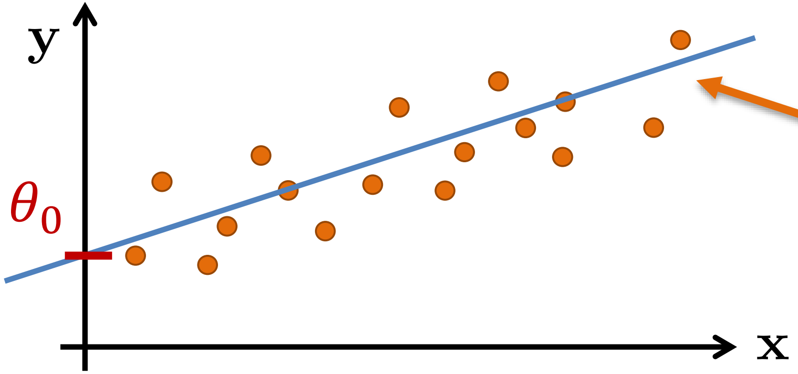
Introduction to Neural Networks

Lecture 2 Recap

Linear Regression

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

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



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

Logistic Regression

- Loss function

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

- Cost function

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

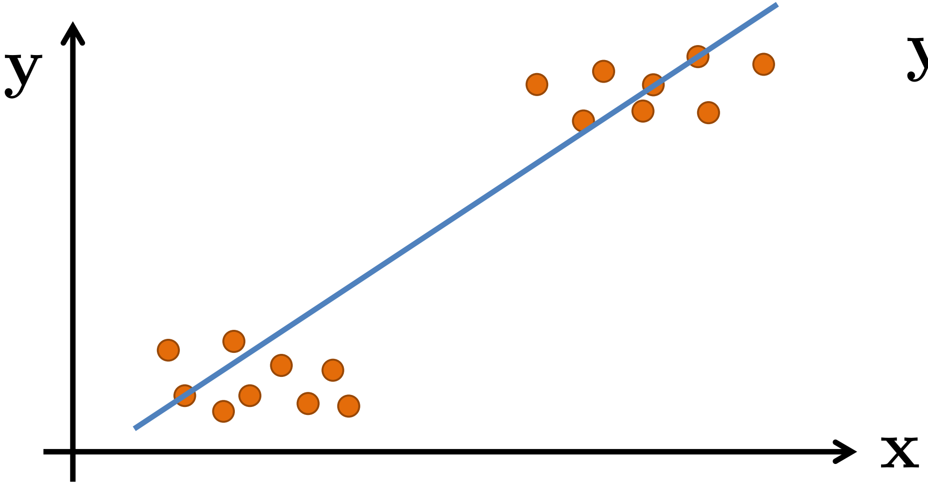
Minimization

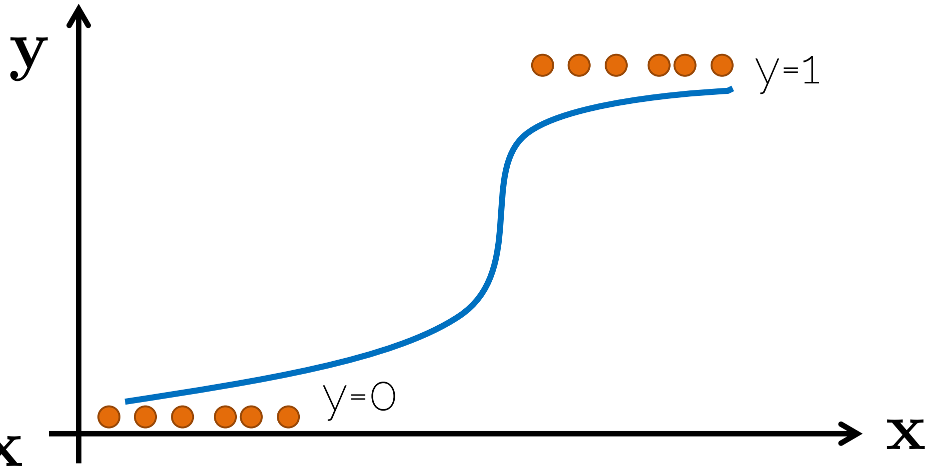
$$\hat{y}_i = \sigma(x_i \boldsymbol{\theta})$$

predictions

Understand

Linear vs Logistic Regression

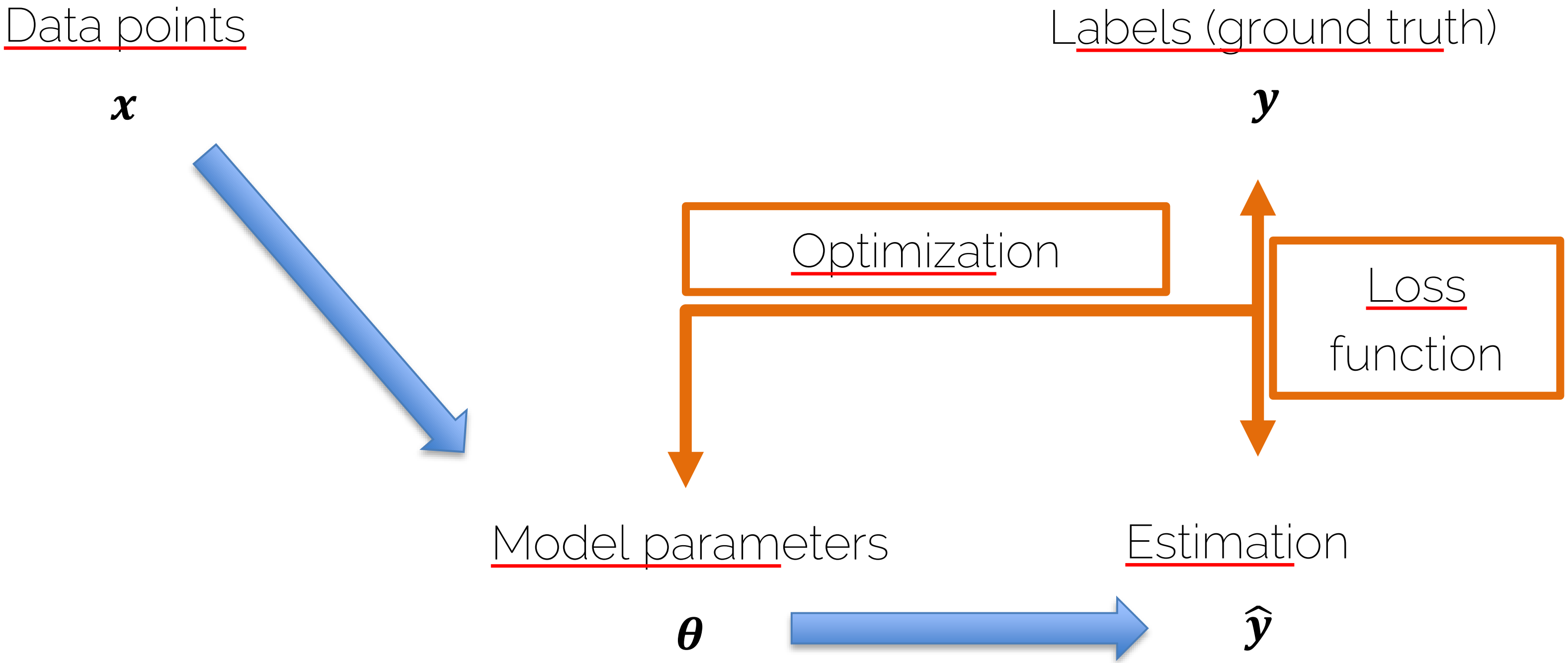




Predictions can exceed the range of the training samples
→ in the case of classification [0;1] this becomes a real issue

Predictions are guaranteed to be within [0;1]

How to obtain the Model?



Linear Score Functions

- Linear score function as seen in linear regression

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

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

* Optimizing in linear score is L2 optim

↳ optimal soln : mean

Linear Score Functions on Images

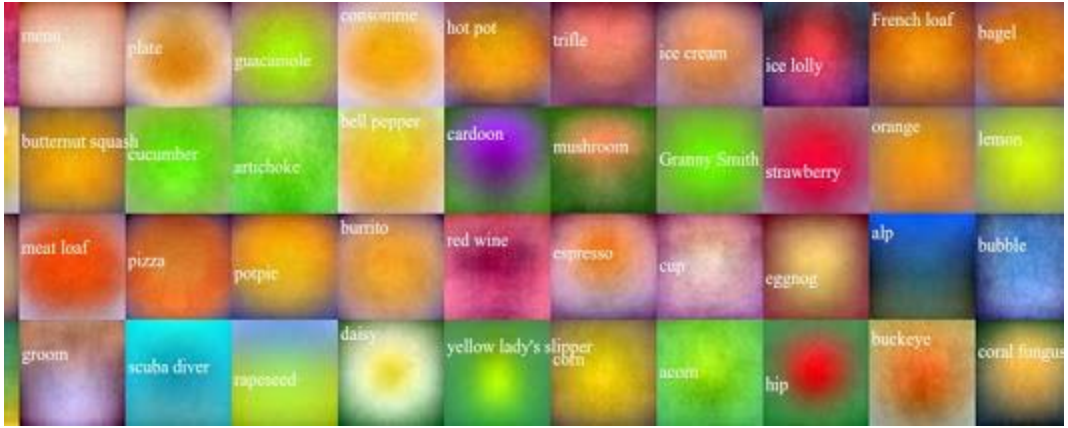
representation of the average image - so to say.

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

On CIFAR-10

object centered in the middle

* Simple dataset might be described by linear fn.



On ImageNet

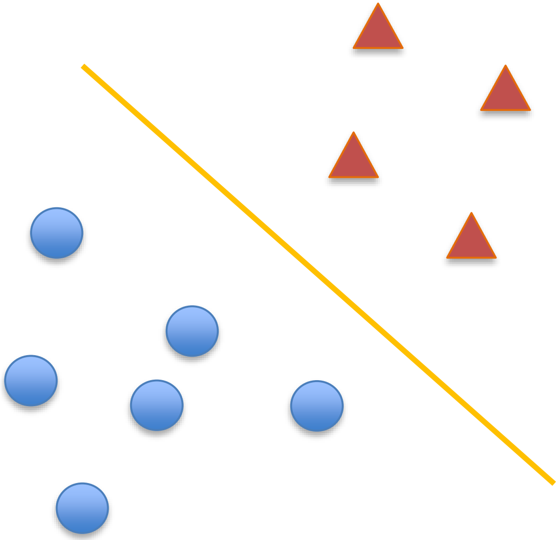
Source:: Li/Karpathy/Johnson

Linear Score Functions?

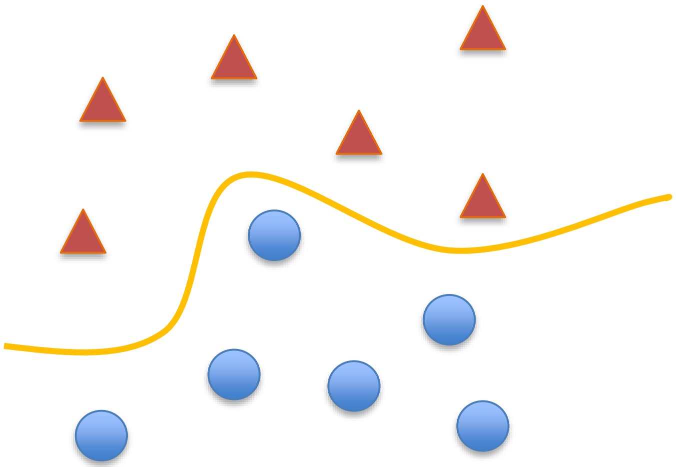
not powerful enough to rep complex fn

Main Problem

Logistic Regression



Linear Separation Impossible!



Linear Score Functions?

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

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

- Operation is still linear.

$$\widehat{\boldsymbol{W}} = \boldsymbol{W_2} \cdot \boldsymbol{W}$$
$$\hat{\boldsymbol{f}} = \widehat{\boldsymbol{W}}\, \boldsymbol{x}$$

- Solution → add non-linearity!!

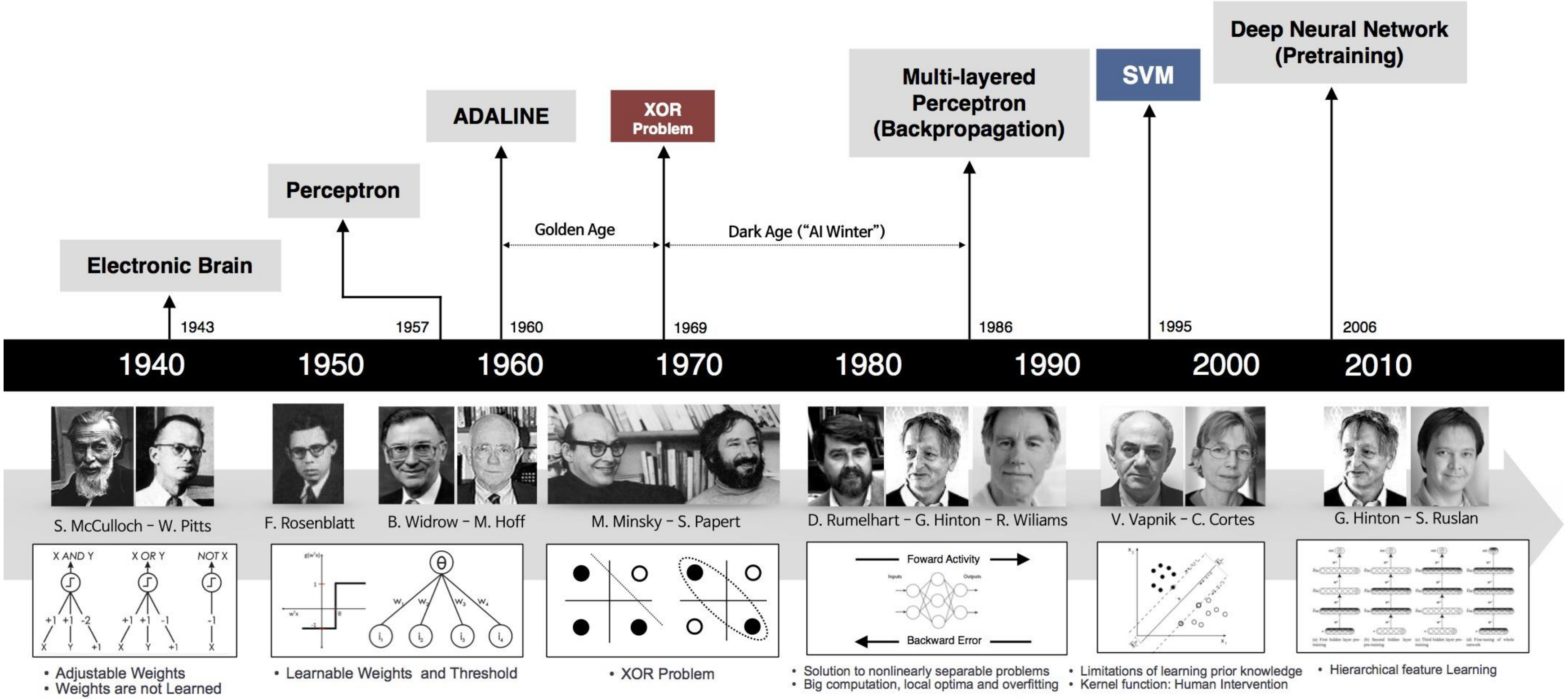
Neural Network

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

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

Introduction to Neural Networks

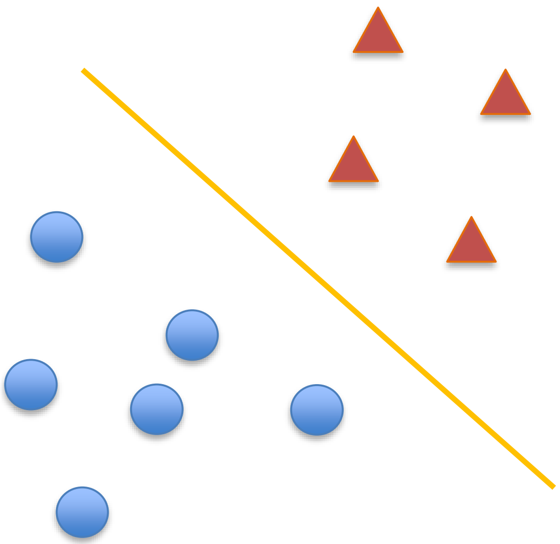
History of Neural Networks



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

Neural Network

Logistic Regression

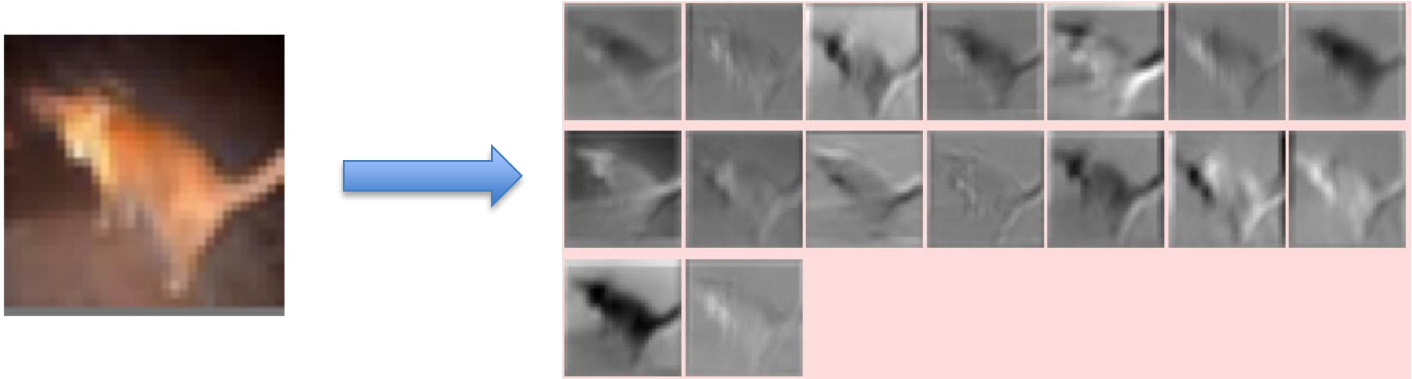


Neural Networks

Neural Network

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

On CIFAR-10



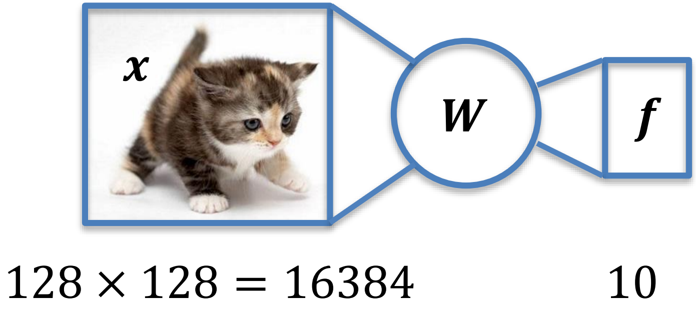
Visualizing activations of first layer.

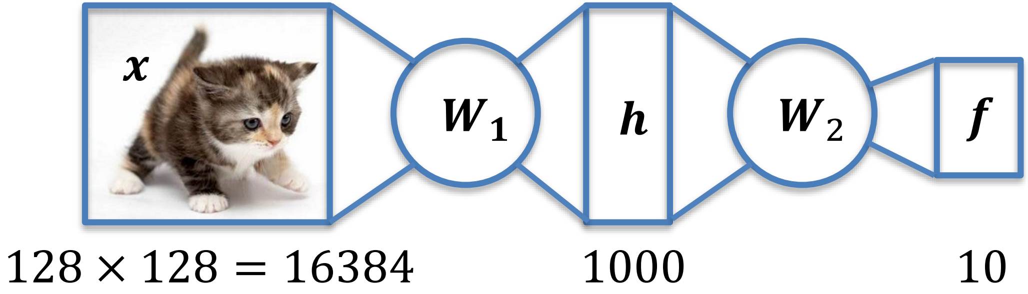
Source: ConvNetJS

Neural Network

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

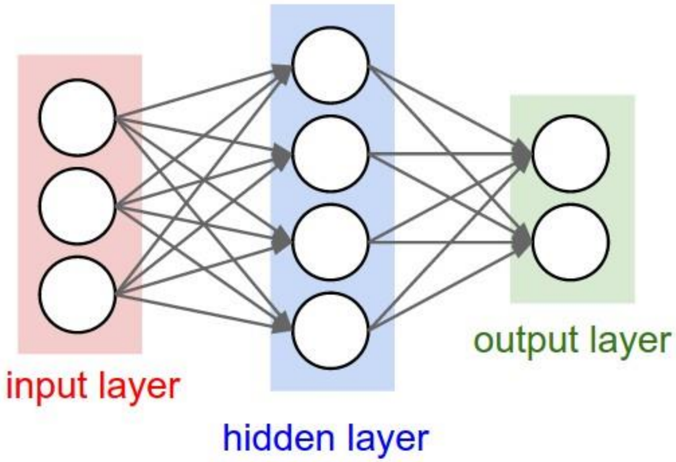
2-layer network: $\boldsymbol{f} = \boldsymbol{W}_2 \max(\boldsymbol{0}, \boldsymbol{W}_1\boldsymbol{x})$



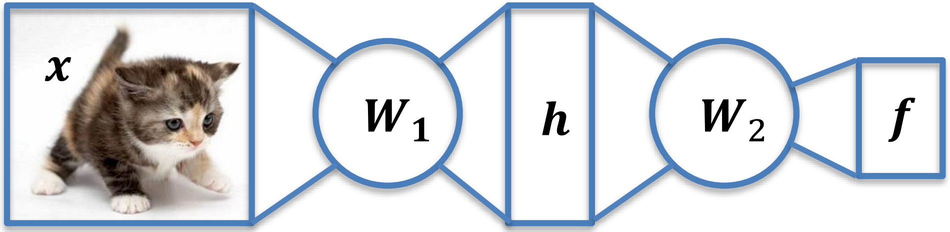


Why is this structure useful?

Neural Network



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

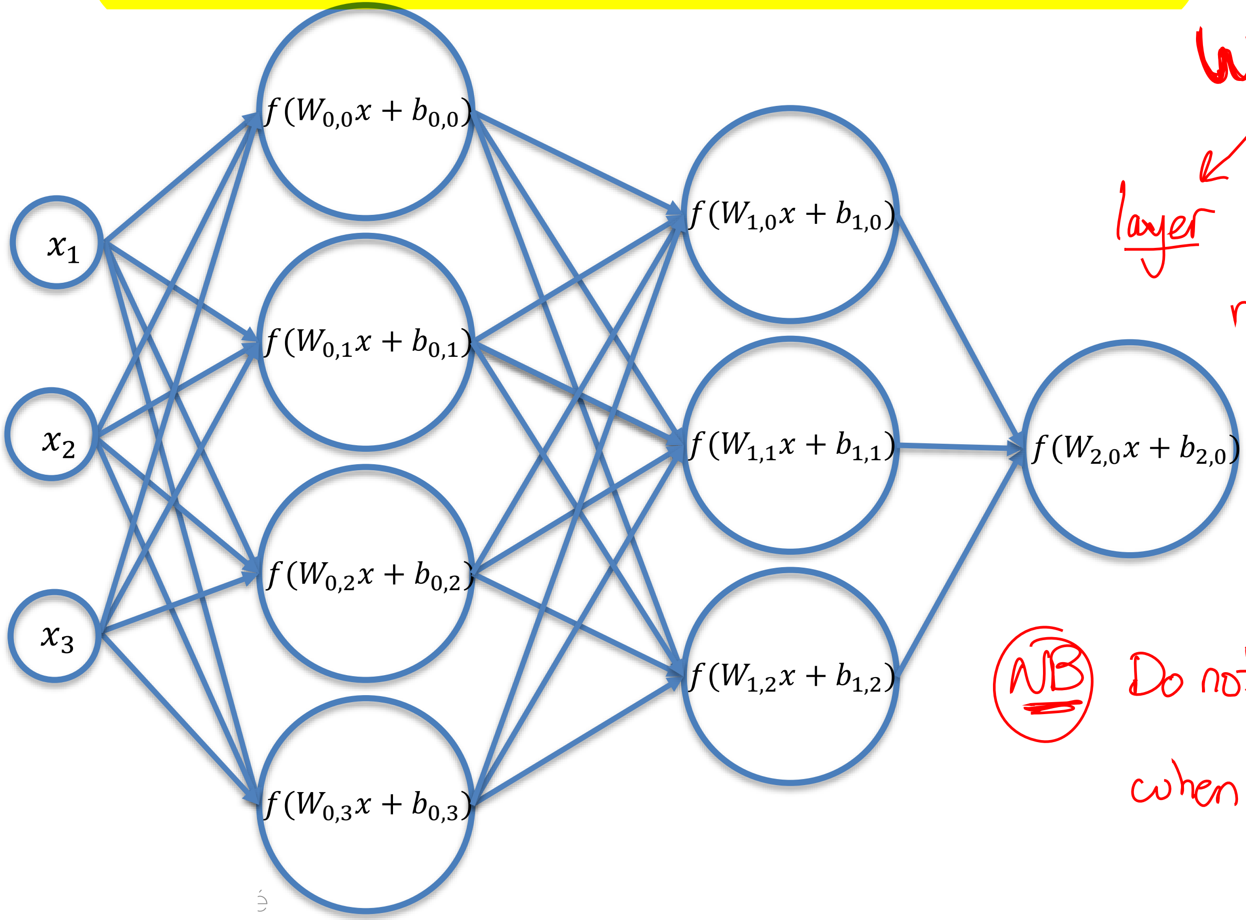


$128 \times 128 = 16384$ 1000 10

Input Layer Hidden Layer Output Layer

dim

Net of Artificial Neurons



$W_{l,m,n}$

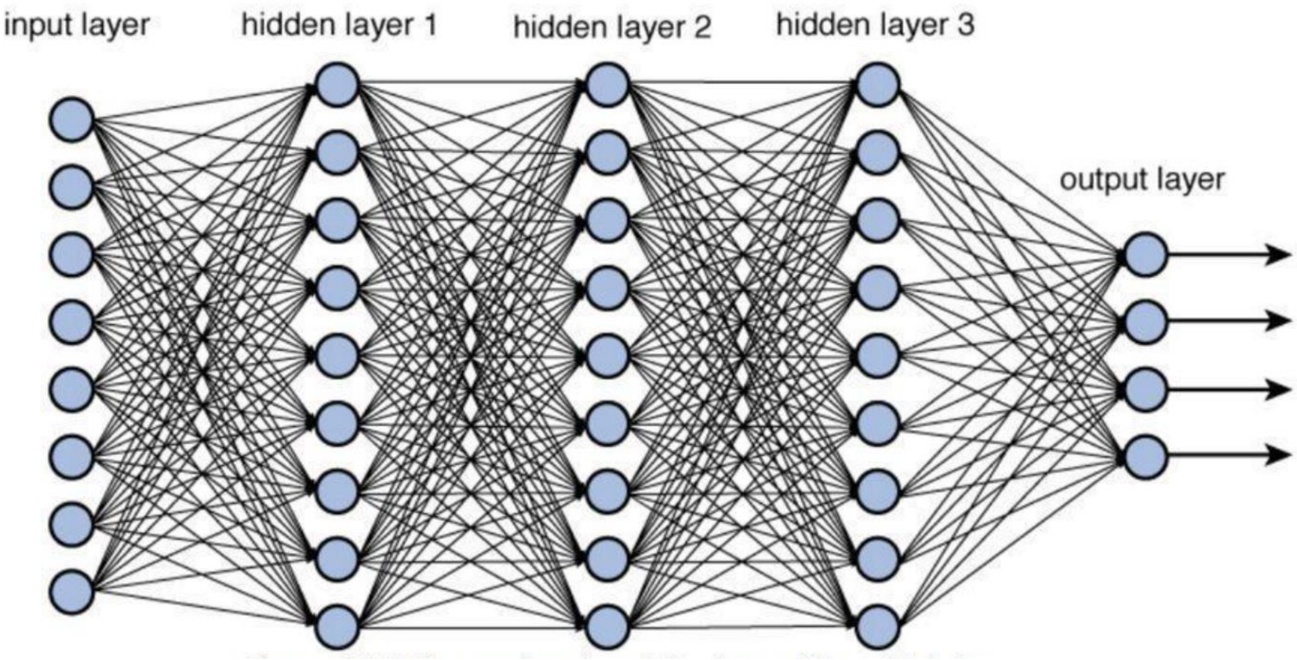
layer

neuron in layer

weight in neuron

NB Do not forget bias when counting weights

Neural Network

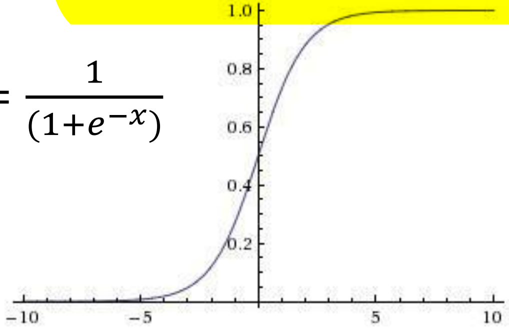


Source: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

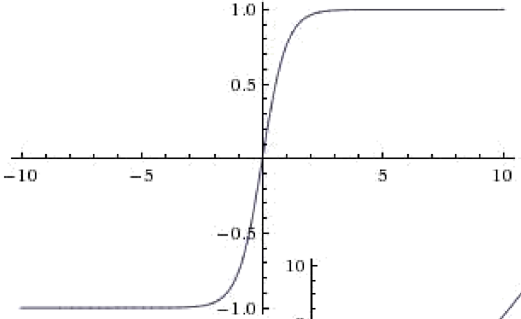
Important

Activation Functions

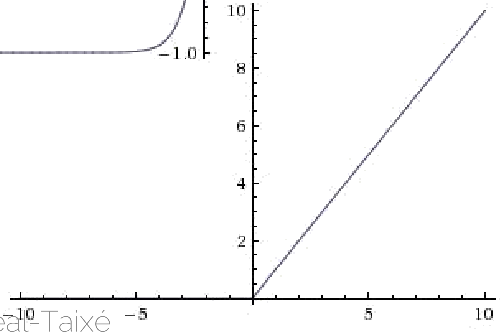
Sigmoid: $\sigma(x) = \frac{1}{(1+e^{-x})}$



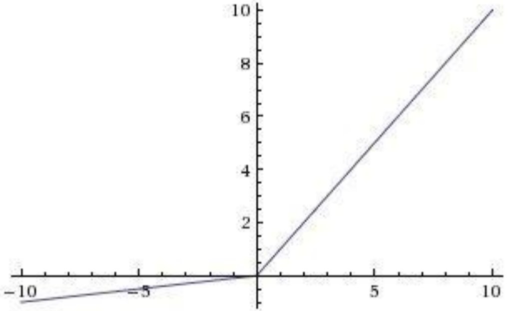
tanh: $\tanh(x)$



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

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

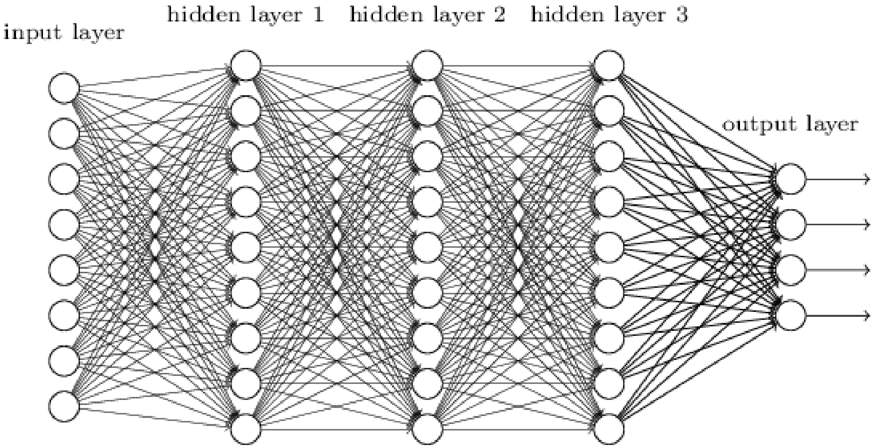
this is still
linear

Why activation functions?

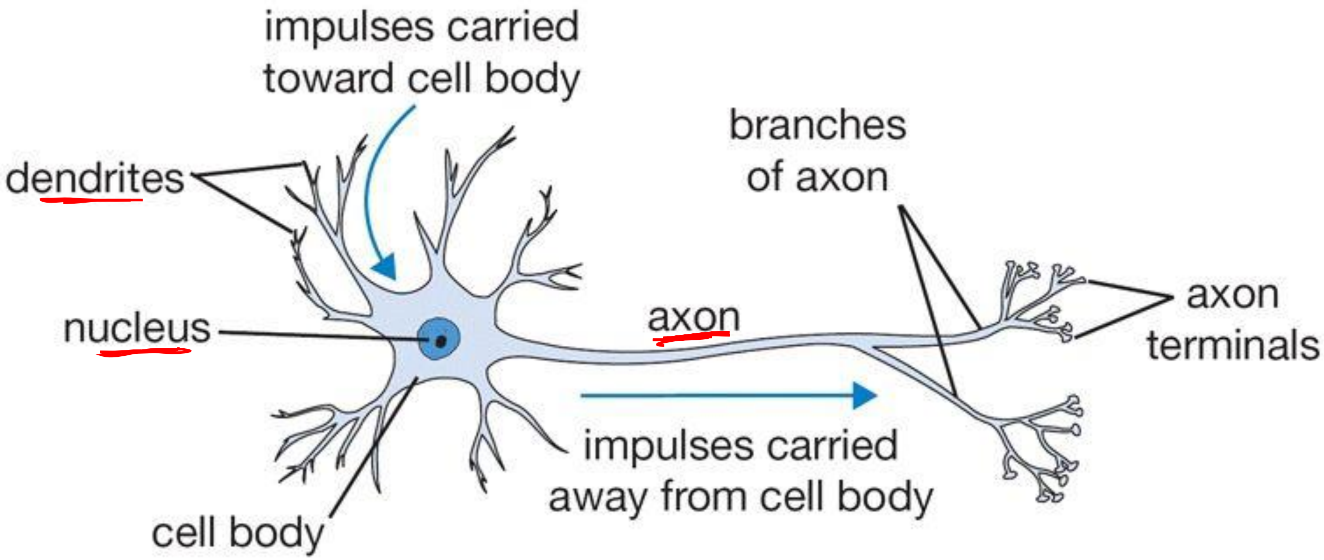
Simply concatenating linear layers would be so much cheaper...

Neural Network

Why organize a neural network into layers?

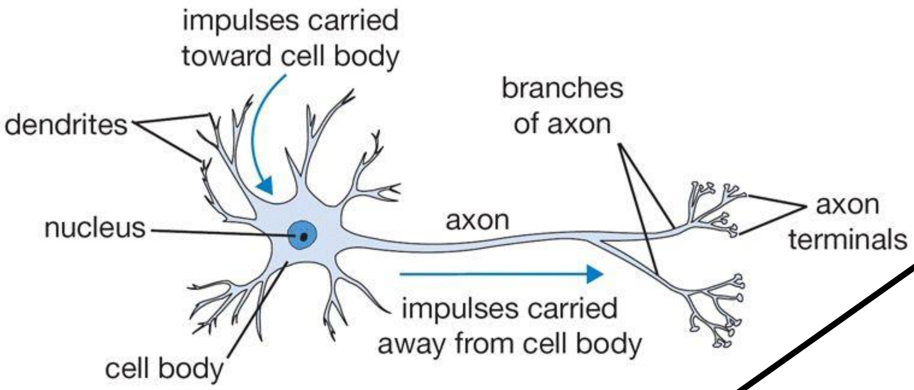


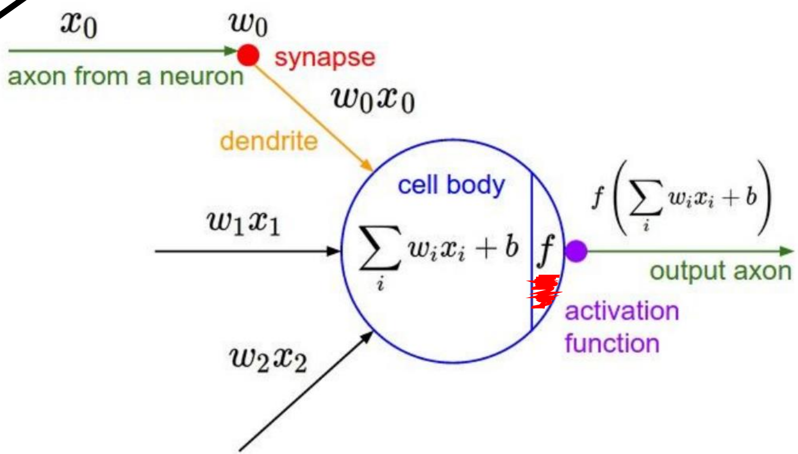
Biological Neurons



Credit: Stanford CS 231n

Biological Neurons





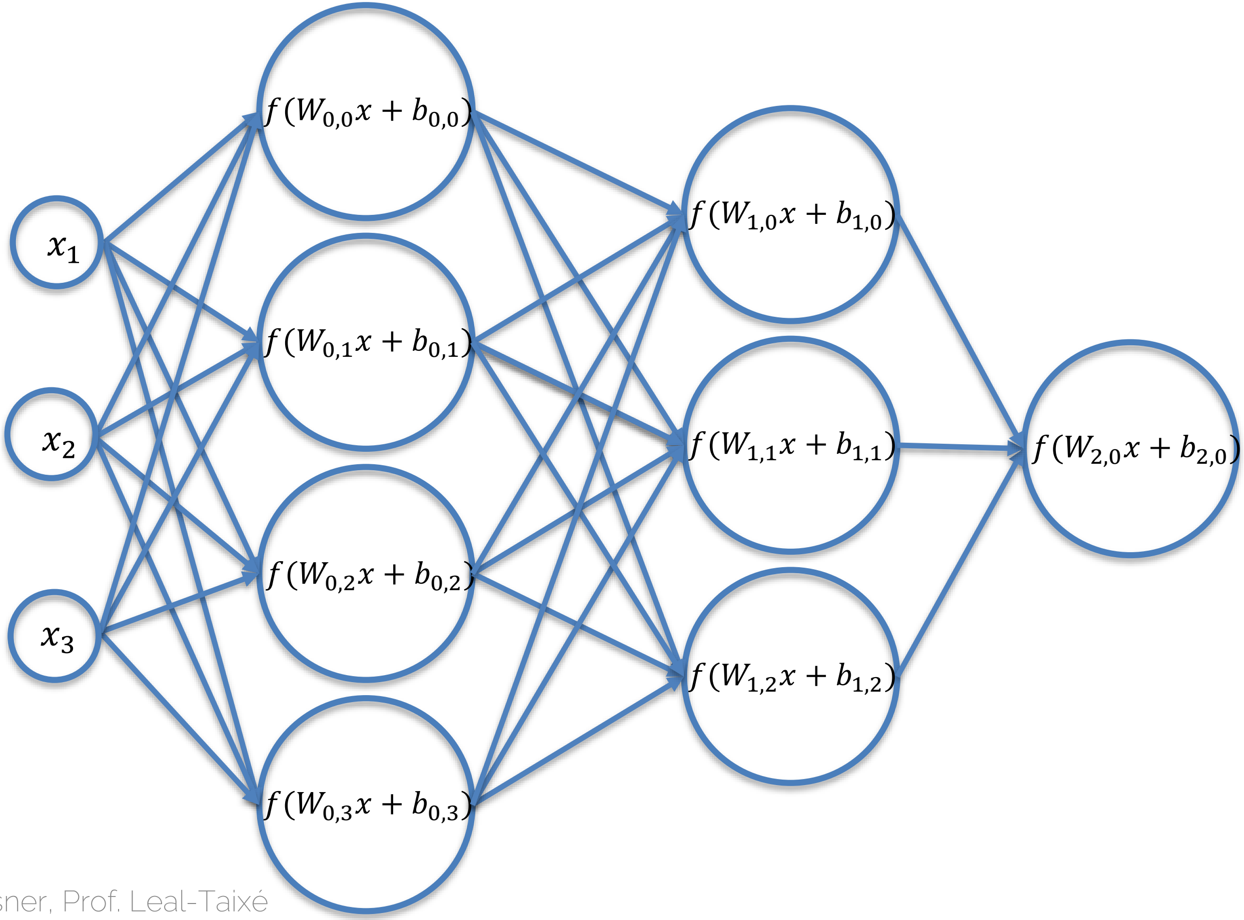
totally irrelevant

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 however... ☺

Artificial Neural Network



Neural Network

- Summary → Supervised
  - Given a dataset with ground truth training pairs $[x_i; y_i]$,

idea → Find optimal weights $\boldsymbol{W}$ 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)

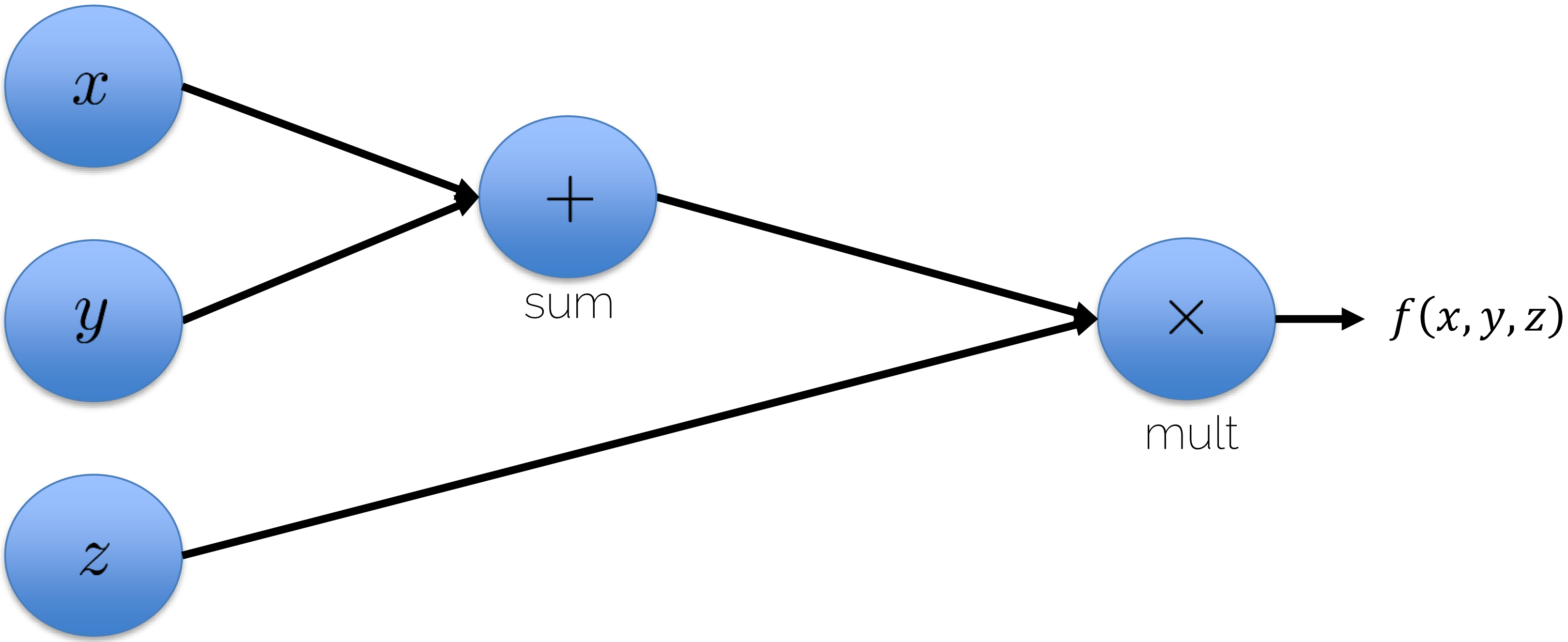
Computational Graphs

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

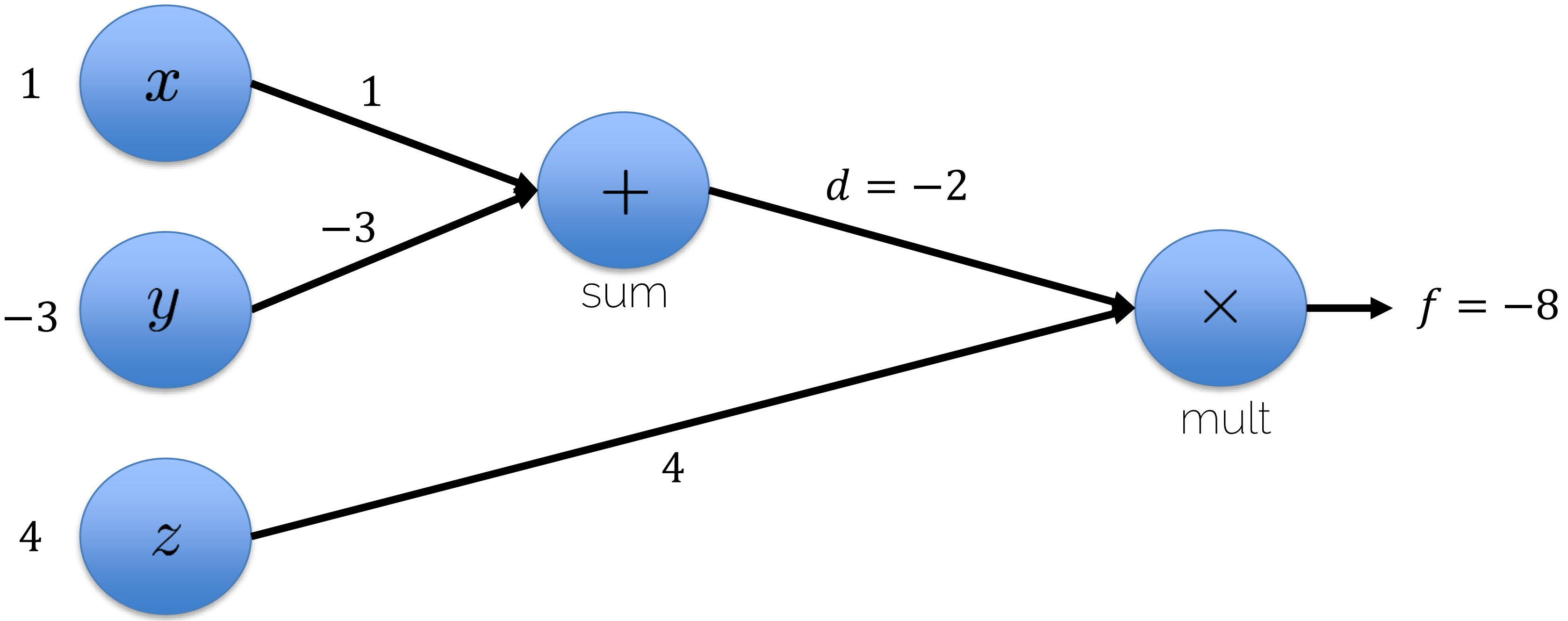
- $f(x, y, z) = (x + y) \cdot z$



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!

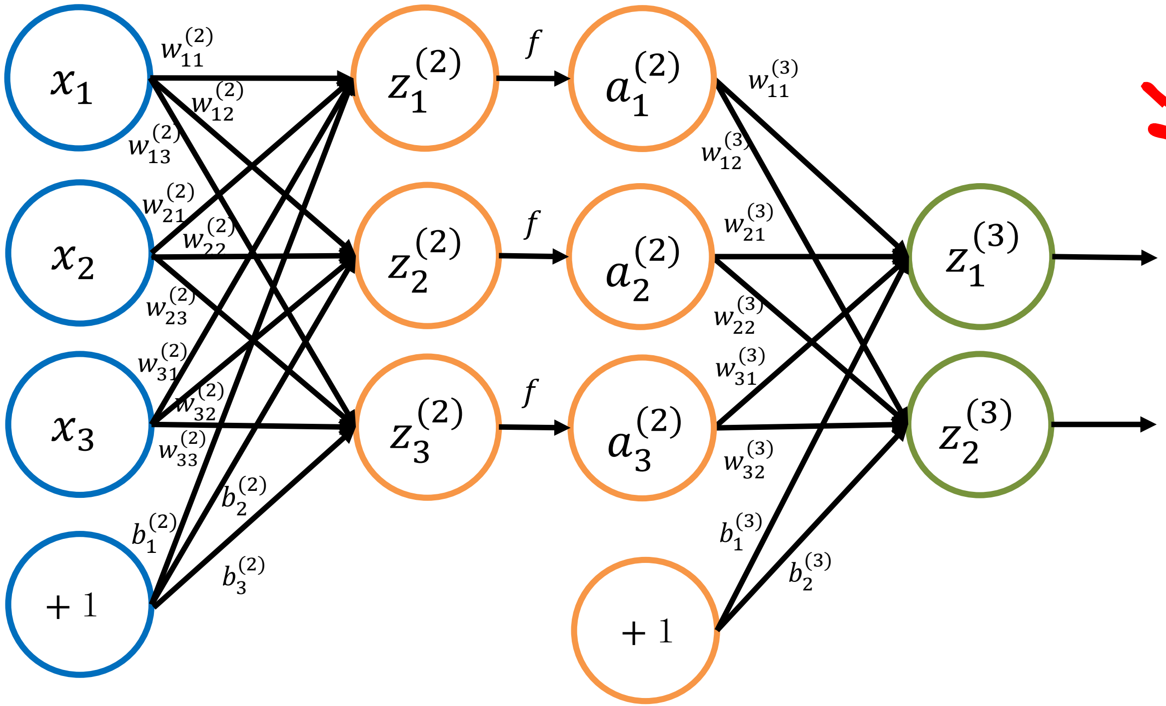
- Represent NN as computational graphs!

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



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

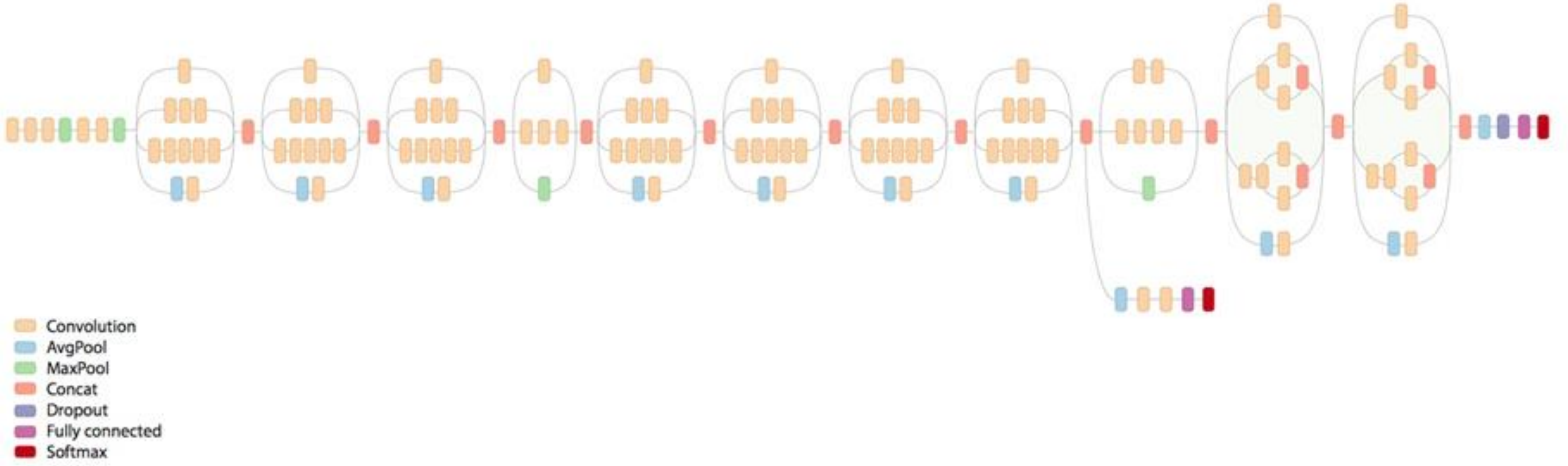
$$a_k^{(2)} = f(z_k^{(2)})$$

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

...

Computational Graphs

- From a set of neurons to a Structured Compute Pipeline

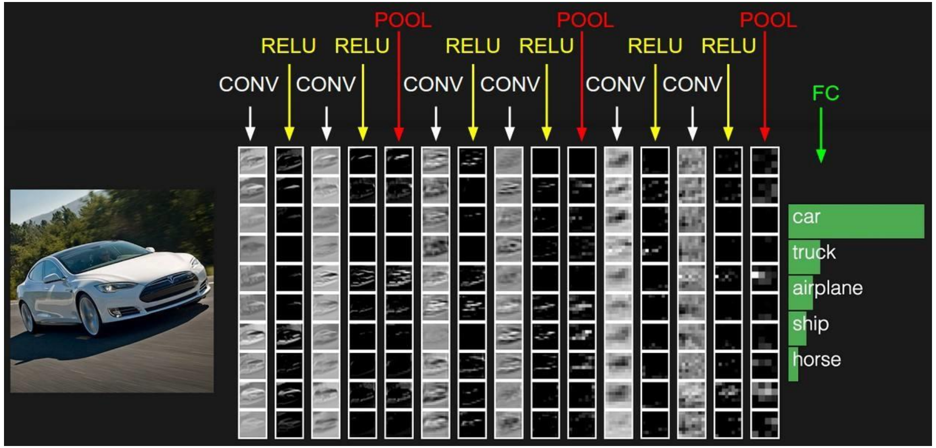


[Szegedy et al.,CVPR'15] Going Deeper with Convolutions

Understand

Computational Graphs

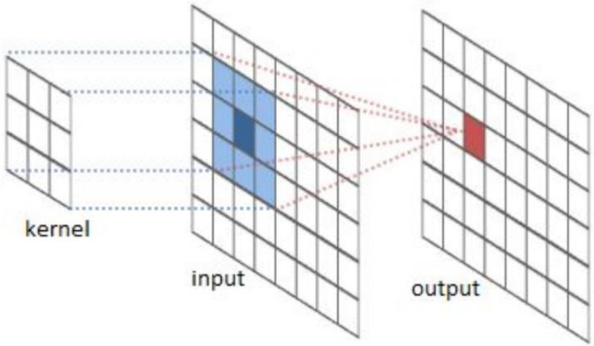
- The computation of Neural Network has further meanings:
  - The multiplication of $\boldsymbol{W_i}$ and $\boldsymbol{x}$: encode input information
  - The activation function: select the key features



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

Computational Graphs

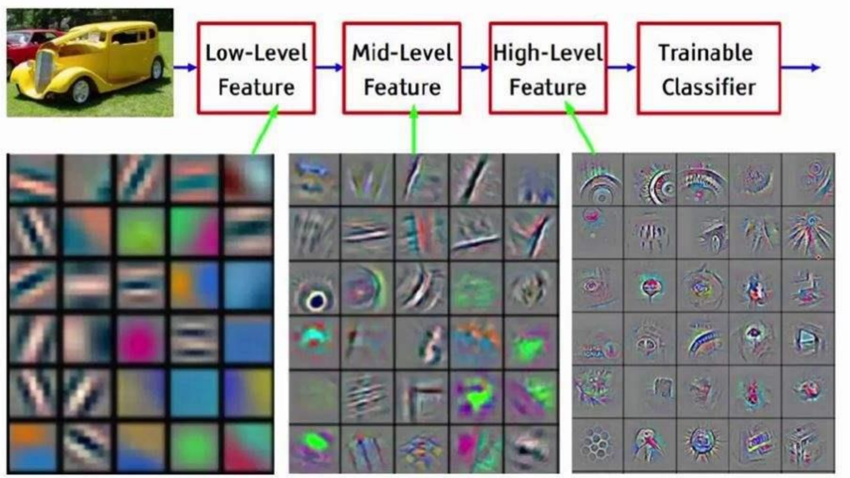
- The computations of Neural Networks have further meanings:
  - The convolutional layers: extract useful features with shared weights



Source: https://www.zcfy.cc/original/understanding-convolutions-colah-s-blog

Computational Graphs

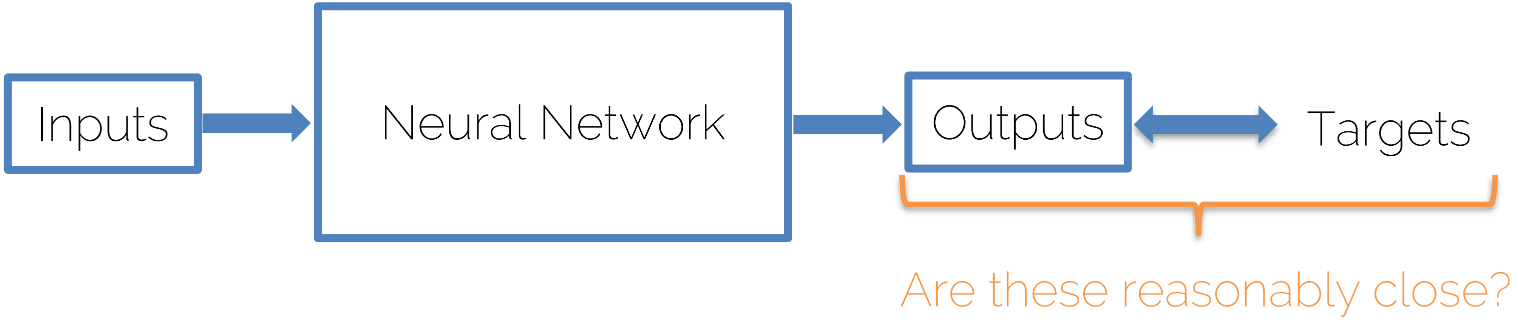
- The computations of Neural Networks have further meanings:
  - The convolutional layers: extract useful features with shared weights



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

Loss Functions

What's Next?

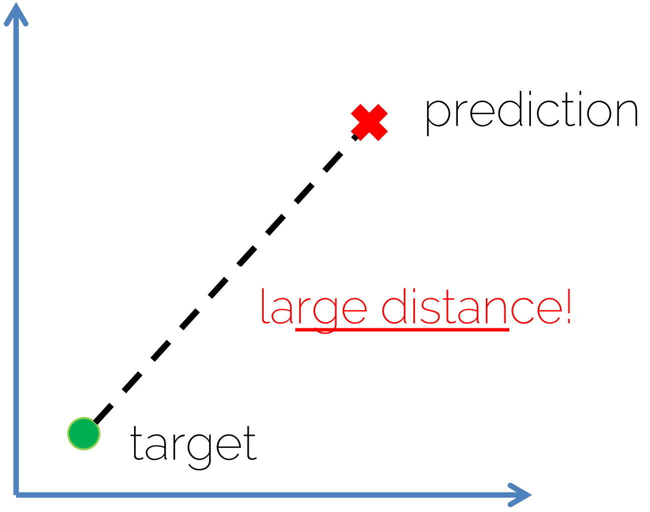


NB

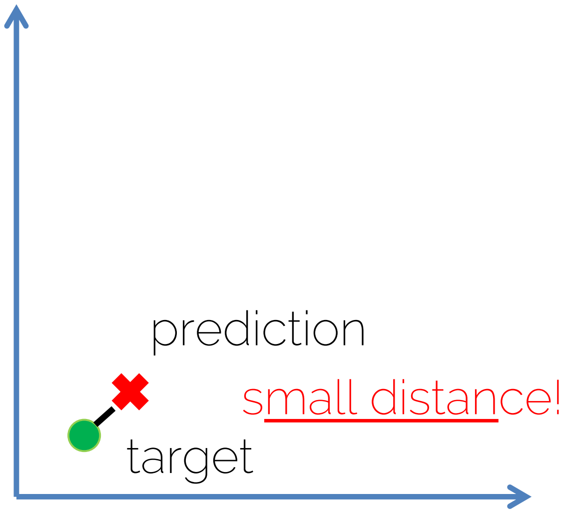
We need a way to describe how close the network's outputs (= predictions) are to the targets!

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

* We normalize to avoid scaling issues

- L1 Loss:

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

- MSE Loss:

L2 Loss

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

highly sensitive to outliers

Binary Cross Entropy

- Loss function for binary (yes/no) classification

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

n → training samples

Yes! (0.8)

No! (0.2)

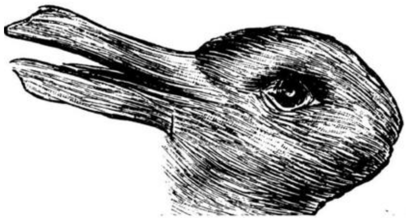
The network predicts the probability of the input belonging to the "yes" class!

Cross Entropy

- loss function for multi-class classification

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

This generalizes the binary case from the slide before!



dog (0.1)

rabbit (0.2)

duck (0.7)

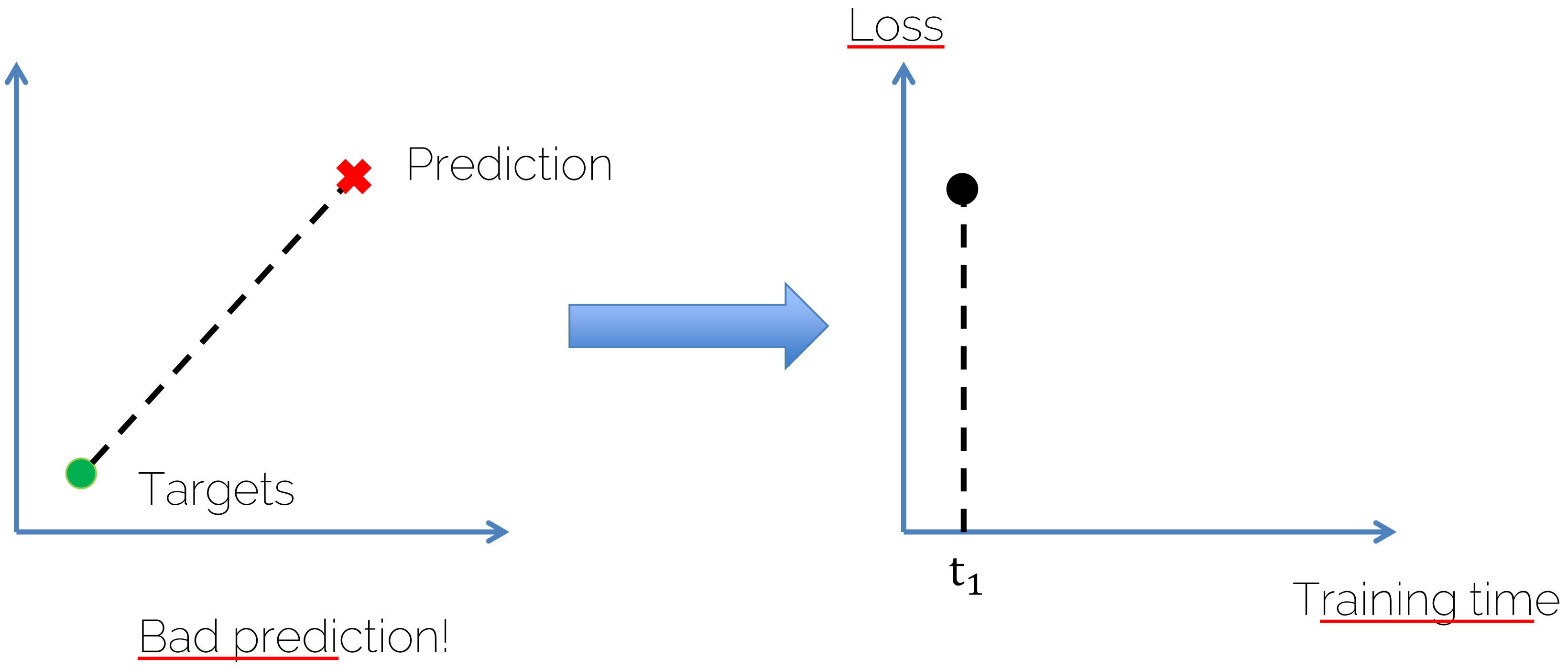
...

k → # classes

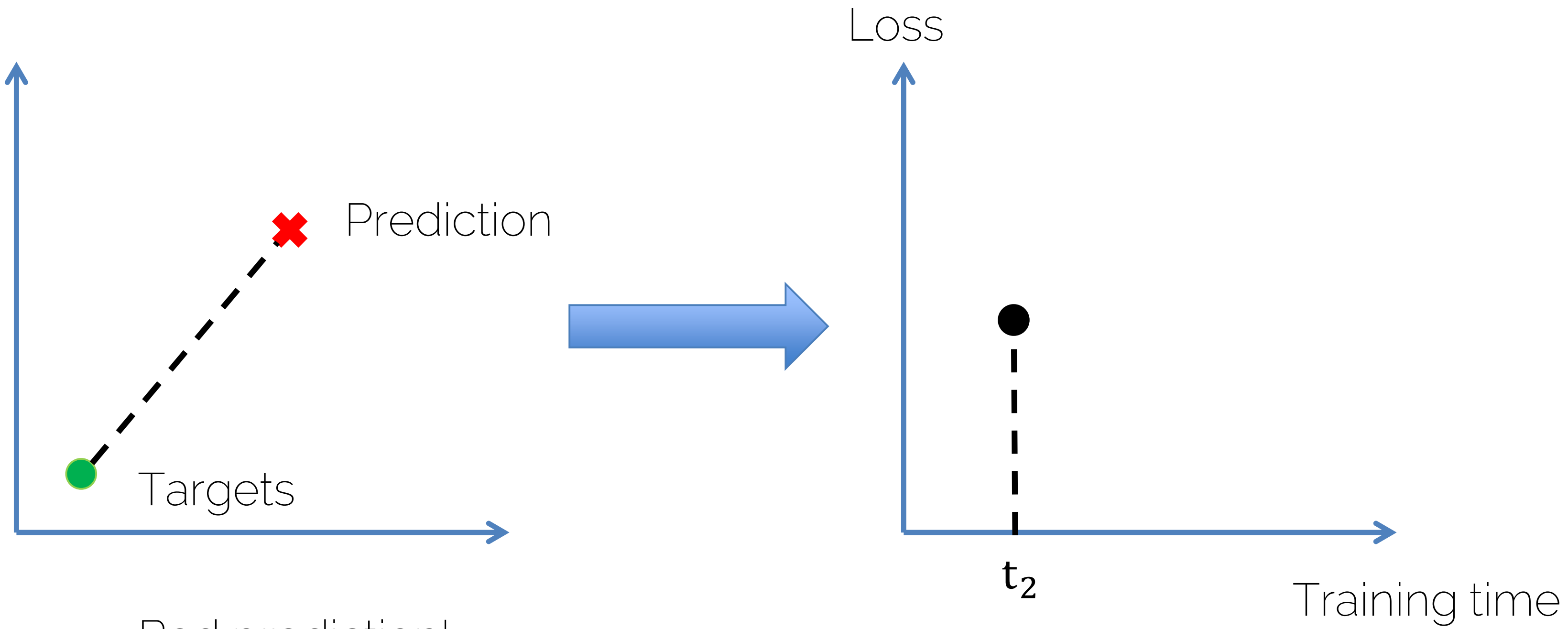
More General Case

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

Initially

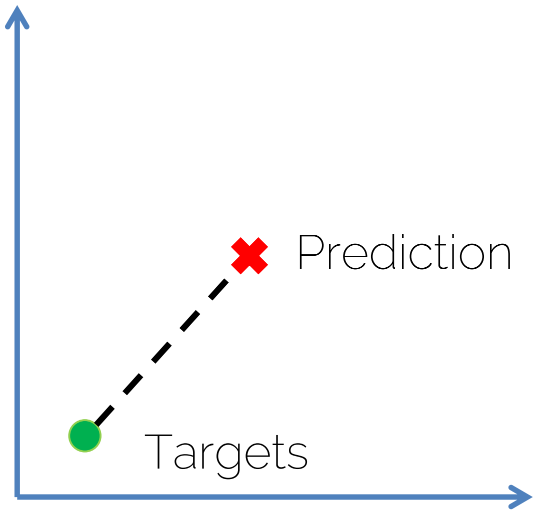


During Training...

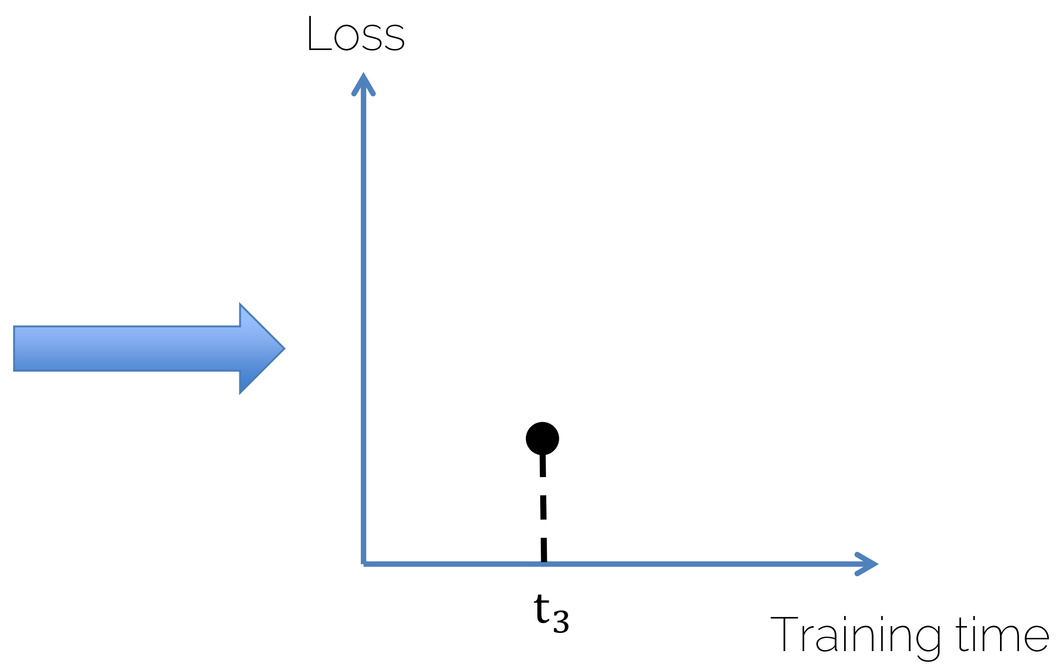


Bad prediction!

During Training…

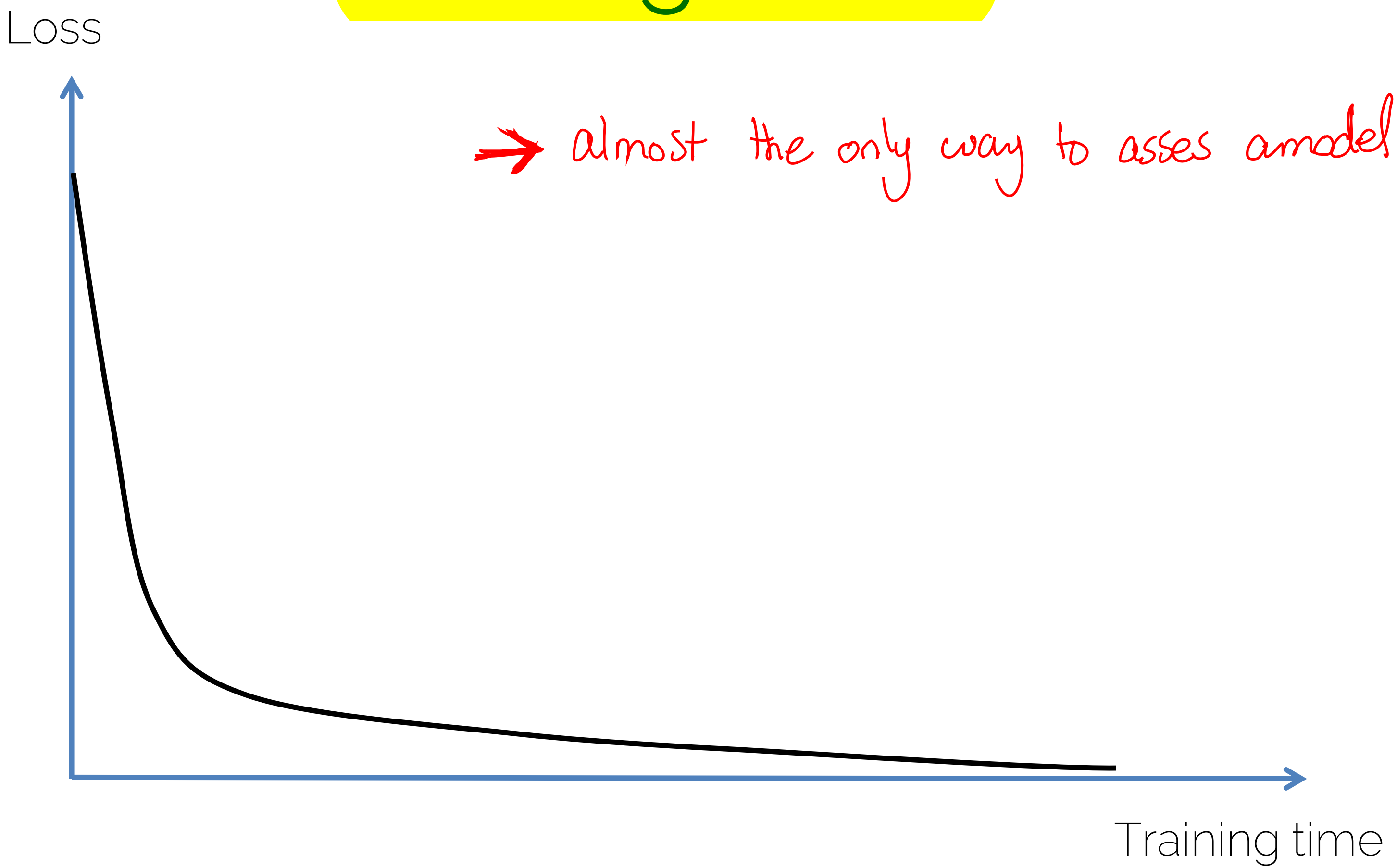


Bad prediction!



Training Curve

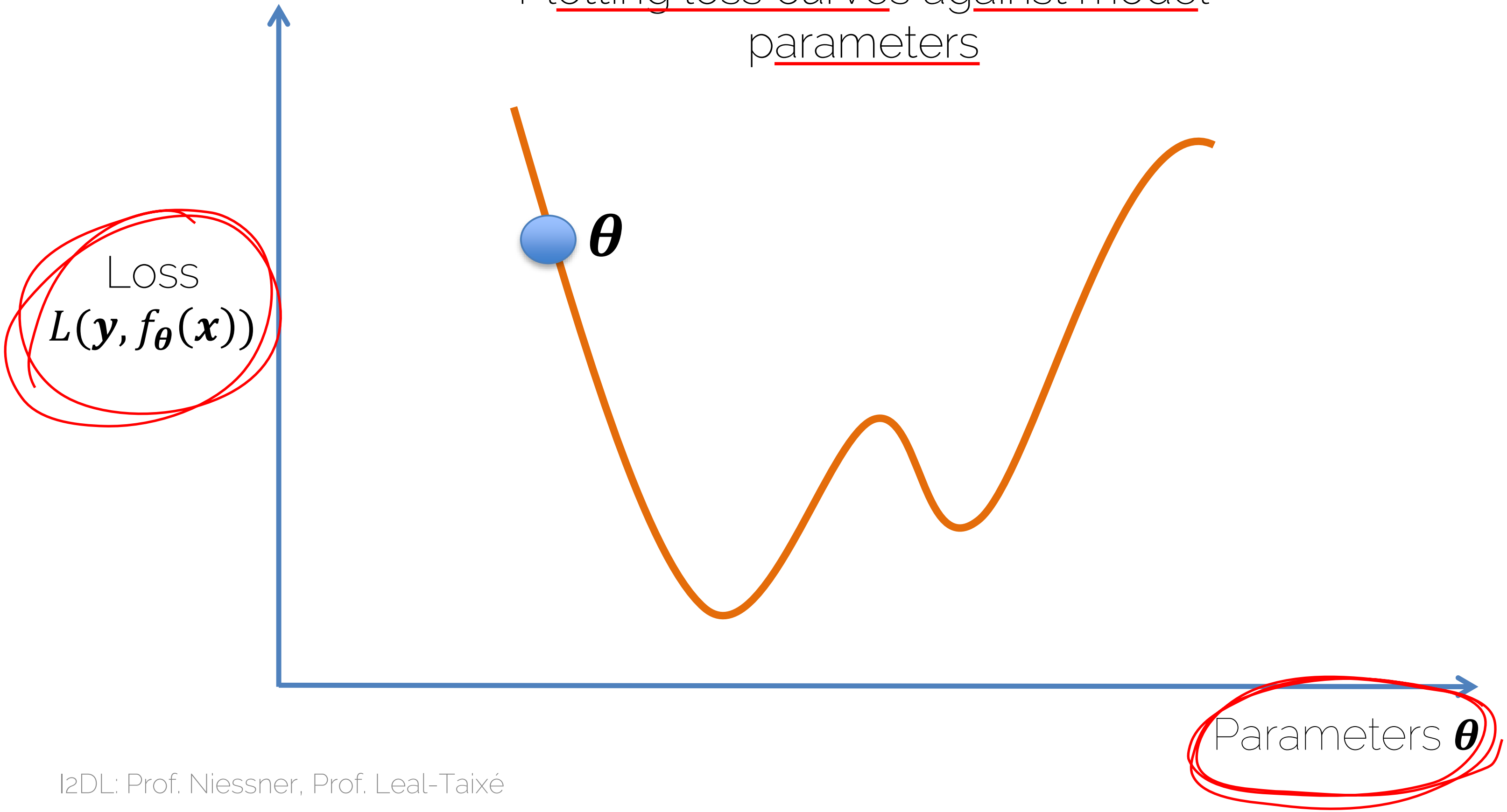
→ almost the only way to asses a model



Understand

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. $\boldsymbol{\theta}$

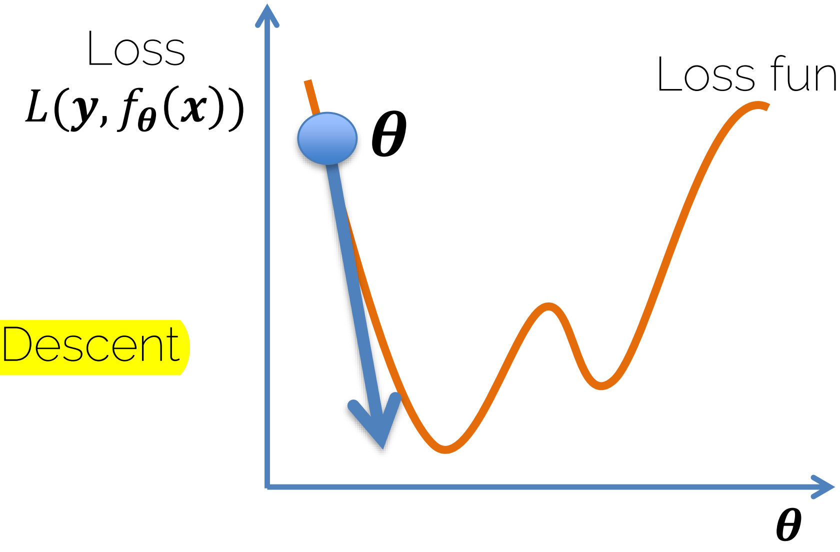
→ Optimization! We train compute graphs with some optimization techniques!

└ for this lec, only gradient_based

How to Find a Better NN?

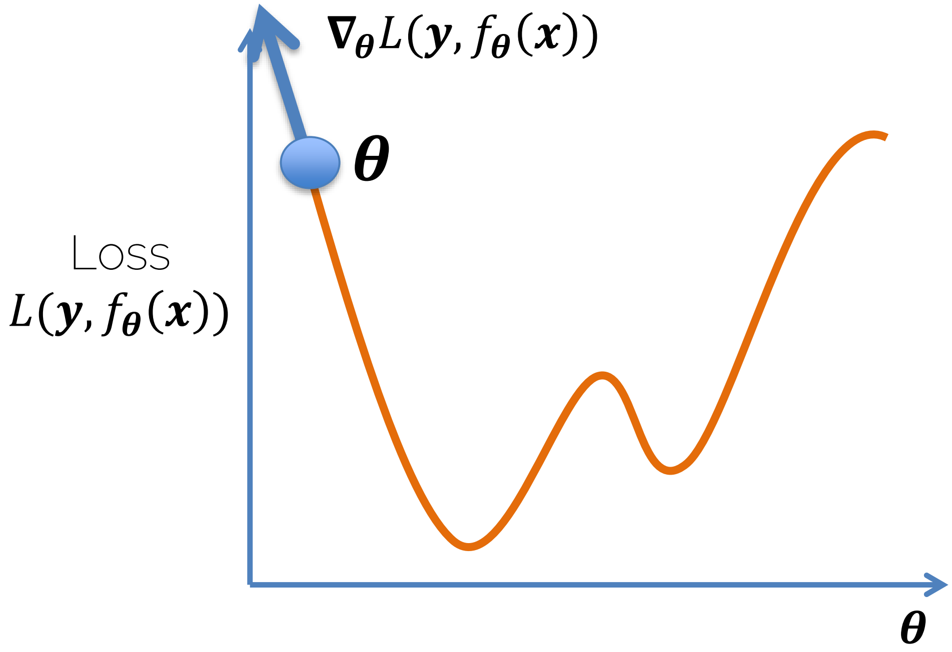
- Minimize: $L(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ w.r.t. $\boldsymbol{\theta}$
- In the context of NN, we use gradient-based optimization

Gradient Descent



How to Find a Better NN?

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



Learning rate

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

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

How to Find a Better NN?

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

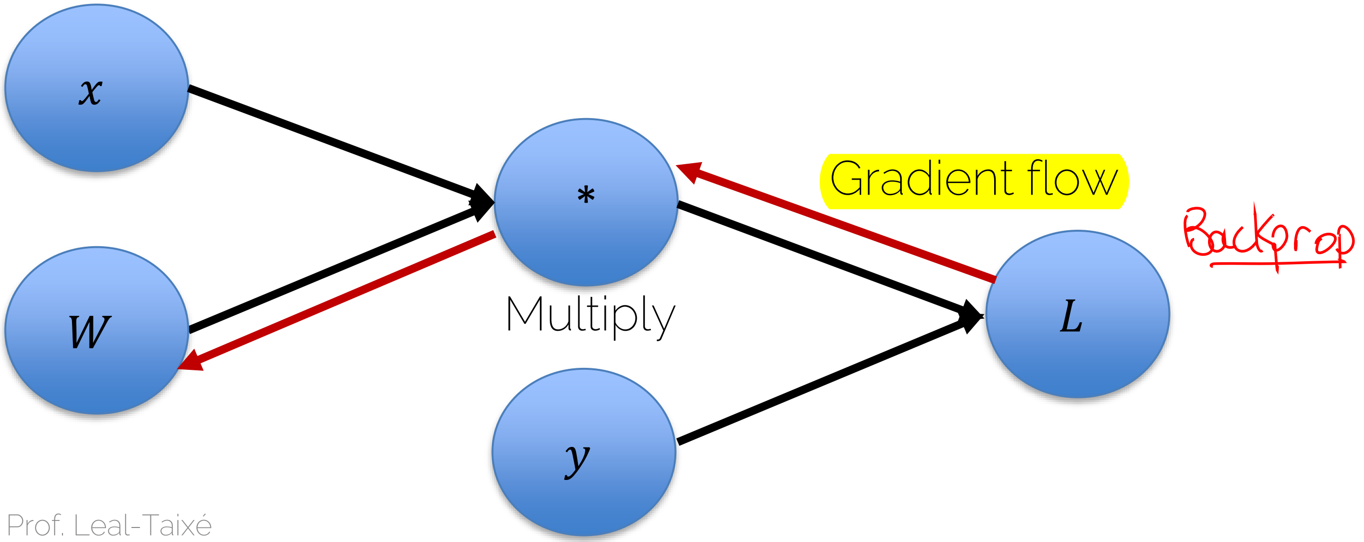
$$\boldsymbol{f_\theta}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}\,, \qquad \boldsymbol{\theta} = \boldsymbol{W}$$

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

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

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: $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}; \boldsymbol{\theta}) = \frac{1}{n} \sum_i^n ||y_i - \boldsymbol{W} \cdot x_i||_2^2$



How to Find a Better NN?

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

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

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

How to Find a Better NN?

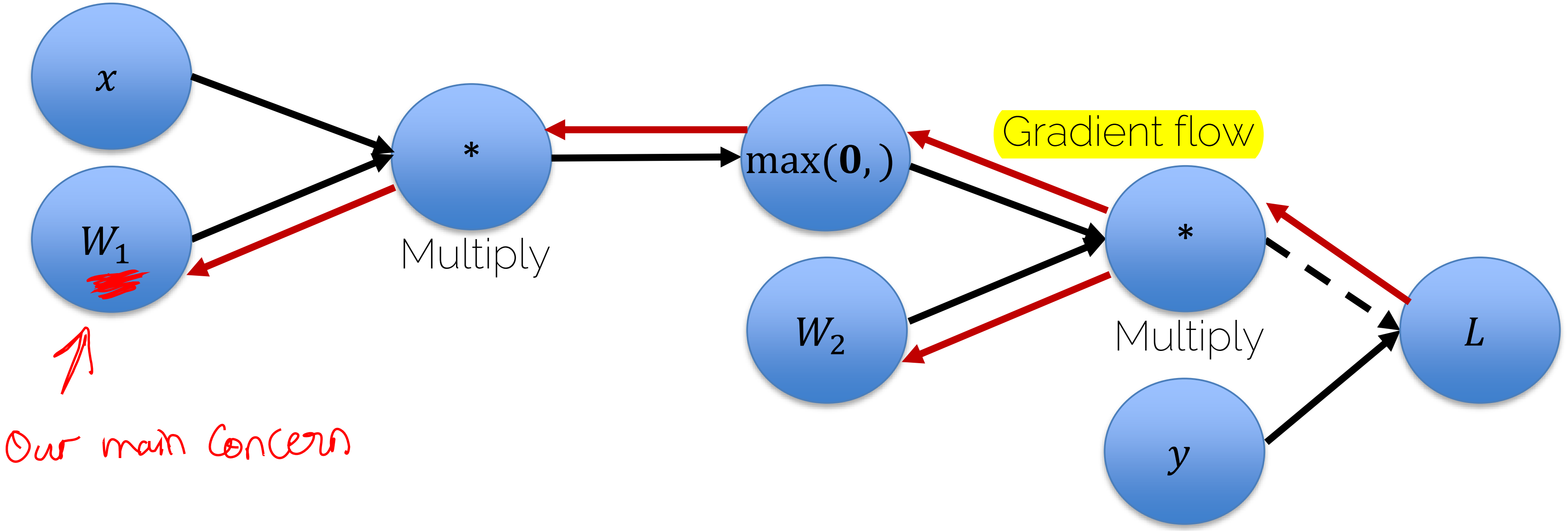
- Given inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$
- Given a multi-layer NN with many activations

$$\boldsymbol{f} = \boldsymbol{W}_5 \sigma(\boldsymbol{W}_4 \tanh(\boldsymbol{W}_3, \max(\boldsymbol{0}, \boldsymbol{W}_2 \max(\boldsymbol{0}, \boldsymbol{W}_1 \boldsymbol{x}))))$$

- Gradient descent for $\boldsymbol{L}(\boldsymbol{y}, \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}))$ 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 $L\big(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x})\big)$ 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?
  - Easy to compute using compute graphs

- efficient with large networks

- 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/moodle 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/