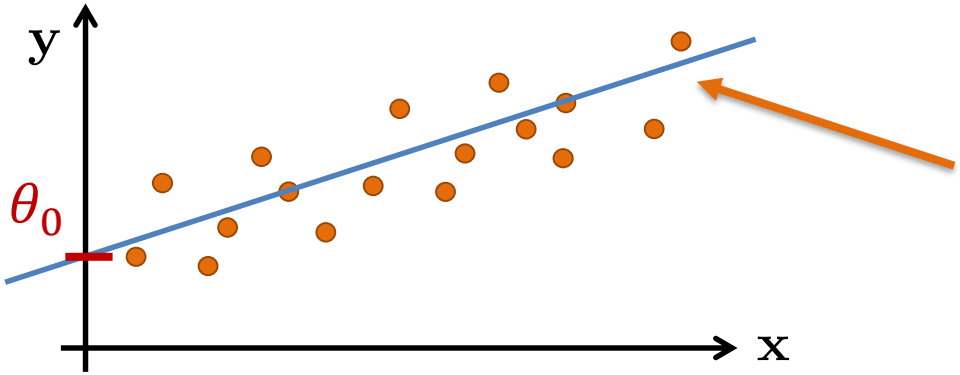
Introduction to Neural Networks

From Linear and Logistic Regression to Neural Networks

Linear Regression

= a **supervised learning** method to find a **linear model** of the form

$$\hat{y}_i = \theta_0 + \sum_{j=1}^{d} x_{ij}\theta_j = \theta_0 + x_{i1}\theta_1 + x_{i2}\theta_2 + \cdots + x_{id}\theta_d$$



Goal: find a model that explains a target y given the input x

Logistic Regression

= a **supervised learning** method for binary classification

- Loss per training sample

$$\mathcal{L}(\hat{y}_i, y_i) = -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

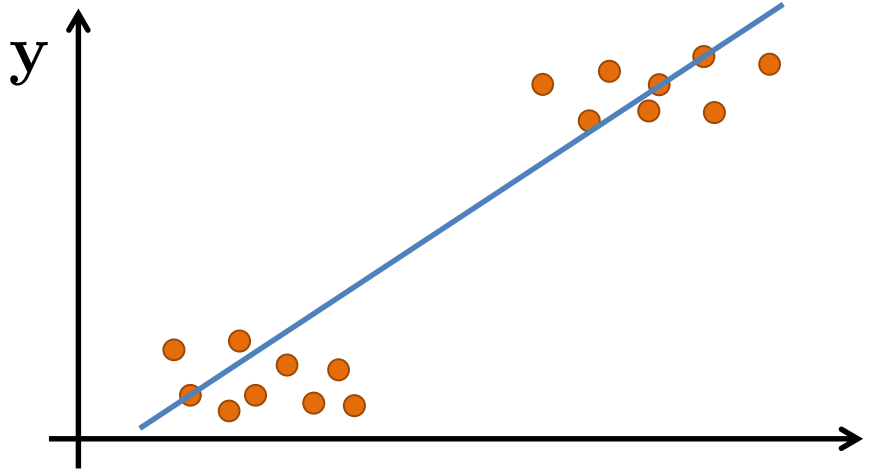
- Overall loss

$$\mathcal{C}(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} (y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

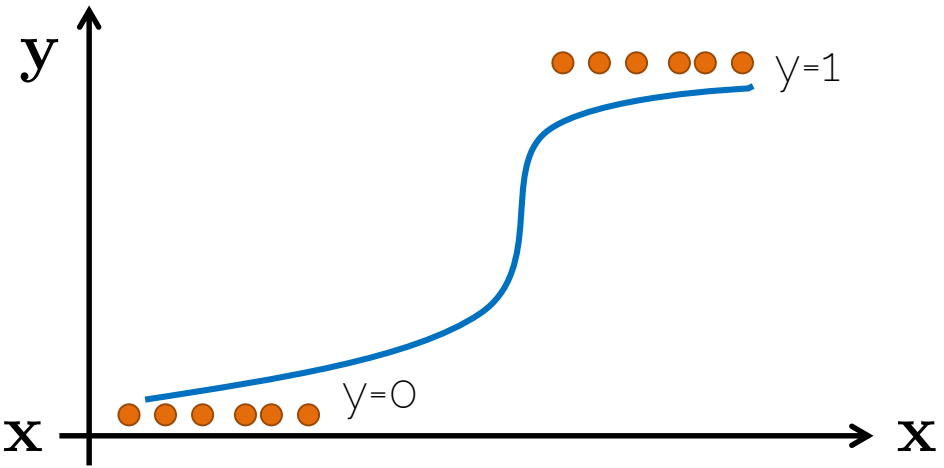
Minimization

$\hat{y}_i = \sigma(x_i \boldsymbol{\theta})$

Linear vs Logistic Regression

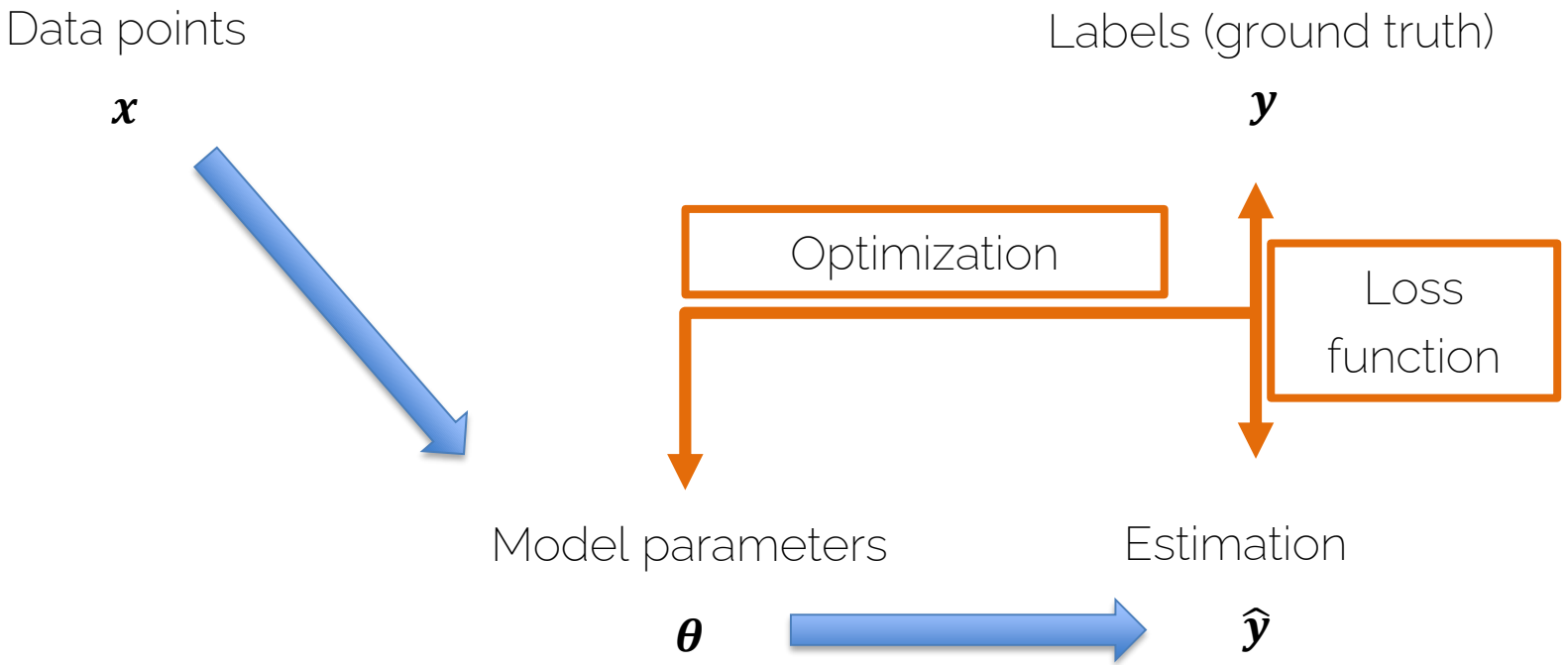


Predictions can exceed the range of the training samples
→ in the case of classification [0;1] this becomes a real issue



Predictions are guaranteed to be within [0;1]

How to obtain the Model?



Linear Score Functions

- Linear score function as seen in linear regression

$$f_i = \sum_j w_{i,j} x_j$$

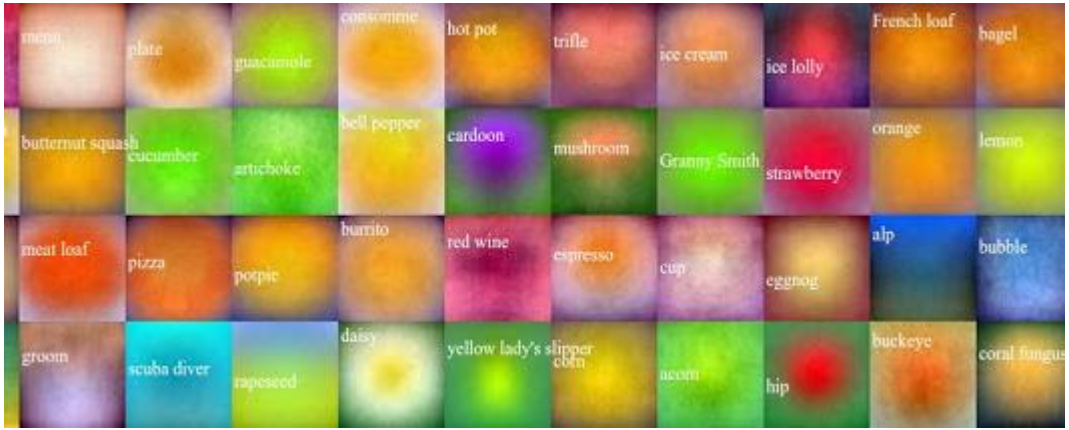
$$\boldsymbol{f} = \boldsymbol{W}\,\boldsymbol{x} \qquad \text{(Matrix Notation)}$$

Linear Score Functions on Images

- Linear score function $\boldsymbol{f} = \boldsymbol{W}\boldsymbol{x}$



On CIFAR-10

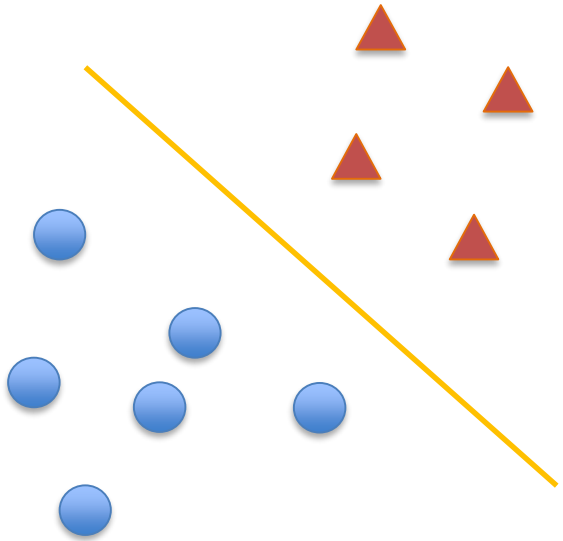


On ImageNet

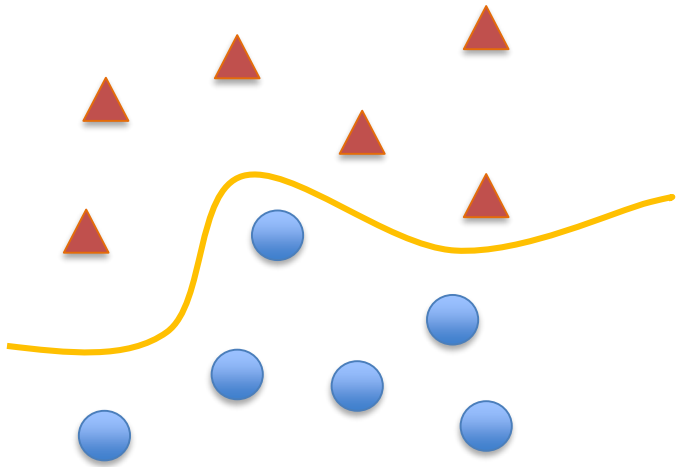
Source:: Li/Karpathy/Johnson

Linear Score Functions?

Logistic Regression

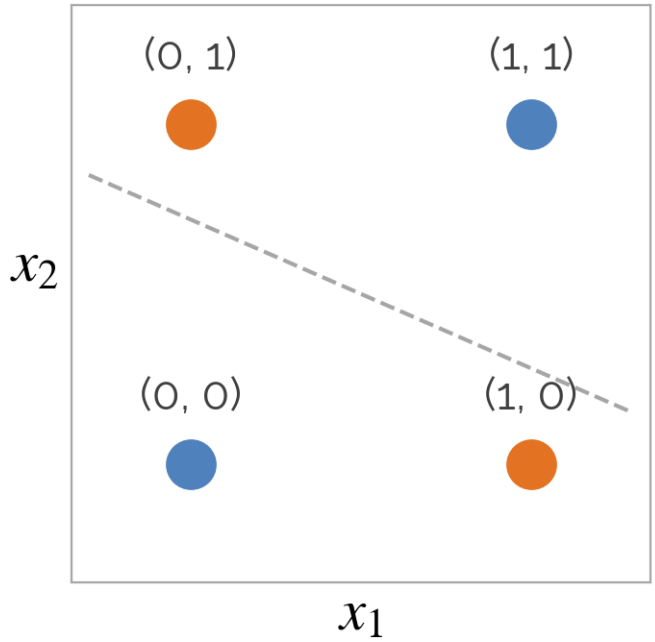


Linear Separation Impossible!



Linear Score Functions?

XOR: no line separates the two classes



Linear Score Functions?

- Can we make linear regression better?
  - Naïve idea: Multiply with another weight matrix $\boldsymbol{W_2}$

$$\hat{\boldsymbol{f}} = \boldsymbol{W_2} \cdot \boldsymbol{W_1} \cdot \boldsymbol{x}$$

- Operation remains linear:

$$\boldsymbol{W} = \boldsymbol{W_2} \cdot \boldsymbol{W_1}$$

$$\hat{\boldsymbol{f}} = \boldsymbol{W}\boldsymbol{x}$$

- Solution → add non-linearity!!

Neural Network

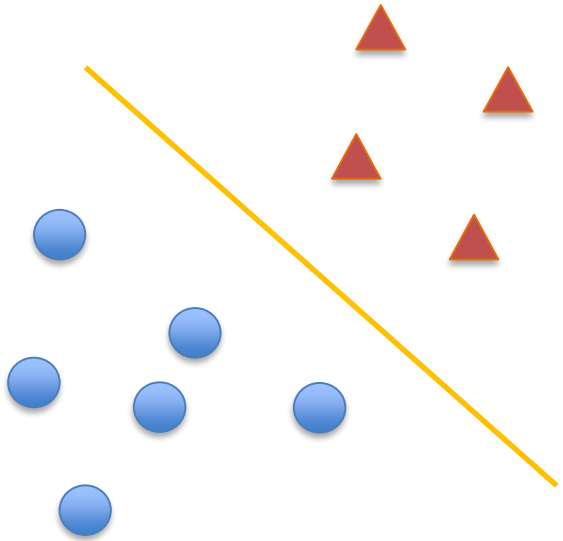
- Linear score function $\boldsymbol{f = Wx}$

- Neural network is a nesting of 'functions'
  - 2-layers: $\boldsymbol{f = W_2 \max(0, W_1 x)}$
  - 3-layers: $\boldsymbol{f = W_3 \max(0, W_2 \max(0, W_1 x))}$
  - 4-layers: $\boldsymbol{f = W_4 \tanh(W_3 \max(0, W_2 \max(0, W_1 x)))}$
  - 5-layers: $\boldsymbol{f = W_5 \sigma(W_4 \tanh(W_3 \max(0, W_2 \max(0, W_1 x))))}$
  - … up to hundreds of layers

Neural Networks

Neural Network

Logistic Regression



Neural Networks



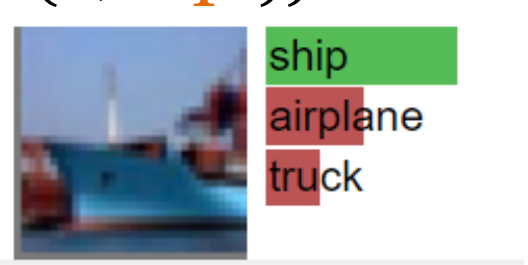
Stacking linear layers with activations enables us to fit **nonlinear decision boundaries**!

Neural Network

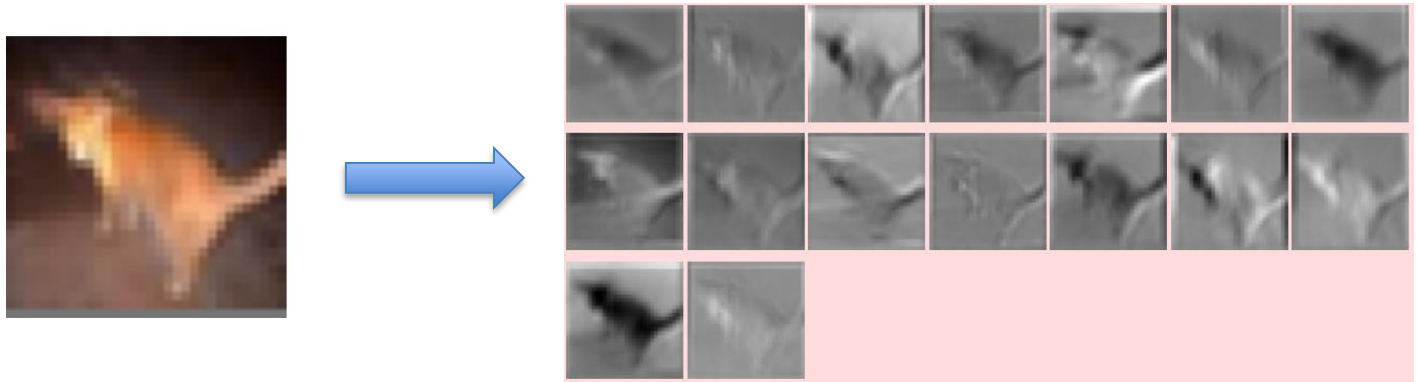
- Non-linear score function $\boldsymbol{f = \ldots(\max(0, W_1 x))}$







On CIFAR-10

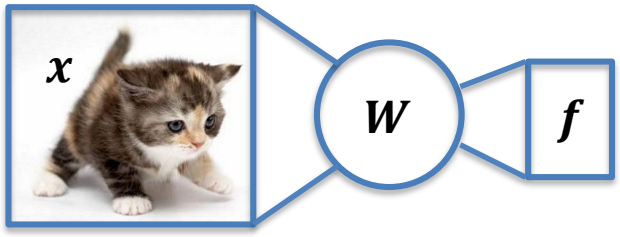


Visualizing activations of the first layer.

Source: ConvNetJS

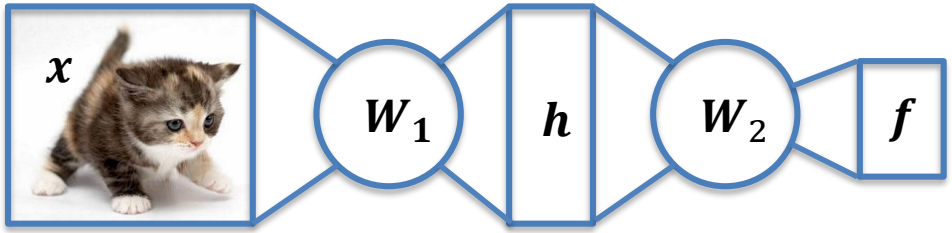
Neural Network

1-layer network: $\boldsymbol{f = Wx}$



$128 \times 128 \times 3 = 49152$ 10

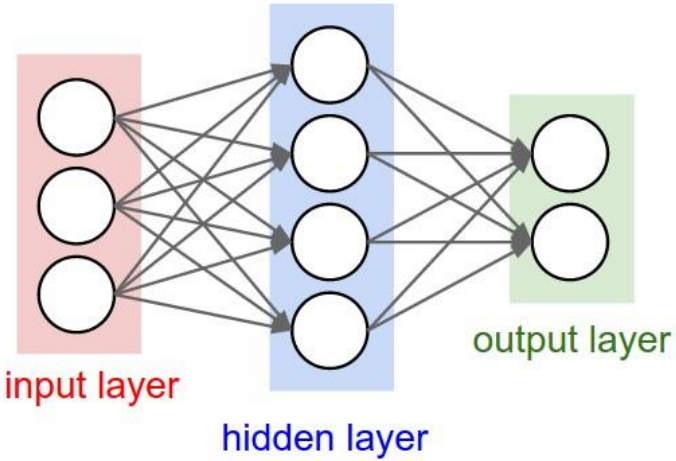
2-layer network: $\boldsymbol{f = W_2 \max(0, W_1 x)}$



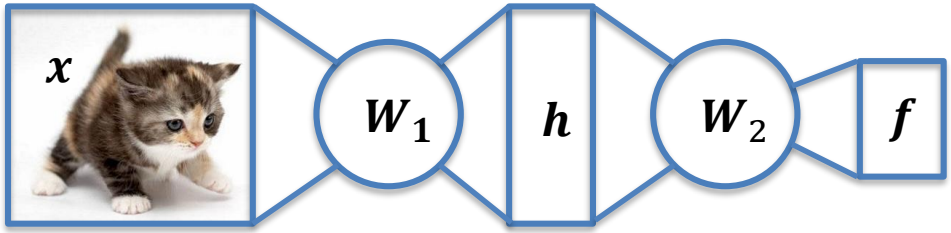
$128 \times 128 \times 3 = 49152$ 1000 10

Why is this structure useful?

Neural Network



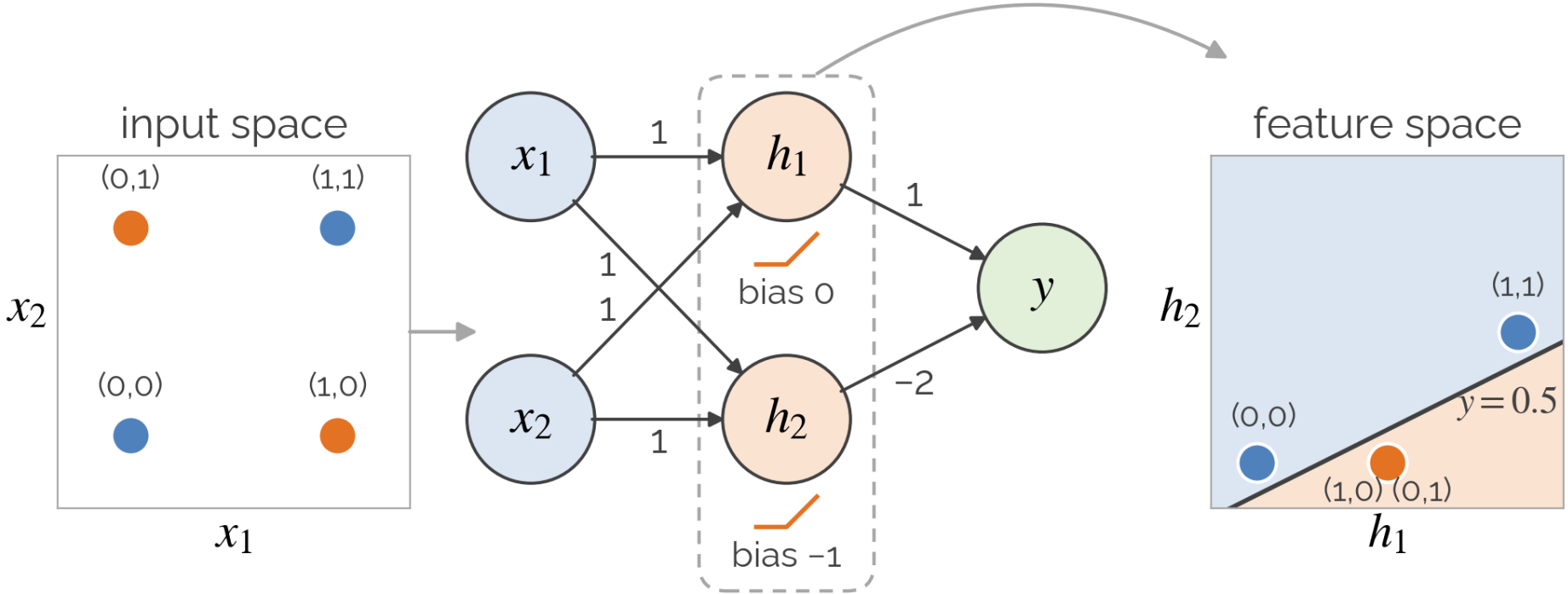
2-layer network: $\boldsymbol{f = W_2 \max(0, W_1 x)}$



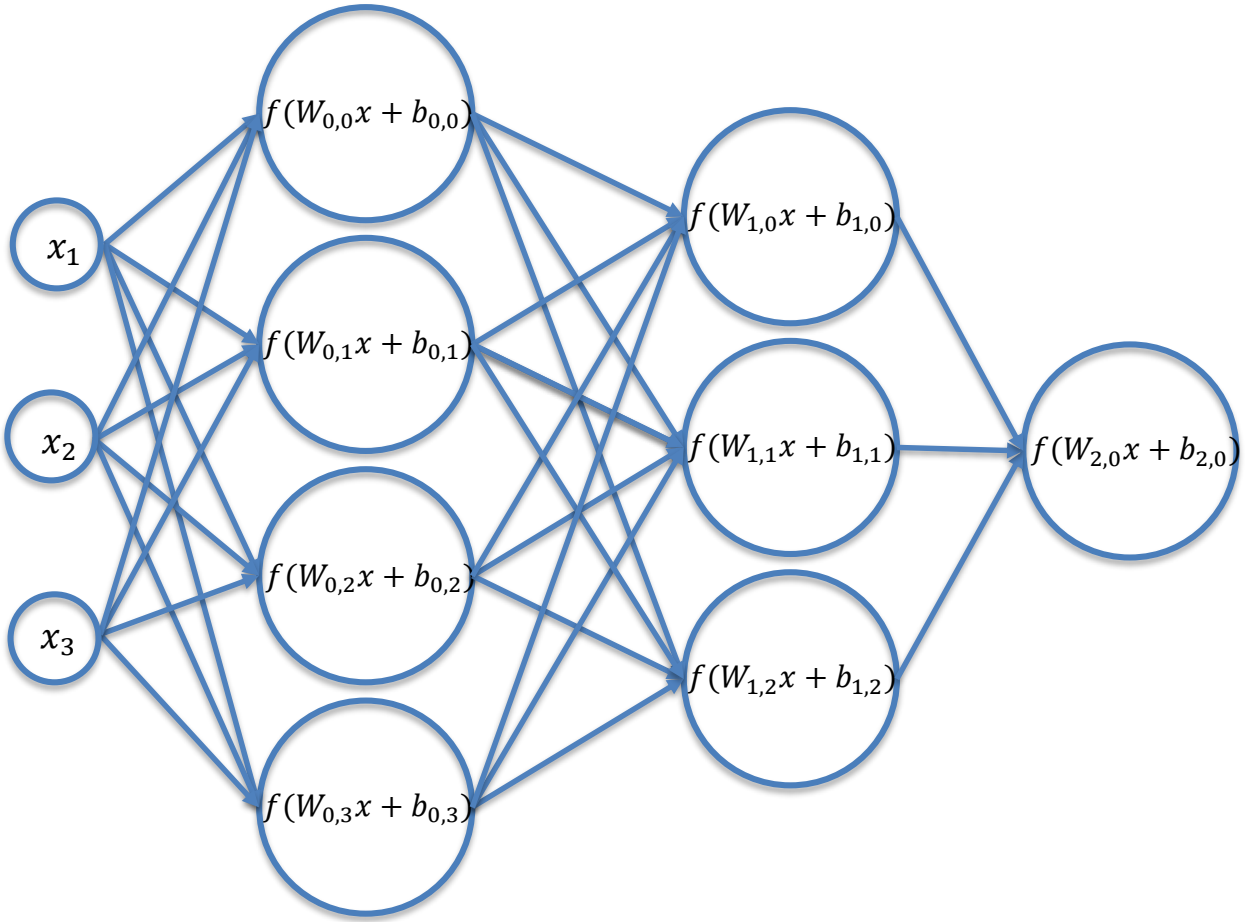
$128 \times 128 \times 3 = 49152$ $\quad$ 1000 $\quad$ 10

Input Layer $\quad$ Hidden Layer $\quad$ Output Layer

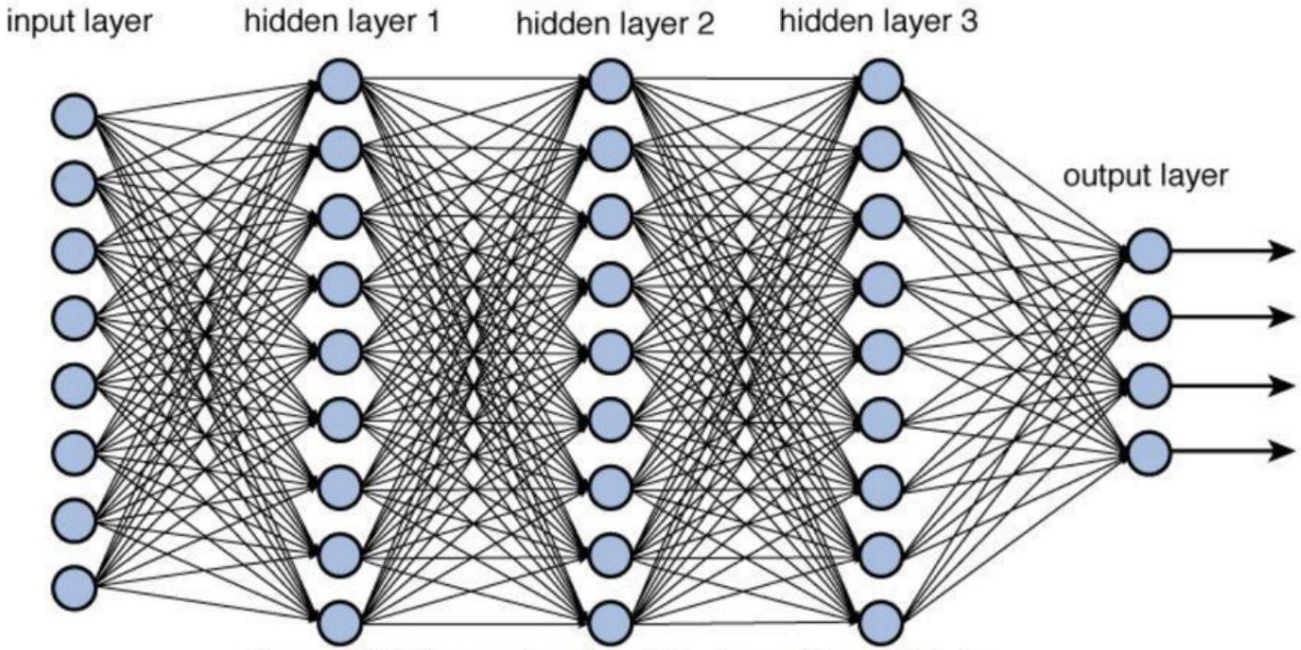
Neural Network



Net of Artificial Neurons



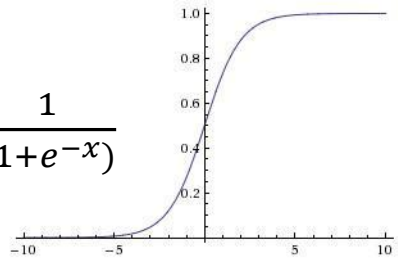
Neural Network



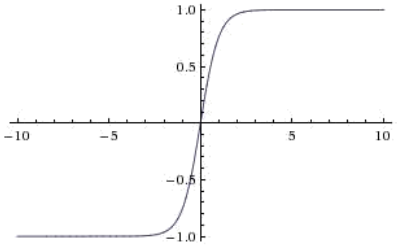
Source: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Activation Functions

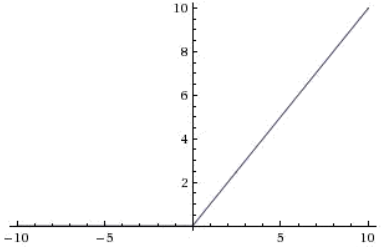
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



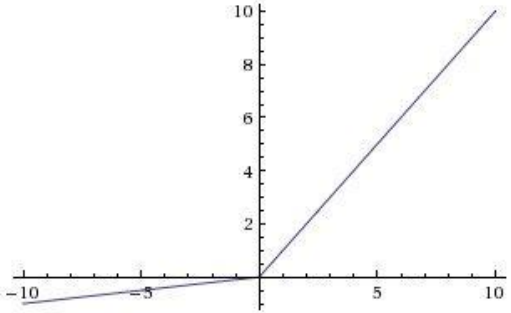
tanh: $\tanh(x)$



ReLU: $\max(0, x)$



Leaky ReLU: $\max(0.01x, x)$



Parametric ReLU: $\max(\alpha x, x)$

Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$

Neural Network

$$\boldsymbol{f = W_3 \cdot (W_2 \cdot (W_1 \cdot x)))}$$

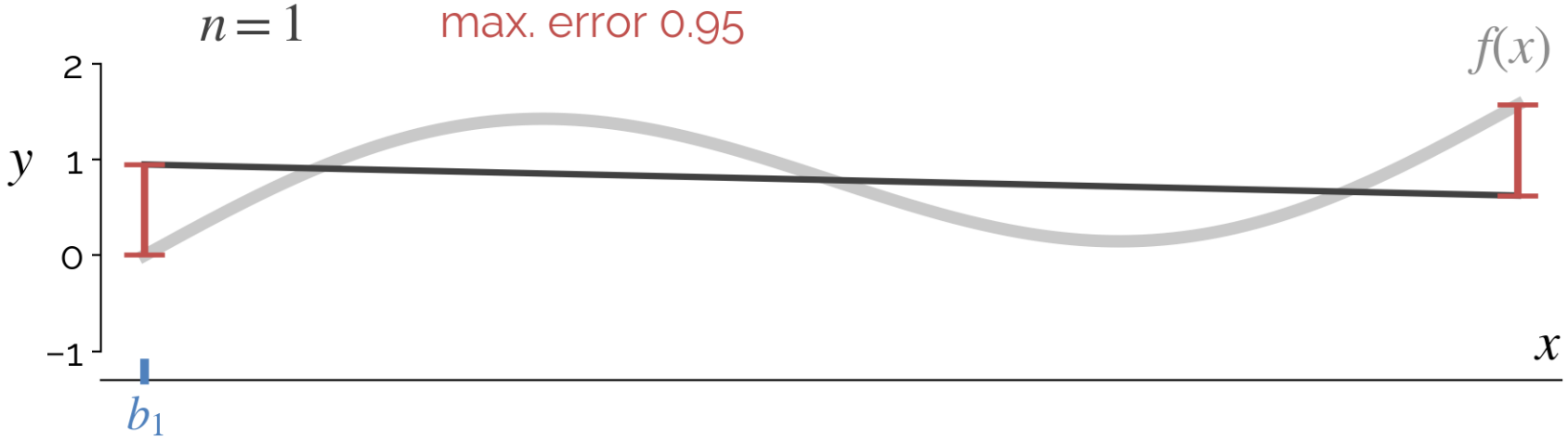
Why activation functions?

Simply concatenating linear layers would be so much cheaper…

Universal Approximation

One hidden layer, n activations (ReLUs):

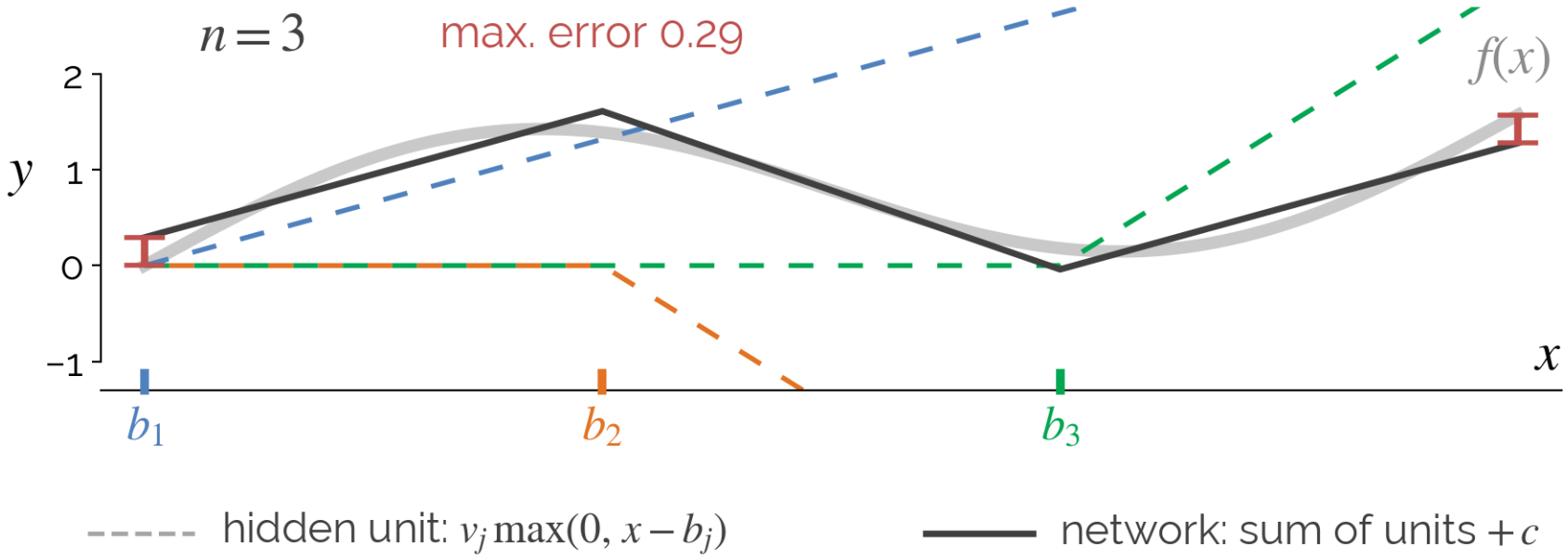
$$\hat{y}(x) = v_1 \max(0, x - b_1) + \ldots + v_n \max(0, x - b_n) + c$$



Universal Approximation

One hidden layer, n activations (ReLUs):

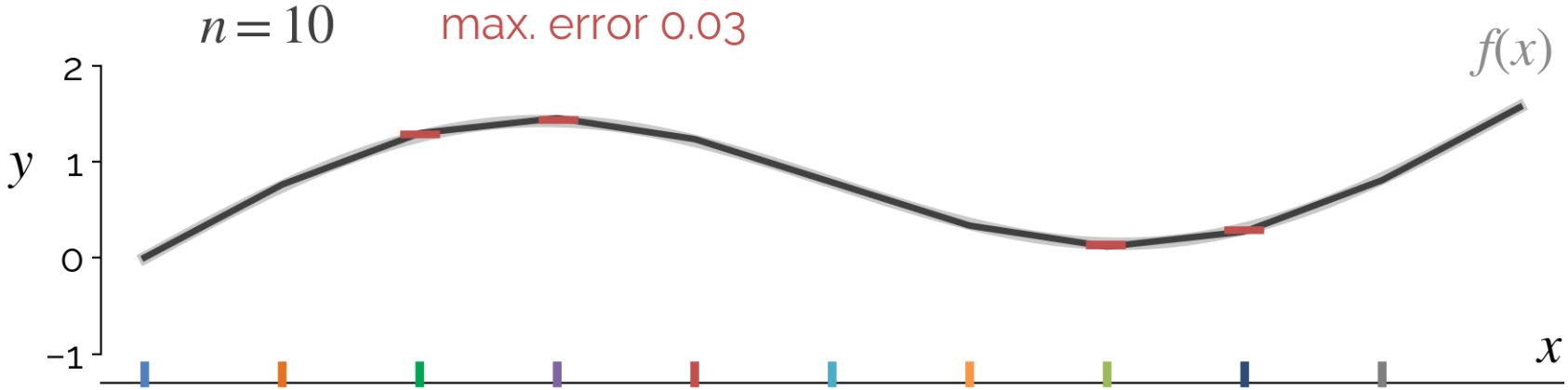
$$\hat{y}(x) = v_1 \max(0, x - b_1) + \ldots + v_n \max(0, x - b_n) + c$$



Universal Approximation

One hidden layer, n activations (ReLUs):

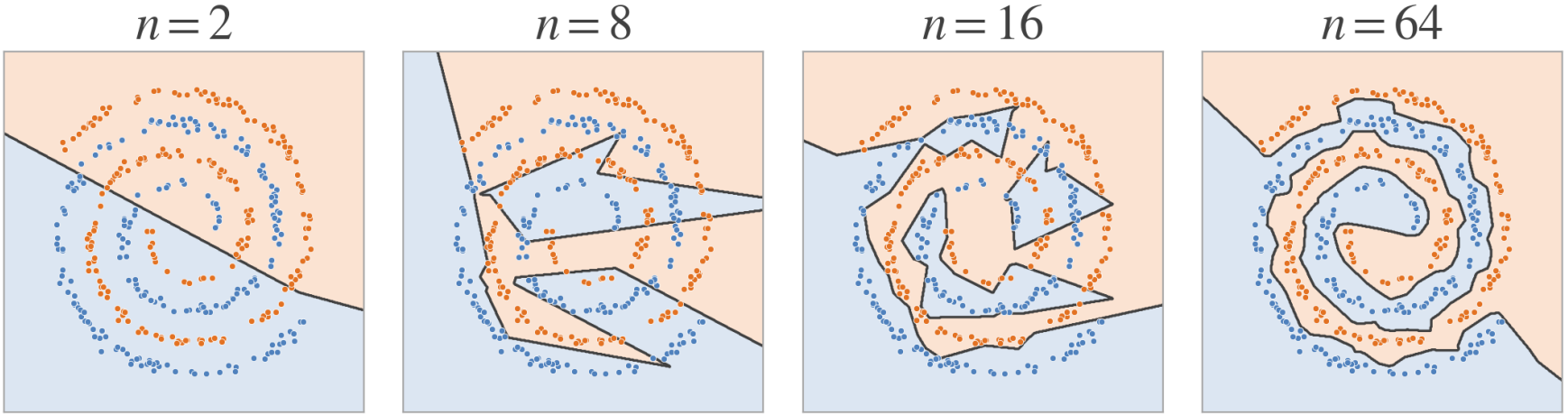
$$\hat{y}(x) = v_1 \max(0, x - b_1) + \ldots + v_n \max(0, x - b_n) + c$$



Universal approximation theorem: one hidden layer with enough units approximates any continuous function arbitrarily well [Cybenko 1989; Leshno et al. 1993]

Universal Approximation

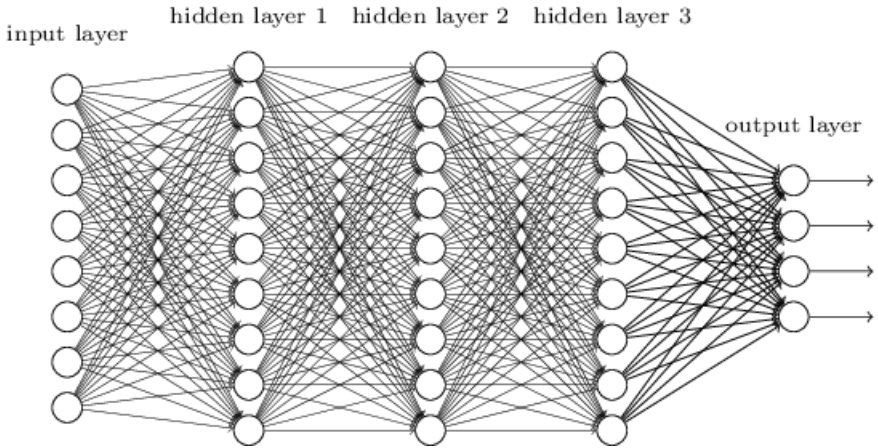
Classifier: one hidden layer, n activations (ReLUs)



Try it yourself: playground.tensorflow.org

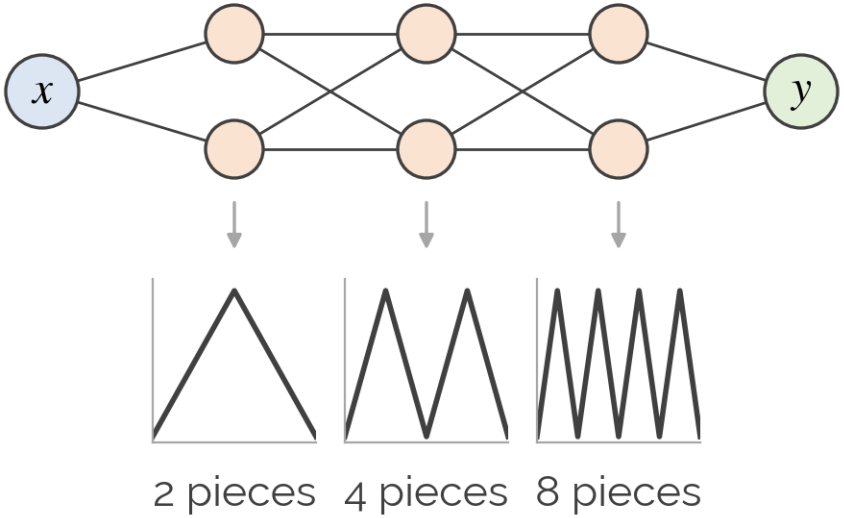
Neural Network

Why organize a neural network into layers?

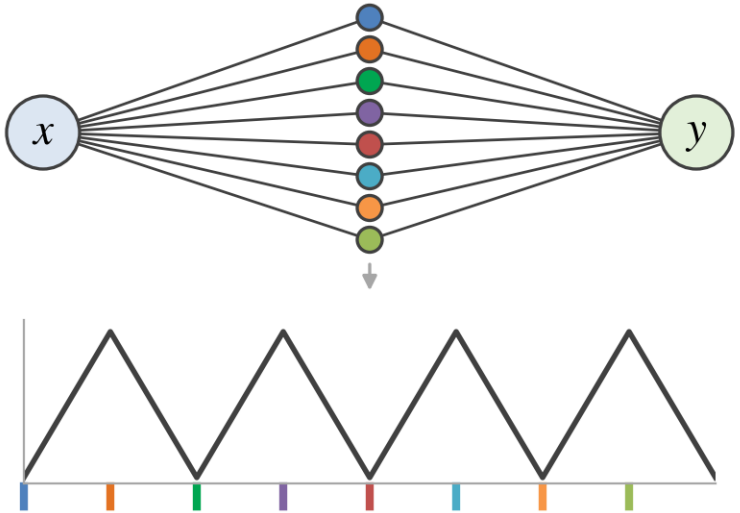


Depth vs. Width

Deep: 3 layers, 2 ReLUs each



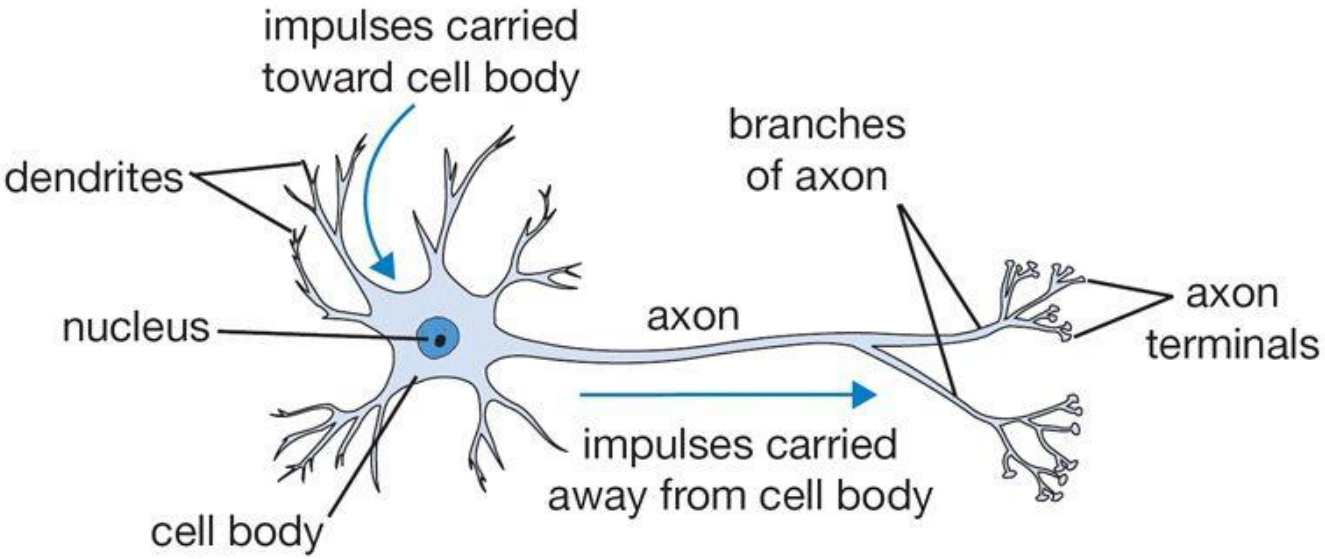
Wide: 1 layer, 8 ReLUs, summed



1024 pieces:

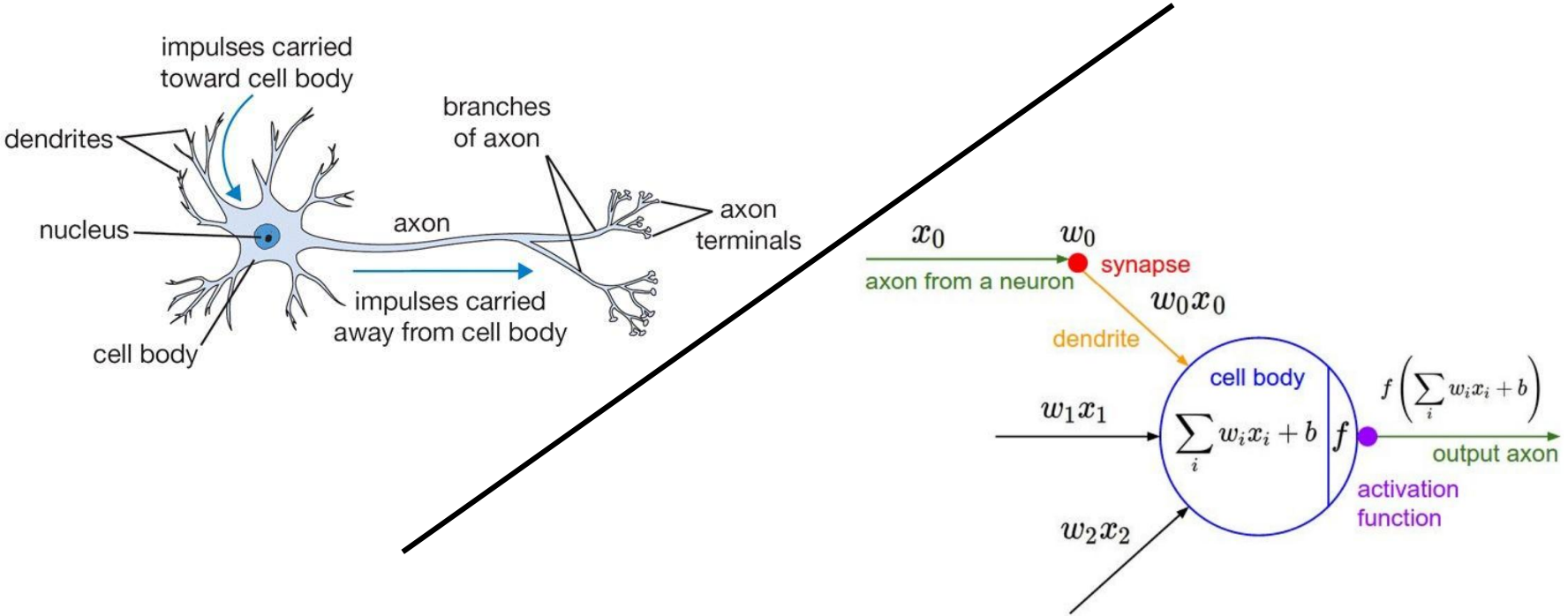


Biological Neurons



Credit: Stanford CS 231n

Biological Neurons



Credit: Stanford CS 231n

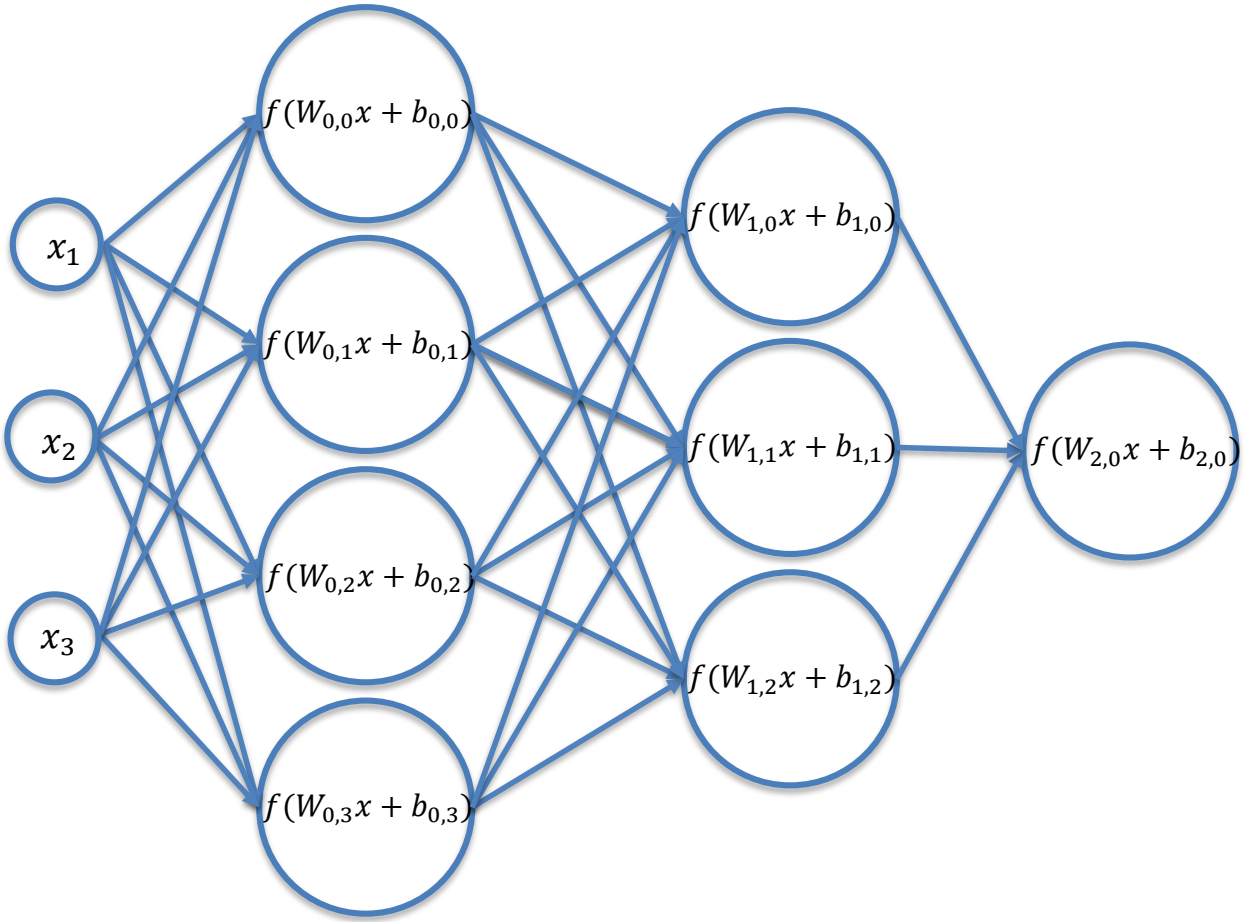
Artificial Neural Networks vs Brain





Artificial neural networks are inspired by the brain,
but not even close in terms of complexity!
The comparison is great for the media and news articles though... ☺

Artificial Neural Network



Neural Network

- Summary
  - Given a dataset with ground truth training pairs $[x_i; y_i]$,
  - Find optimal weights and biases $\boldsymbol{W}$ using stochastic gradient descent, such that the loss function is minimized
    - Compute gradients with backpropagation (use batch-mode; more later)
    - Iterate many times over training set (SGD; more later)

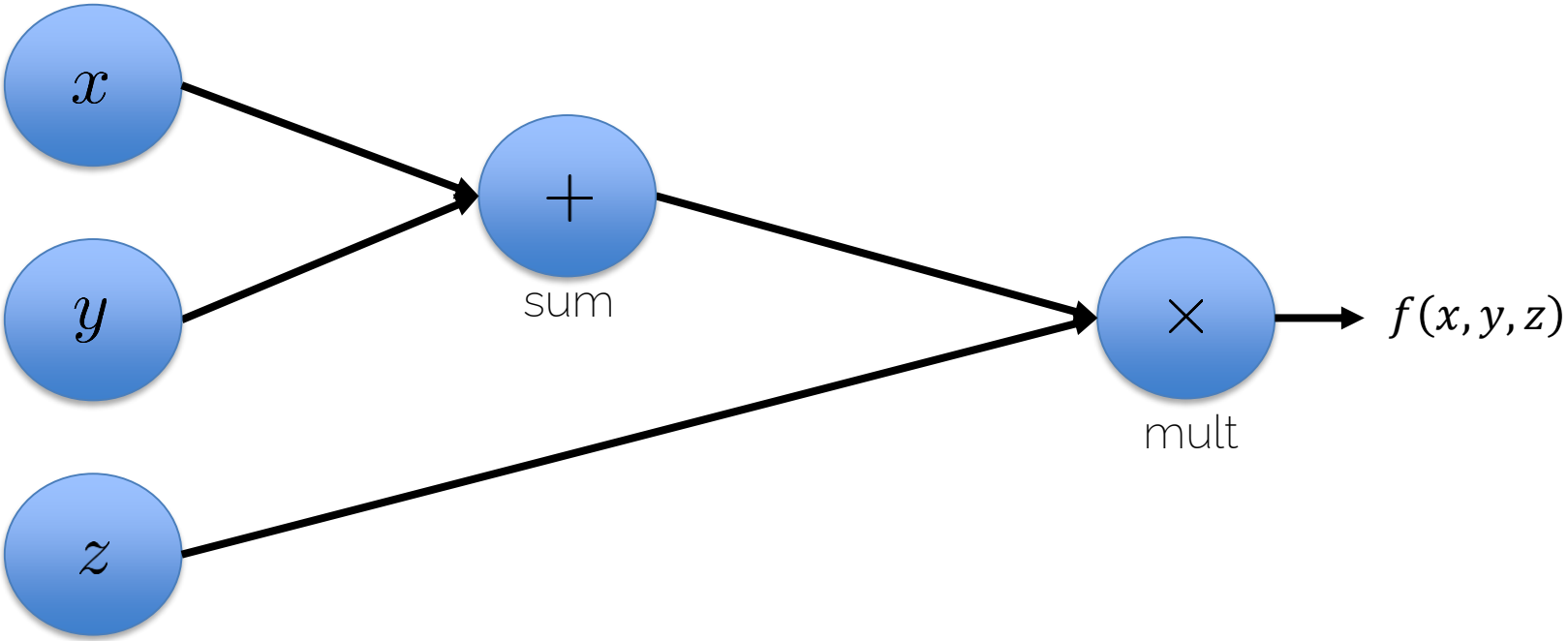
Computational Graphs

Computational Graphs

- Directional graph
- Matrix operations are represented as compute nodes.
- Vertex nodes are variables or operators like +, -, *, /, log(), exp() …
- Directional edges show flow of inputs to vertices

Computational Graphs

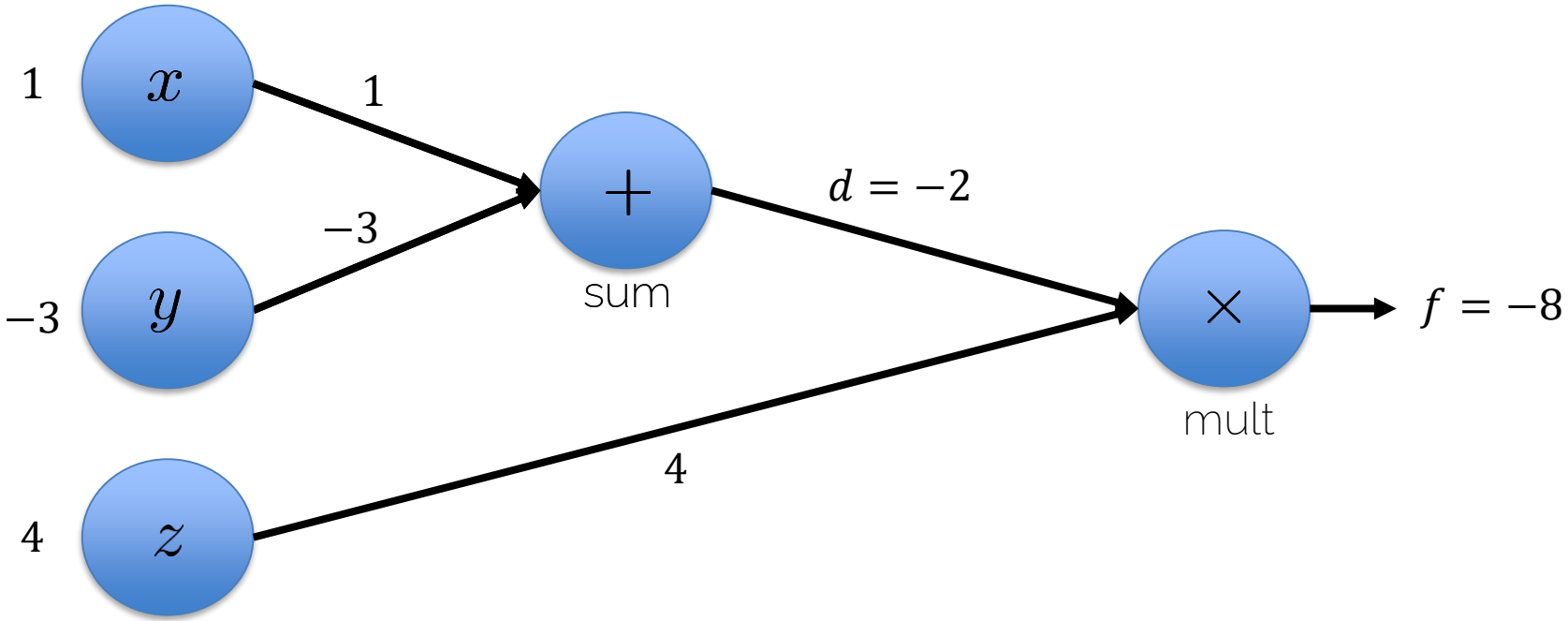
- $f(x, y, z) = (x + y) \cdot z$



Evaluation: Forward Pass

- $f(x, y, z) = (x + y) \cdot z$

Initialization $x = 1, y = -3, z = 4$



Computational Graphs

- Why discuss compute graphs?
- Neural networks have complicated architectures

$$\boldsymbol{f} = \boldsymbol{W_5}\sigma(\boldsymbol{W_4}\tanh(\boldsymbol{W_3}\max(\boldsymbol{0}, \boldsymbol{W_2}\max(\boldsymbol{0}, \boldsymbol{W_1}\boldsymbol{x}))))$$

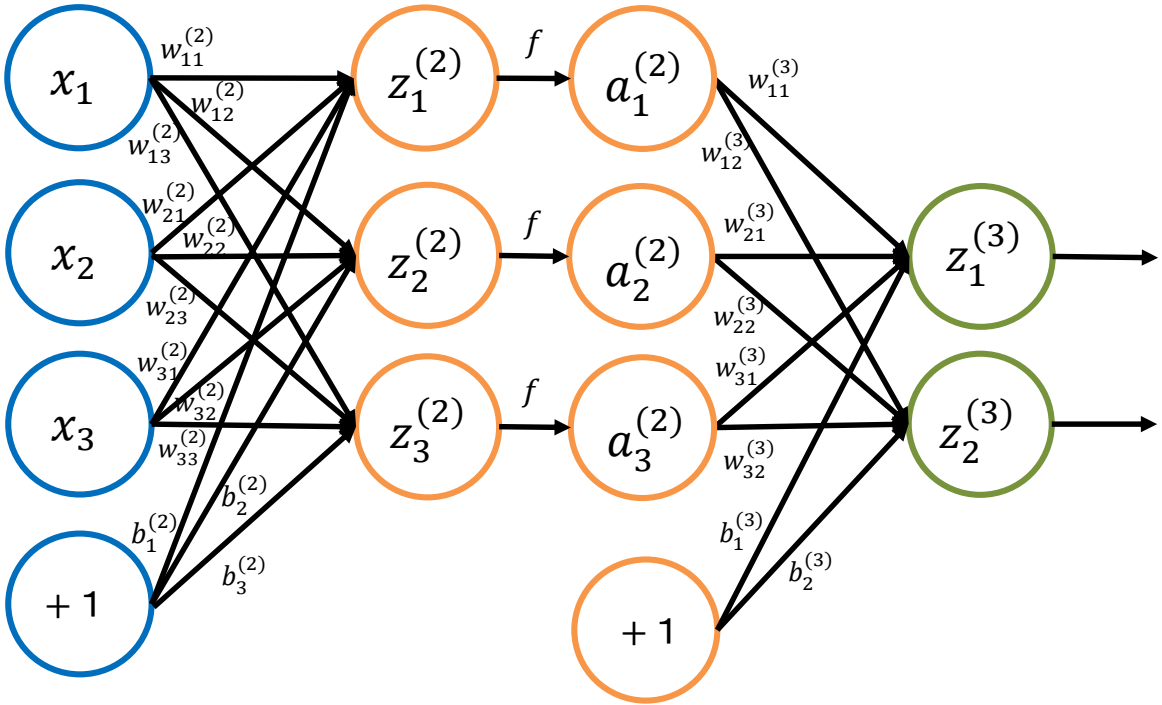
- Lot of matrix operations!
- Represent NN as computational graphs!

Computational Graphs

A neural network can be represented as a computational graph…

- it has compute nodes (operations)
- it has edges that connect nodes (data flow)
- it is directional
- it can be organized into ‘layers’

Computational Graphs



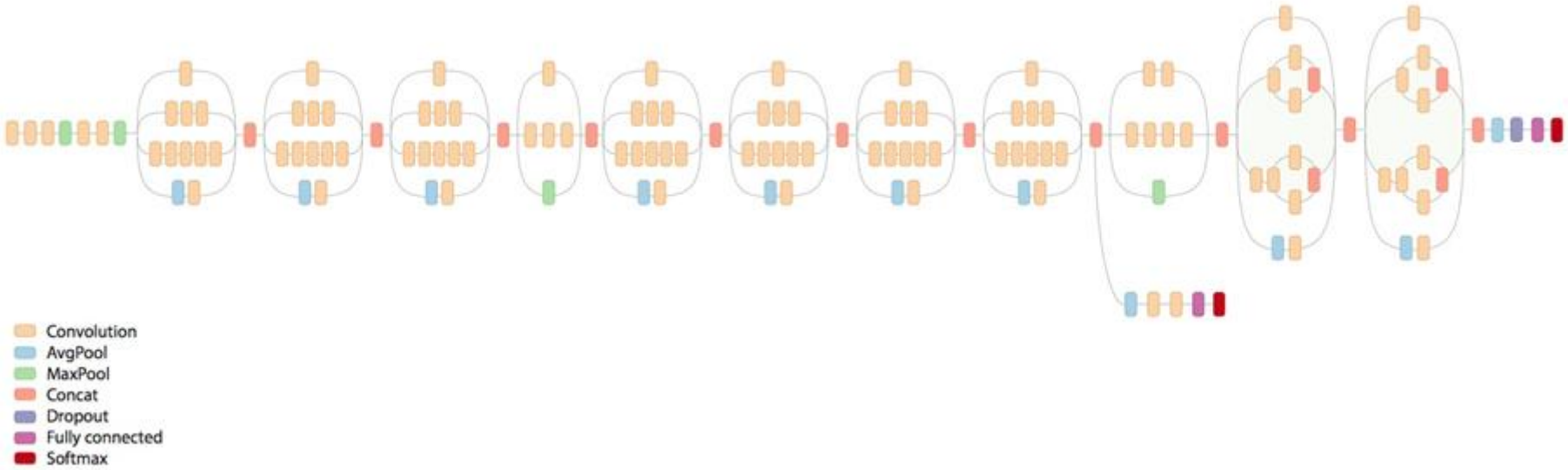
$$z_k^{(2)} = \sum_i x_i w_{ik}^{(2)} + b_k^{(2)}$$

$$a_k^{(2)} = f(z_k^{(2)})$$

$$z_k^{(3)} = \sum_i a_i^{(2)} w_{ik}^{(3)} + b_k^{(3)}$$

Computational Graphs

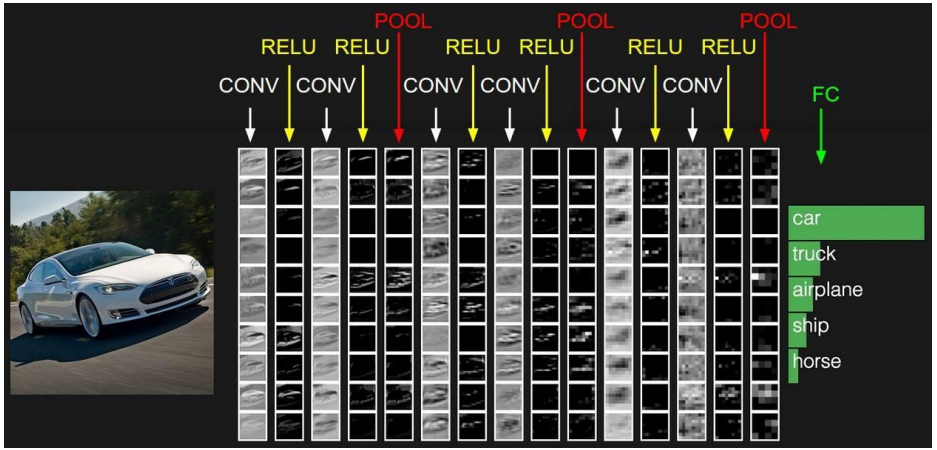
- From a set of neurons to a Structured Compute Pipeline



[Szegedy et al.,CVPR'15] Going Deeper with Convolutions

Computational Graphs

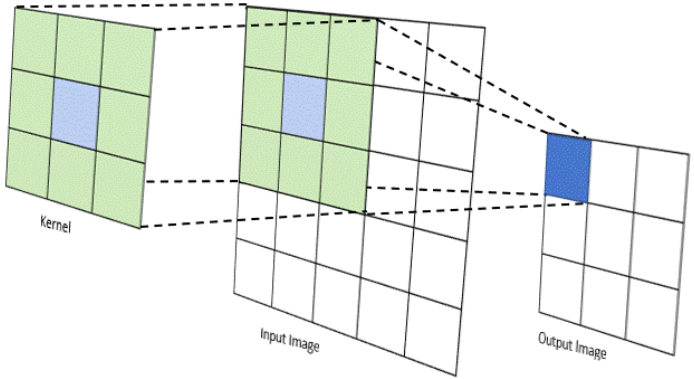
- The computation of Neural Network has further meanings:
  - The multiplication of $\boldsymbol{W}$ and $\boldsymbol{x}$: encode input information
  - The activation function: select the key features



Source; https://www.zybuluo.com/liuhui0803/note/981434

Computational Graphs

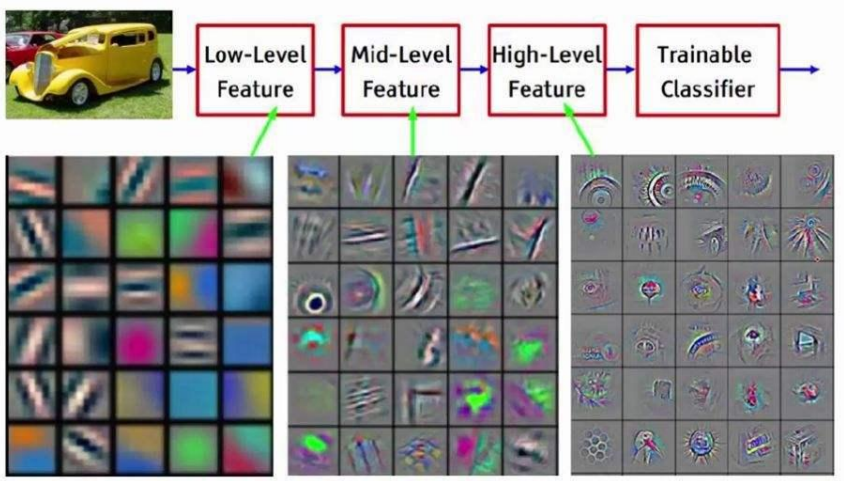
- The computations of Neural Networks have further meanings:
  - The convolutional layers: extract useful features with shared weights



Source: https://medium.com/@timothy_terati/image-convolution-filtering-a54dce7c786b

Computational Graphs

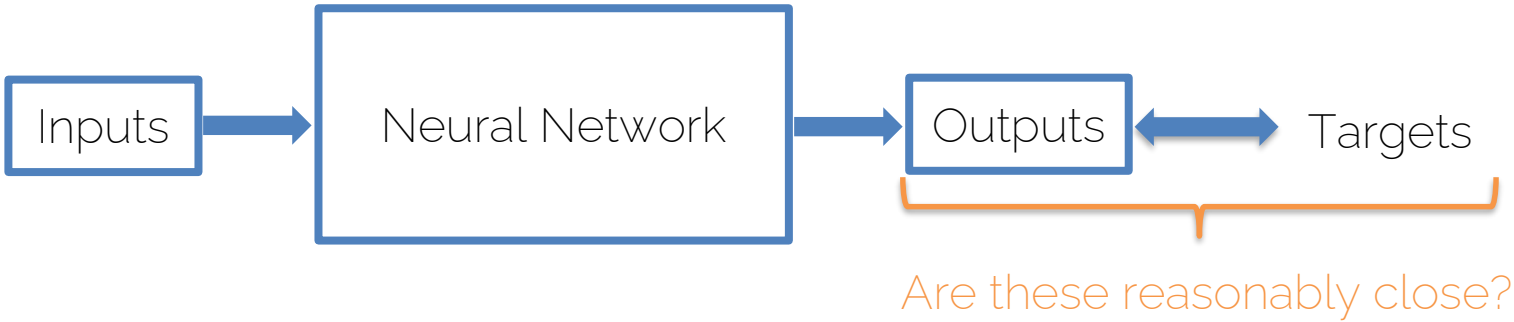
- The computations of Neural Networks have further meanings:
  - The convolutional layers: extract useful features with shared weights



Source: https://www.zybuluo.com/liuhui0803/note/981434

Loss Functions

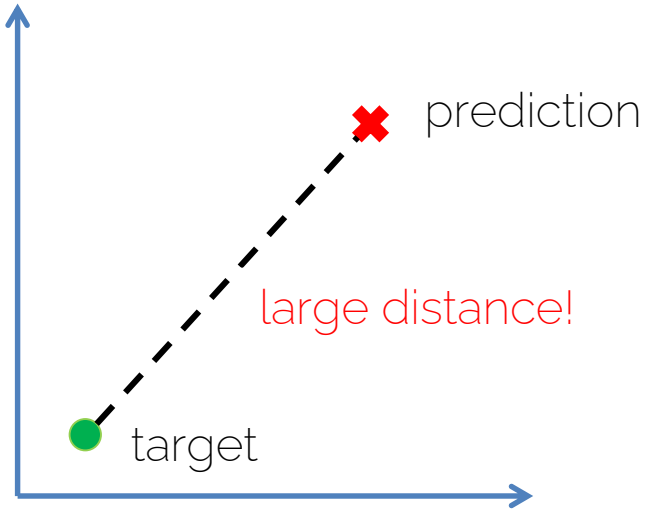
What's Next?



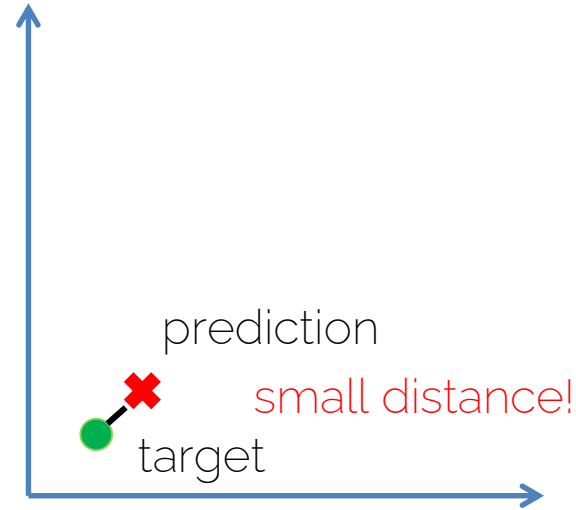
We need a way to describe how close the network's outputs (= predictions) are to the targets!

What's Next?

Idea: calculate a '**distance**' between prediction and target!



bad prediction



good prediction

Loss Functions

- A function to **measure the goodness of the predictions** (or equivalently, the network's performance)

Intuitively, …

- a **large loss indicates bad predictions**/performance (→ performance needs to be improved by training the model)
- the choice of the loss function depends on the concrete problem or the distribution of the target variable

Regression Loss

- L1 Loss:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_{i}^{n} ||y_i - \hat{y}_i||_1$$

- MSE Loss:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_{i}^{n} ||y_i - \hat{y}_i||_2^2$$

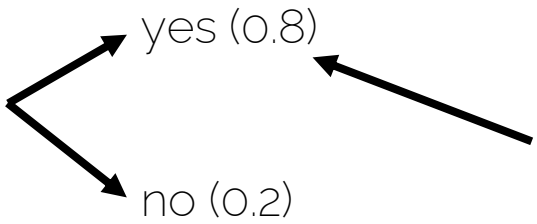
Binary Cross Entropy

- Loss function for binary (yes/no) classification

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta}) = -\frac{1}{n}\sum_{i}^{n}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$$

Cat?





The network predicts the probability of the input belonging to the "yes" class!

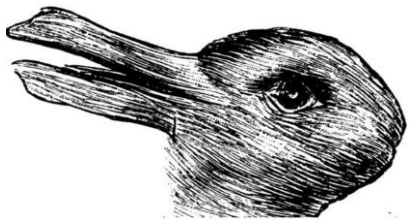
"Cool Kitty" by ~Pawsitive~Candie_N, Flickr, CC BY 2.0

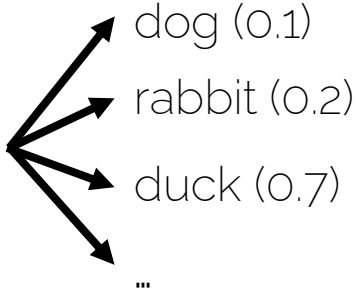
Cross Entropy

Generalizes to multi-class classification:

$$y_{ik} = \begin{cases} 1 & if\ x_i \in class\ k \\ 0 & else \end{cases}$$

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}(y_{ik} \cdot \log \hat{y}_{ik})$$

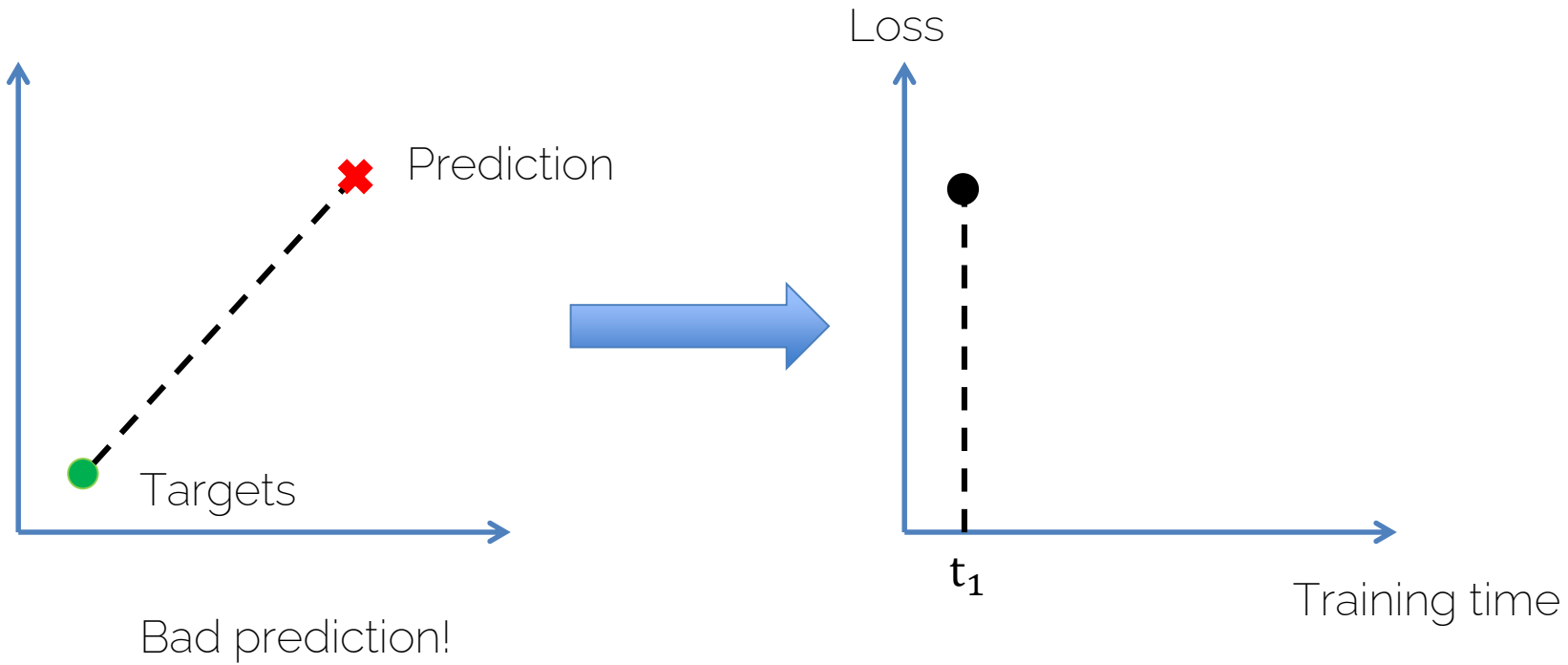




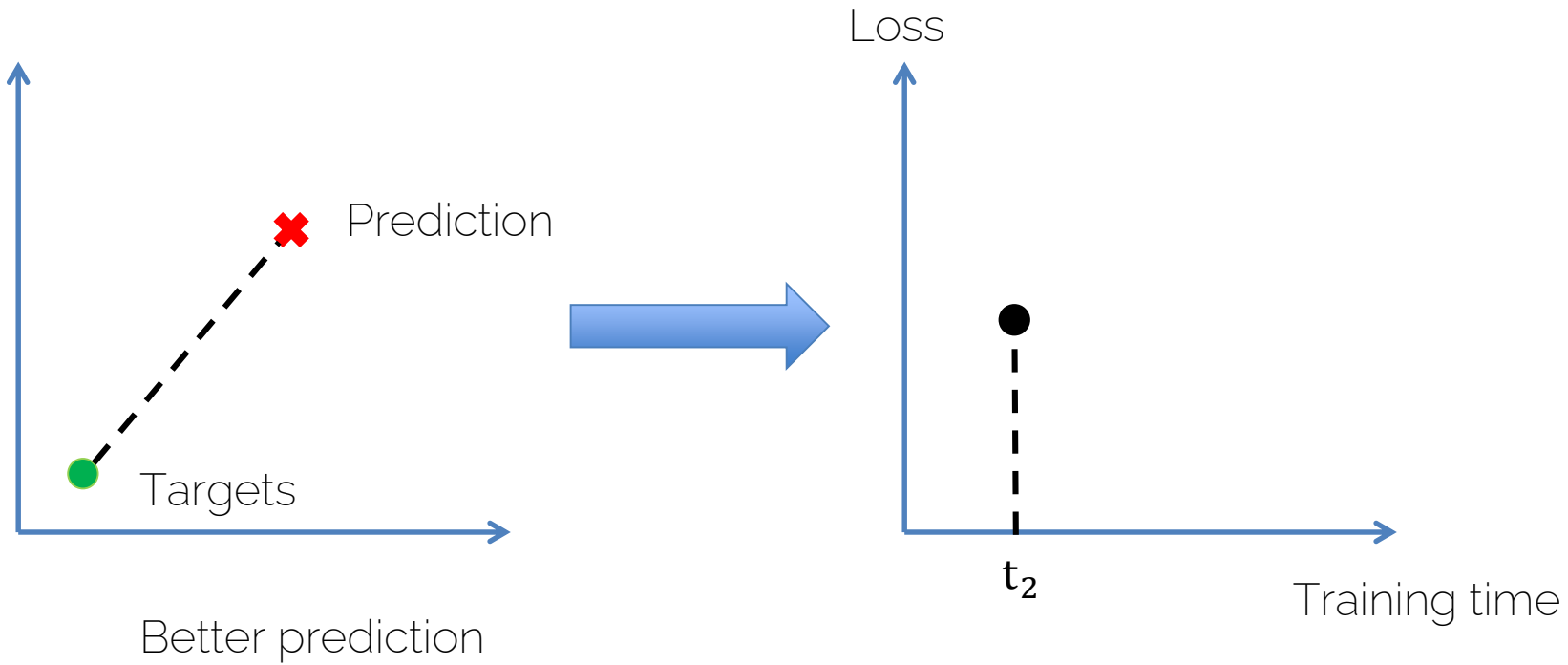
More General Case

- Ground truth: $\boldsymbol{y}$
- Prediction: $\hat{\boldsymbol{y}}$
- Loss function: $\boldsymbol{L(y, \hat{y})}$
- Motivation:
  - minimize the loss <=> find better predictions
  - predictions are generated by the NN
  - find better predictions <=> find better NN

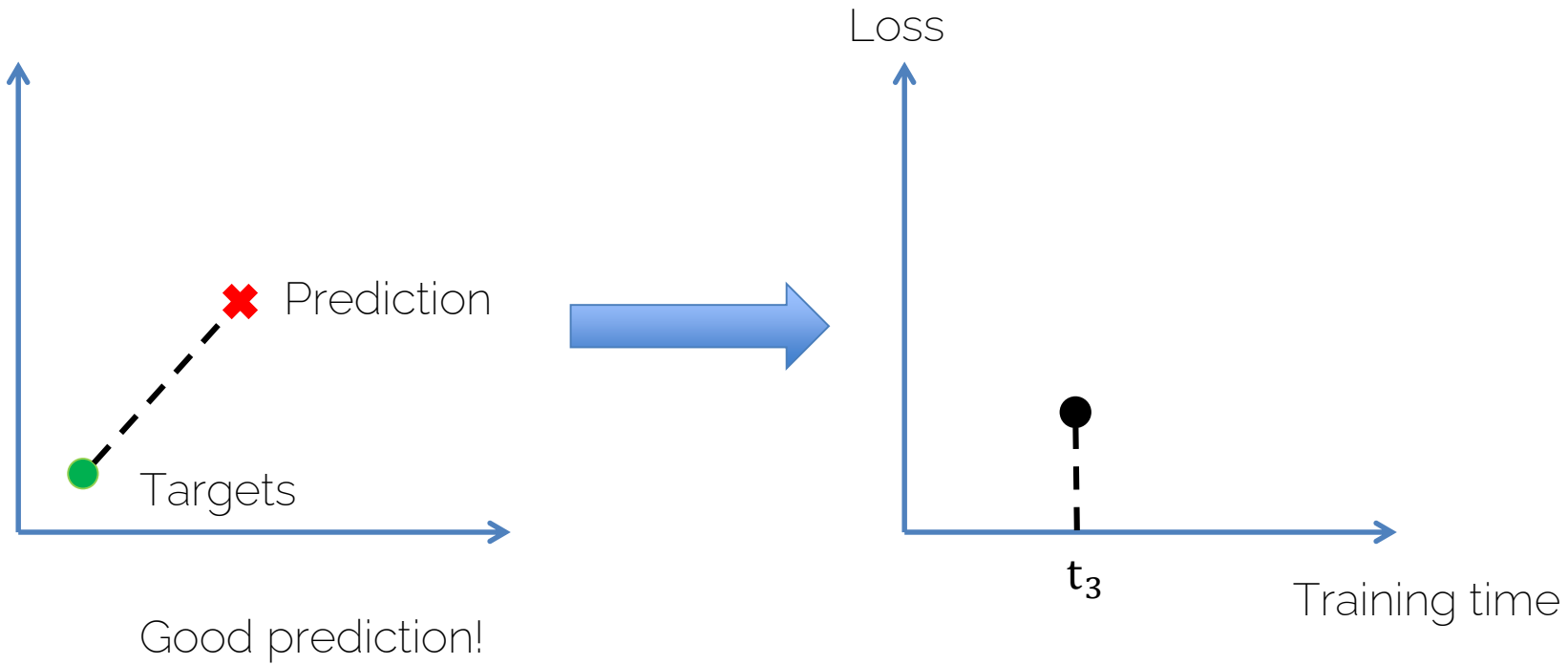
Initially



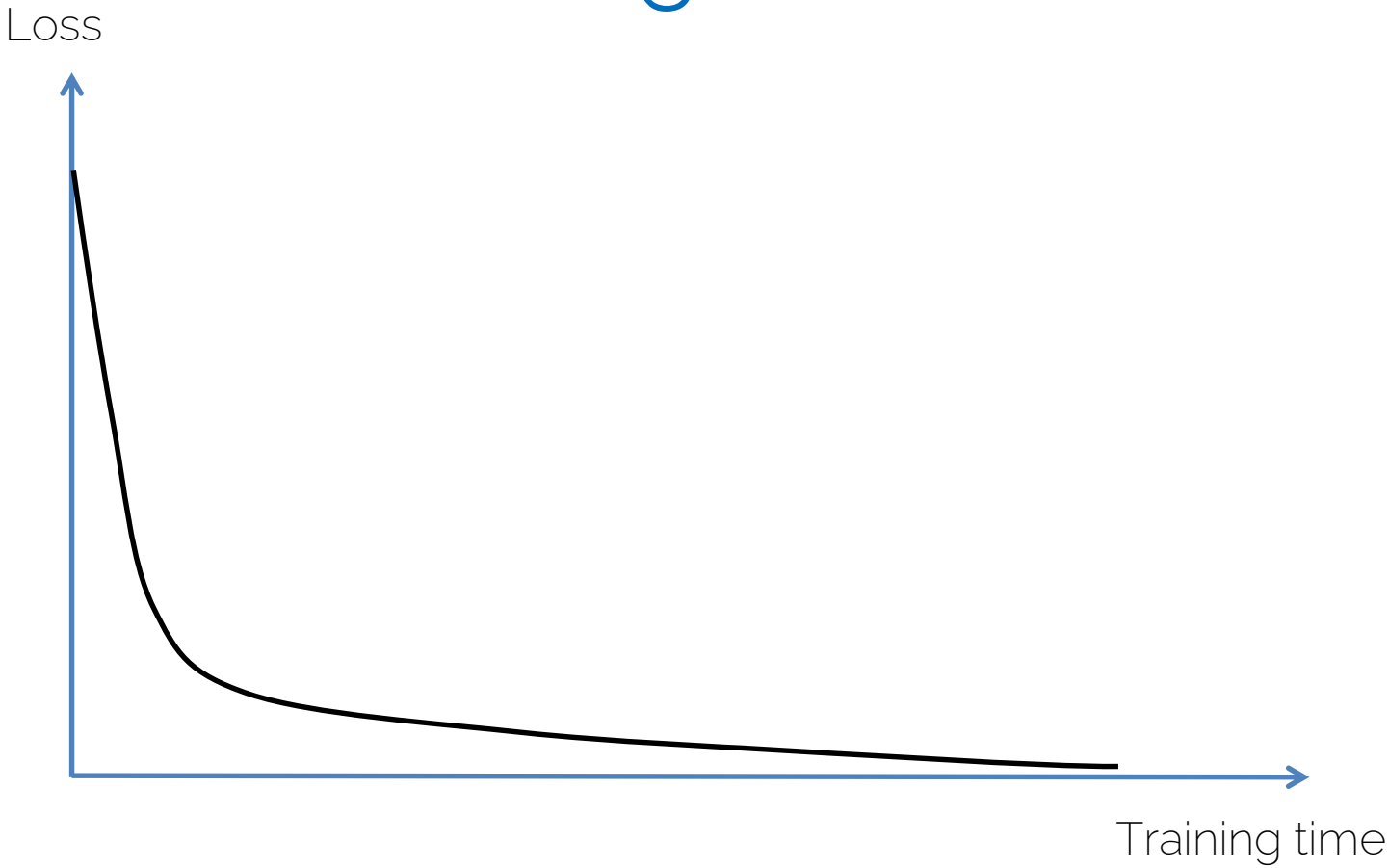
During Training…



During Training…

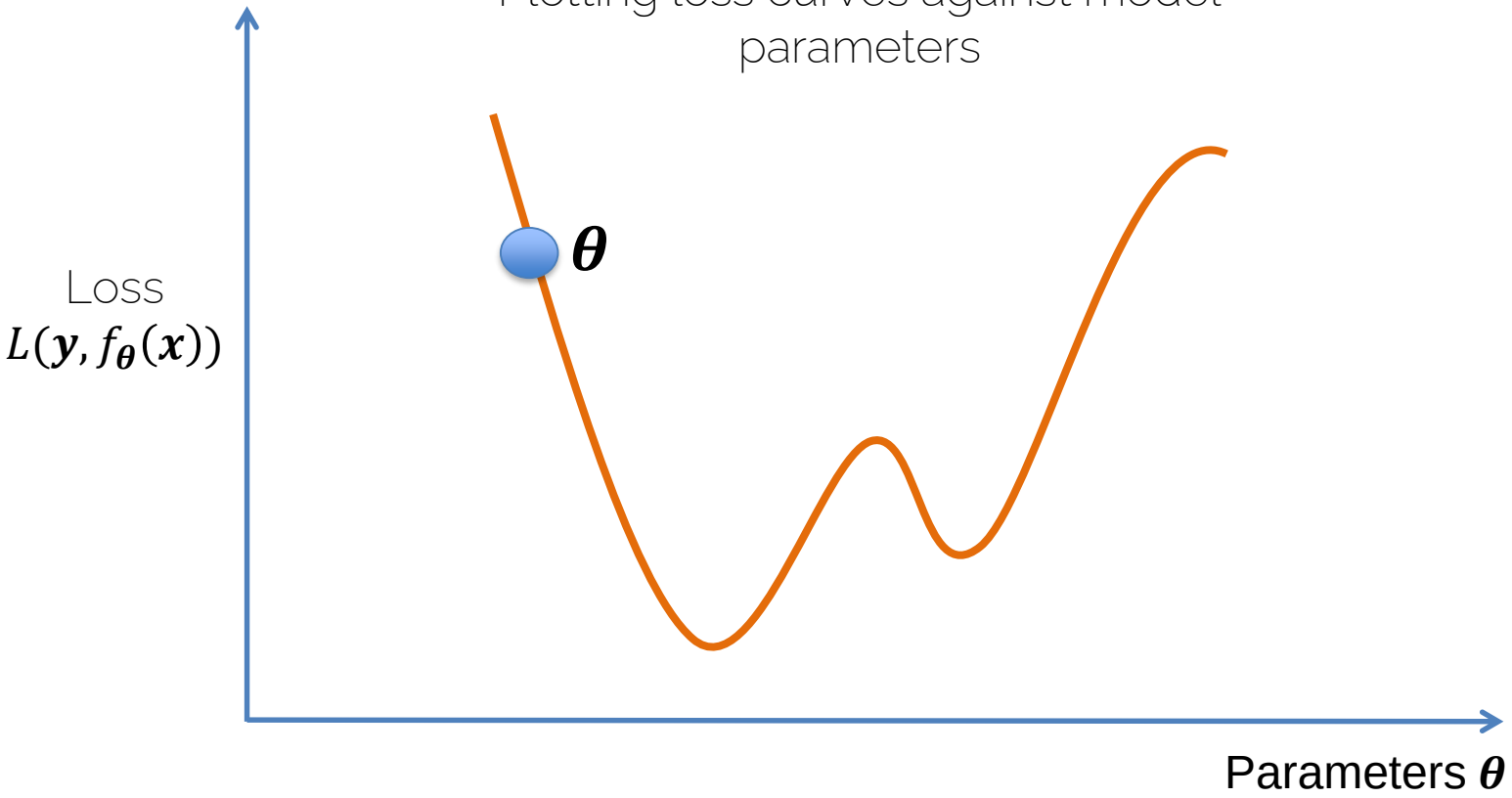


Training Curve



How to find a better NN?

Plotting loss curves against model parameters



How to find a better NN?

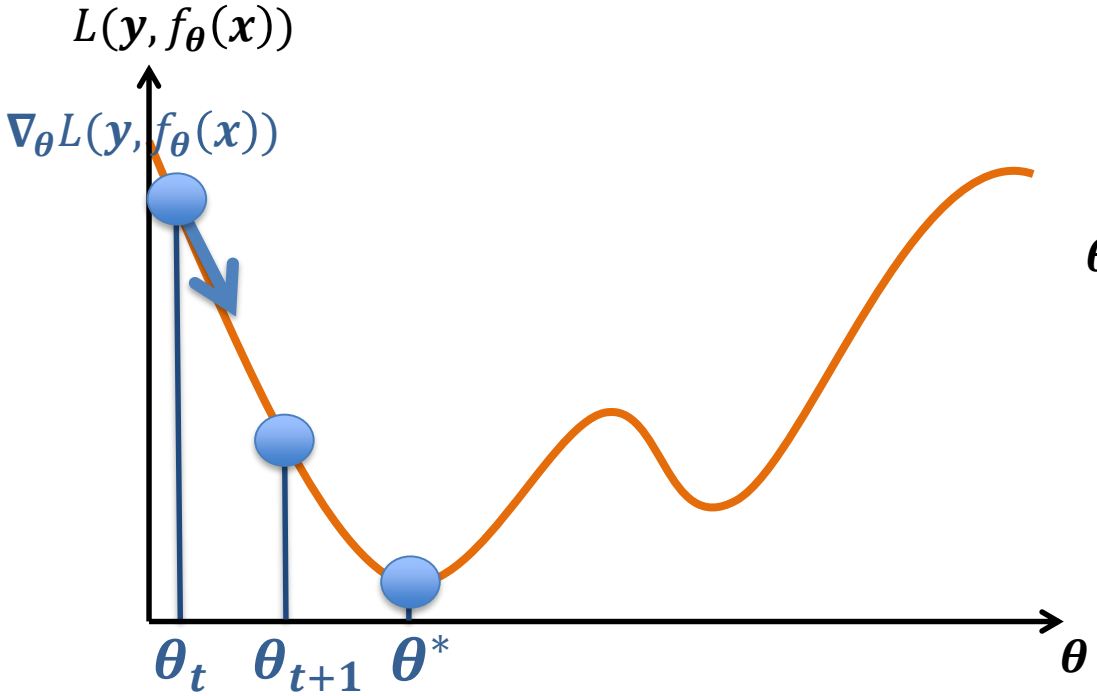
- Loss function: $L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = L(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$
- Neural Network: $f_{\boldsymbol{\theta}}(\boldsymbol{x})$
- Goal:
  - minimize the loss w. r. t. $\boldsymbol{\theta}$



Optimization! We train compute graphs with some optimization techniques!

Gradient Descent

Minimize loss $L\big(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\big)$ w.r.t. $\boldsymbol{\theta}$:



step size / learning rate

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L\big(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\big)\Big|_{\boldsymbol{\theta}_t}$$

$$t \to \infty$$

$$\boldsymbol{\theta}^* = \arg\min L\big(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\big)$$

a (local) minimum, for a small enough learning rate

How to find a better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given one layer NN with no activation function

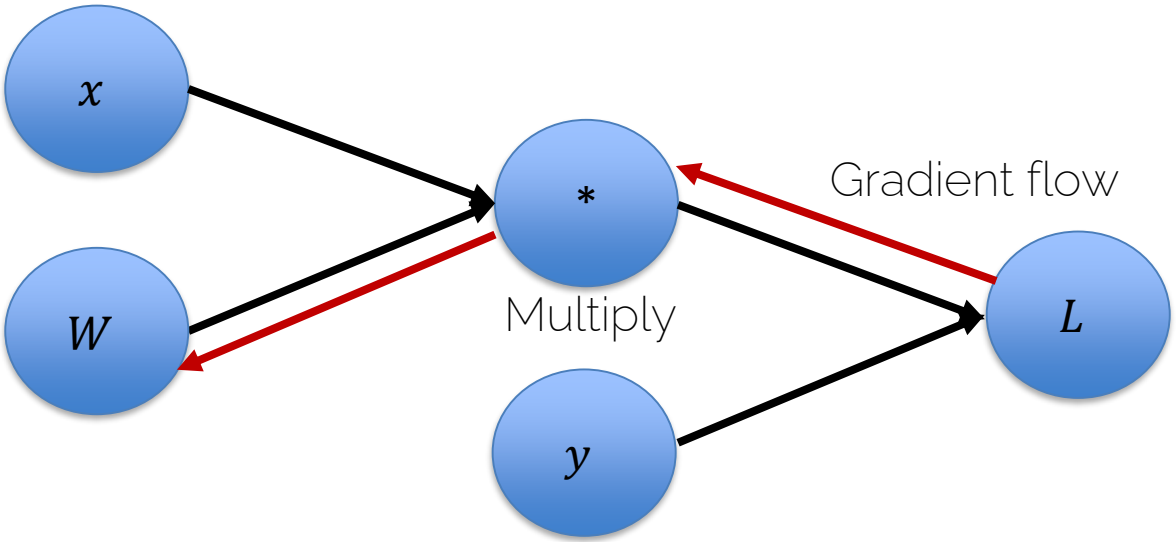
$$\boldsymbol{f_\theta(x) = Wx}\,, \qquad \boldsymbol{\theta = W}$$

Later $\boldsymbol{\theta = \{W, b\}}$

- Given MSE Loss: $\boldsymbol{L(y, \hat{y}; \theta)} = \frac{1}{n}\sum_i^n ||y_i - \hat{y}_i||_2^2$

How to find a better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given one layer NN with no activation function
- Given MSE Loss: $L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\boldsymbol{W} \cdot x_i - y_i||_2^2$



How to find a better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given one layer NN with no activation function

$$f_\theta(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}, \qquad \boldsymbol{\theta} = \boldsymbol{W}$$

- Given MSE Loss: $L(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n}\sum_i^n ||\boldsymbol{W} \cdot x_i - y_i||_2^2$
- $\nabla_\theta L(\boldsymbol{y}, f_\theta(\boldsymbol{x})) = \frac{2}{n}\sum_i^n (\boldsymbol{W} \cdot x_i - y_i) \cdot x_i^T$

How to find a better NN?

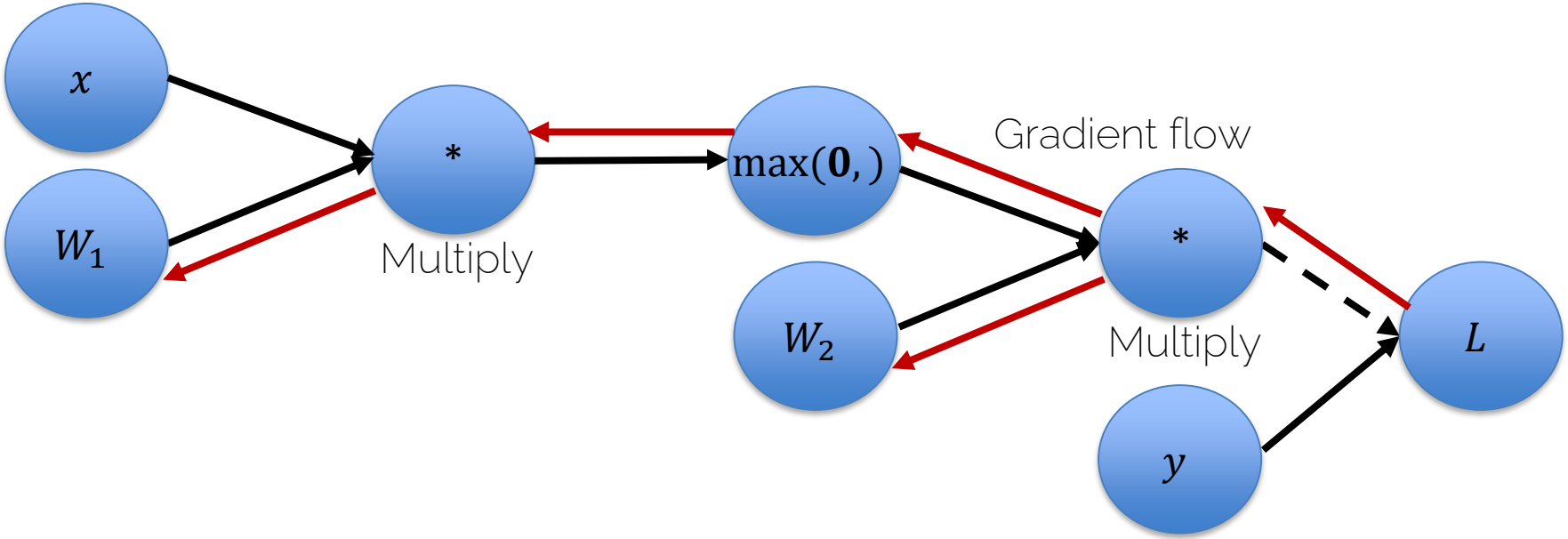
- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given a multi-layer NN with many activations

$$\boldsymbol{f} = \boldsymbol{W_5}\sigma(\boldsymbol{W_4}\tanh(\boldsymbol{W_3}\max(\mathbf{0}, \boldsymbol{W_2}\max(\mathbf{0}, \boldsymbol{W_1}\boldsymbol{x}))))$$

- Gradient descent for $L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ w. r. t. $\boldsymbol{\theta}$
  - Need to propagate gradients from end to first layer ($\boldsymbol{W_1}$).

How to find a better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given multi-layer NN with many activations



How to find a better NN?

- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given multilayer layer NN with many activations

$$\boldsymbol{f} = \boldsymbol{W_5}\sigma(\boldsymbol{W_4}\tanh(\boldsymbol{W_3}\max(\boldsymbol{0}, \boldsymbol{W_2}\max(\boldsymbol{0}, \boldsymbol{W_1}\boldsymbol{x}))))$$

- Gradient descent solution for $L\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)$ w. r. t. $\boldsymbol{\theta}$
  - Need to propagate gradients from end to first layer ($\boldsymbol{W_1}$)
- Backpropagation: Use chain rule to compute gradients
  - Compute graphs come in handy!

How to find a better NN?

- Why gradient descent?
  - The graph gives all gradients for about twice the cost of the forward pass
  - Scales to millions of parameters
- Other methods
  - Second-order methods (Newton, L-BFGS): too expensive for large networks
  - Improved gradient descent (momentum, Adam): comes later

Summary

- Neural Networks are computational graphs
- Goal: for a given train set, find optimal weights

- Optimization is done using gradient-based solvers
  - Many options (more in the next lectures)

- Gradients are computed via backpropagation
  - Nice because can easily modularize complex functions

Next Lectures

- Next Lecture:
  - Backpropagation and optimization of Neural Networks

- Check for updates on website/piazza regarding exercises

See you next week ☺

Further Reading

- Books, free online:
  - S. Prince. Understanding Deep Learning. MIT Press, 2023, Ch. 3 and 4. udlbook.github.io/udlbook
  - I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016, Ch. 6. deeplearningbook.org

- General Machine Learning book:
  - C. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.