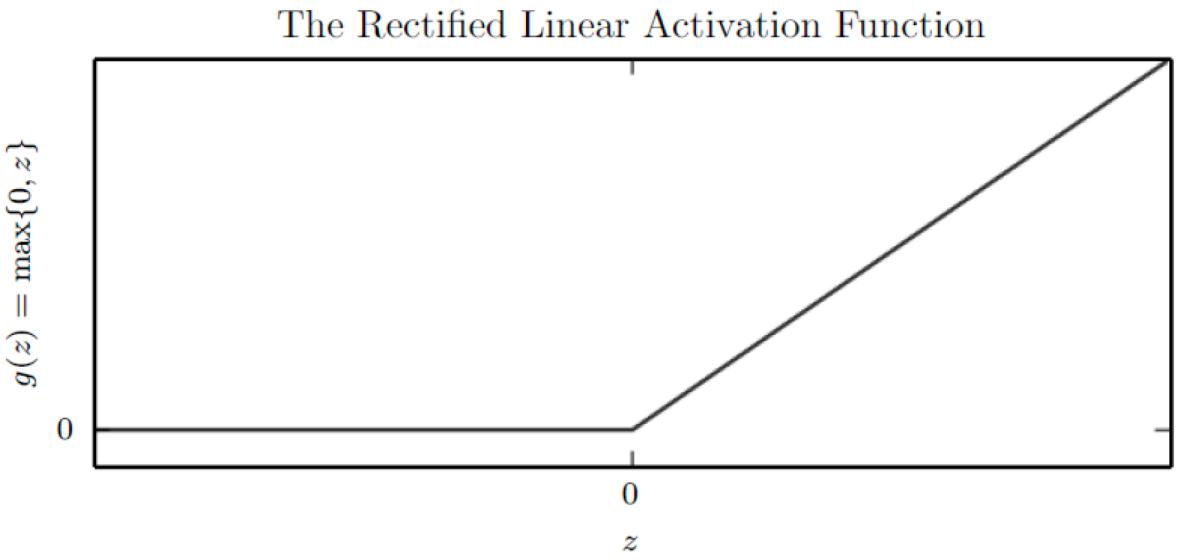


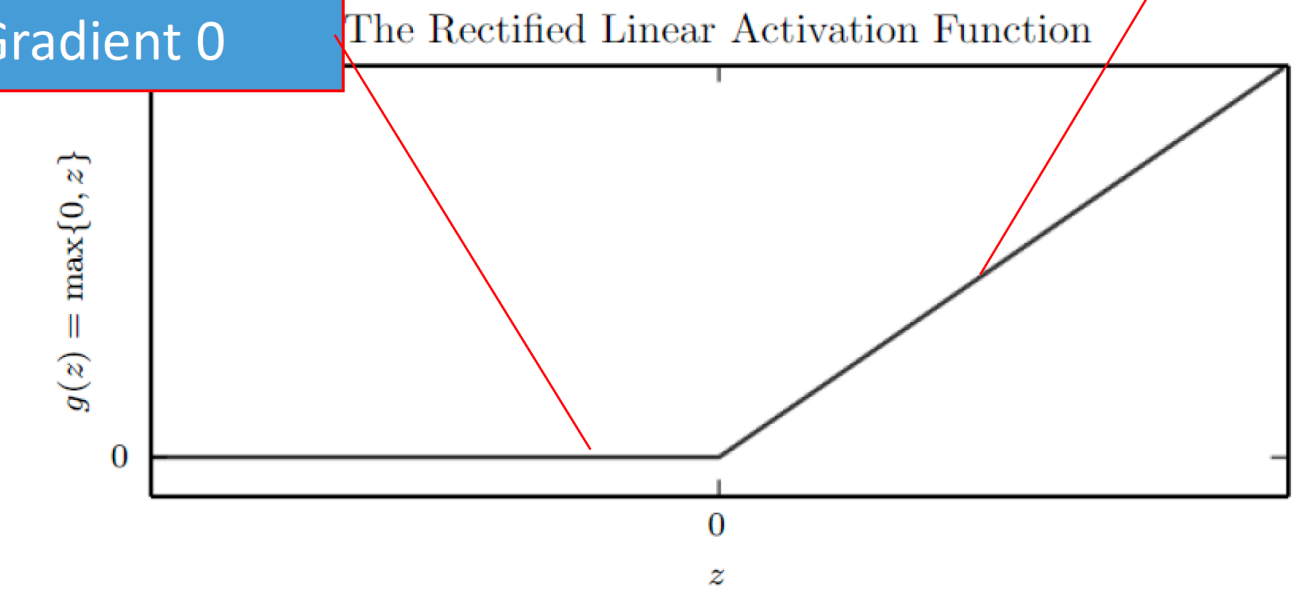
Figure from *Deep learning*, by Goodfellow, Bengio, Courville.

ReLU gradient

- Activation function ReLU (rectified linear unit)
  - $\mathrm{ReLU}(z) = \max\{z, 0\}$

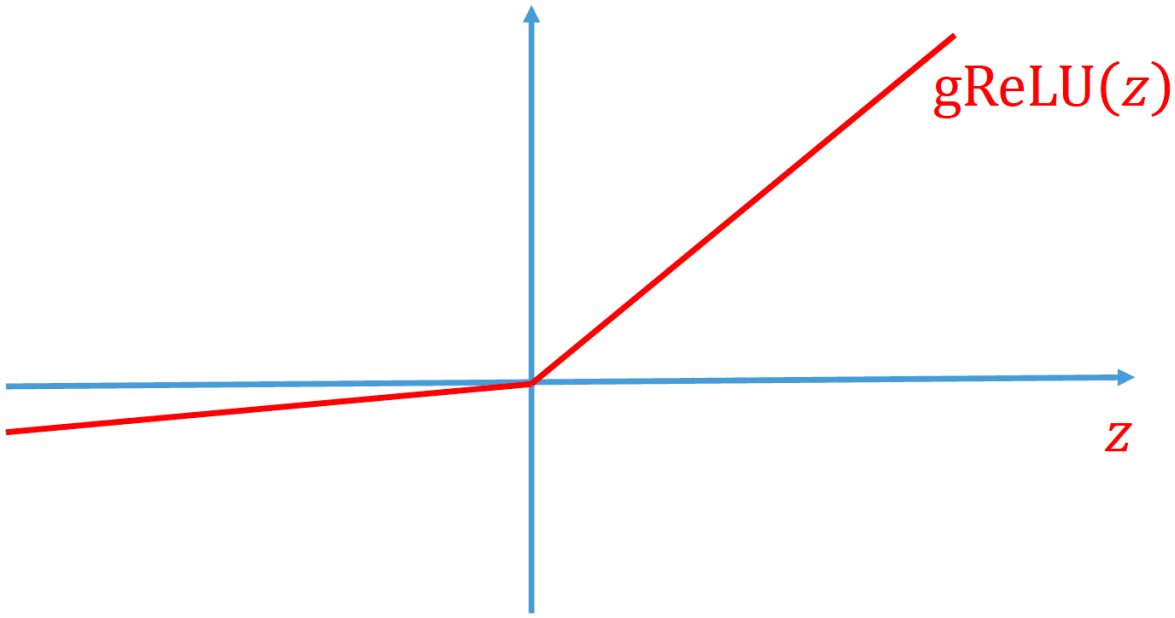
Gradient 1

Gradient 0



ReLU generalizations

- $\text{gReLU}(z) = \max\{z, 0\} + \alpha \min\{z, 0\}$
  - Leaky-ReLU$(z) = \max\{z, 0\} + 0.01 \min\{z, 0\}$
  - Parametric-ReLU(z): α learnable



Acknowledgements

- A lot of content from online resources
  - Yingyu Liang's slides https://www.cs.princeton.edu/courses/archive/spring16/cos495/
  - Goodfellow's slides https://www.deeplearningbook.org/