

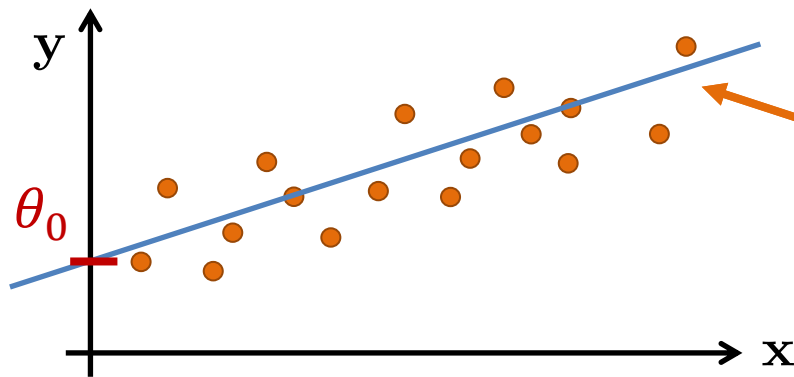
Introduction to Neural Networks

Lecture 2 Recap

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

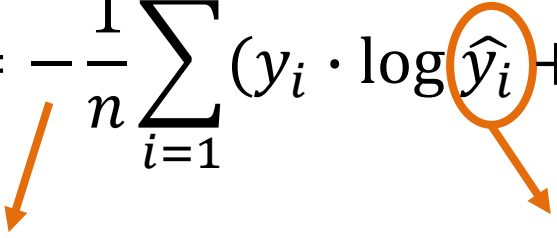
Logistic Regression

- Loss function

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

- Cost function

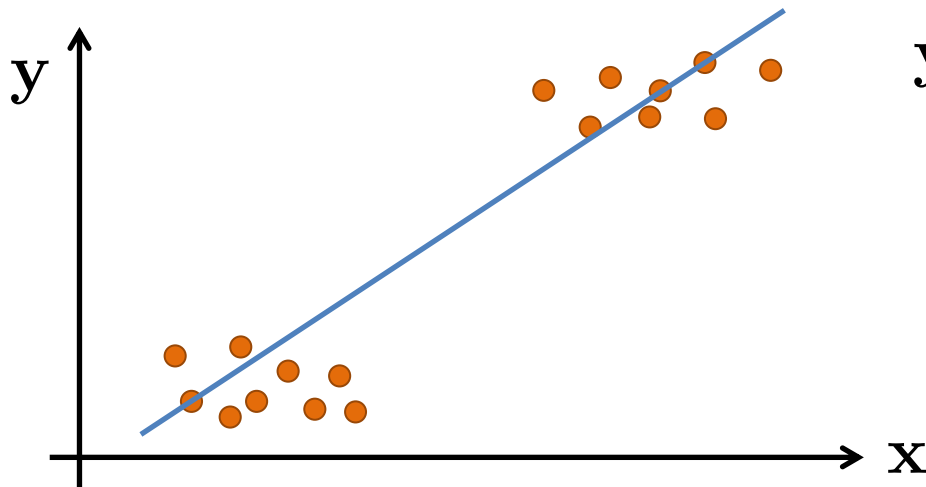
$$\mathcal{C}(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^n (y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$



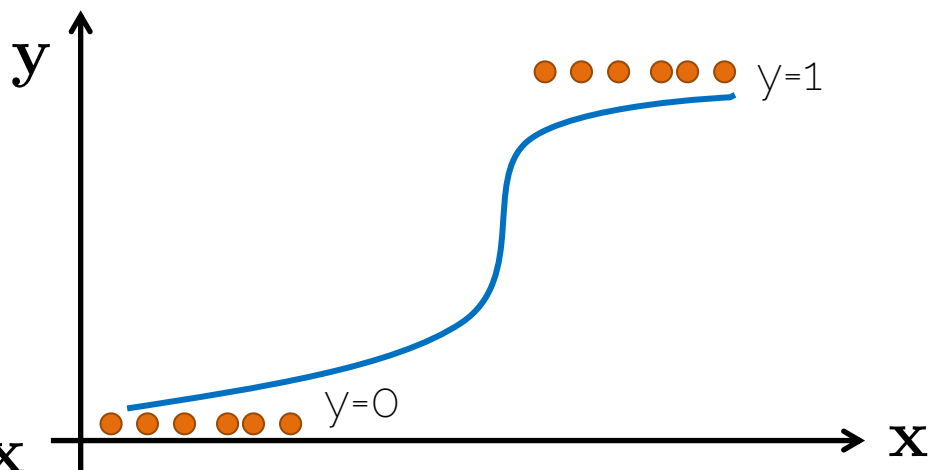
Minimization

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

Linear vs Logistic Regression

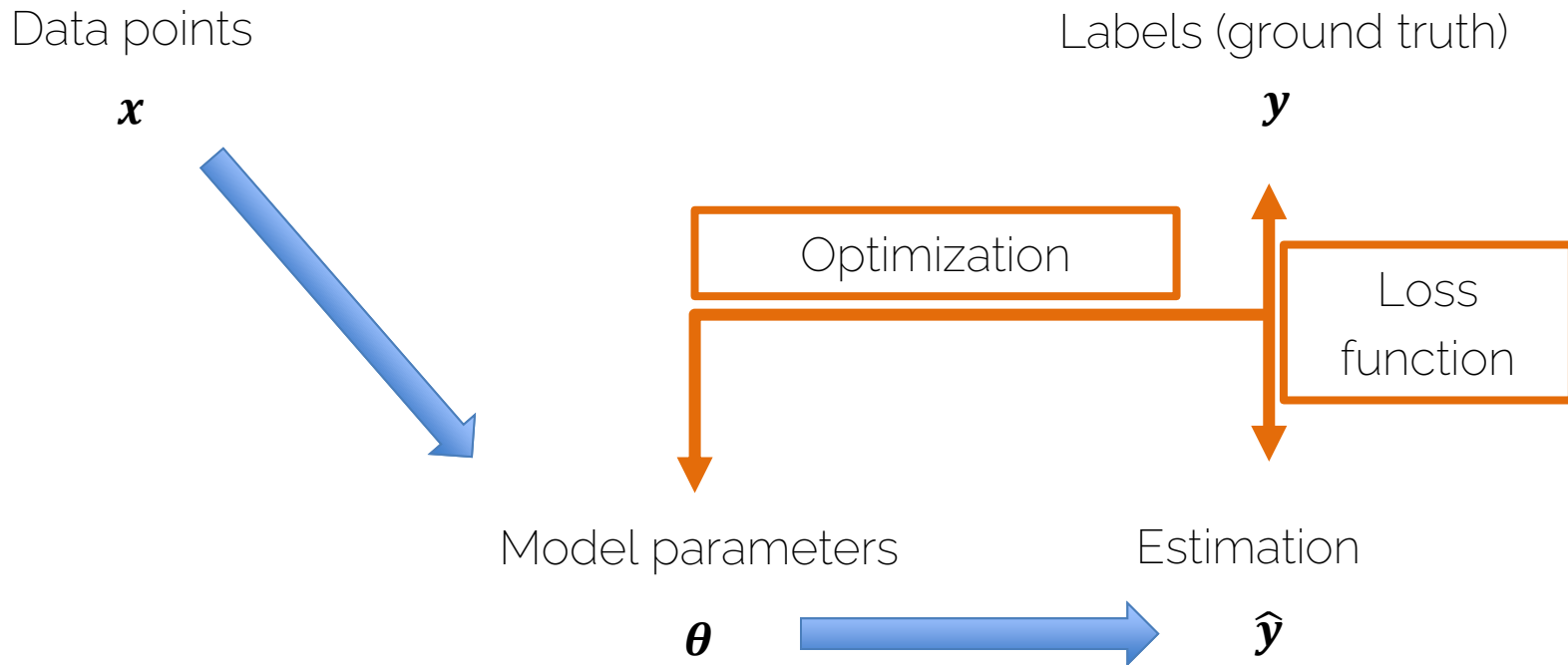


Predictions can exceed the range of the training samples
→ in the case of classification $[0;1]$ this becomes a real issue




Predictions are guaranteed to be within $[0;1]$

How to obtain the Model?



Linear Score Functions

- Linear score function as seen in linear regression

$$f_i = \sum_j w_{k,j} x_{j,i}$$


$$\mathbf{f} = \mathbf{W} \mathbf{x} \quad (\text{Matrix Notation})$$

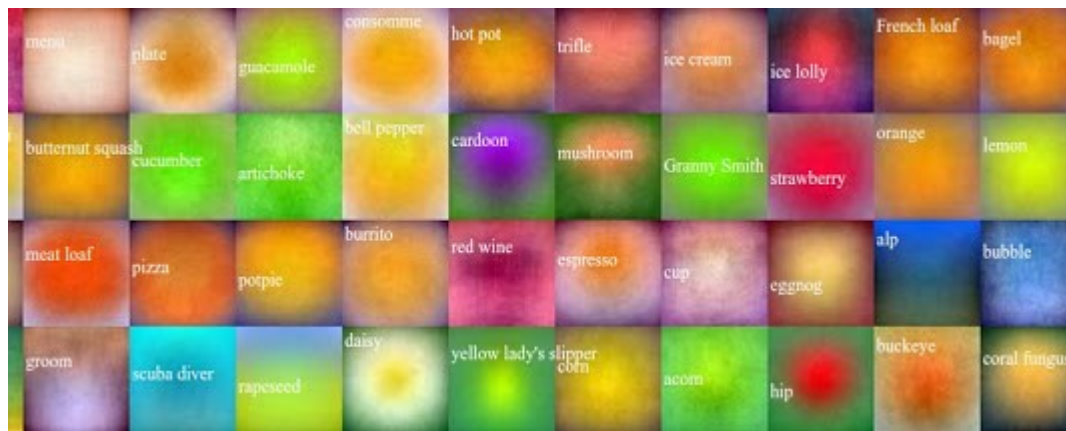
Linear Score Functions on Images

optimal weight

- Linear score function $f = Wx$



On CIFAR-10

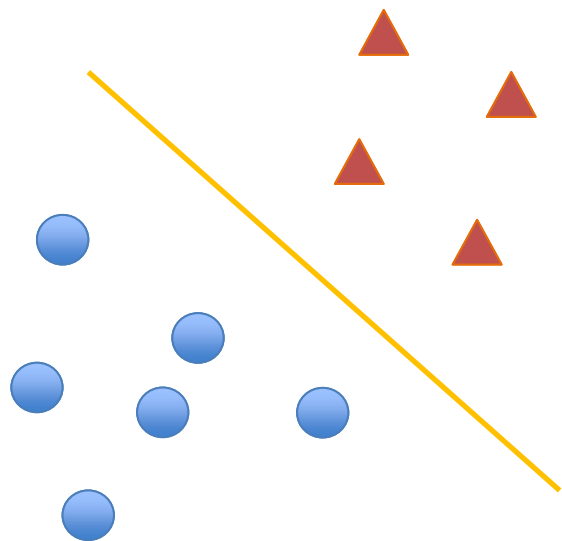


On ImageNet

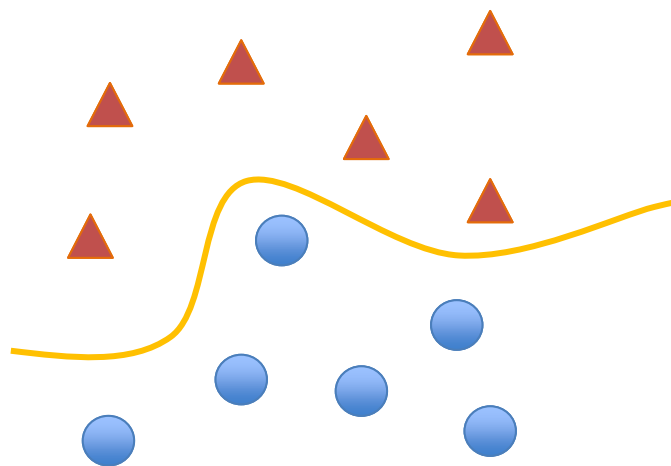
Source: Li/Karpathy/Johnson

Linear Score Functions?

Logistic Regression



Linear Separation Impossible!



Linear Score Functions?

- Can we make linear regression better?
 - Multiply with another weight matrix \mathbf{W}_2

$$\hat{\mathbf{f}} = \mathbf{W}_2 \cdot \mathbf{f}$$

$$\hat{\mathbf{f}} = \mathbf{W}_2 \cdot \mathbf{W} \cdot \mathbf{x}$$

- Operation is still linear.

$$\widehat{\mathbf{W}} = \mathbf{W}_2 \cdot \mathbf{W}$$

$$\hat{\mathbf{f}} = \widehat{\mathbf{W}} \mathbf{x}$$

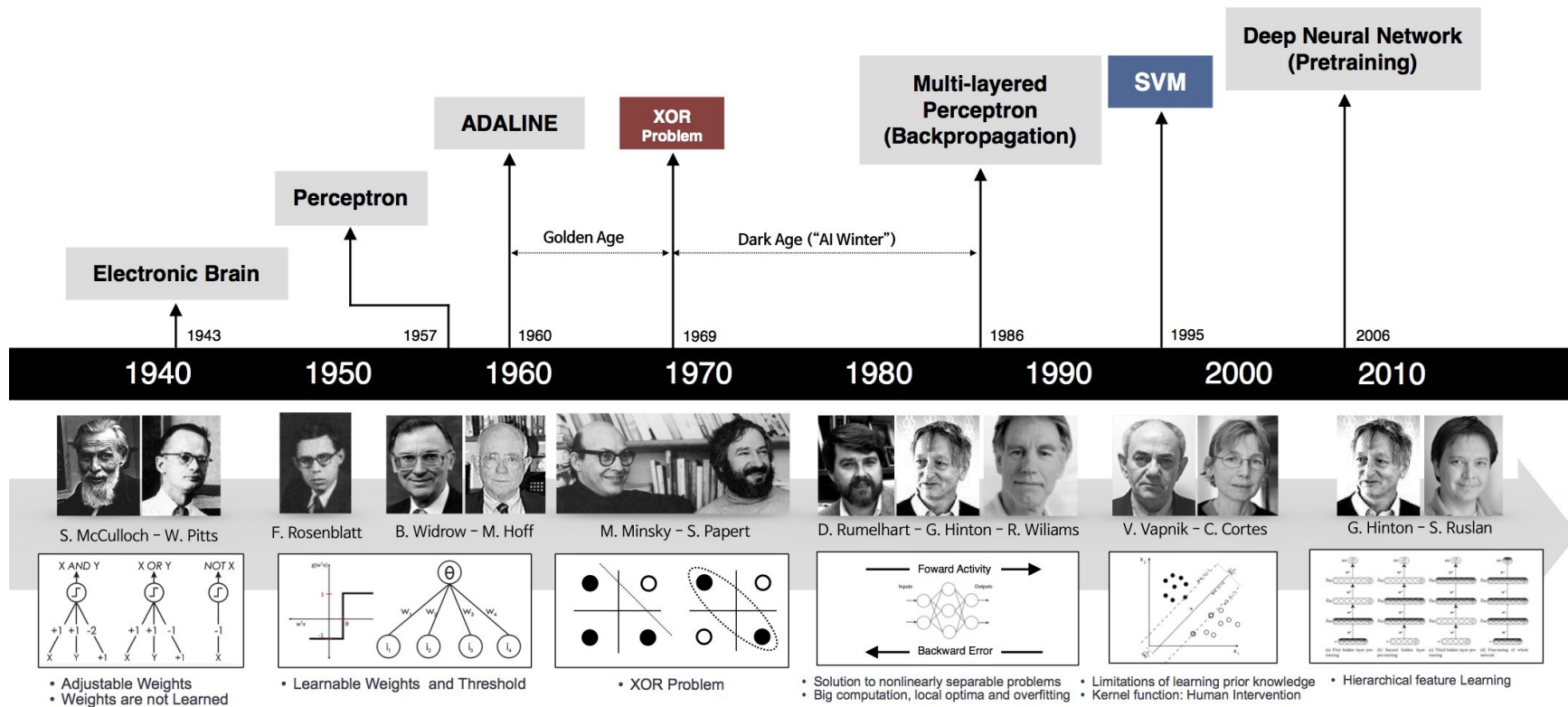
- Solution \rightarrow add non-linearity!!

Neural Network

- Linear score function $\mathbf{f} = \mathbf{W}\mathbf{x}$
- Neural network is a nesting of 'functions'
 - 2-layers: $\mathbf{f} = \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x})$
 - 3-layers: $\mathbf{f} = \mathbf{W}_3 \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x}))$
 - 4-layers: $\mathbf{f} = \mathbf{W}_4 \tanh(\mathbf{W}_3, \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x})))$
 - 5-layers: $\mathbf{f} = \mathbf{W}_5 \sigma(\mathbf{W}_4 \tanh(\mathbf{W}_3, \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x}))))$
 - ... up to hundreds of layers

Introduction to Neural Networks

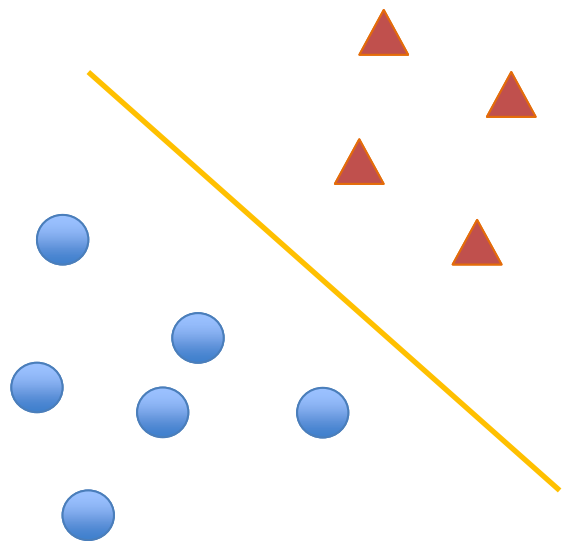
History of Neural Networks



Source: http://beamlab.org/deeplearning/2017/02/23/deep_learning_101_part1.html

Neural Network

Logistic Regression

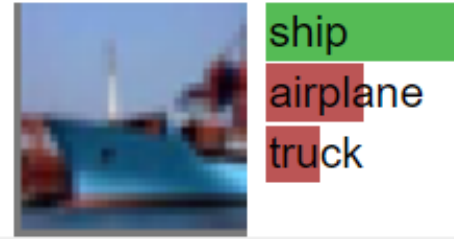
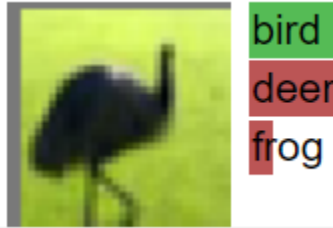


Neural Networks

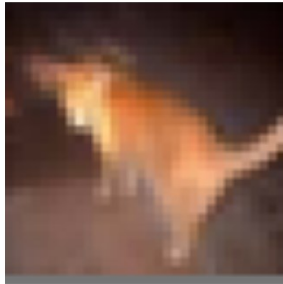


Neural Network

- Non-linear score function $f = \dots (\max(0, W_1 x))$



On CIFAR-10

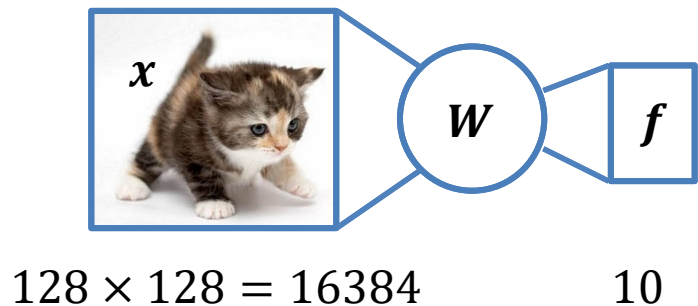


Visualizing activations of the first layer.

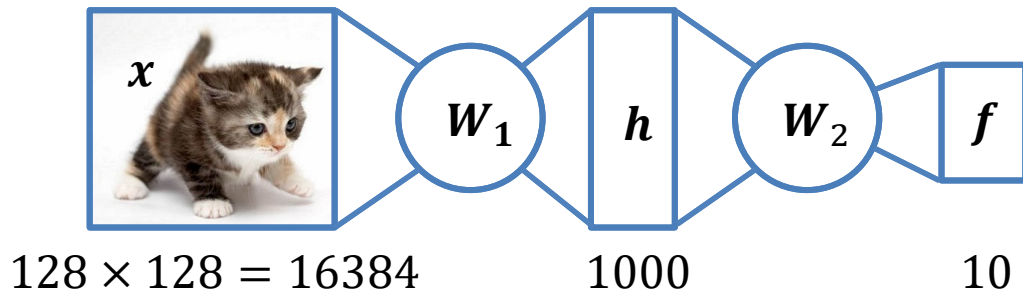
Source: ConvNetJS

Neural Network

1-layer network: $f = Wx$



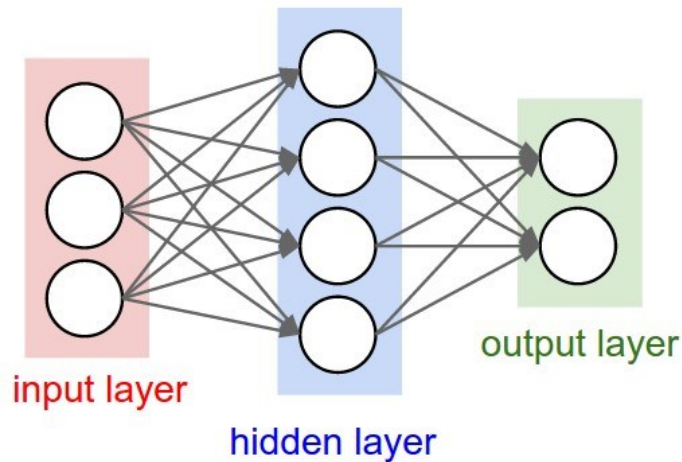
2-layer network: $f = W_2 \max(\mathbf{0}, W_1 x)$



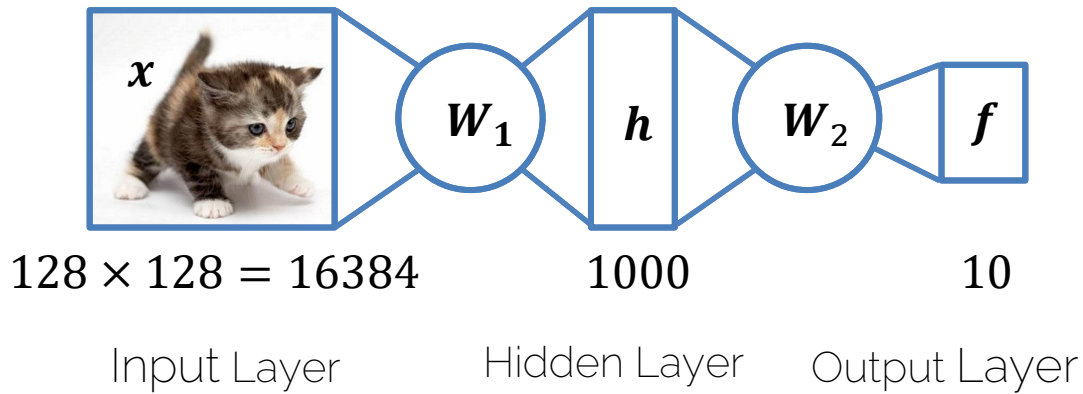
complex mode to deal with high d task

Why is this structure useful?

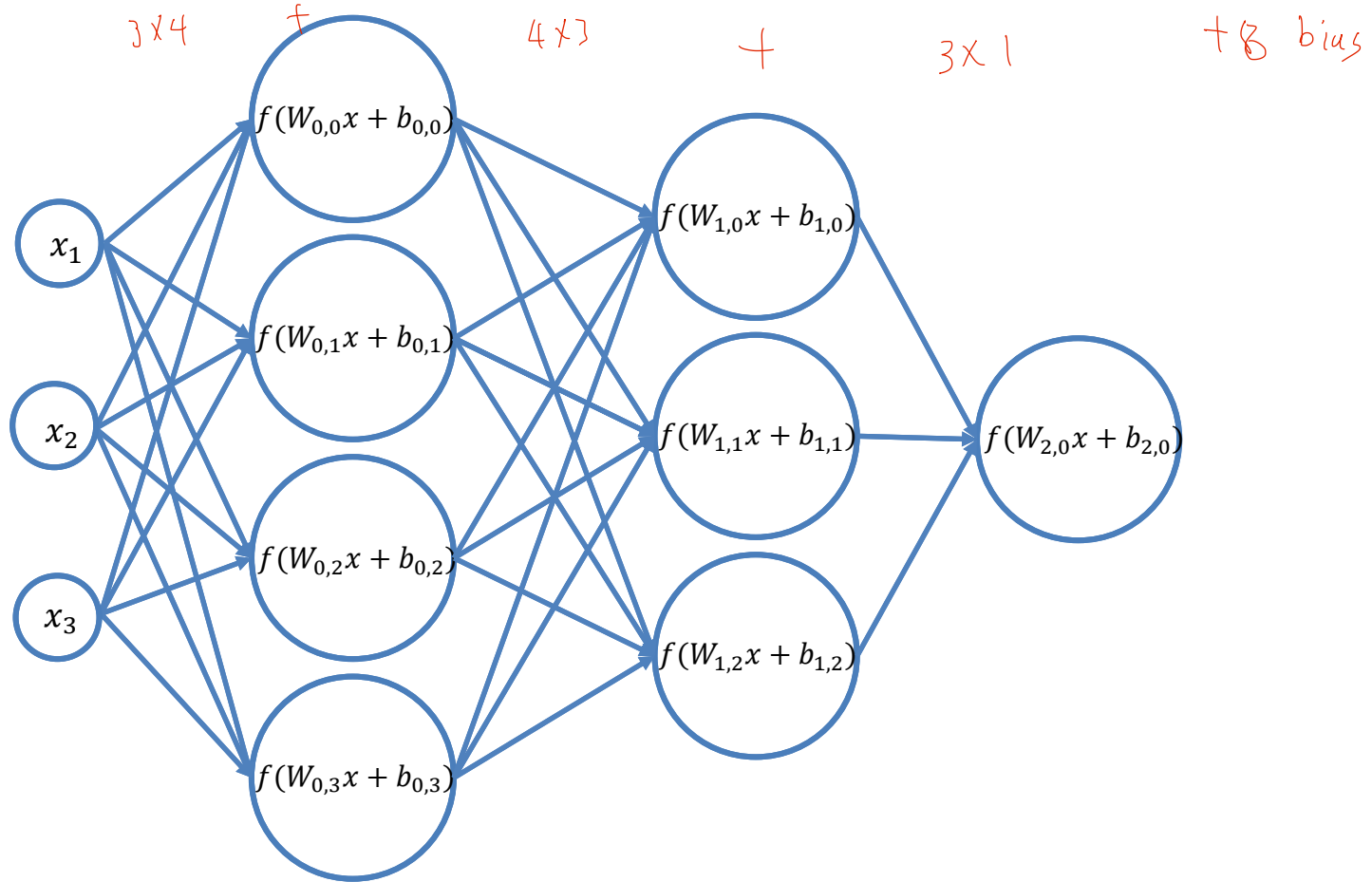
Neural Network



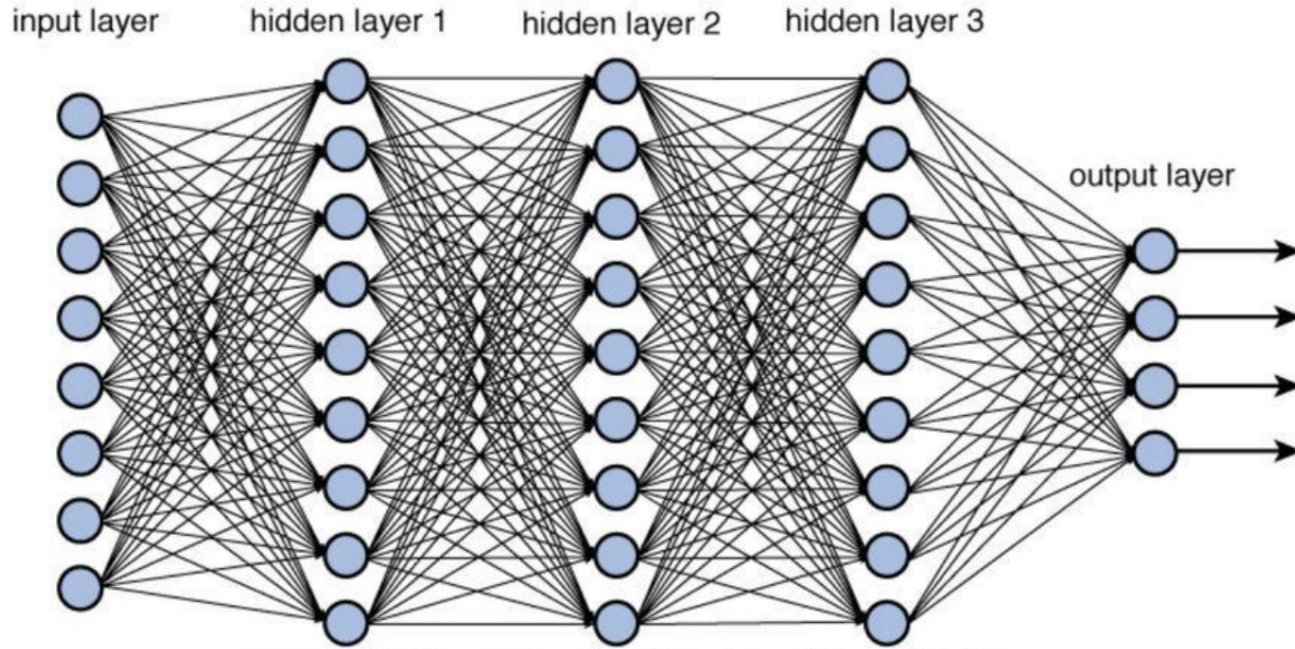
2-layer network: $\mathbf{f} = \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x})$



Net of Artificial Neurons



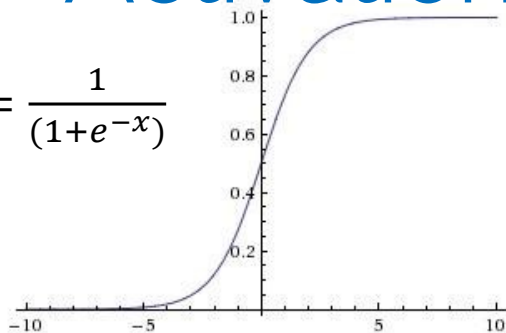
Neural Network



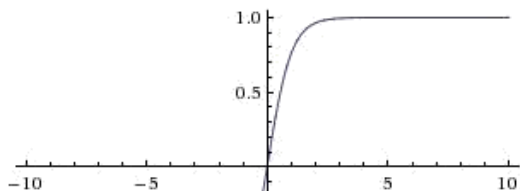
Source: <https://towardsdatascience.com/training-deep-neural-networks-gfdb1964b964>

Activation Functions

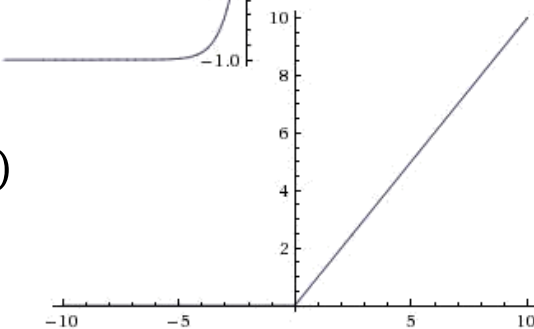
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



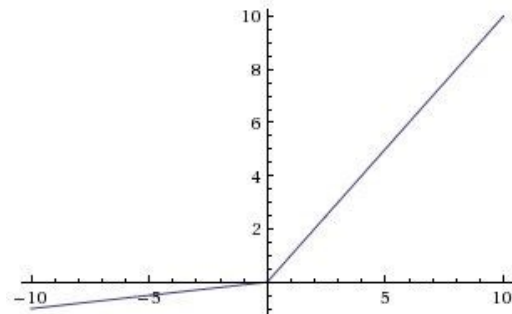
tanh: $\tanh(x)$



ReLU: $\max(0, x)$



Leaky ReLU: $\max(0.1x, x)$



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

$$\mathbf{f} = \mathbf{W}_3 \cdot (\mathbf{W}_2 \cdot (\mathbf{W}_1 \cdot \mathbf{x}))$$

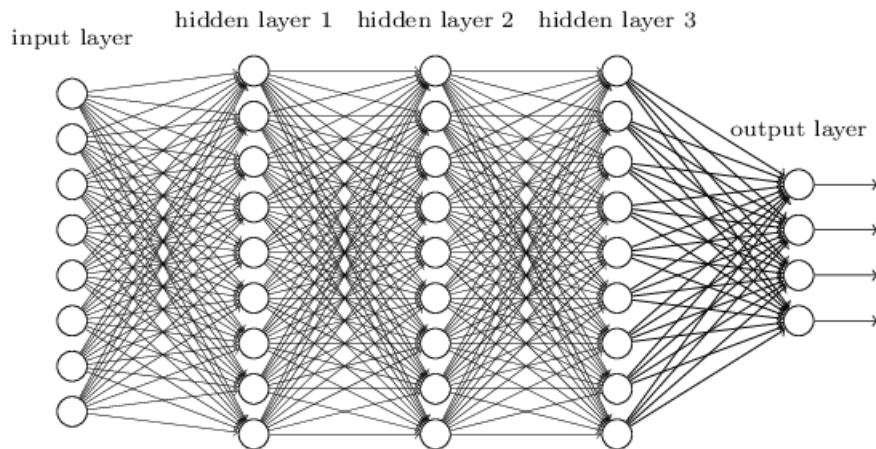
Why activation functions?

Simply concatenating linear layers would be so much cheaper...

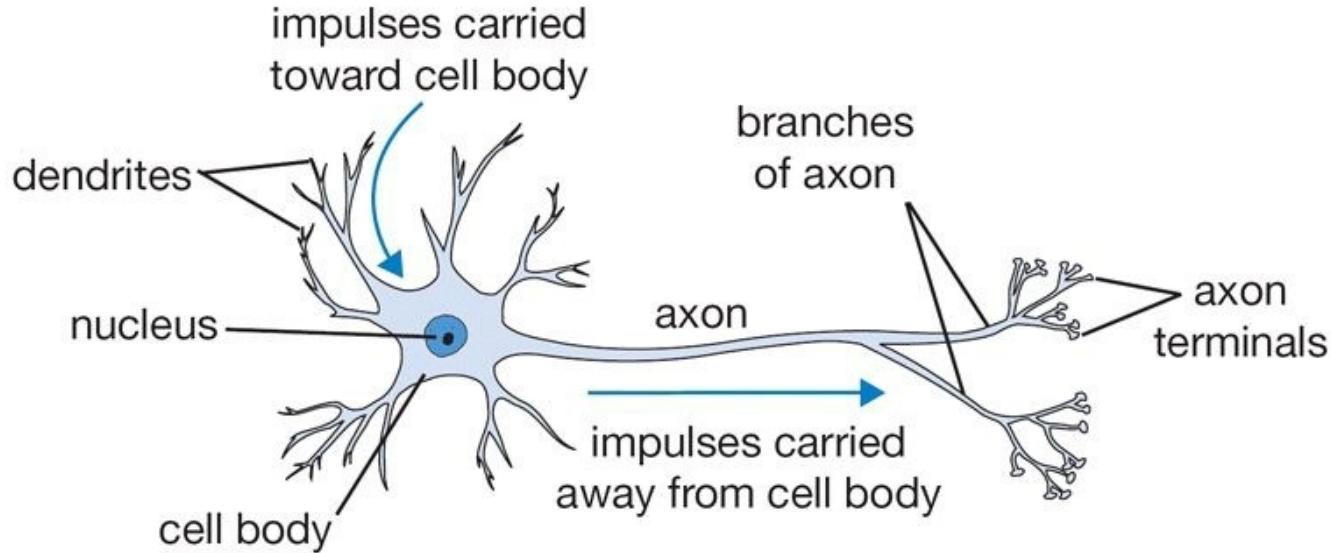
model complexity

Neural Network

Why organize a neural network into layers?

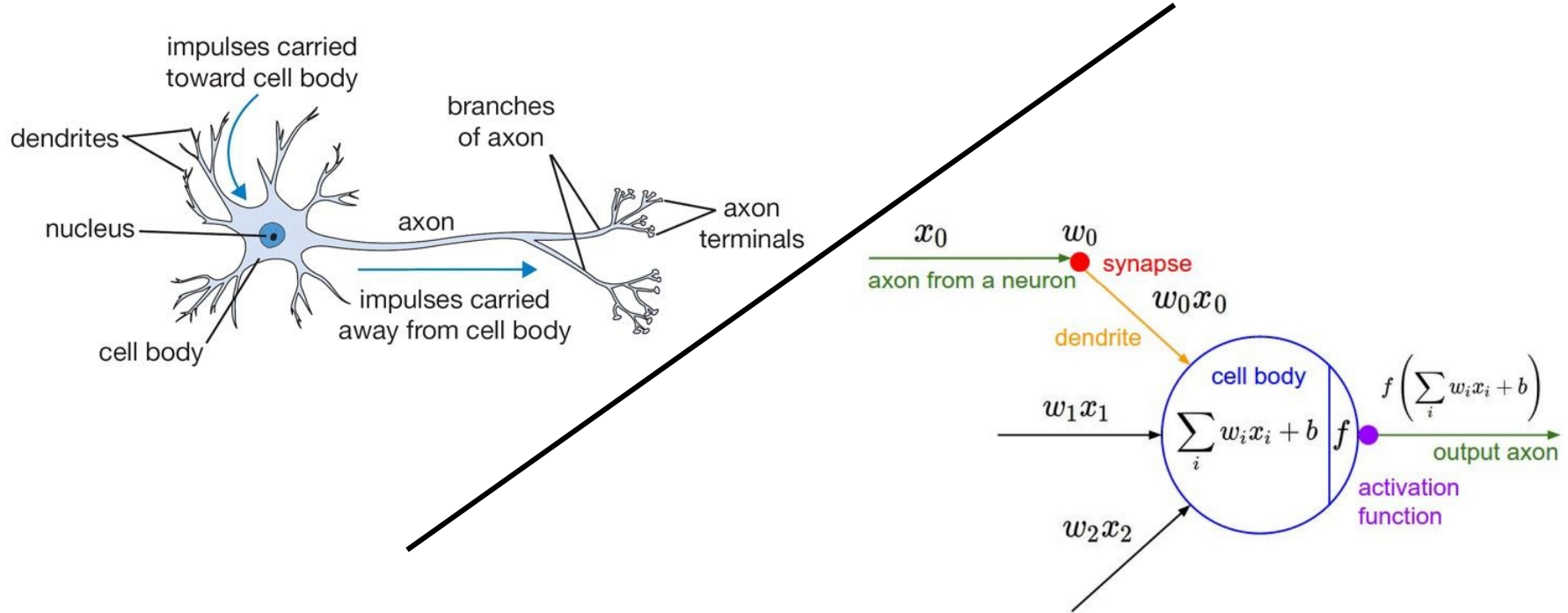


Biological Neurons



Credit: Stanford CS 231n

Biological Neurons



Credit: Stanford CS 231n

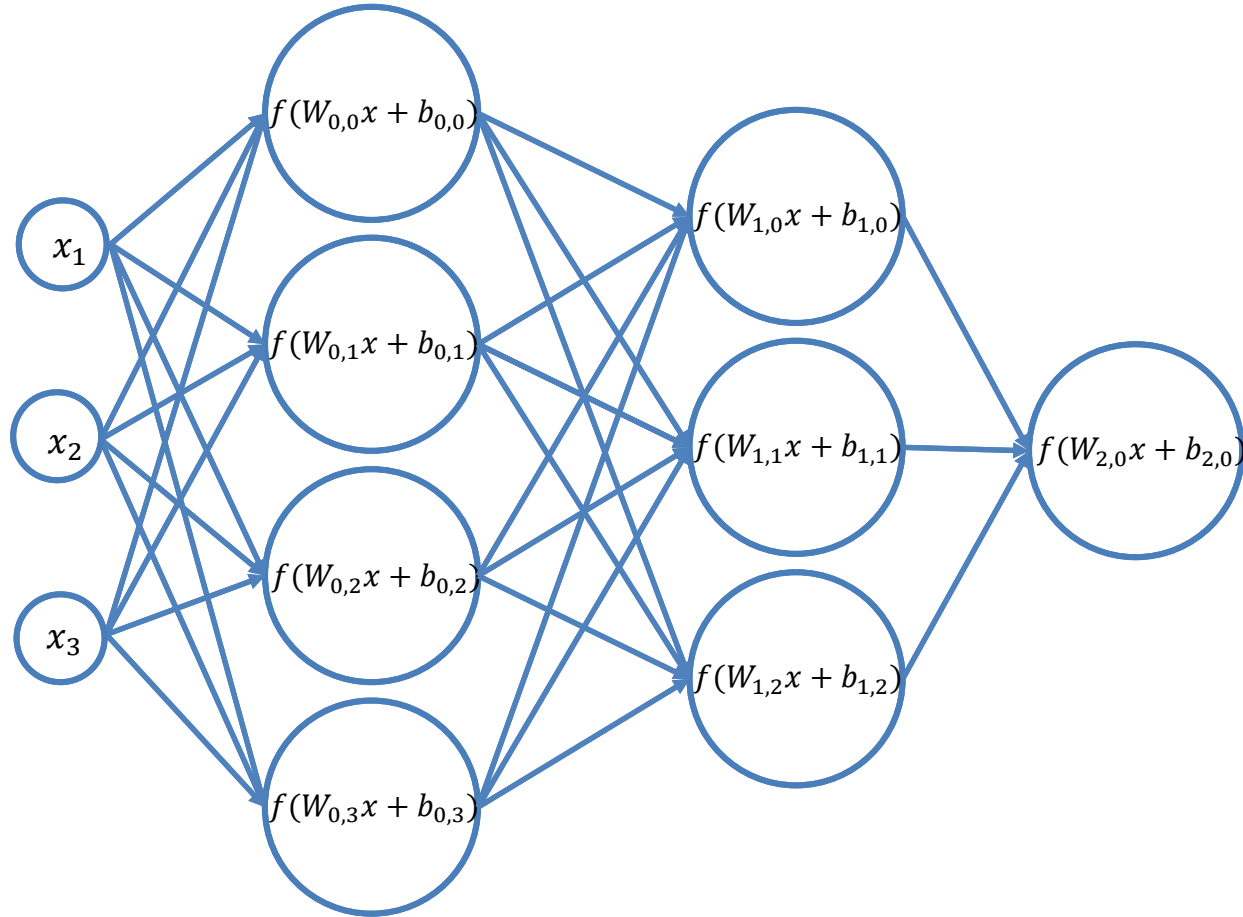
Artificial Neural Networks vs Brain



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 \mathbf{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)

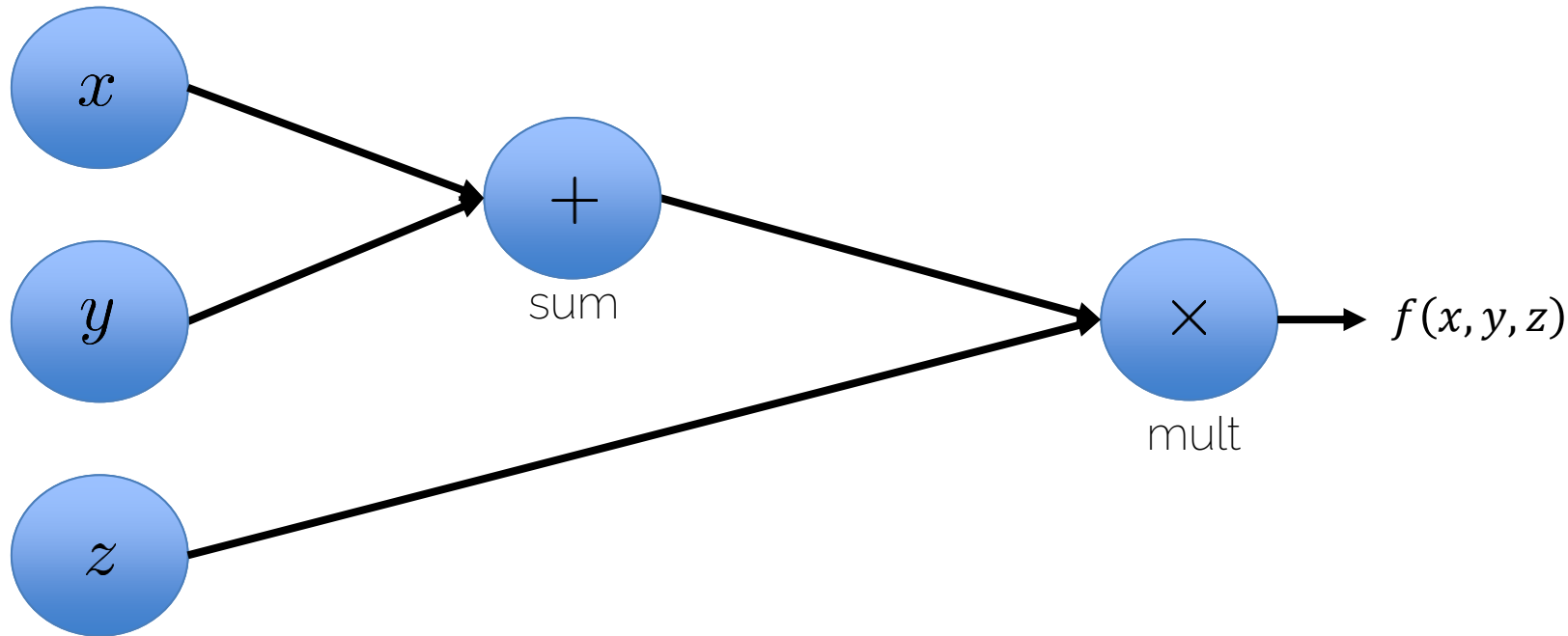
Computational Graphs

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

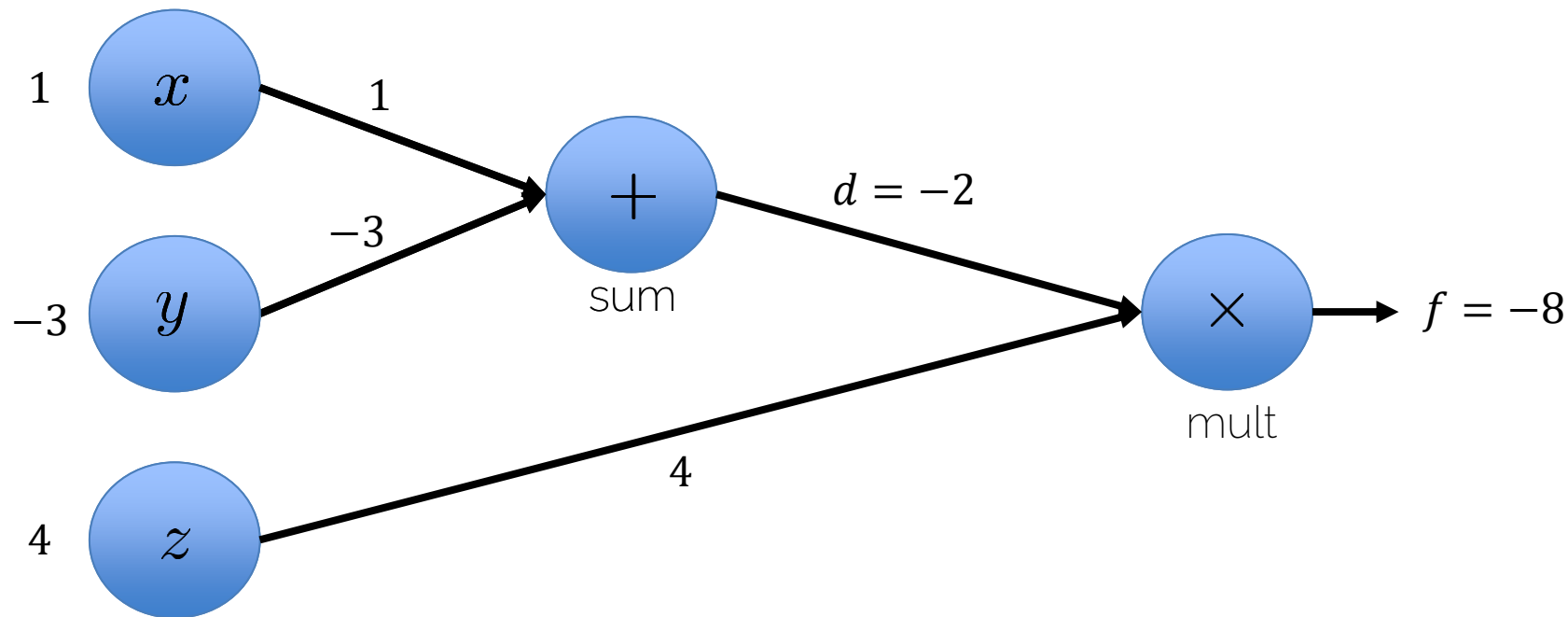
- $f(x, y, z) = (x + y) \cdot z$



Evaluation: Forward Pass

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

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



Computational Graphs

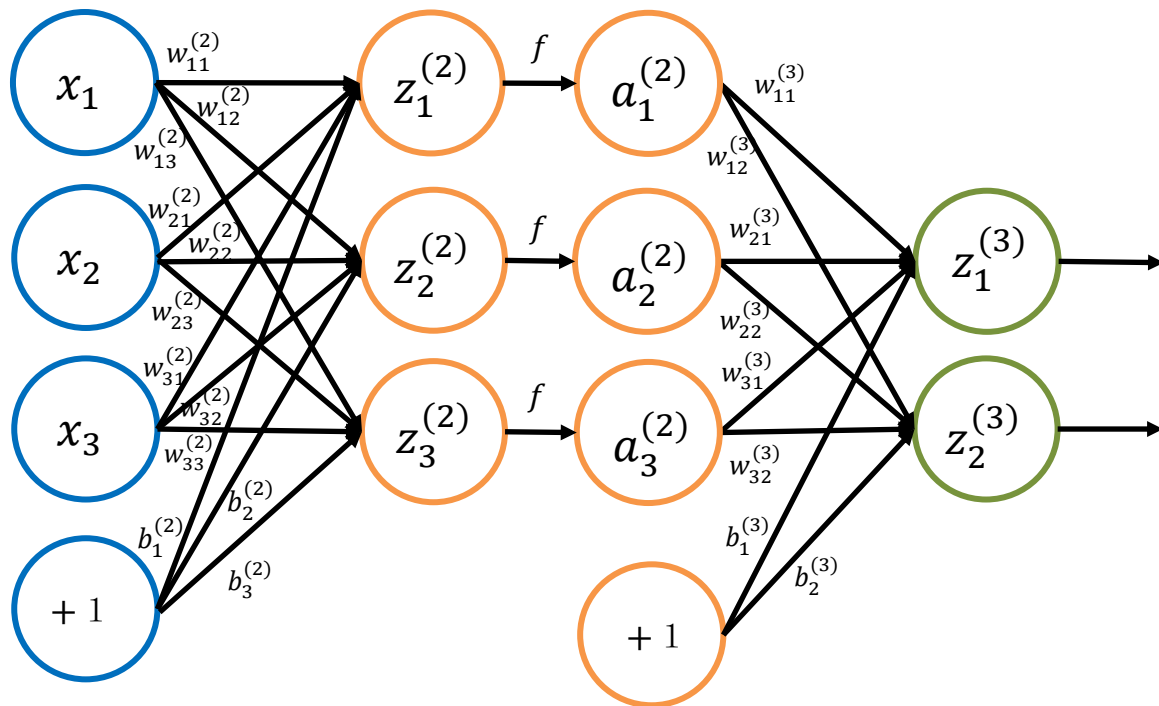
- Why discuss compute graphs?
- Neural networks have complicated architectures
$$f = W_5 \sigma(W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1 x))))$$
- Lot of matrix operations!
- Represent NN as computational graphs!

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



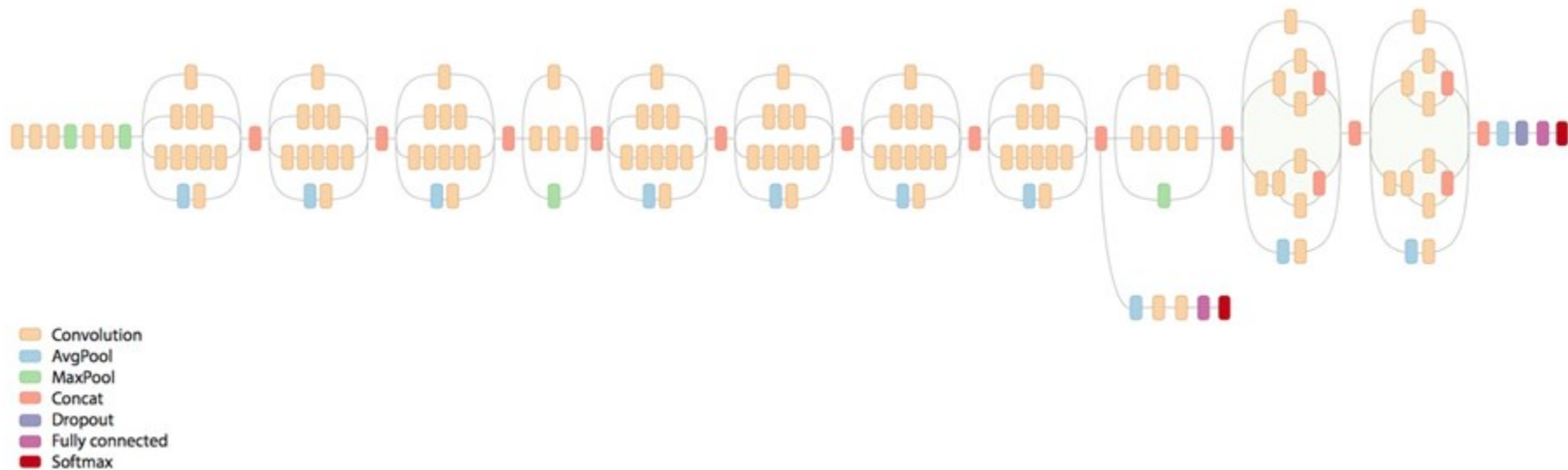
$$z_k^{(2)} = \sum_i x_i w_{ik}^{(2)} + b_k^{(2)}$$

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

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

Computational Graphs

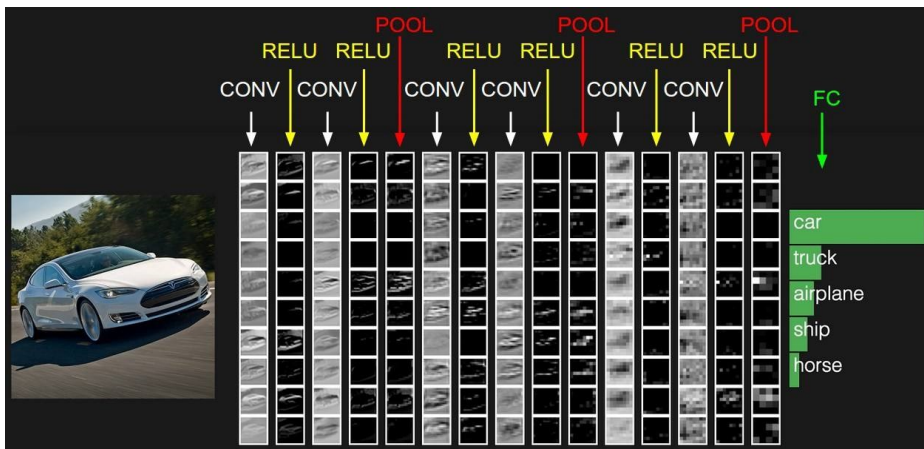
- From a set of neurons to a Structured Compute Pipeline



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

Computational Graphs

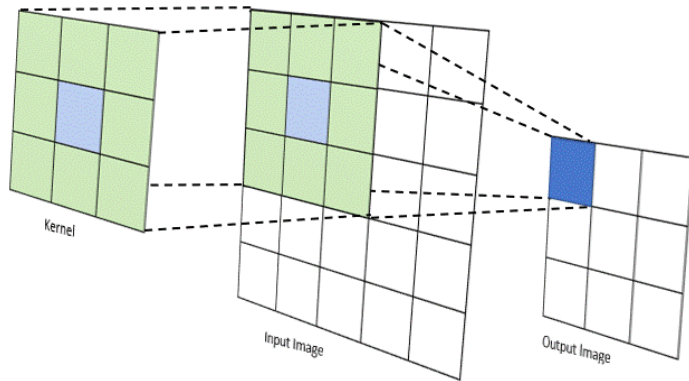
- The computation of Neural Network has further meanings:
 - The multiplication of \mathbf{W}_i and \mathbf{x} : encode input information
 - The activation function: select the key features



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

Computational Graphs

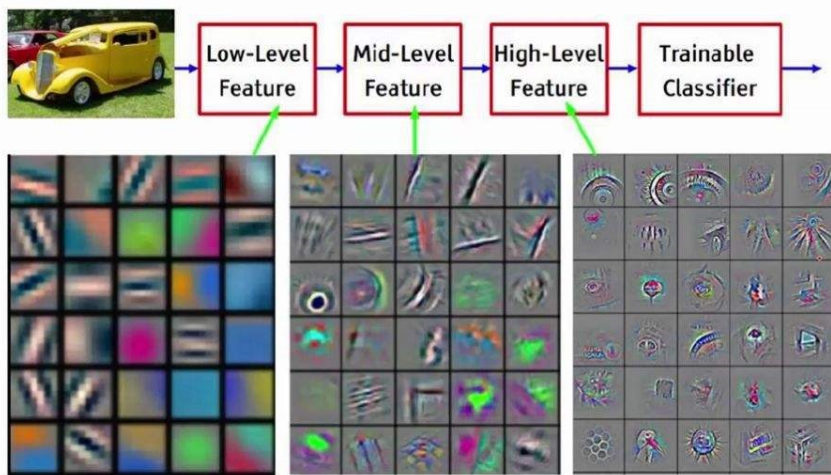
- The computations of Neural Networks have further meanings:
 - The convolutional layers: extract useful features with shared weights



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

Computational Graphs

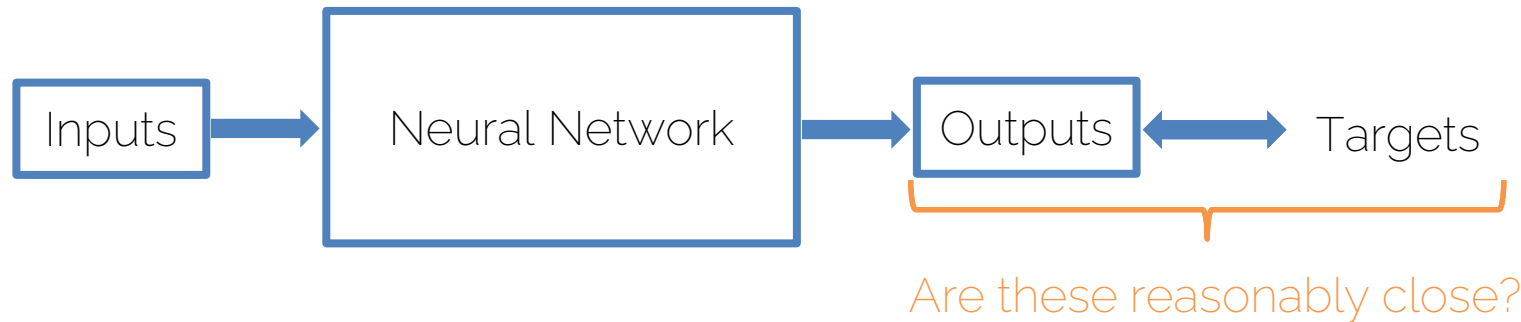
- The computations of Neural Networks have further meanings:
 - The convolutional layers: extract useful features with shared weights



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

Loss Functions

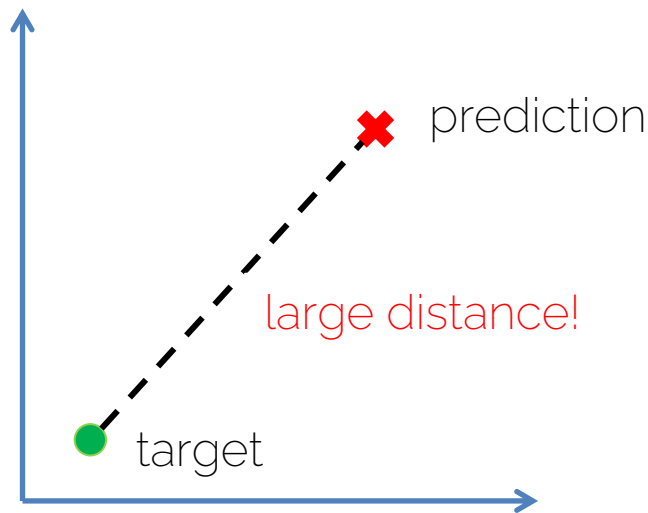
What's Next?



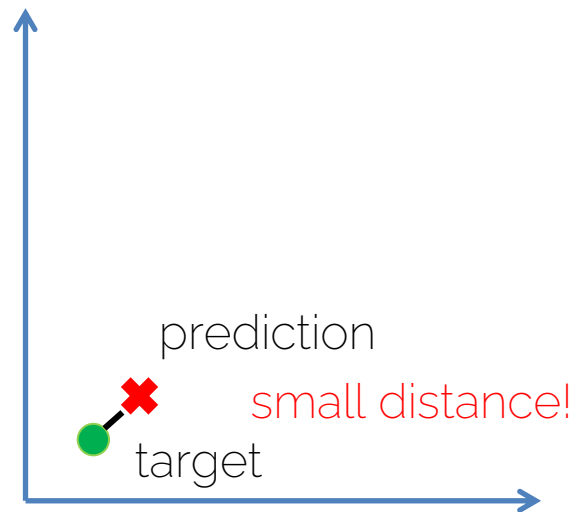
We need a way to describe how close the network's outputs (= predictions) are to the targets!

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:

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

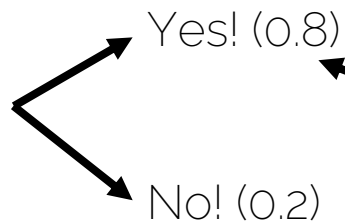
- MSE Loss:

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

Binary Cross Entropy

- Loss function for binary (yes/no) classification

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

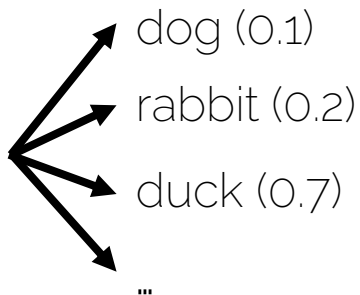
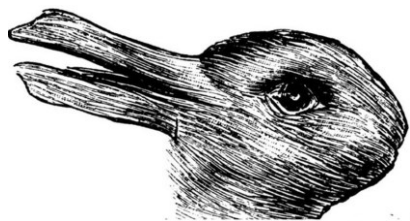


The network predicts the probability of the input belonging to the "yes" class!

Cross Entropy

= loss function for multi-class classification

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

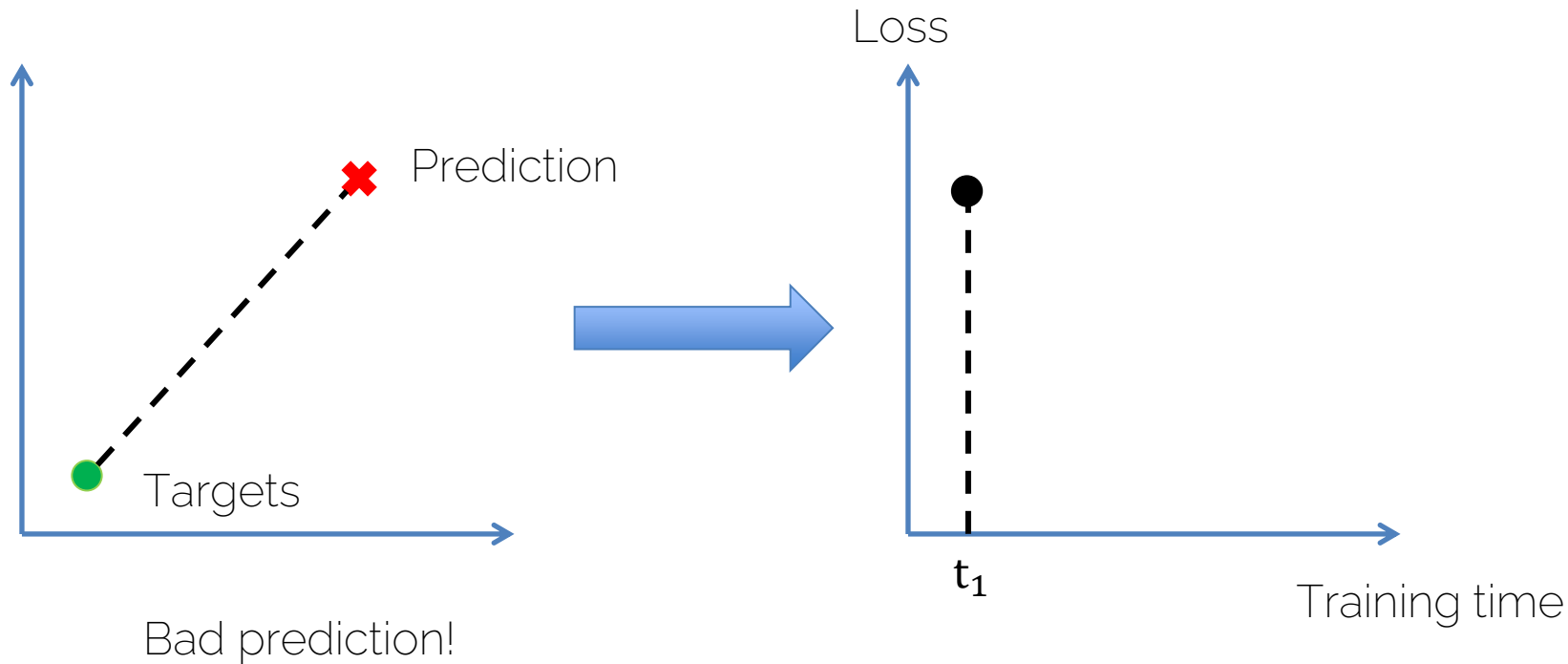


This generalizes the binary case from the slide before!

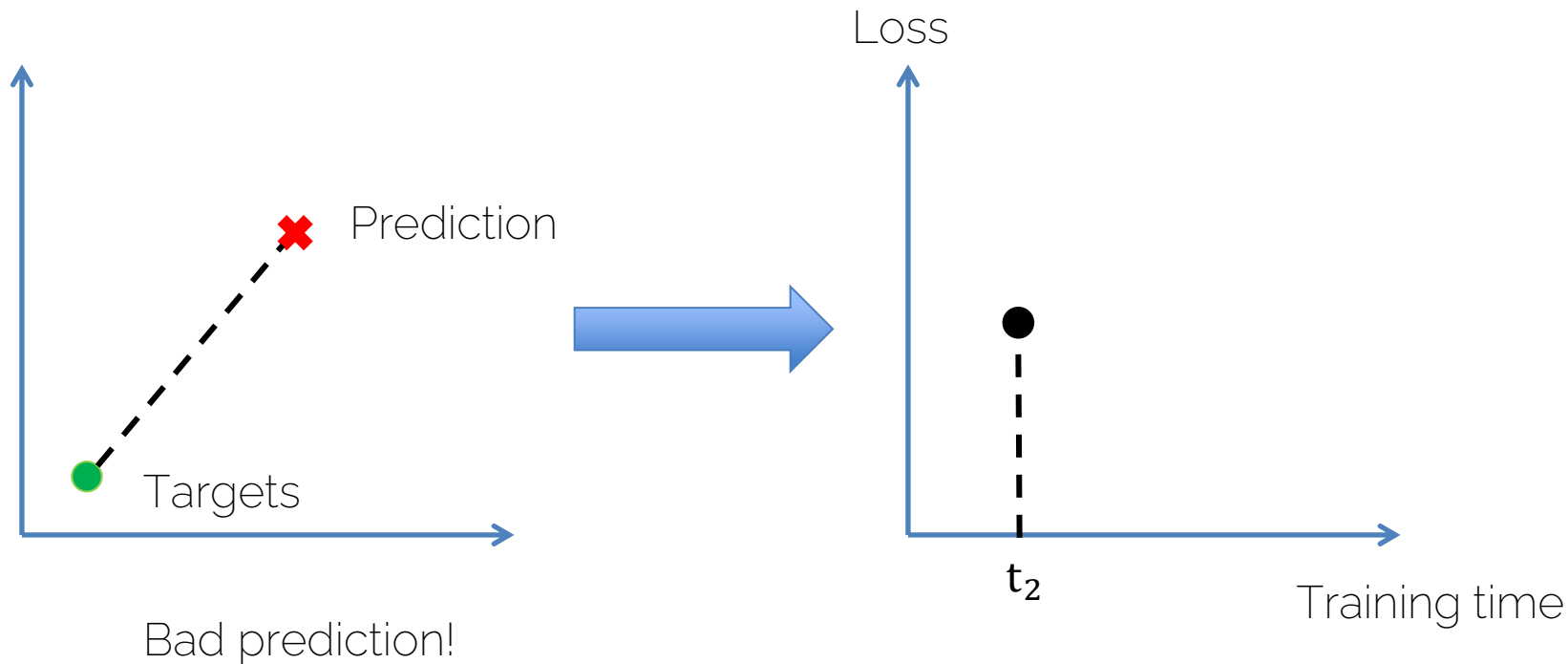
More General Case

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

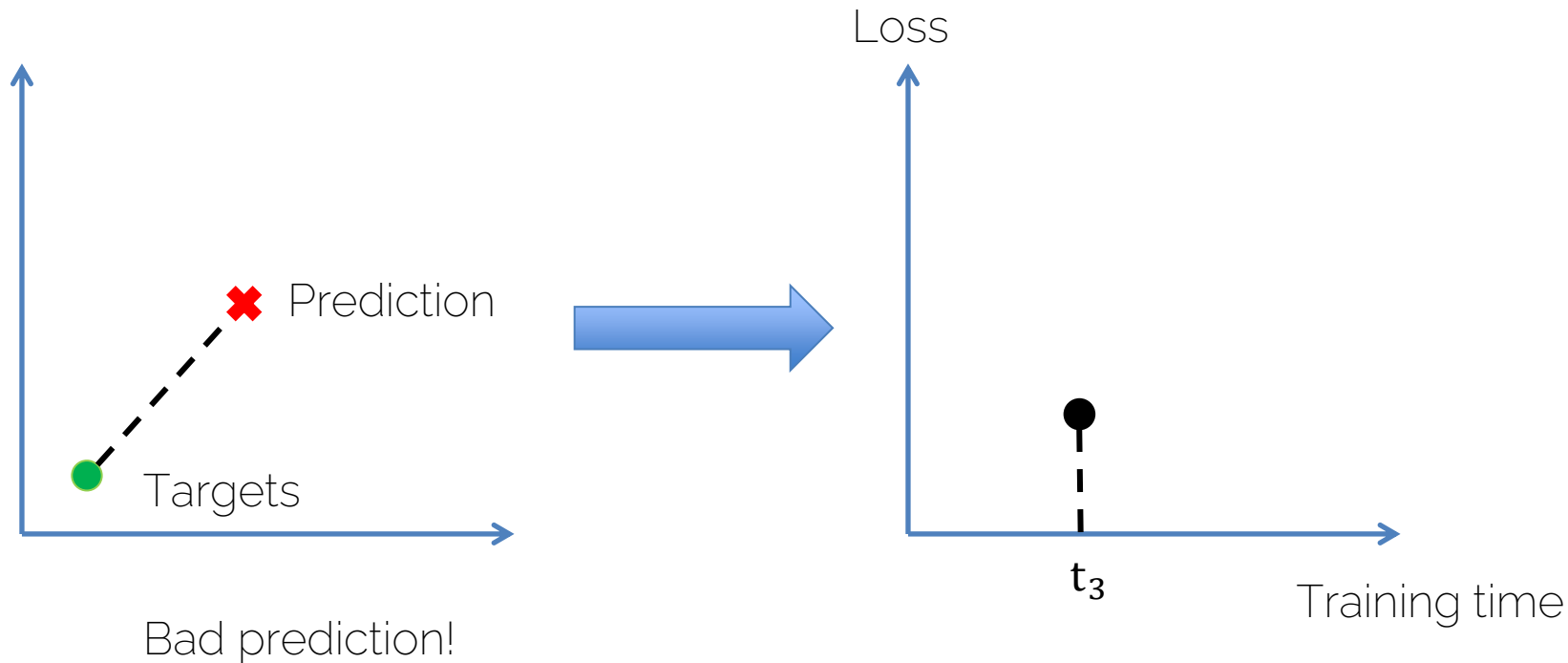
Initially



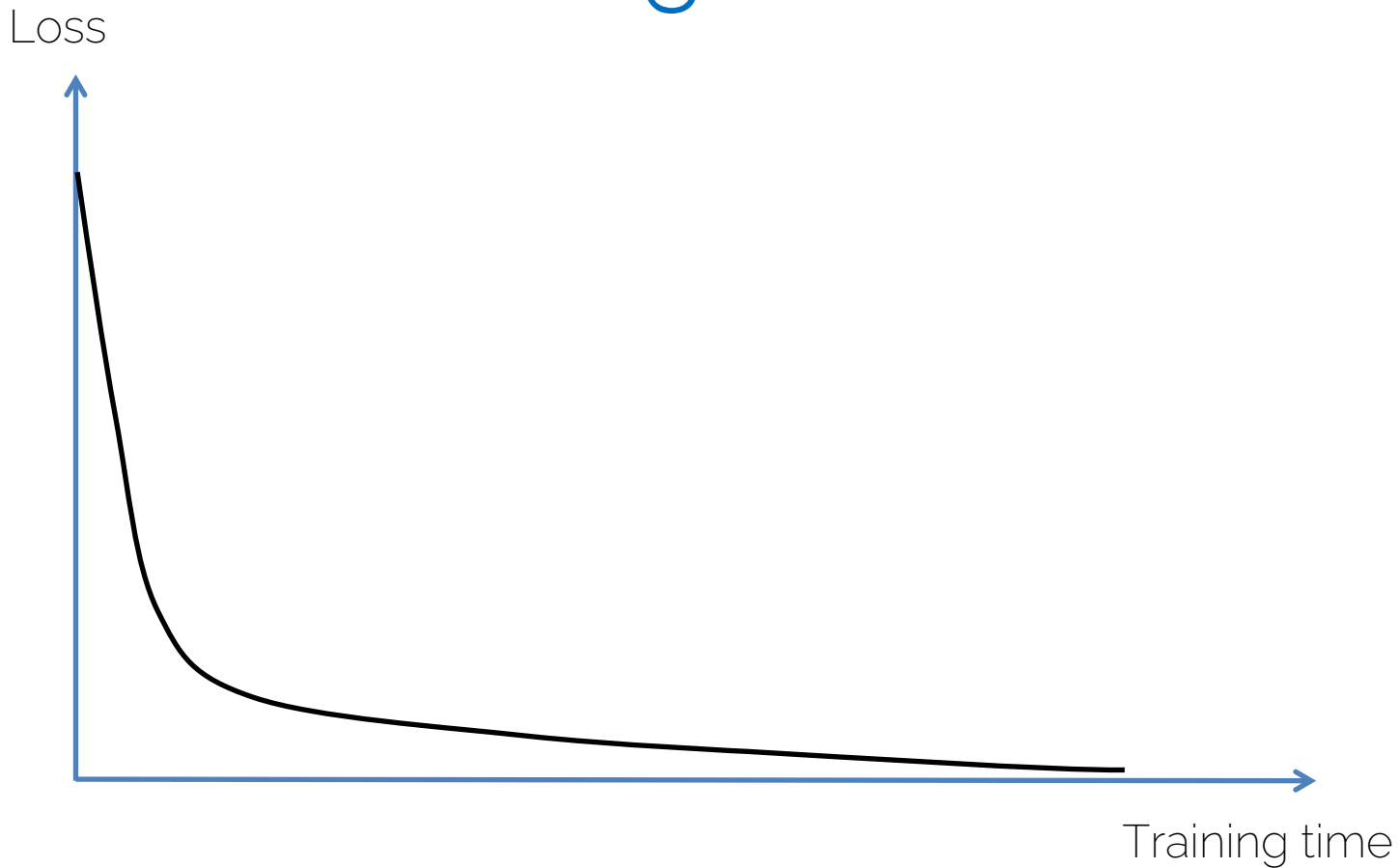
During Training...



During Training...

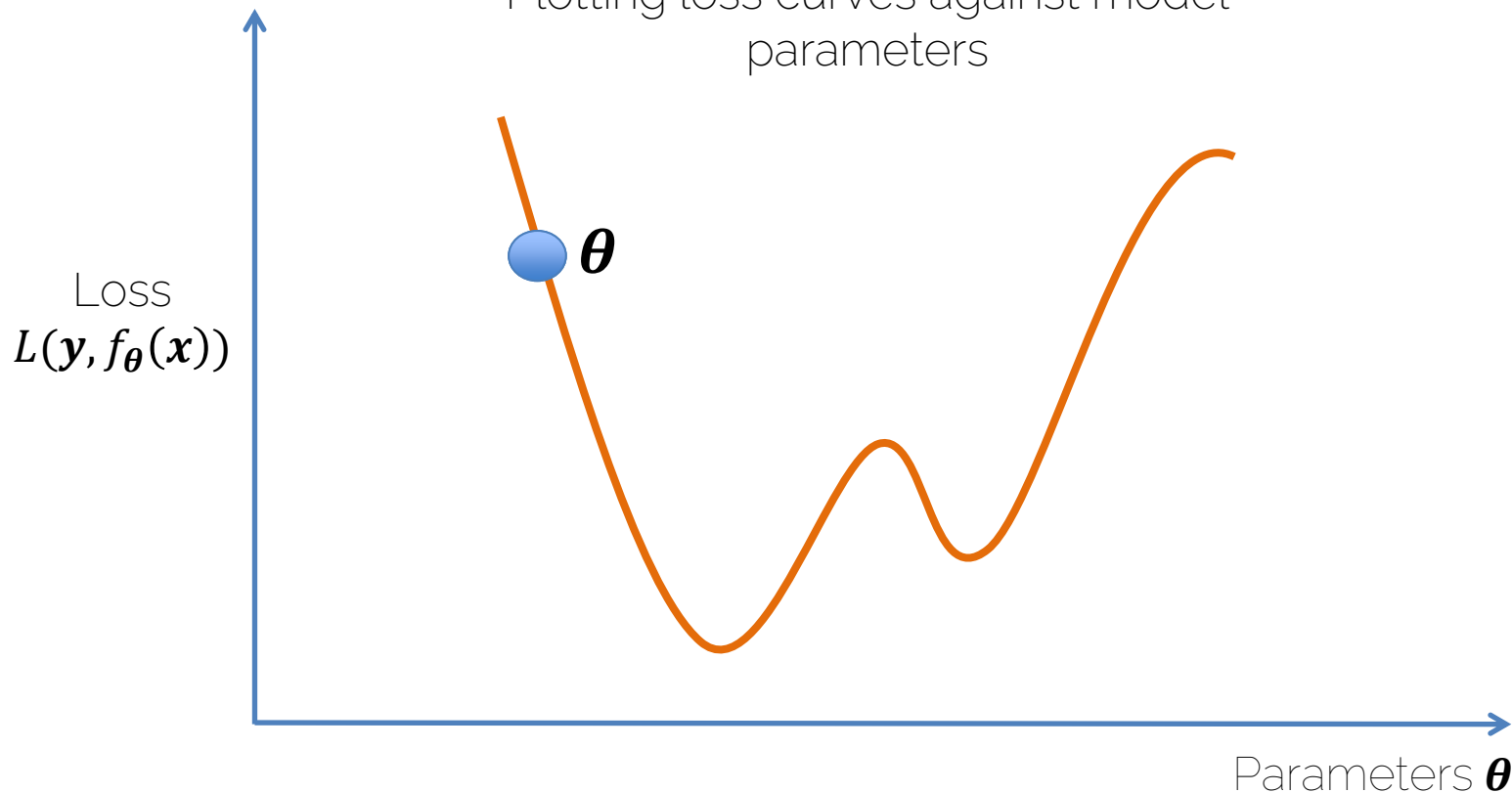


Training Curve



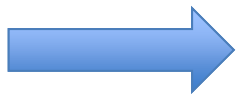
How to Find a Better NN?

Plotting loss curves against model parameters



How to Find a Better NN?

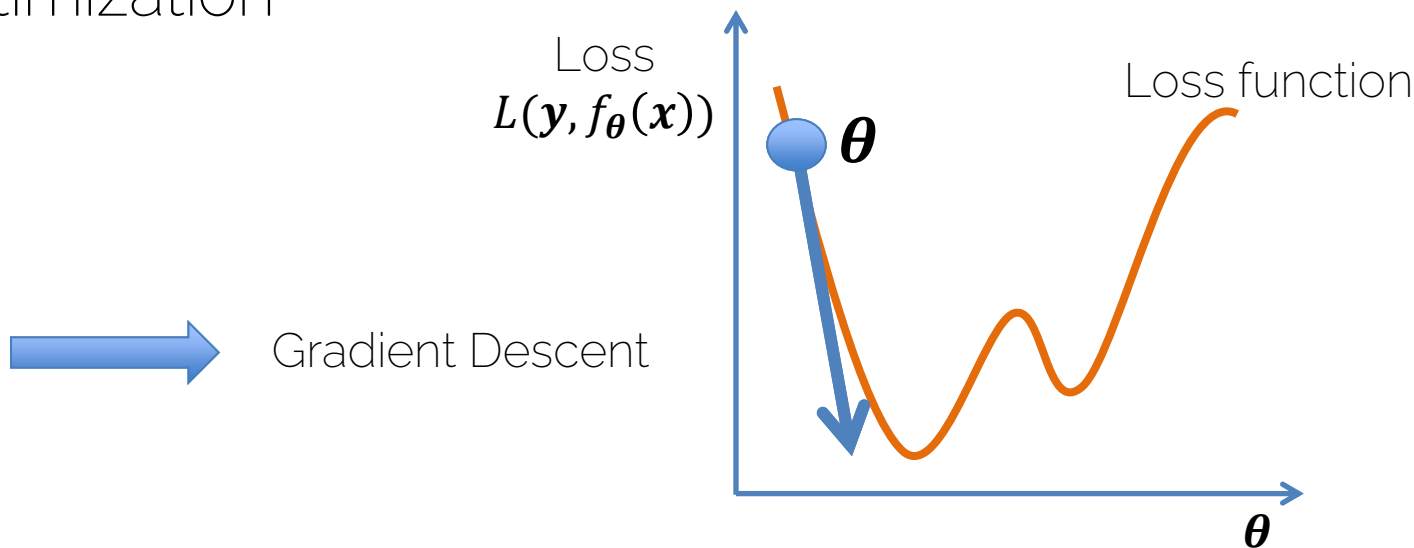
- Loss function: $L(\mathbf{y}, \hat{\mathbf{y}}) = L(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$
- Neural Network: $f_{\boldsymbol{\theta}}(\mathbf{x})$
- Goal:
 - minimize the loss w. r. t. $\boldsymbol{\theta}$



Optimization! We train compute graphs with some optimization techniques!

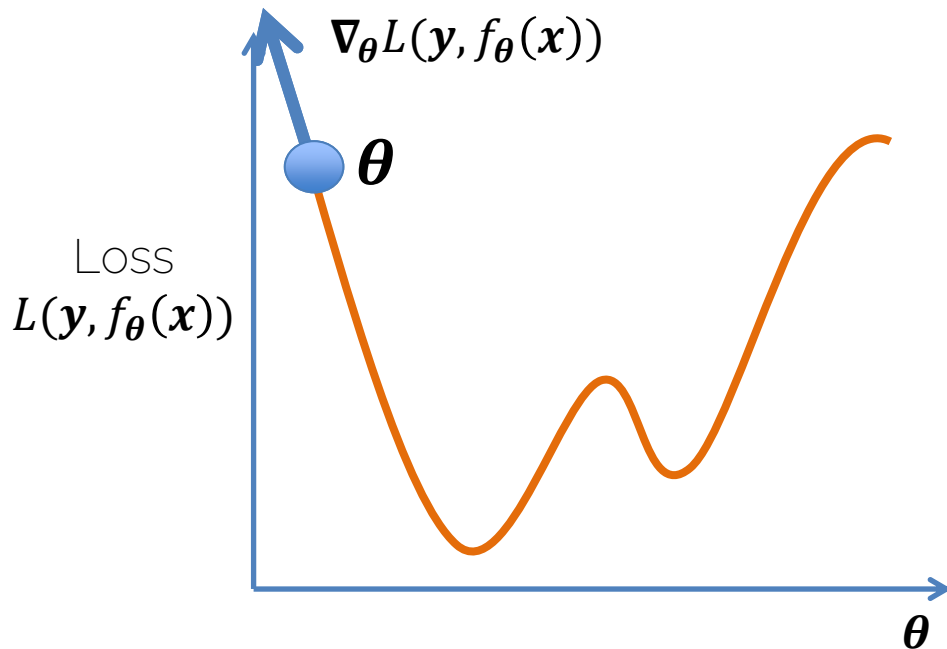
How to Find a Better NN?

- Minimize: $L(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$ w.r.t. $\boldsymbol{\theta}$
- In the context of NN, we use gradient-based optimization



How to Find a Better NN?

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



Learning rate

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$$

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

How to Find a Better NN?

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

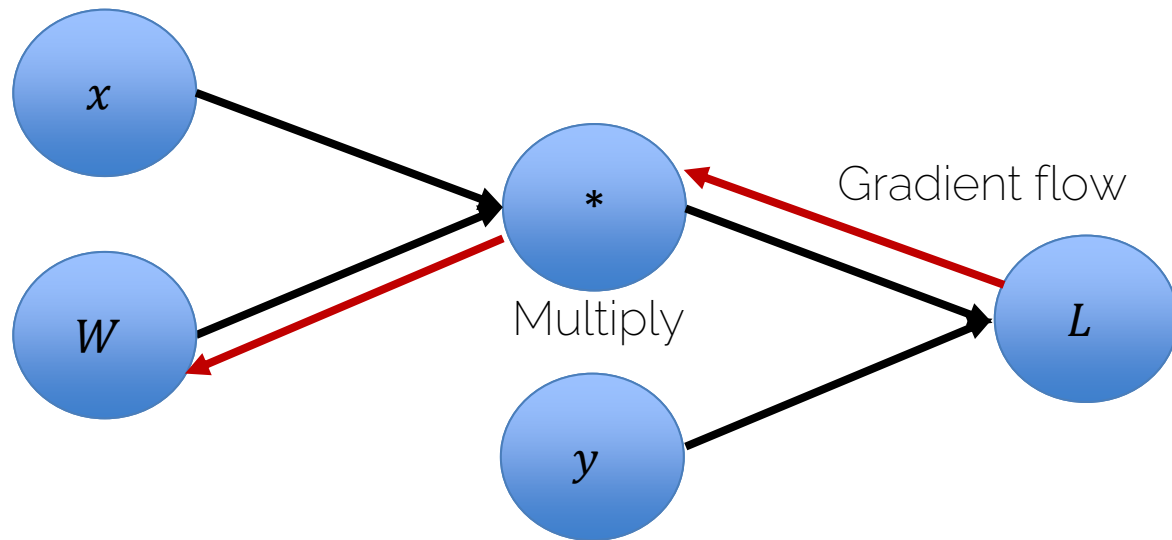
$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}\mathbf{x}, \quad \boldsymbol{\theta} = \mathbf{W}$$

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

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

How to Find a Better NN?

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



How to Find a Better NN?

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

$$f_{\theta}(\mathbf{x}) = \mathbf{W}\mathbf{x}, \quad \theta = \mathbf{W}$$

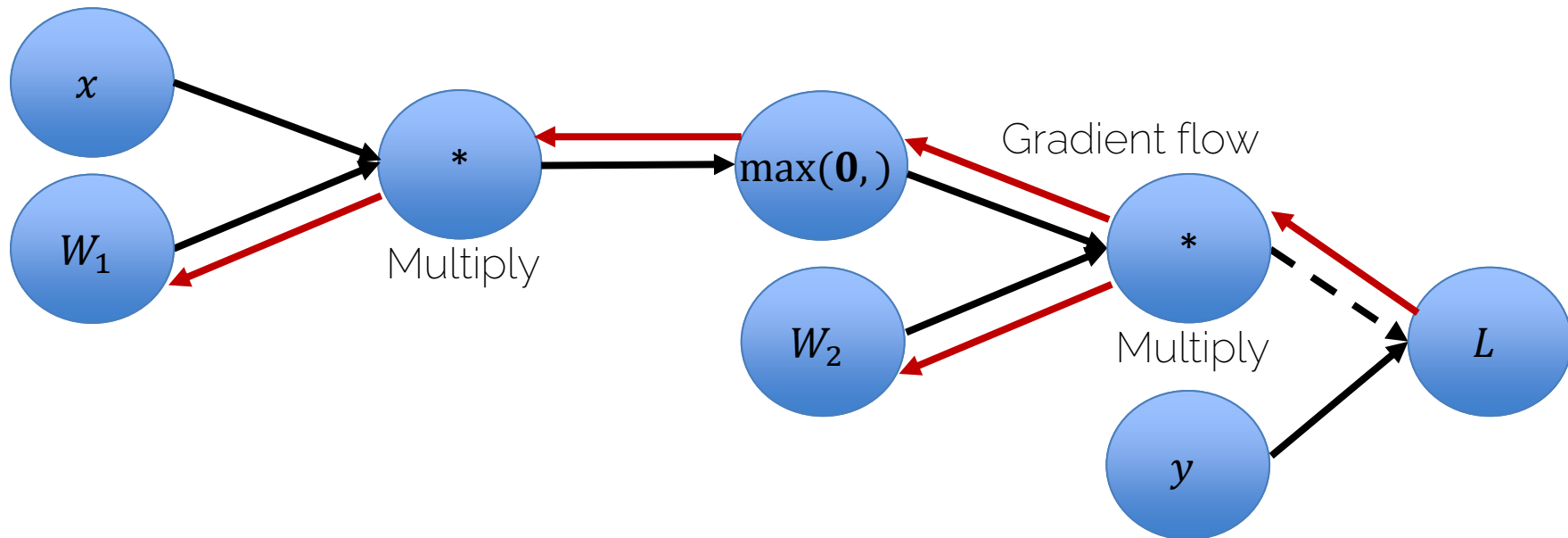
- Given MSE Loss: $L(\mathbf{y}, \hat{\mathbf{y}}; \theta) = \frac{1}{n} \sum_i^n ||\mathbf{W} \cdot \mathbf{x}_i - y_i||_2^2$
- $\nabla_{\theta} L(\mathbf{y}, f_{\theta}(\mathbf{x})) = \frac{2}{n} \sum_i^n (\mathbf{W} \cdot \mathbf{x}_i - y_i) \cdot \mathbf{x}_i^T$

How to Find a Better NN?

- Given inputs \mathbf{x} and targets \mathbf{y}
- Given a multi-layer NN with many activations
$$\mathbf{f} = \mathbf{W}_5 \sigma(\mathbf{W}_4 \tanh(\mathbf{W}_3, \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x}))))$$
- Gradient descent for $L(\mathbf{y}, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}))$ w. r. t. $\boldsymbol{\theta}$
 - Need to propagate gradients from end to first layer (\mathbf{W}_1).

How to Find a Better NN?

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



How to Find a Better NN?

- Given inputs \mathbf{x} and targets \mathbf{y}
- Given multilayer layer NN with many activations
$$\mathbf{f} = \mathbf{W}_5 \sigma(\mathbf{W}_4 \tanh(\mathbf{W}_3, \max(\mathbf{0}, \mathbf{W}_2 \max(\mathbf{0}, \mathbf{W}_1 \mathbf{x}))))$$
- Gradient descent solution for $L(\mathbf{y}, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}))$ w. r. t. $\boldsymbol{\theta}$
 - Need to propagate gradients from end to first layer (\mathbf{W}_1)
- Backpropagation: Use chain rule to compute gradients
 - Compute graphs come in handy!

How to Find a Better NN?

- Why gradient descent?
 - Easy to compute using compute graphs
- Other methods include
 - Newtons method
 - L-BFGS
 - Adaptive moments
 - Conjugate gradient

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

- Optimization:
 - <http://cs231n.github.io/optimization-1/>
 - <http://www.deeplearningbook.org/contents/optimization.html>
- General concepts:
 - Pattern Recognition and Machine Learning – C. Bishop
 - <http://www.deeplearningbook.org/>