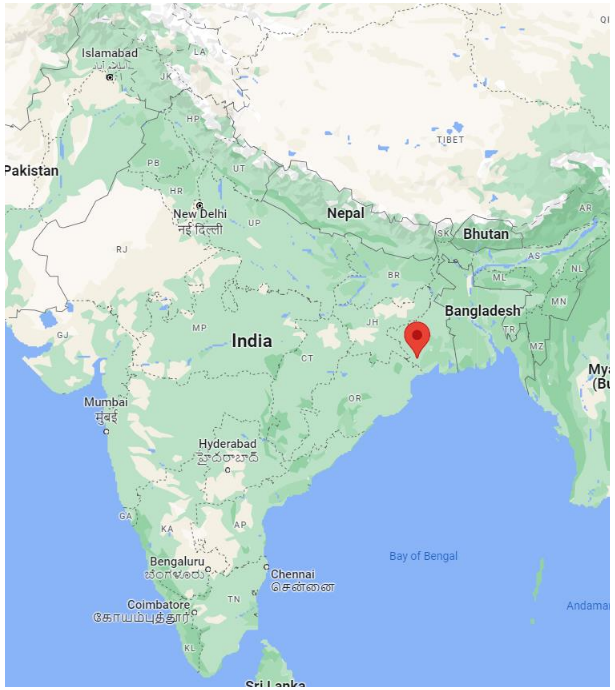
# Deep Learning Foundations

Winter term 2024/25

Sandipan Sikdar

# Who is giving the course?




- Assistant Professor at the Faculty of Electrical Engineering and Computer Science (Nov’22 - )
- Postdoctoral Researcher at RWTH Aachen (2018 – 2022)
- PhD, Computer Science and Engineering from Indian Institute of Technology Kharagpur

# Deep Learning Revolution



ChatGPT

# Deep Learning

- Single/Muli-layer perceptrons
- Training/optimization

We will cover in this course

- Generative modeling
- Self-supervised learning
- Contrastive learning
- Multi-task learning
- Meta learning

Text Mining

- Deeper neural nets
- Convolutional neural nets
- Recurrent neural nets
- Transformers
- ….

# Setup

- Lectures: WED 8:00 – 9:30 (2 * 45 minutes)
- Exercises: WED 15:00 – 16:30 (2 * 45 minutes)
- Lecture
  - Mostly theory, algorithms, findings
- Exercise
  - Presentation on how to do in practice
  - Hands on implementations (Python/Pytorch)
  - Finish exercises at home (own responsibilty)

# Setup

- Home assignments
  - 3 home assignments (3 weeks time)
  - Teams of upto 3 students
  - No plagiarism! (Can result in exclusion from the course)
  - Tentative dates:
    - Home assignment 1 (13.11.2024)
    - Home assignment 2 (04.12.2024)
    - Home assignment 3 (08.01.2025)

# Grading

- Final exam:
  - Pen and paper exam
  - The final exam will be scaled to 100 points. The minimum passing score is 50 points.
  - Bonus points for home assignments: Points from the home assignments will be scaled to 10 points and added to the final score.

# Course Plan

| Week | Date | Theme | Topics |
|---|---|---|---|
| 1 | 16.10 | Intro + Preliminaries | Neural network basics |
| 2 | 23.10 | No lectures | |
| 3 | 30.10 | Neural Networks | Training neural networks |
| 4 | 06.11 | Deeper architectures | CNNs/RNNs/Transformers |
| 5 | 13.11 | Generative models I | Preliminaries, latent variable models |
| 6 | 20.11 | Generative models II | Variational Auto Encoder (VAE) |
| 7 | 27.11 | Generative models III | Diffusion models |
| 8 | 04.12 | Generative models IV | Flow Models |
| 9 | 11.12 | Generative models V | GANs |
| 10 | 18.12 | Learning paradigm | Self-supervised and contrastive learning |
| 11 | 08.01 | Learning paradigm | Multi-task and Meta learning |
| 12 | 15.01 | Adversarial Robustness | Poisoning attacks |
| 13 | 22.01 | Adversarial Robustness | Attacks and defenses |
| 14 | 29.01 | Summary and outlook | |

# Preliminaries

# Supervised Learning

- We are given a set of training examples of the form

  $\{(x^1, y^1), \dots, (x^n, y^n)\}$

- $x^i$ represents the feature vector for the $i$th training example

- $y^i$ represents the ground truth

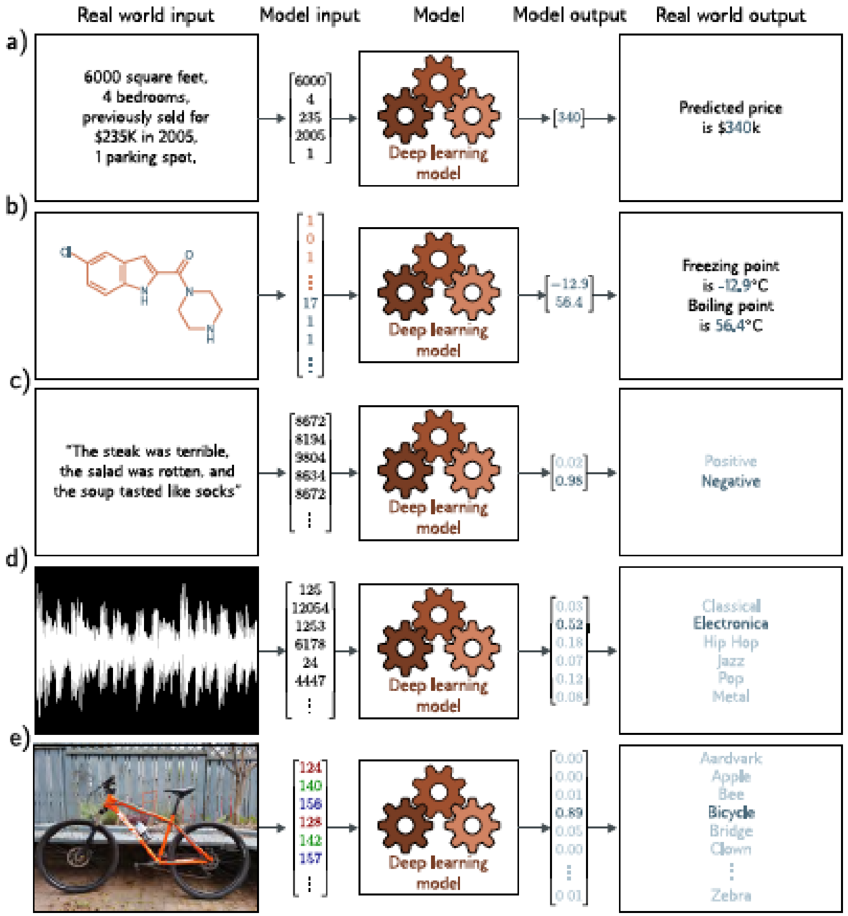
  - Classification: $y^i \in [c]$

  - Regression: $y^i \in R$

# Supervised Learning

- We are given a set of training examples of the form $\{(x^1, y^1), \ldots, (x^n, y^n)\}$
- $x^i$ represents the feature vector for the $i$th training example
- $y^i$ represents the ground truth
- Goal: Find $f_W: R^d \rightarrow R, f_W(x^i) \approx y^i$
- $W$: model parameters

# Supervised Learning



# Supervised Learning

- Goal: Find $f_W : R^d \to R, f_W(x^i) \approx y^i$
- How do we find $W$?

$$\min_W \frac{1}{n} \sum_{i=1}^{n} l(f_W(x^i), y^i)$$

- $l$ -> loss/cost function
- Emperical risk minimization

# Loss Function

- A function that measures, for each value of the $\mathrm{W}s$ how close $f_W(x^i)$'s are to the corresponding $y^i$s
- Regression:

$$l(f_W(x^i), y^i) = \frac{1}{2}\left(f_W(x^i) - y^i\right)^2$$

# Loss Function

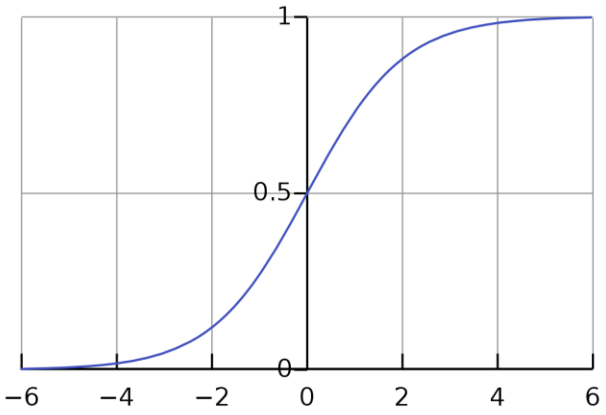
- A function that measures, for each value of the $\mathrm{W}s$ how close $f_W(x^i)$'s are to the corresponding $y^i$s
- Classification:

$$y^i = \begin{cases} 1 \\ 0 \end{cases}$$

# Loss Function

- Convert the output of the model to probabilities

- Sigmoid function: $g(x) = \frac{1}{1+e^{(-x)}}$




$$\begin{cases} P\left(y^i = 1 \middle| x^i, W\right) = g\left(f_W\left(x^i\right)\right) \\ P\left(y^i = 0 \middle| x^i, W\right) = 1 - g(f_W(x^i)) \end{cases}$$

# Loss Function

$$\begin{cases} P\left(y^i = 1 \middle| x^i, W\right) = g\left(f_W\left(x^i\right)\right) \\ P\left(y^i = 0 \middle| x^i, W\right) = 1 - g(f_W(x^i)) \end{cases}$$

$$\Downarrow$$

$$P\left(y^i \middle| x^i, W\right) = g\left(f_W(x_i)\right)^{y^i} \left(1 - g\left(f_W\left(x^i\right)\right)\right)^{1-y^i}$$

# Loss Function

$$\max_W \prod_{i=1}^{n} P(y^i|x^i, W)$$

$$= \max_W \sum_{i=1}^{n} \log P(y^i|x^i, W)$$

$$= \min_W - \sum_{i=1}^{n} \log P(y^i|x^i, W)$$

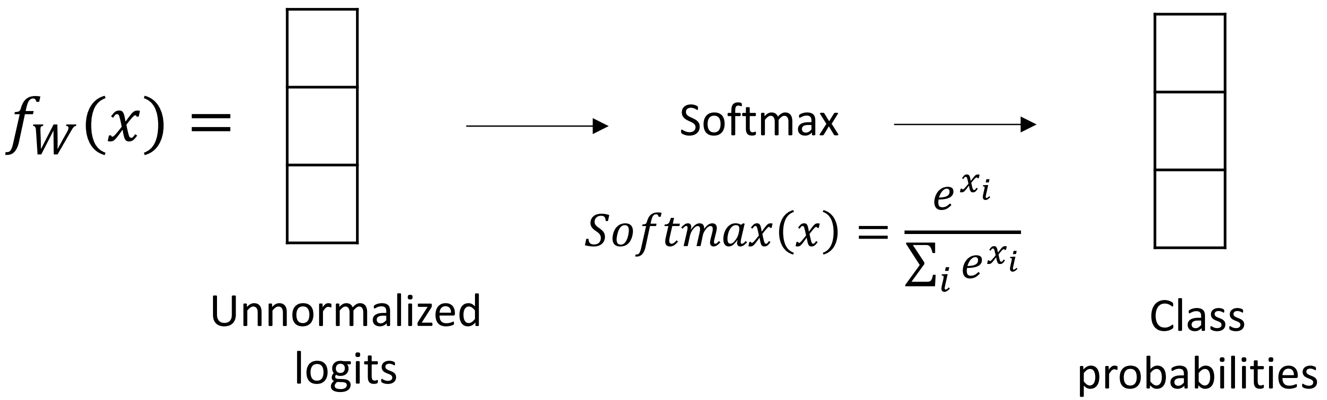
# Loss Function

$$\min_{W} - \sum_{i=1}^{n} \log P(y^i | x^i, W)$$

$$= \underbrace{\min_{W} - \sum_{i=1}^{n} y^i \log\left(g\left(f_W(x)\right)\right) + \left(1 - y^i\right) \log(1 - g(f_W(x^i)))}_{\text{Cross entropy loss}}$$

# Loss Function

- Generalizing cross entropy to multiple class
- Model output -> vector of the dimension of the number of classes
- Use softmax instead of sigmoid



# Loss Function

- Class: $c_j : j \in \{0,1,2\}$
- Each class represented by a 1-hot vector

$$c_j = \begin{cases} 0, [0,0,1] \\ 1, [0,1,0] \\ 2, [1,0,0] \end{cases}$$

- Cross-entropy loss for multiclass classification is then computed as -

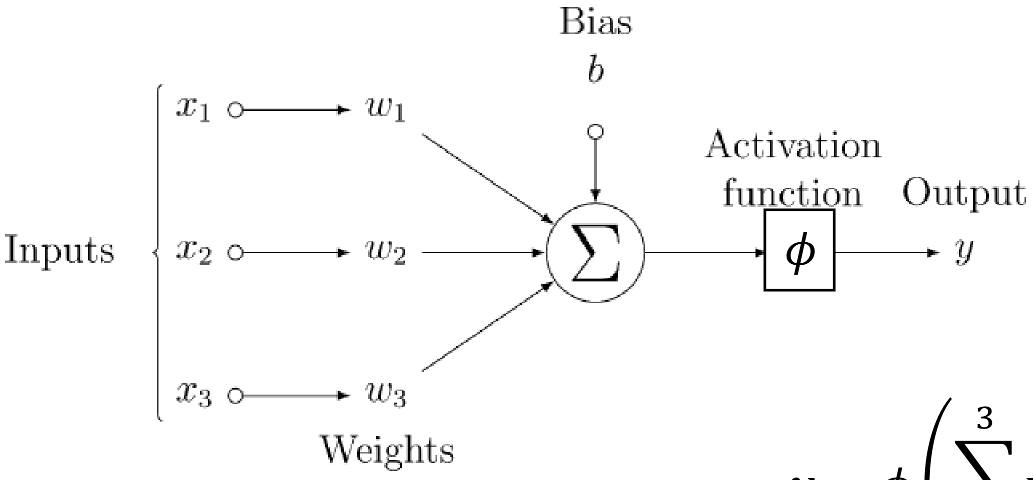
$$-\sum_{j=0}^{|c|} c[j] \log(f_W(x^i)[j])$$

# Choice of model

- $f_W: R^d \rightarrow R, f_W(x^i) \approx y^i$
- What should $f_W$ look like?
- Linear function
- $f_W(x) = W^T x + b$
- + Convex optimization
- - Limited representation

# Choice of model

- Non-linear functions

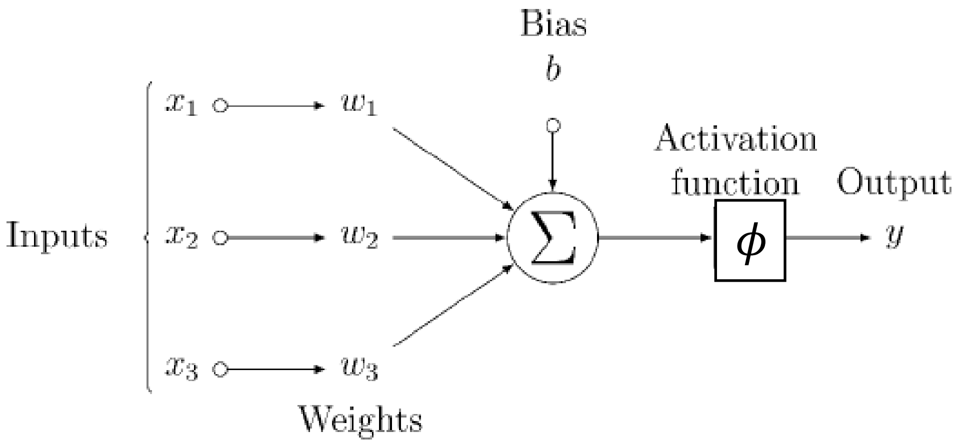



$$y = \phi\left(\sum_{i=1}^{3} w_i x_i + b\right)$$

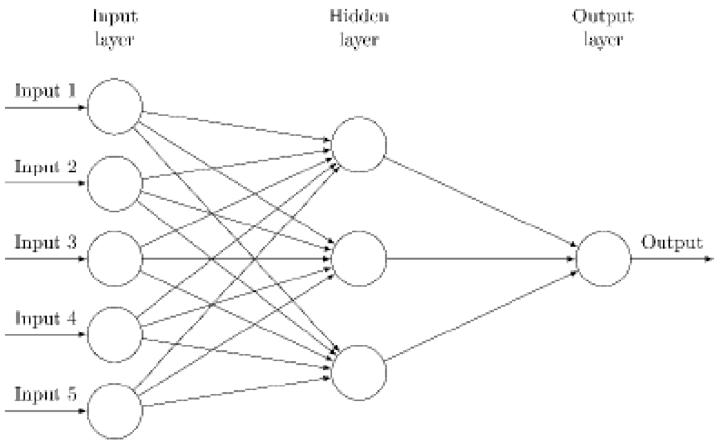
Simplest neural network

# Neural Network

- Cascaded layers

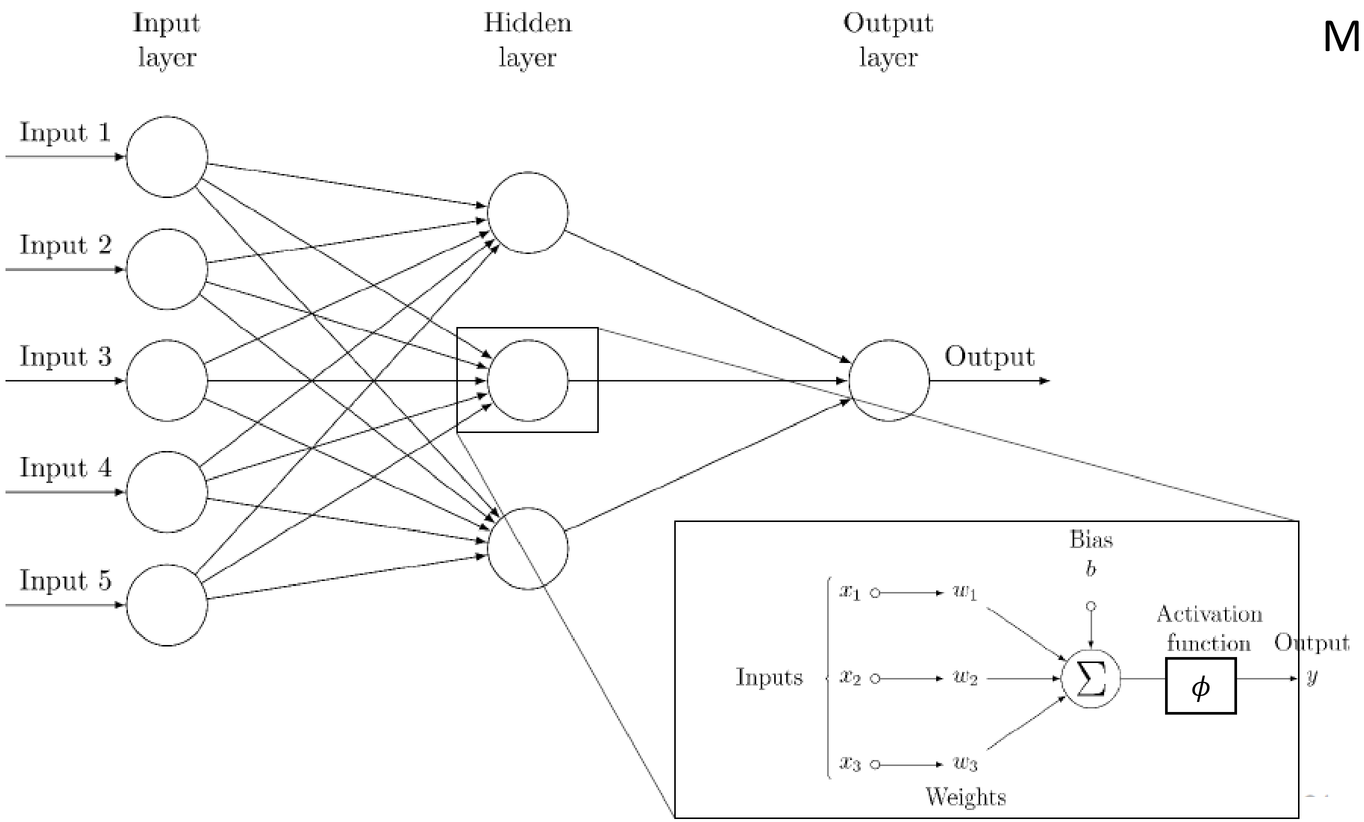



Single Neuron




Neural Network

# Neural Network

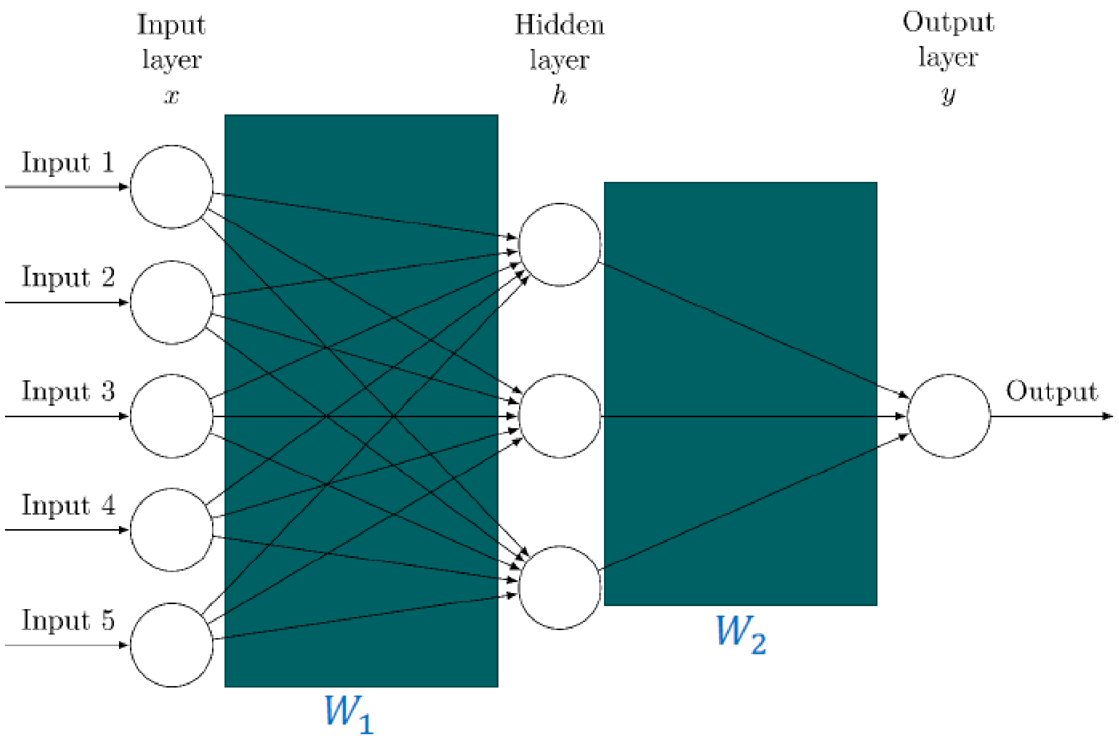



Multi-layer perceptron

# Neural Network – Forward Pass

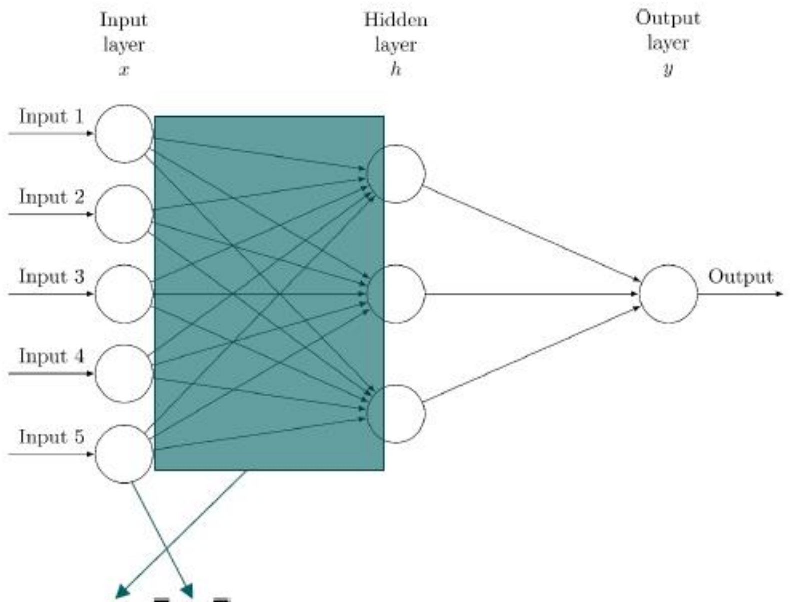
- Given
  - A neural network with a fixed set of weights
  - An input sample to be classified
- How to get the classification?
- A layer is applied by multiplying its weight matrix with its input vector

# Neural Network – Forward Pass




$$h = \phi(W_1 x + b_1)$$
$$y = \phi(W_2 h + b_2)$$

# Neural Network – Forward Pass




$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 + w_{15}x_5 + b_1 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + w_{24}x_4 + w_{25}x_5 + b_2 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + w_{34}x_4 + w_{35}x_5 + b_3 \end{bmatrix} \xrightarrow[\text{Activation}]{} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

# Chained Linear Classifiers

- Remember
  - A linear classifier provides class scores by calculating $y = \phi(Wx + b)$
  - A neural network is a chain of linear classifiers with activation functions

- A neural network is some function like this:

$$y = \phi(W_3\phi(W_2\phi(W_1x + b_1) + b_2) + b_3)$$

- We can chain this as deep as we want

# Activation Functions

- After each layer, an **activation function** is used
- An activation function can be **any function**
- However, **non-linearity** is required for a more expressive model

- Why do we need this?
  - Remember: Linear classifiers can separate data linearly
  - A neural network with a linear activation function is still a linear classifier
  - Non-linear activation functions enable us to learn non-linear relations

# Using Linear Activations

- Consider a linear activation $\phi(a) = a$

$$
\begin{aligned}
y &= \phi(W_3\phi(W_2\phi(W_1x + b_1) + b_2) + b_3) \\
&= W_3(W_2(W_1x + b_1) + b_2) + b_3) \\
&= W_3(W_2W_1x + W_2b_1 + b_2) + b_3) \\
&= W_3W_2W_1x + \underbrace{W_3W_2b_1 + W_3b_2 + b_3}_{\text{constant term}} \\
&= \boldsymbol{W}x + \boldsymbol{b}
\end{aligned}
$$

# Chained Linear Classifiers

$$y = \phi(W_3\phi(W_2\phi(W_1x + b_1) + b_2) + b_3)$$

$$\Downarrow$$

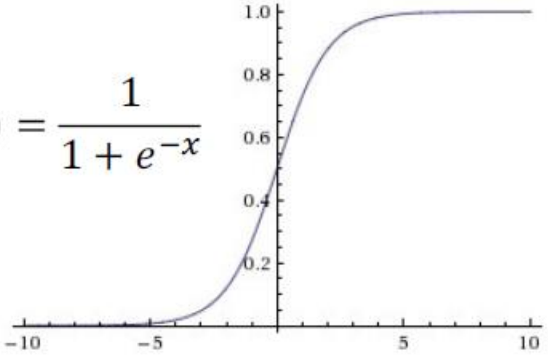
$$y = softmax(W_3 \tanh(W_2 \tanh(W_1x + b_1) + b_2) + b_3)$$

Activation function

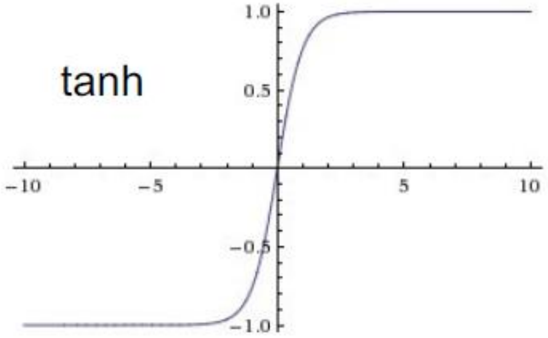
# Activation Functions

- Some common activation functions:

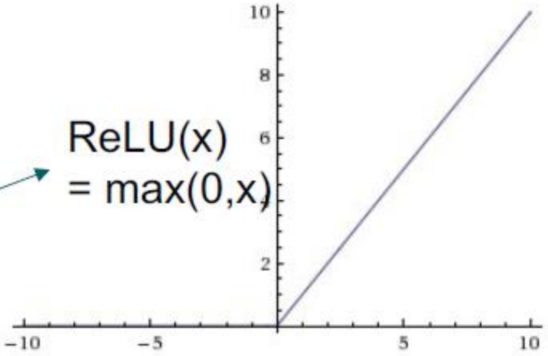
$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$$




tanh




ReLU(x) = max(0,x)

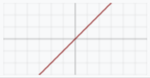
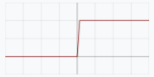
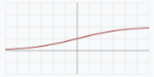
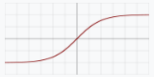
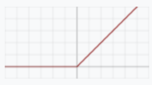
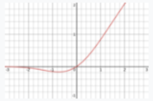
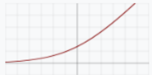
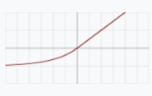
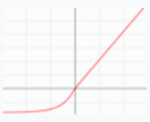
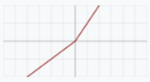



Use ReLU!
Works best in practice

Some more:

- Heaviside
- Leaky ReLU
- SeLU
- …

# Activation Functions

| Name | Plot | Function, $g(x)$ | Derivative of $g$, $g'(x)$ | Range | Order of continuity |
|---|---|---|---|---|---|
| Identity |  | $x$ | $1$ | $(-\infty, \infty)$ | $C^\infty$ |
| Binary step |  | $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$ | $0$ | $\{0, 1\}$ | $C^{-1}$ |
| Logistic, sigmoid, or soft step |  | $\sigma(x) \doteq \frac{1}{1+e^{-x}}$ | $g(x)(1-g(x))$ | $(0, 1)$ | $C^\infty$ |
| Hyperbolic tangent (tanh) |  | $\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$ | $1 - g(x)^2$ | $(-1, 1)$ | $C^\infty$ |
| Rectified linear unit (ReLU)[8] |  | $(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$ | $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$ | $[0, \infty)$ | $C^0$ |
| Gaussian Error Linear Unit (GELU)[5] |  | $\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$ | $\Phi(x) + x\phi(x)$ | $(-0.17\ldots, \infty)$ | $C^\infty$ |
| Softplus[9] |  | $\ln(1 + e^x)$ | $\frac{1}{1+e^{-x}}$ | $(0, \infty)$ | $C^\infty$ |
| Exponential linear unit (ELU)[10] |  | $\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter $\alpha$ | $\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$ | $(-\alpha, \infty)$ | $\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$ |
| Scaled exponential linear unit (SELU)[11] |  | $\lambda\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$ | $\lambda\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$ | $(-\lambda\alpha, \infty)$ | $C^0$ |
| Leaky rectified linear unit (Leaky ReLU)[12] |  | $\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ | $\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$ | $(-\infty, \infty)$ | $C^0$ |

And many more!!

# Universal Approximation Theorem

- “A neural network can approximate any continuous function”

**Theorem 1.** *Let $\sigma$ be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(\overset{w_j}{y_j^{T}} x + \overset{b_j}{\theta_j}) \qquad (2)$$

*are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which*

$$|G(x) - f(x)| < \varepsilon \qquad \text{for all} \quad x \in I_n.$$

Cybenko G, “Approximation by Superpositions of a Sigmoidal Function”

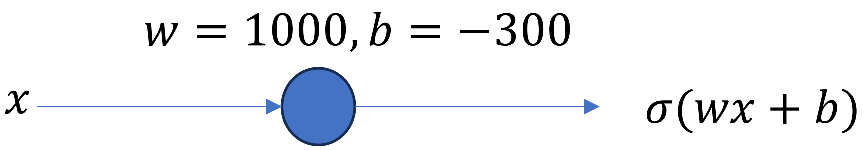
# Universal Approximation Theorem

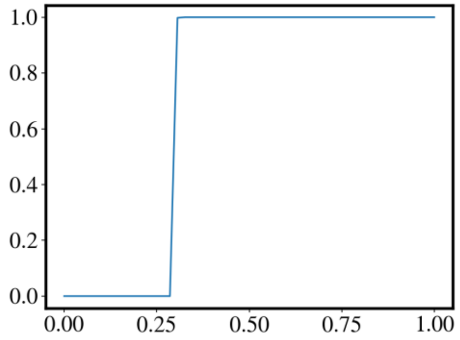
- The theorem states that $\exists N$ and $\exists w_j, \theta_j, \alpha_j$ for $j = 1,2, \ldots, N$ such that

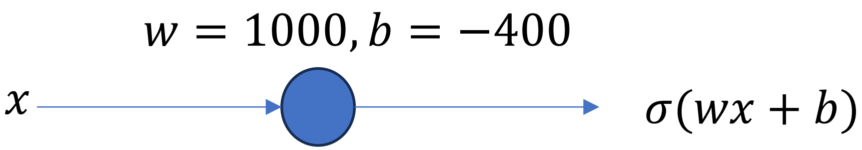
$$|G(x) - f(x)| < \epsilon$$

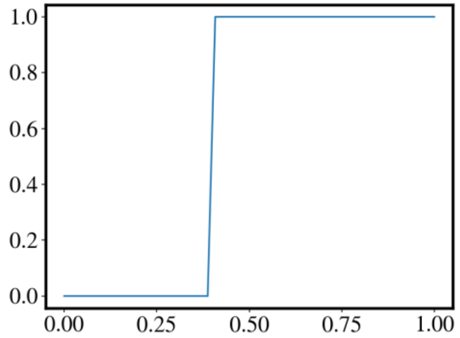
Sigmoid: $\sigma\left(w_j x + b_j\right)$

# Universal Approximation Theorem



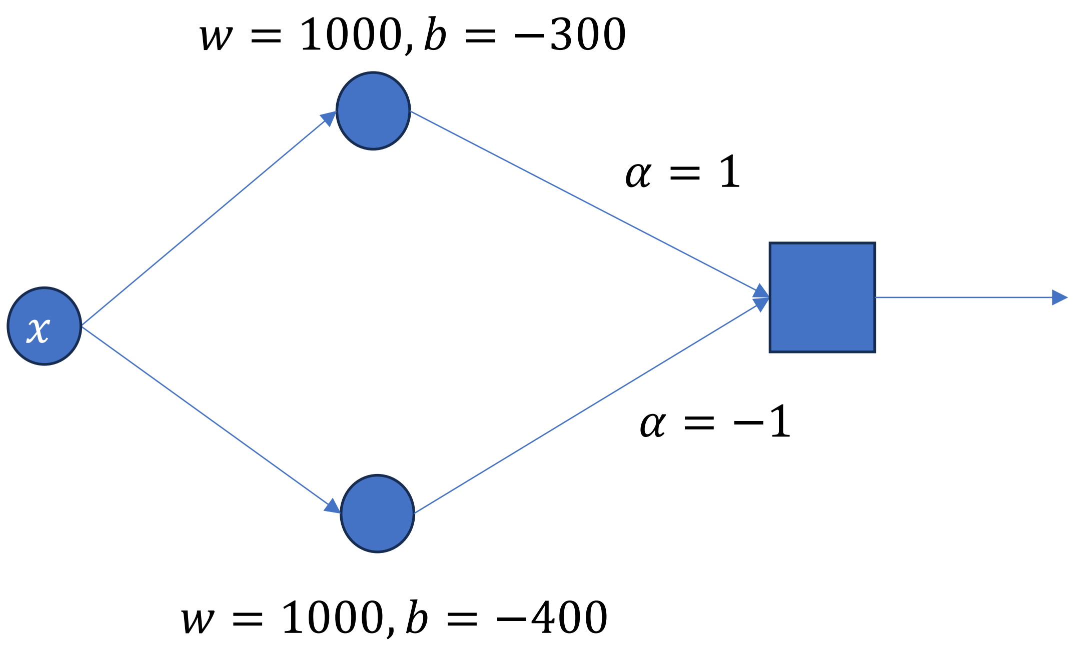



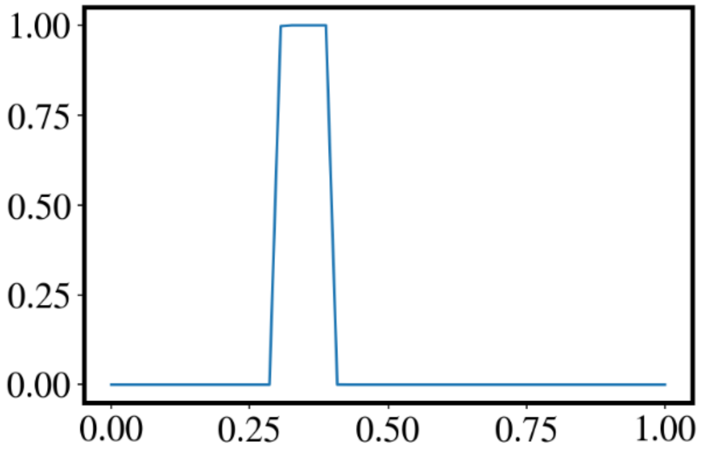




Piece

# Universal Approximation Theorem



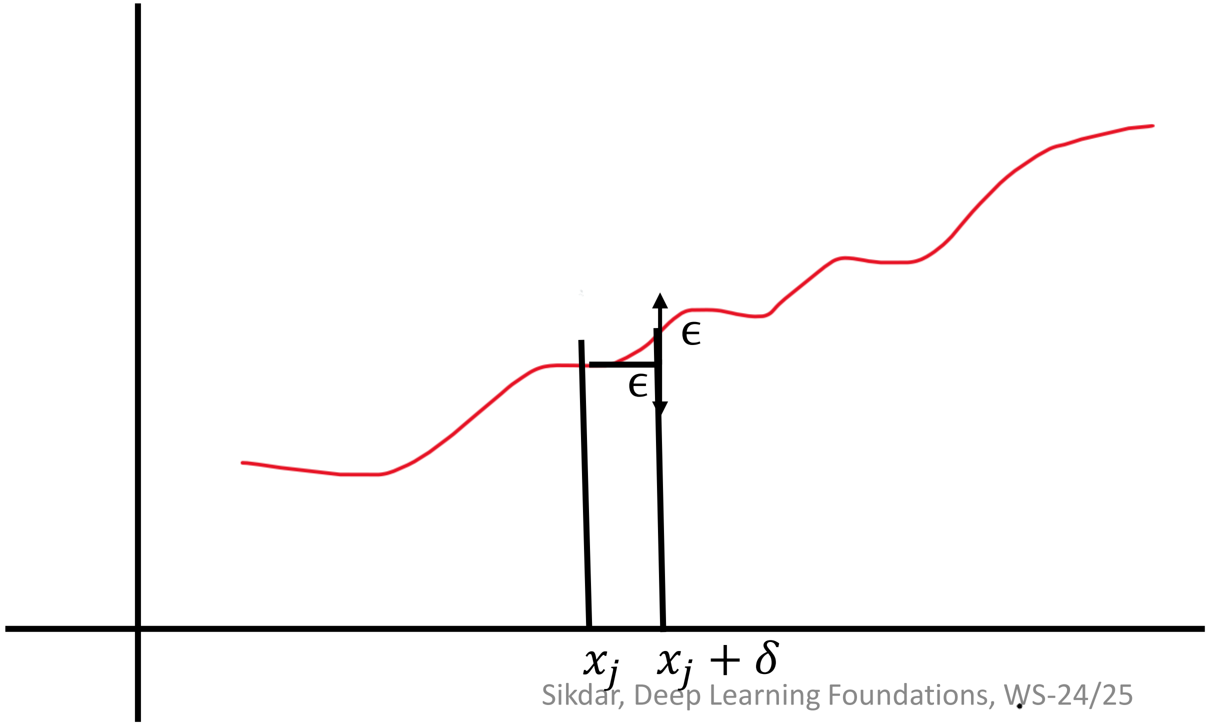

$\sigma(1000x - 300) - \sigma(1000x - 400)$



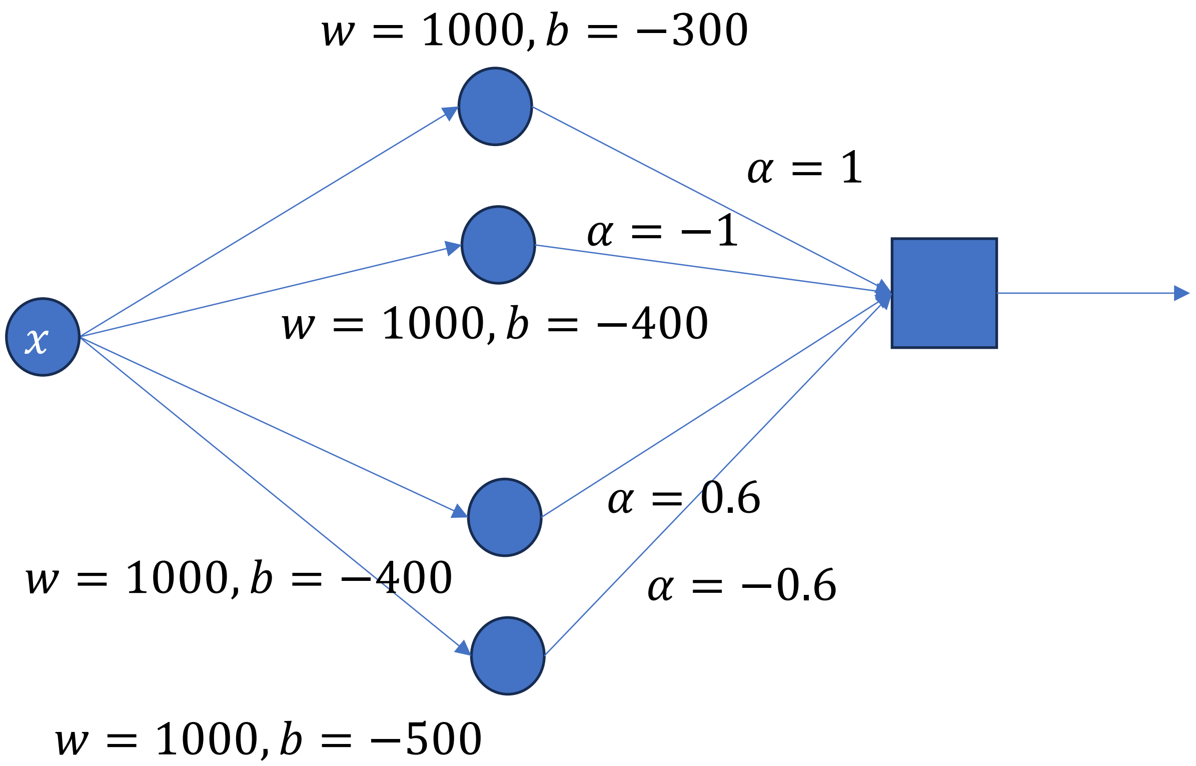
# Universal Approximation Theorem

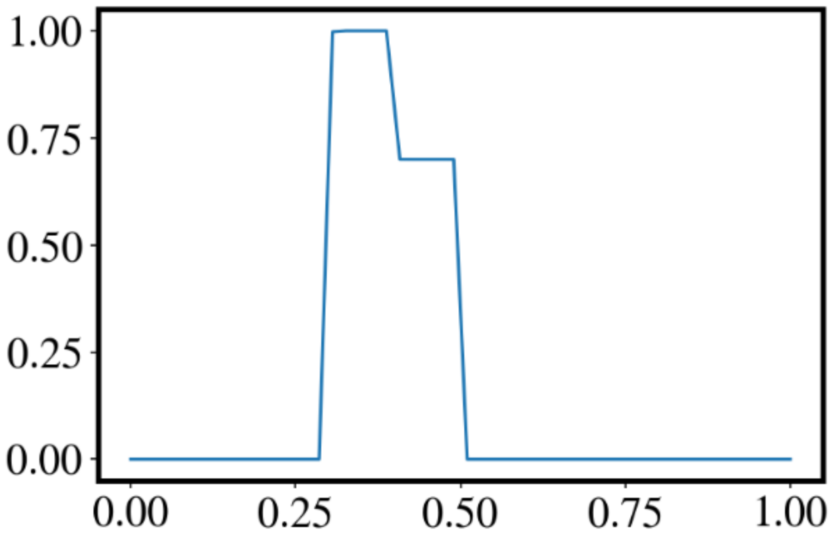
- The theorem states that $\exists N$ and $\exists w_j, b_j, \alpha_j$ for $j = 1,2, \ldots, N$ such that

$$|G(x) - f(x)| < \epsilon$$



# Universal Approximation Theorem

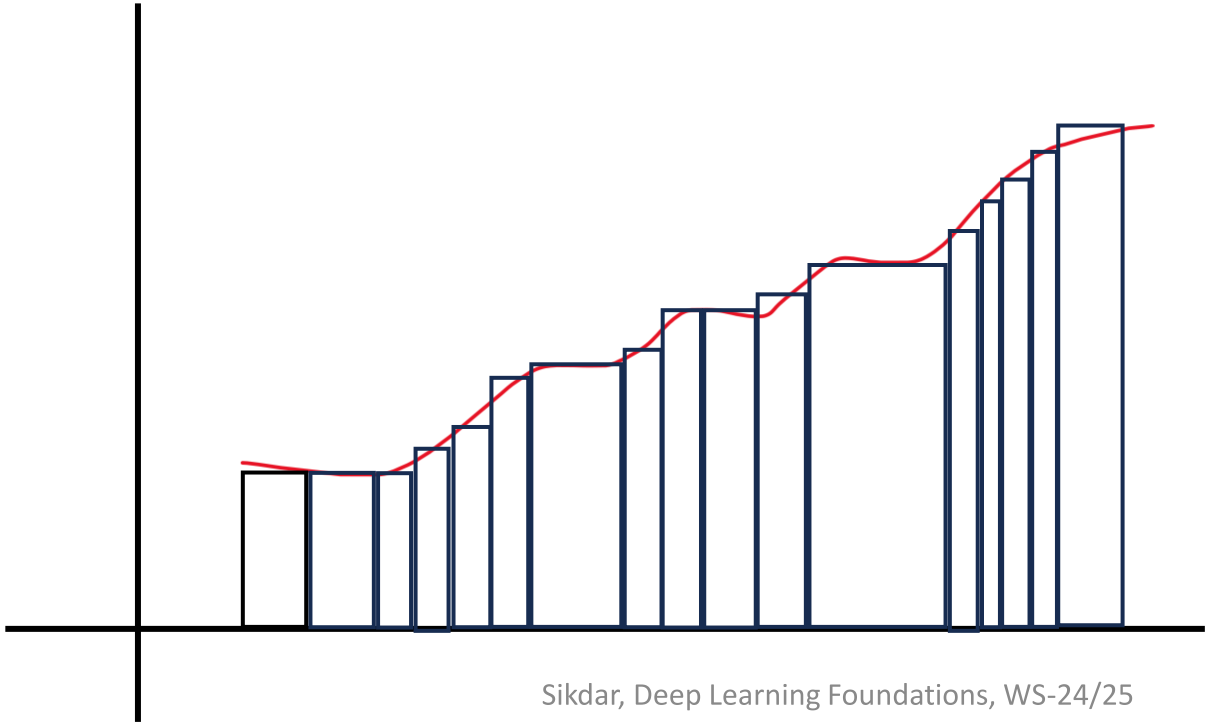


# Universal Approximation Theorem

- The theorem states that $\exists N$ and $\exists w_j, \theta_j, \alpha_j$ for $j = 1,2, \ldots, N$ such that

$$|G(x) - f(x)| < \epsilon$$



# Training/Optimization

$$\min_{W} \frac{1}{n} \sum_{i=1}^{n} l(f_W(x^i), y^i)$$

- Use a search algorithm that starts with some initial guess and repeatedly changes $W$ such that the loss gets smaller and smaller until we reach a point where the loss is minimized
- Gradient descent algorithm

# Training/Optimization

$$L(W) = \frac{1}{n}\sum_{i=1}^{n} l(f_W(x^i), y^i)$$

- Gradient descent algorithm

$$W_j = W_j - \alpha \frac{\partial}{\partial W_j} L(W)$$

Learning rate

# Training/Optimization

- Computing the gradient

$$\frac{\partial}{\partial \theta_j} L(W) = \frac{\partial}{\partial W_j} \frac{1}{2}(f_W(x) - y)^2$$

$$= 2.\frac{1}{2}(f_W(x) - y).\frac{\partial}{\partial W_j}(f_W(x) - y)$$

If we consider the linear model: $f_W(x) = W^T x$

$$= (f_W(x) - y).\frac{\partial}{\partial W_j}(\sum_i W_i x_i \ - y)$$

$$= (f_W(x) - y)x_j$$

# Training/Optimization

```
Loop {

        for i=1 to n, {
```
$$W_j = W_j - \alpha\left(f_W(x^i) - y^i\right)x_j^i \quad \texttt{(for every j)}$$
```
        }

    }
```

# Evaluation

- How do we know that the trained model is good?
- Split the data into train/validation/test


- Training only on the train split
- Periodically check performance on validation split
- Select model with the best performance in validation split
- Evaluate on the test split

# Evaluation

- Measuring performance
- Regression:
  - Mean squared error, Mean absolute error
- Classification
  - Accuracy: fraction of cases where the predicted labels are correct

# Evaluation

- K-fold cross validation
- Simple training-and-test can be subject to random fluctuations
- Better: (ten-fold) cross-validation
  - Split labeled randomly data into 10 parts
  - Use 9 parts for training, last part for testing
  - Repeat 10 times, each part is used for testing once
  - Average results
- Applied if limited training data is available and training time is reasonable

# Summary

- Supervised learning
  - Linear/shallow neural models
  - Loss
  - Optimization
- Universal approximation theorem
- Next time
  - Optimization and regularization of deep neural models