Deep Neural Networks - Introduction

Pattern Recognition

Fall 2019

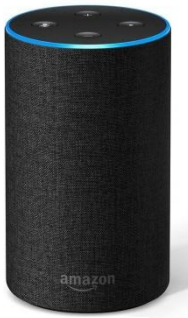
Dennis Madsen

Under construction ...

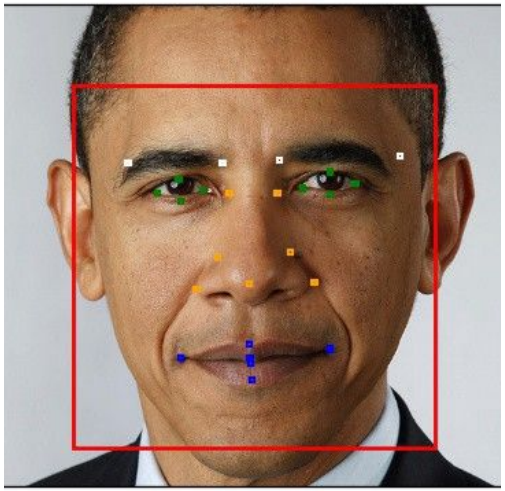
- This is new course content.
- We are always happy about feedback, corrections and suggestions.

Deep Learning is everywhere

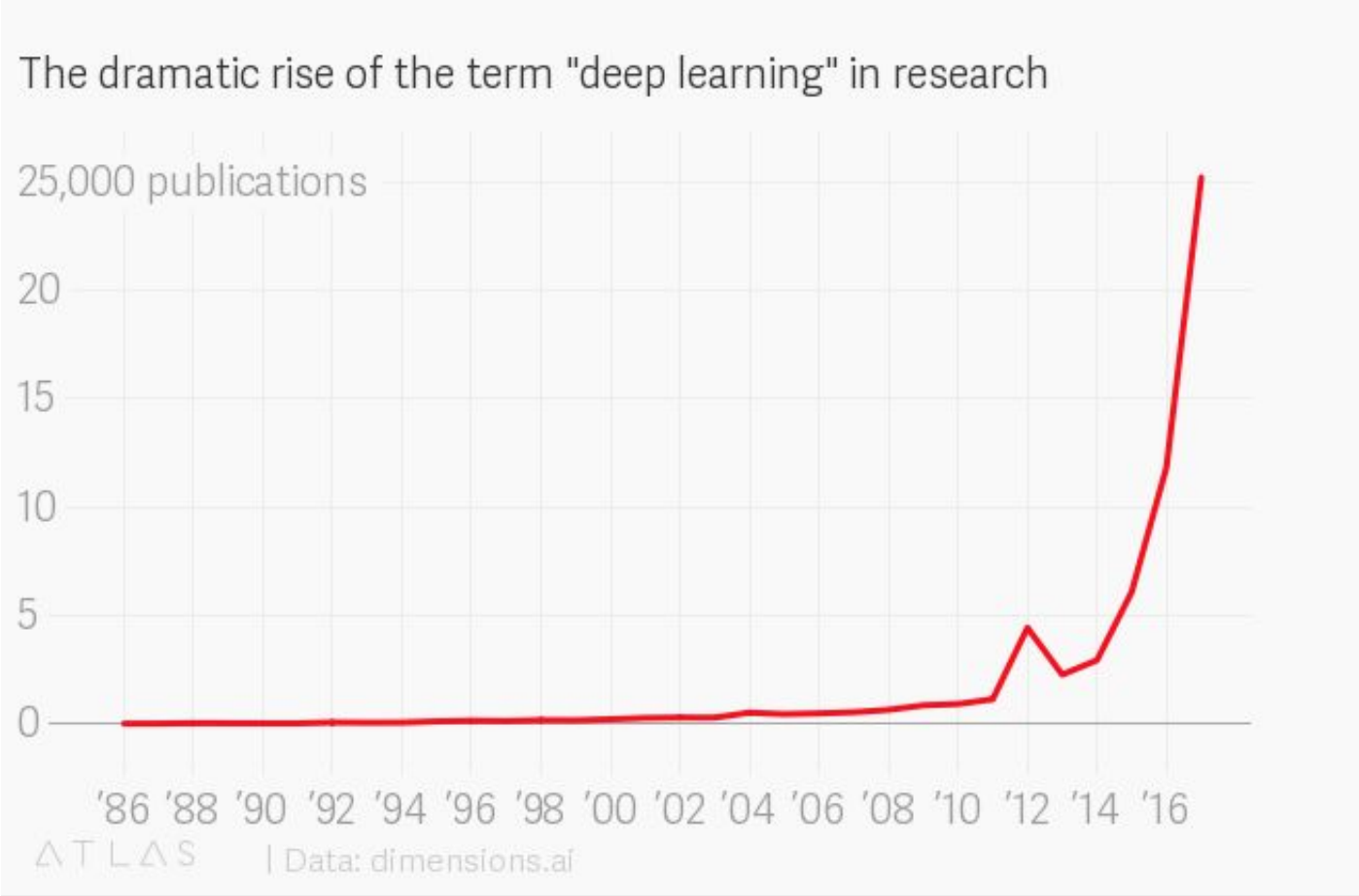
- Speech/text



- From images



Deep Learning is everywhere



Topic overview

- Neural Networks (recap) and Deep Learning
- Improving DNN: Hyperparameter tuning, regularization, optimization
- Convolutional Neural Networks (CNN)
- CNN popular architectures
- Sequence Models/Recurrent neural networks (RNN)
- Beyond the basics (object detection and segmentation)

Today

- **Neural Networks (recap) and Deep Learning**
  - Why Deep Learning (motivation)
  - Computational Graphs
  - Artificial Neurons
  - Forward propagation
  - Activation functions for non-linearity
  - Back propagation
  - Loss-functions
  - History of Deep Learning
  - PyTorch and Automatic differentiation

- *Improving DNN: Hyperparameter tuning, regularization, optimization*

Deep Learning Material

Books:

- https://www.deeplearningbook.org/
- http://neuralnetworksanddeeplearning.com/

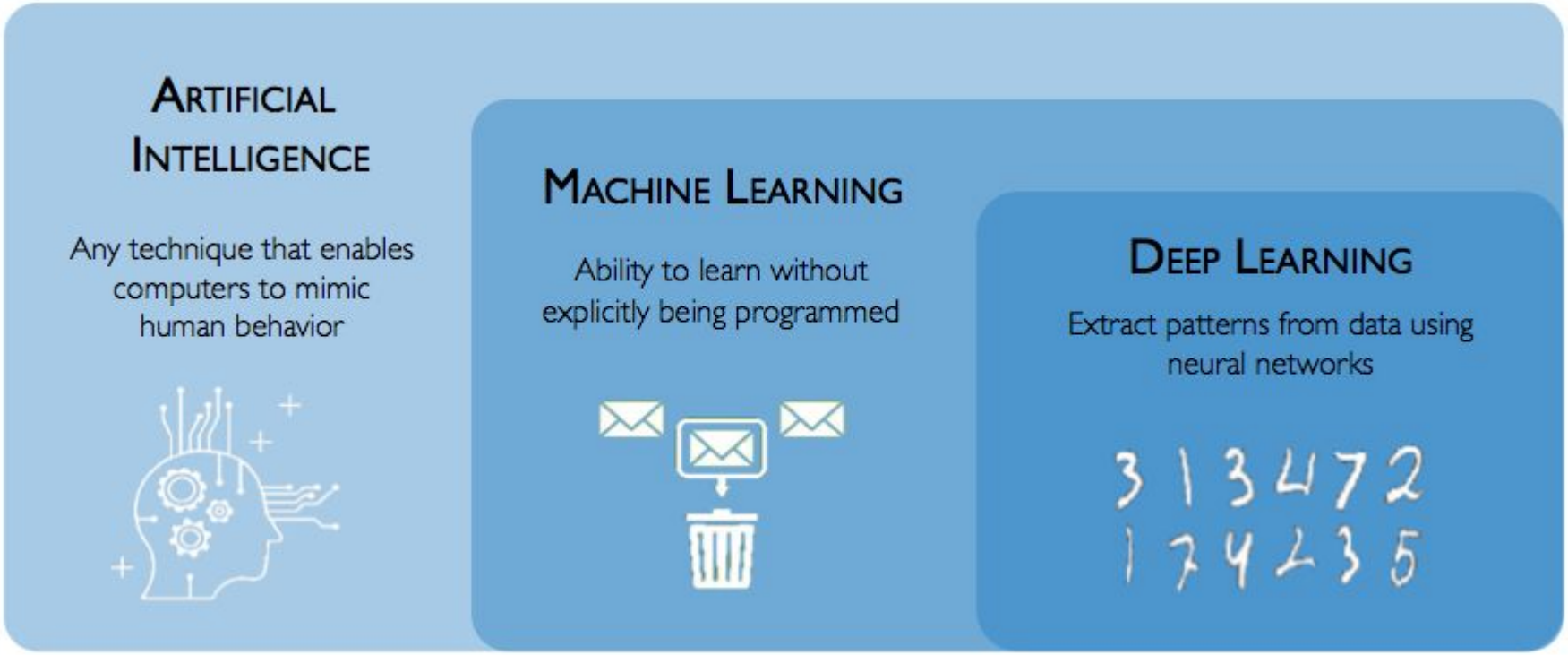
Online Course from MIT:

- http://introtodeeplearning.com/

Online course from Stanford University:

- https://www.coursera.org/specializations/deep-learning?

What is Deep Learning?

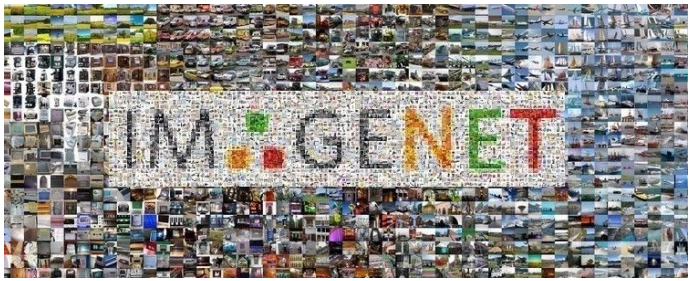


Bishop, Preface: *"Pattern recognition has its origins in engineering, whereas machine learning grew out of computer science. However, these activities can be viewed as two facets of the same field ".*

Source: http://introtodeeplearning.com/

Why now?

1. **Big Data**
   - Larger Datasets
   - Easier Data collection and Storage

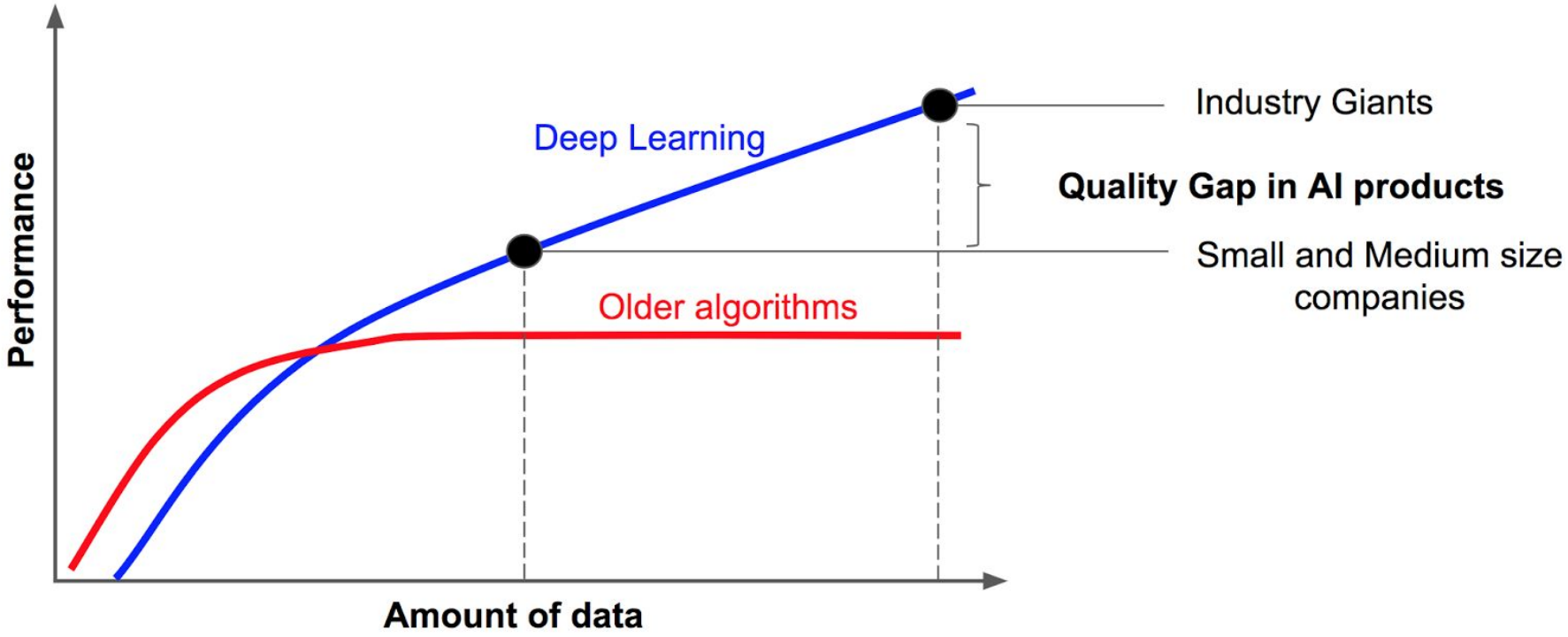


2. **Hardware**
   - Graphics cards
   - Parallelizable algorithms

3. **Software**
   - Open source toolboxes
   - Open source and pre-trained models

Chainer mxnet Caffe2
Microsoft CNTK TensorFlow Keras
GLUON PYTORCH theano

Why is Deep Learning taking off?



Big Data Example: ImageNet Dataset

14 million images divided into 21.841 categories

Definition: elongated crescent-shaped yellow fruit with soft sweet flesh.

1409 pictures of bananas!

Source: http://introtodeeplearning.com/ and http://www.image-net.org/search?q=banana

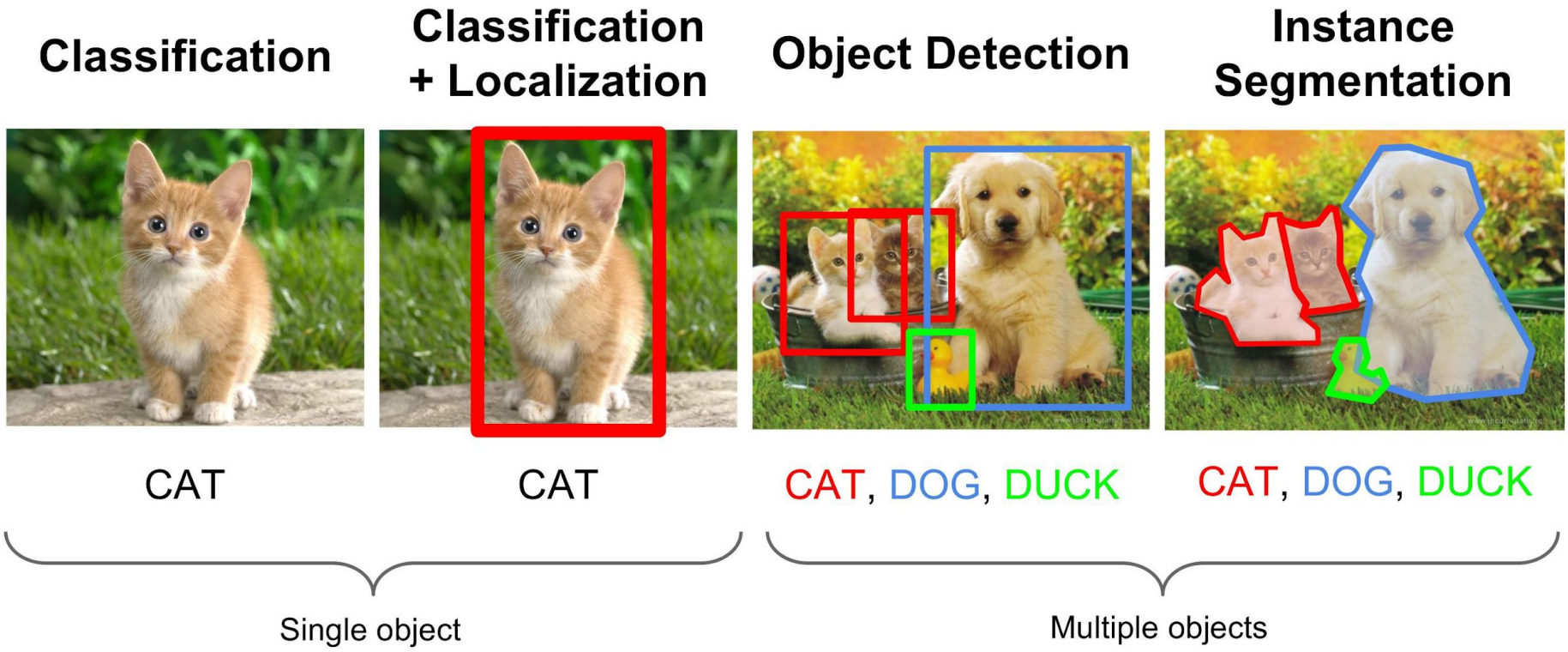
ImageNet Dataset

Classification task: produce a list of object categories present in an image from 1000 categories.

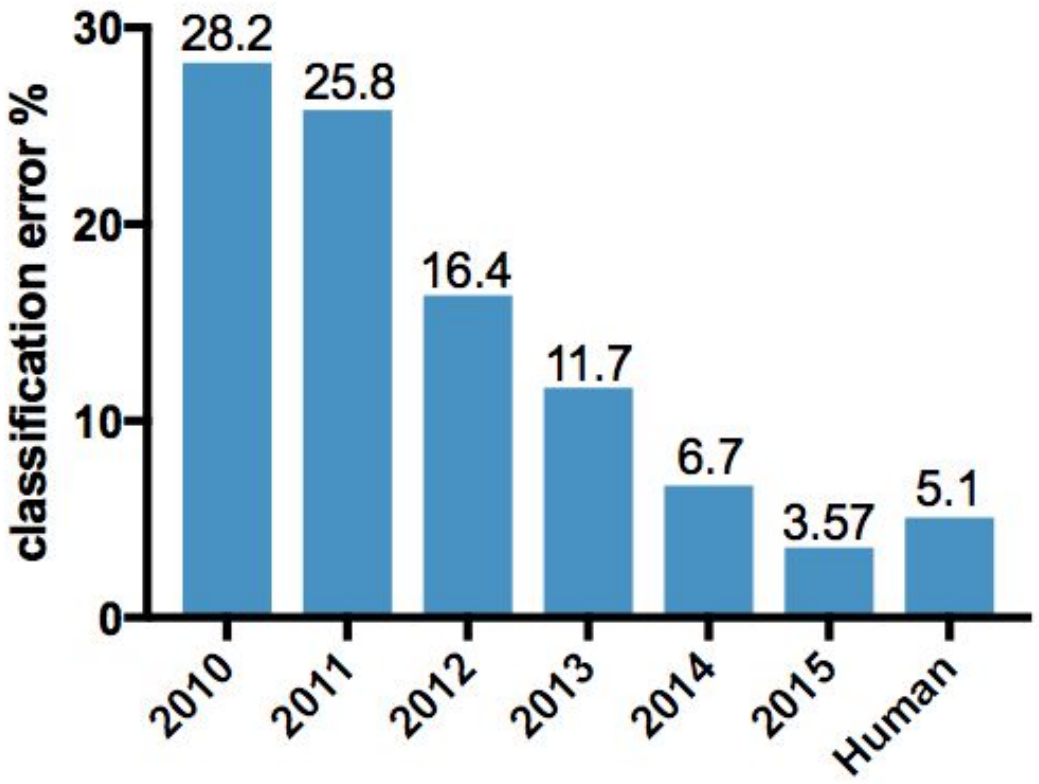
"Top 5 error": rate at which the model does not output correct label in top 5 predictions.

Source: http://introtodeeplearning.com/

Classification vs Detection



ImageNet Dataset



2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

2013: ZFNet
- 8 layers, more filters

2014: VGG
- 19 layers

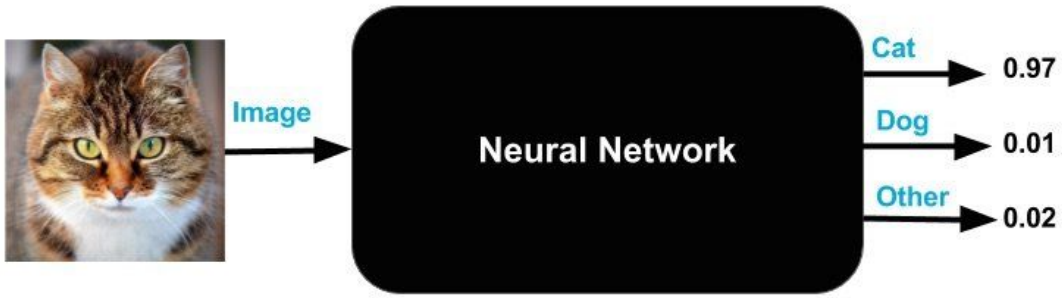
2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

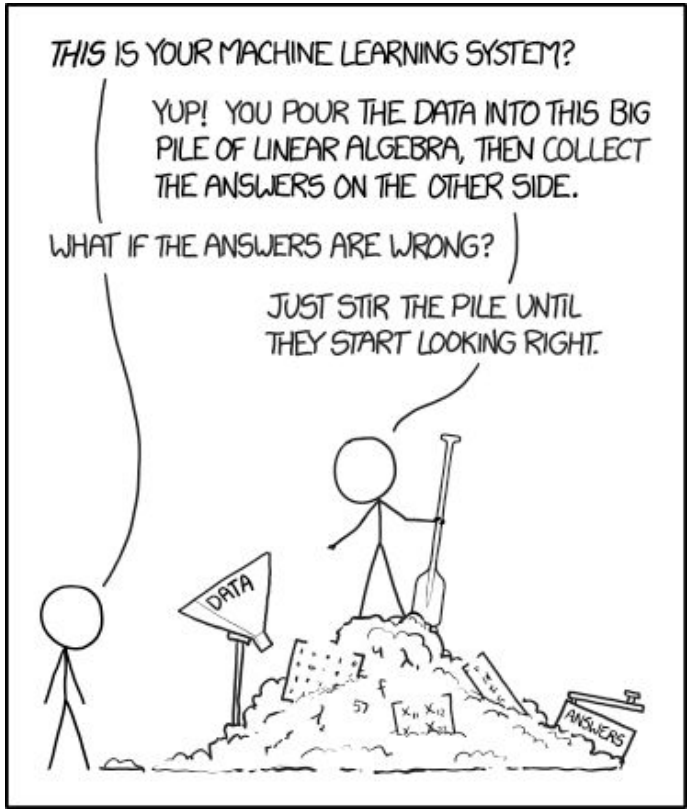
2015: ResNet
- 152 layers

Source: http://introtodeeplearning.com/

What is a Deep Neural Network?

- What is inside the “magical black box”?



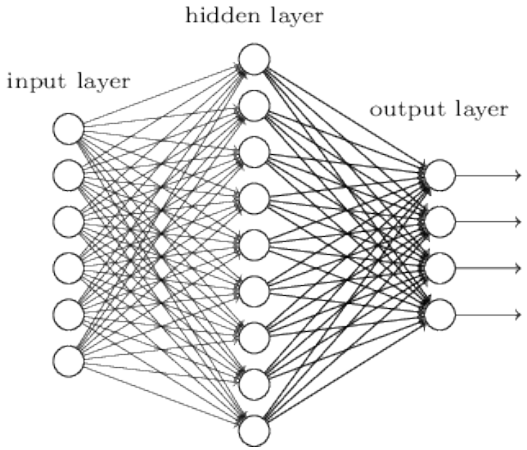


Source: https://www.learnopencv.com/neural-networks-a-30000-feet-view-for-beginners/

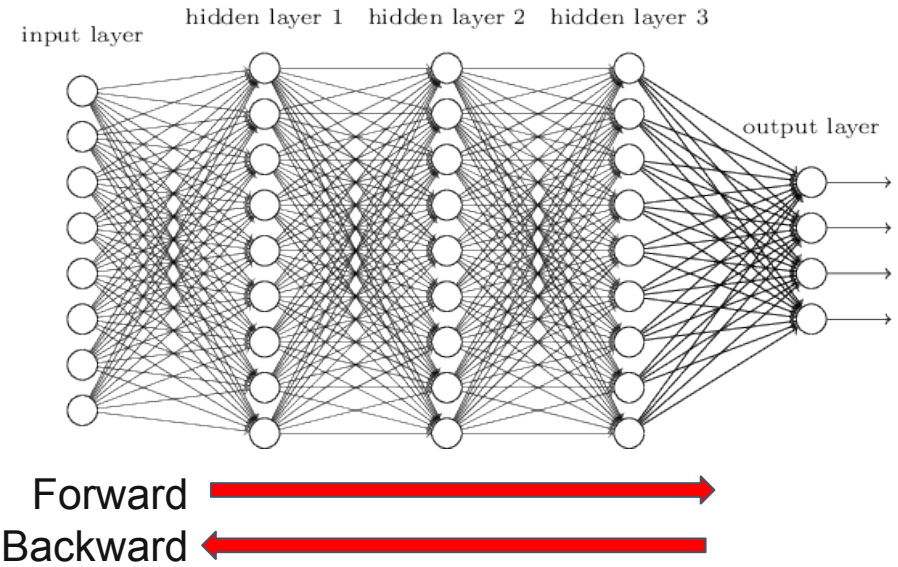
What is a Deep Neural Network?

- An **Artificial** Neural Network consist of simple elements called neurons.
- A neuron can make a simple mathematical decision.
- By combining neurons we can analyze complex problems.

Shallow Artificial Neural Network



Deep Artificial Neural Network: more than 1 hidden layer

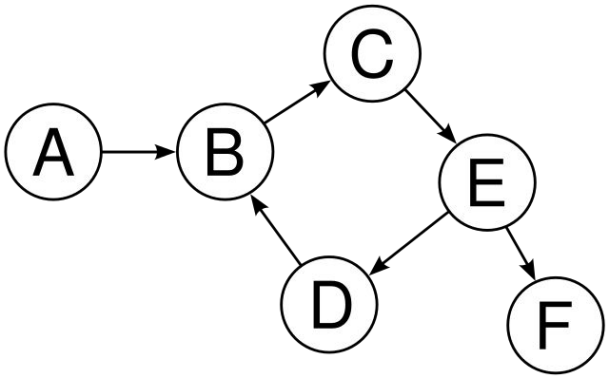


Network of Neurons - Computational graph

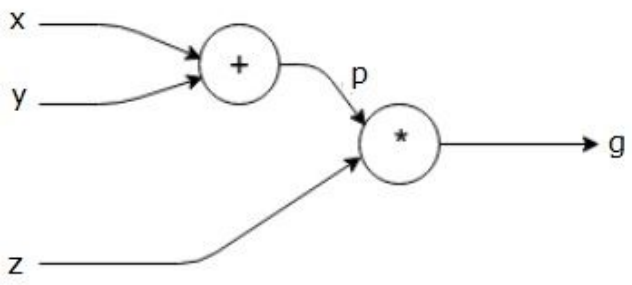
A computational graph is a directed graph where the nodes correspond to operations or variables. Variables can feed their value into operations, and operations can feed their output into other operations.

This way, every node in the graph defines a function of the variables.

Directed graph



Computational Graph



Computational graph

Graphically visualize the computation of a function using a directed graph

Forward: Compute function output
Backward: Compute gradients

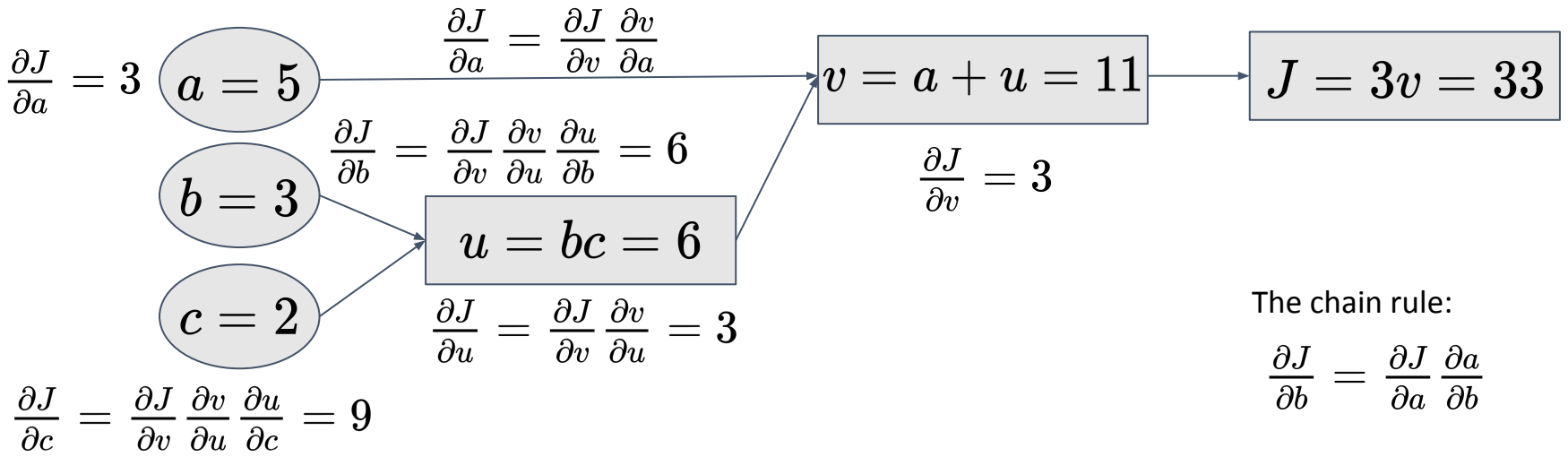
Example: Goal is to compute the function J

$$J(a,b,c) = 3(a+bc)$$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



The chain rule:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial b}$$

Backward: Compute gradients

Example on blackboard

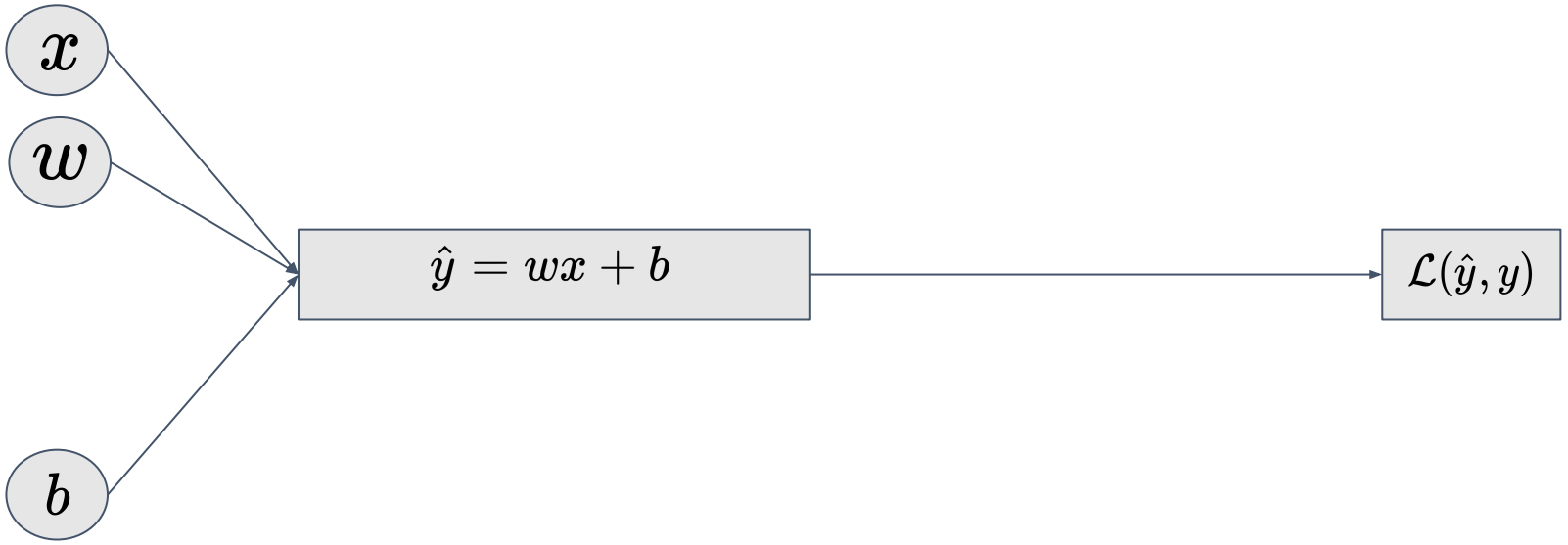
Source: https://www.coursera.org/learn/neural-networks-deep-learning/

Computational graph - Linear Regression

Representing linear repression in a computational graph.
Example with regression on a 1D dataset:

$$\hat{y} = wx + b$$

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$



Source: https://www.coursera.org/learn/neural-networks-deep-learning/

Computational graph - Linear Regression

Representing linear repression in a computational graph.
Example with regression on a 1D dataset:

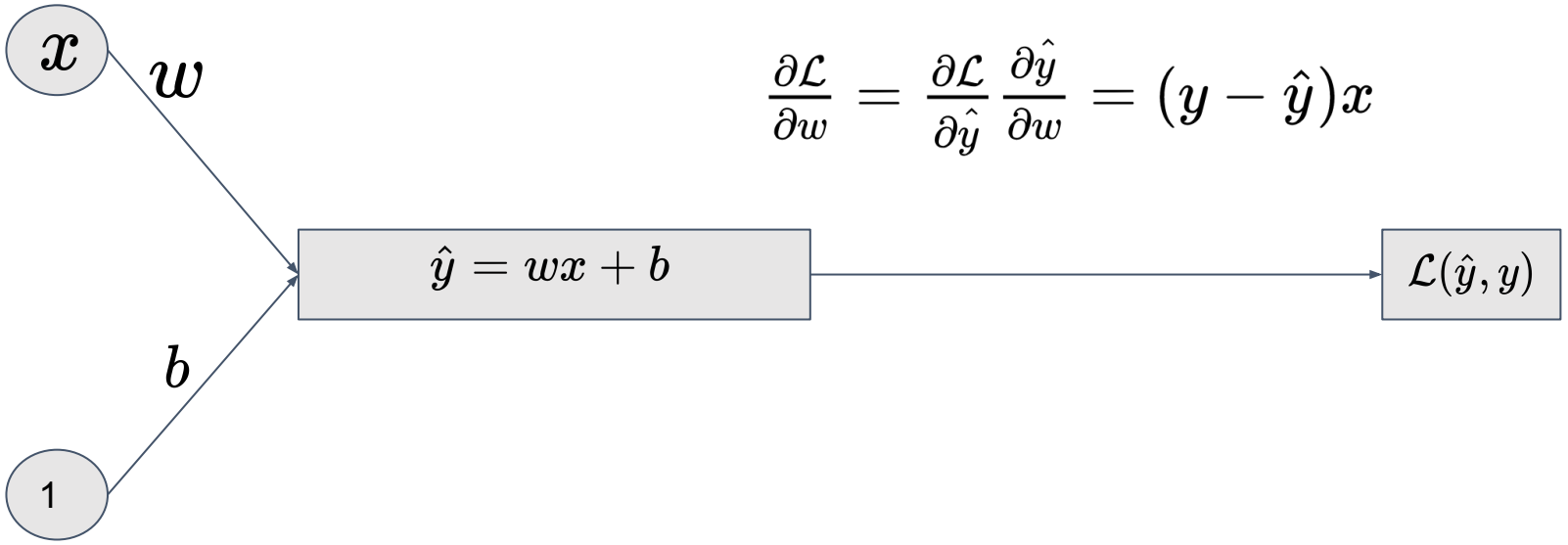
$$\hat{y} = wx + b$$

$$\mathcal{L}(y, \hat{y}) = \tfrac{1}{2}(y - \hat{y})^2$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (y - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial w} = x$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w} = (y - \hat{y})x$$



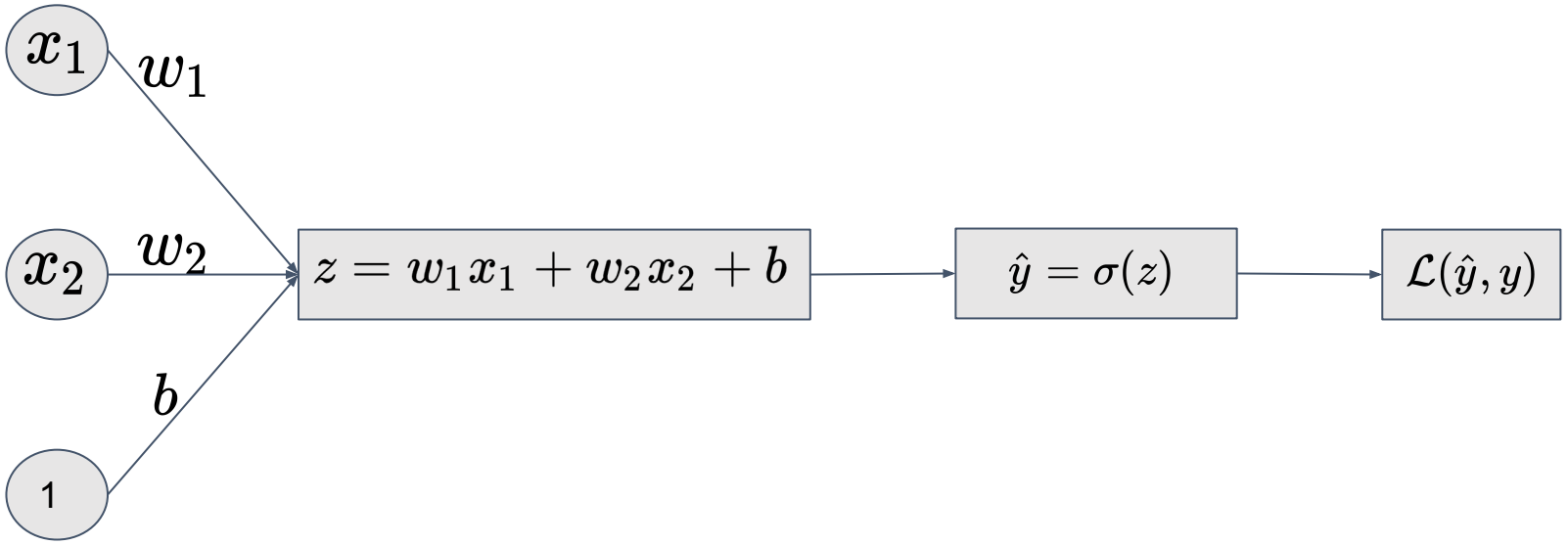
Weights are represented on the edges between nodes.

Source: https://www.coursera.org/learn/neural-networks-deep-learning/

Computational graph - Logistic Regression

Representing logistic regression classification in a computational graph.
Logistic regression on a 2D dataset:

$$z = w^T x + b$$
$$\hat{y} = \sigma(z) \qquad \sigma(z) = \frac{1}{1+e^{-z}}$$
$$\mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$



Source: https://www.coursera.org/learn/neural-networks-deep-learning/

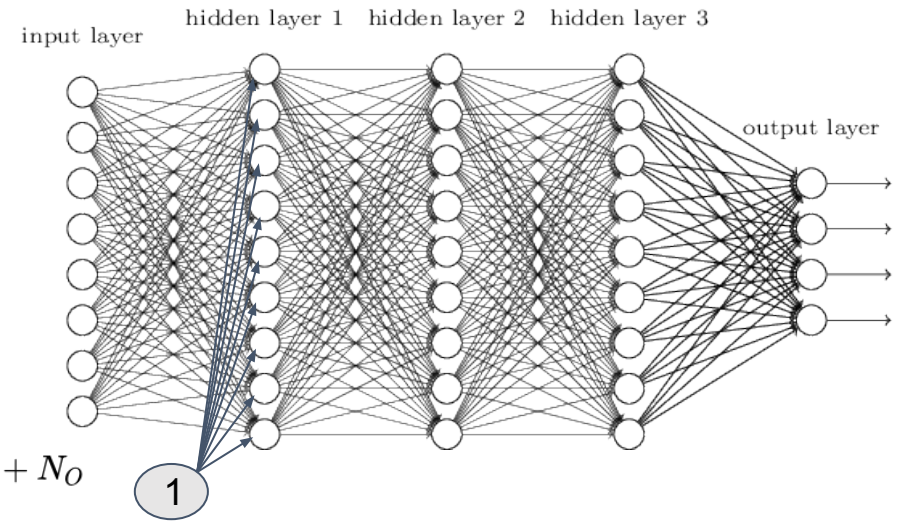
Architecture Design - Fully connected

Neural networks are organized into groups of units called layers. Most neural network architectures arrange these layers in a chain structure, with each layer being a function of the layer that preceded it.

Multi-layer networks are preferable over 3-layered networks because they often generalize better

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right),$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right),$$



Number of parameters in fully connected network:

$$N_F = N_I H_1 + H_1 + H_1 H_2 + H_2 + H_2 H_3 + H_3 + H_3 N_O + N_O$$

- The bias term is always set to 1 and shared among all the neurons in a layer.
- The bias is often left out in architecture visualizations.

Backprop is not only Gradient Descent

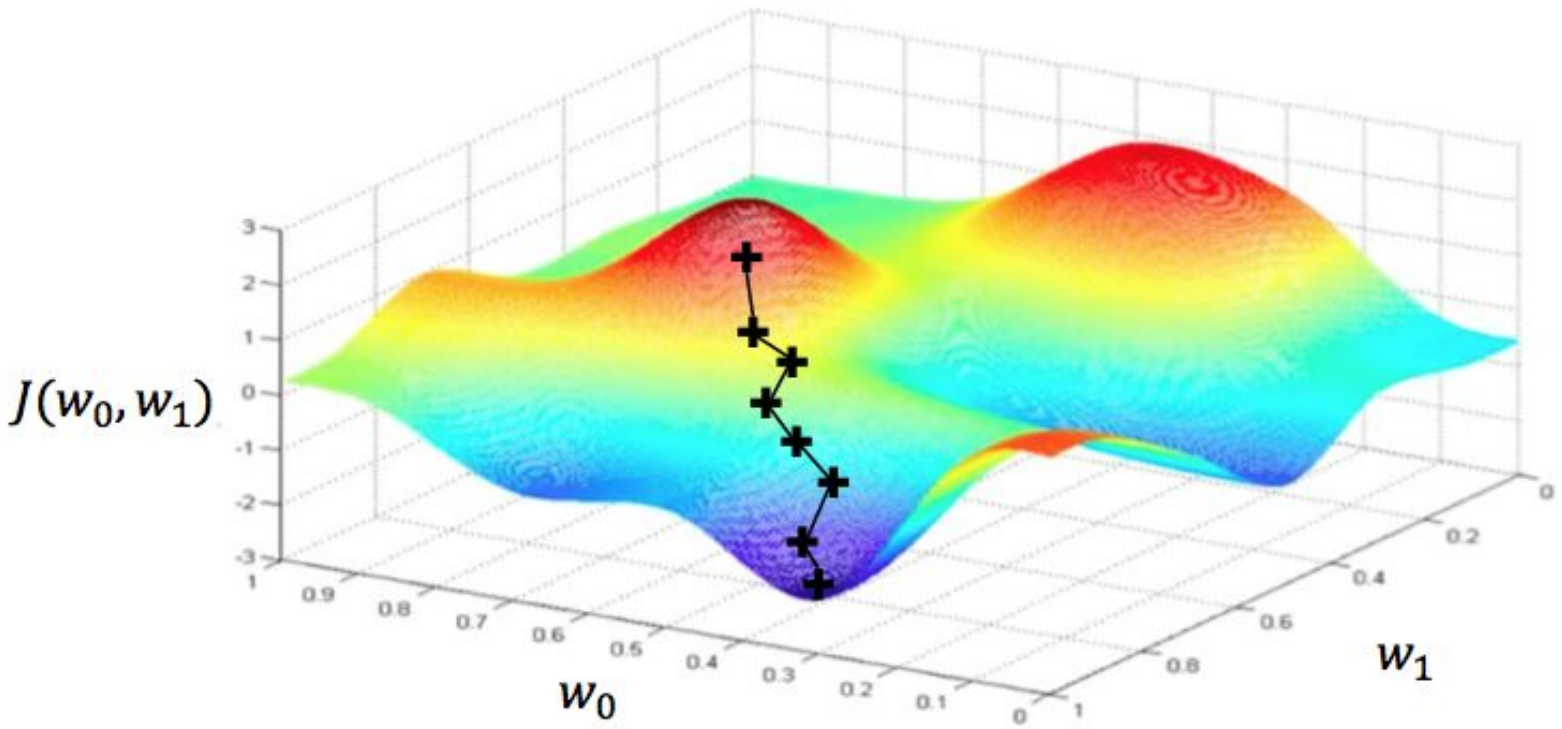
Backpropagation "Backprop" refers to the process of backpropagating the error through a computational graph to compute the gradients.

Gradient Descent iterative optimization algorithm to find the minimum of a function. The algorithm takes steps proportional to the negative of the size of the gradient.

1. Loop until convergence, optimizing **w**
   a. Compute gradients $\frac{\partial J}{\partial w}$
   b. Update weights, $w_{t+1} \leftarrow w_t - \lambda \frac{\partial J}{\partial w}$
2. Return weights **w**

Gradient Descent

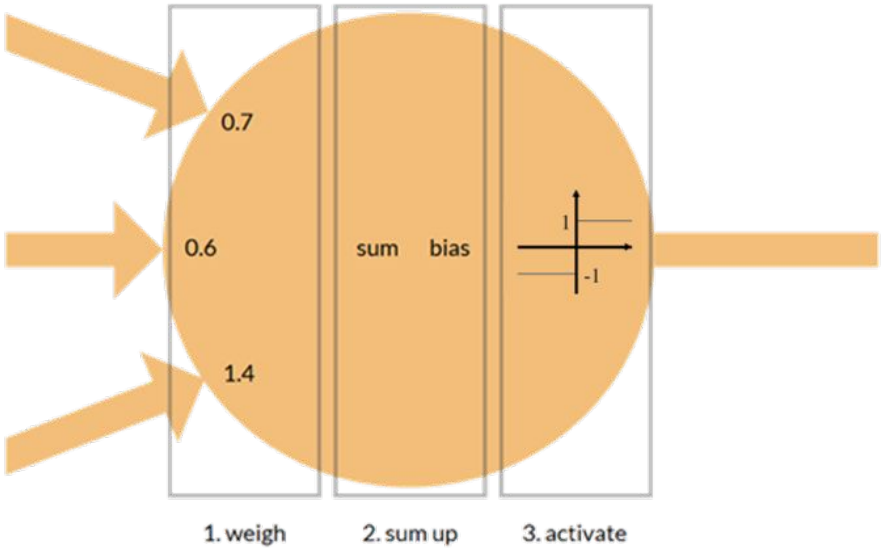
- Take small steps in opposite direction of the gradient.
- Repeat until convergence.



Source: http://introtodeeplearning.com/

A Neuron modelled as a Perceptron

- The structural building block of Neural Networks.
- Perceptrons are also referred to as “artificial neurons”, highlighting the original **inspiration** from biological neurons.



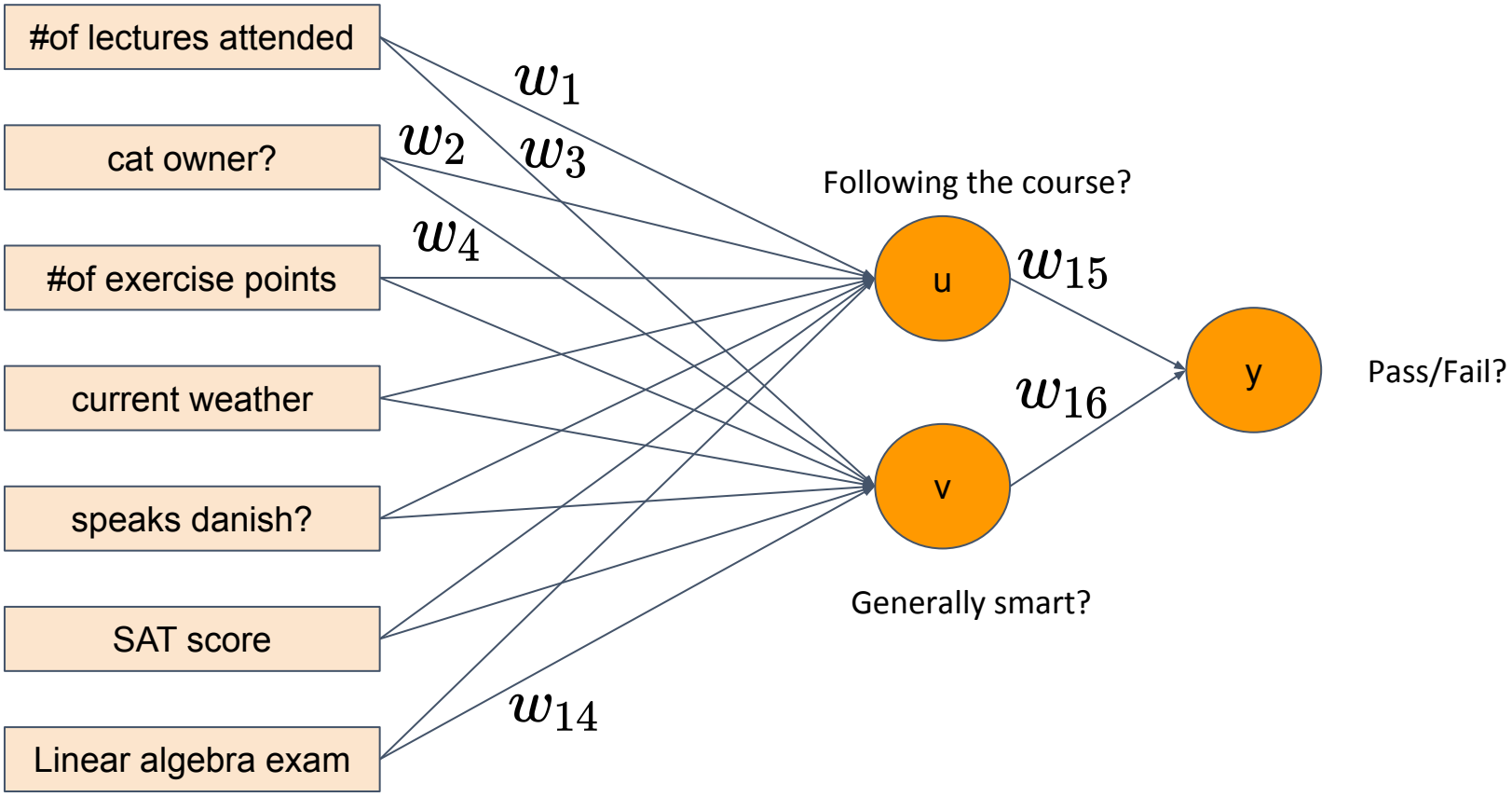
$$w^T x + b > 0 \text{ class } 1$$
$$w^T x + b < 0 \text{ class } 0$$

Estimate the weights **w** using gradient descent

*A Neural network (feedforward network) is a **function approximation machine** that is designed to achieve **statistical generalization**, which **occasionally** draws some insight from what we know about the brain. Neural networks are **NOT** models of brain functions.*

Student example

Will I pass the pattern recognition exam?

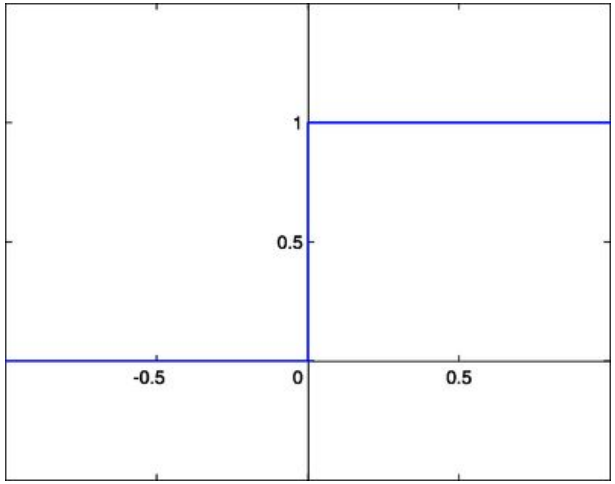


Problem with the step function?

Gradient descent requires the activation function to be differentiable.

1. It is not differentiable at 0
2. 0 derivative everywhere else

The gradient will not give us any information about the direction to go.

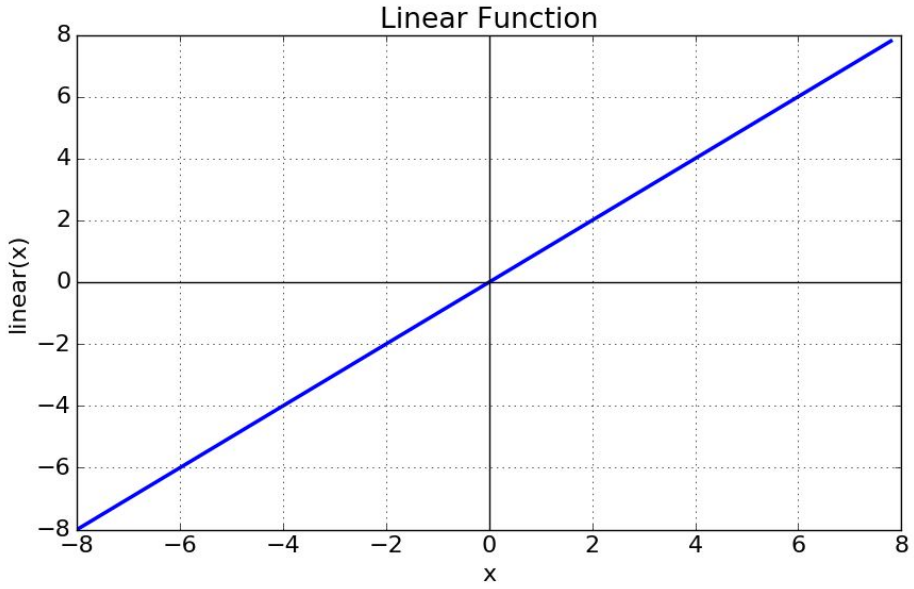


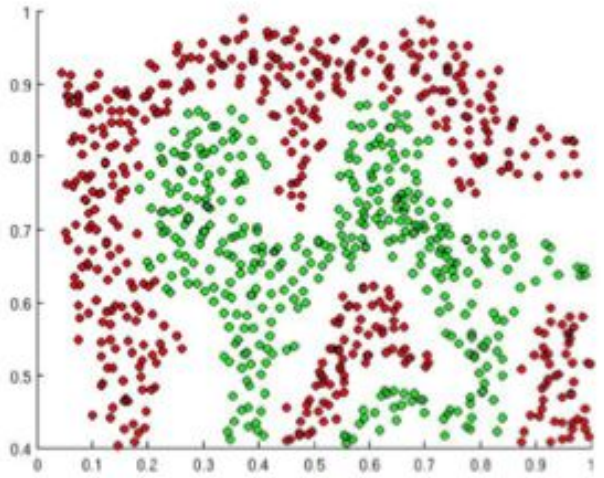
Linear activation functions

The linear function is differentiable everywhere.

Problem:

1. Can only model linear functions.

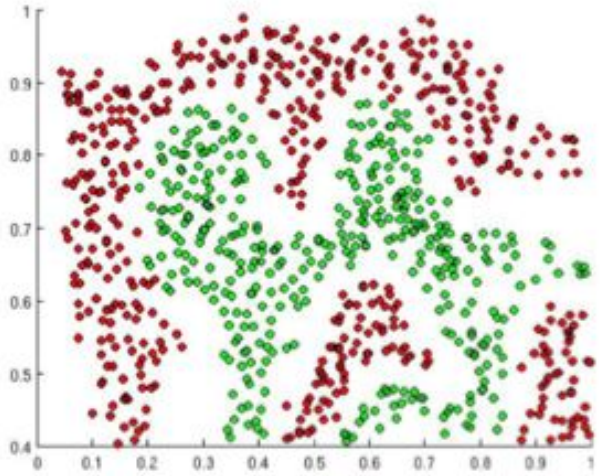
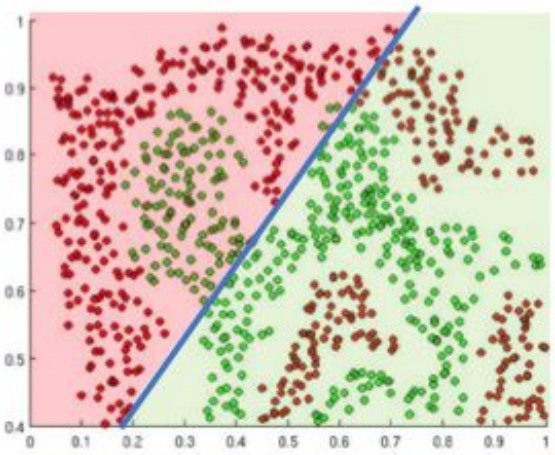
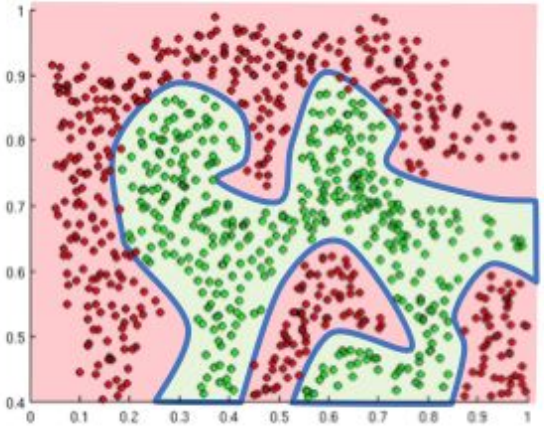




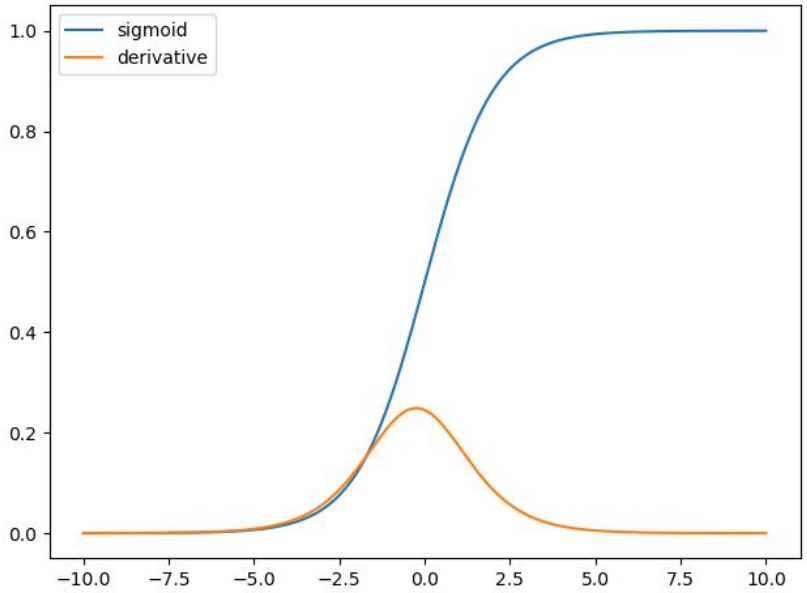
Non-linear activation functions

Why do we need non-linearity in our graph?

To be able to estimate non-linear functions.

Activation functions: Sigmoid

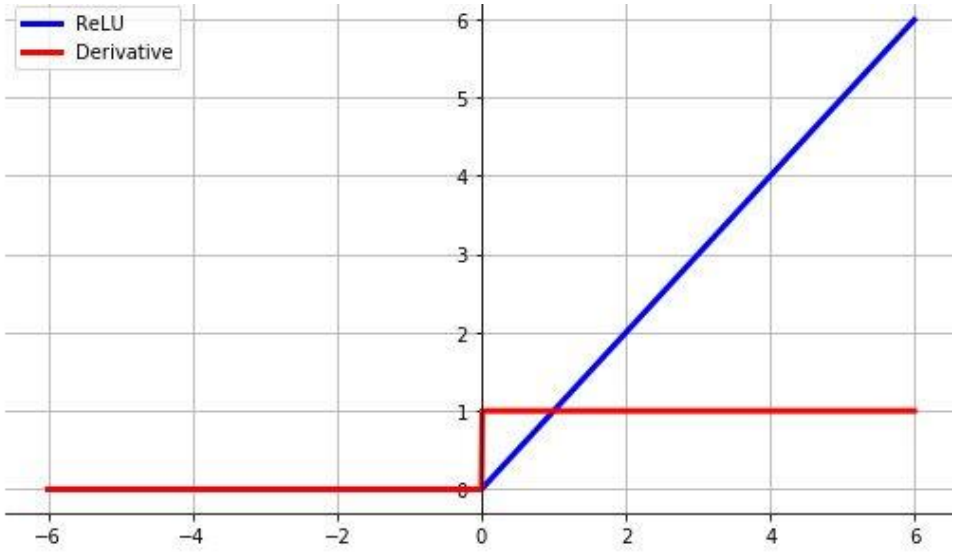


$$f(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial f(x)}{\partial x} = f(x)(1 - f(x))$$

- Input is mapped into the range [0,1] -> probabilistic interpretation
- Reduces the gradient for large inputs -> **vanishing gradients**
  - With *n* layers, *n* small derivatives are multiplied together -> gradient decreases exponentially backwards through the layers.
  - Small gradient -> weights and biases will not be updated.

Activation functions: ReLU

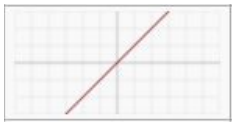
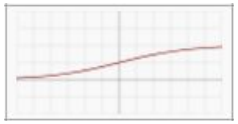
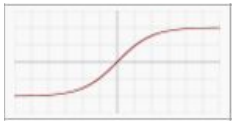
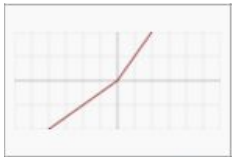


$$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$$

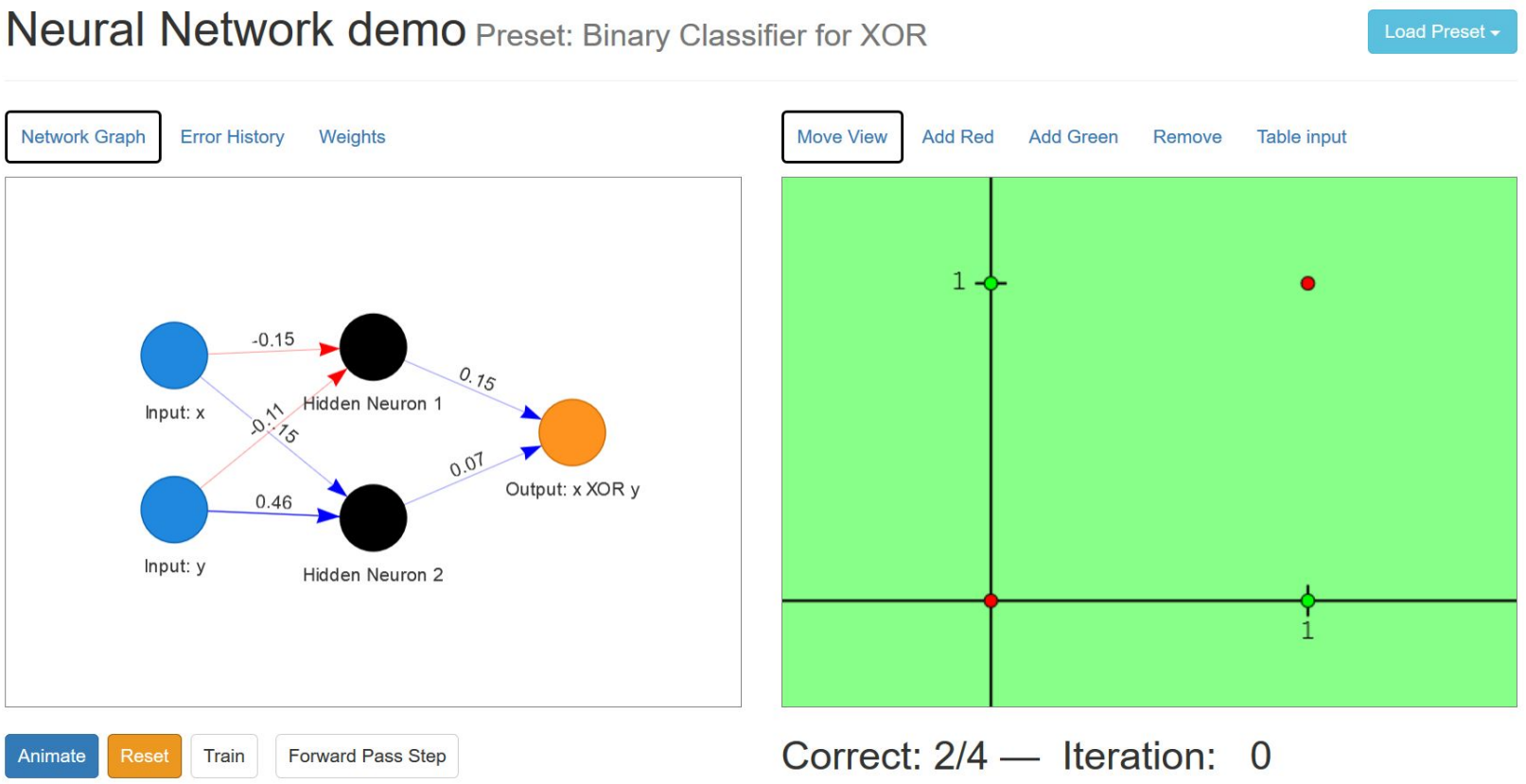
$$\frac{\partial f(x)}{\partial x} = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x > 0 \end{cases}$$

- “Rectified linear unit”
- Efficient to compute
- Smaller risk of vanishing gradients

Activation functions overview

Name	Activation function	Derivative
• Identity/linear 	$f(x) = x$	$\frac{\partial f(x)}{\partial x} = 1$
• Binary step 	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$	$\frac{\partial f(x)}{\partial x} = \begin{cases} 0 \text{ for } x \neq 0 \\ ? \text{ for } x = 0 \end{cases}$
• Logistic (Sigmoid) 	$f(x) = \frac{1}{1+e^{-x}}$	$\frac{\partial f(x)}{\partial x} = f(x)(1 - f(x))$
• TanH 	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	$\frac{\partial f(x)}{\partial x} = 1 - f(x)^2$
• Rectified Linear Unit (ReLU) 	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$\frac{\partial f(x)}{\partial x} = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x > 0 \end{cases}$
• Leaky ReLU 	$f(x) = \begin{cases} 0.001x \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$\frac{\partial f(x)}{\partial x} = \begin{cases} 0.001 \text{ for } x < 0 \\ 1 \text{ for } x > 0 \end{cases}$
• Parametric ReLU	$f(x) = \begin{cases} \alpha x \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$\frac{\partial f(x)}{\partial x} = \begin{cases} \alpha \text{ for } x < 0 \\ 1 \text{ for } x > 0 \end{cases}$

Example Training App - XOR problem



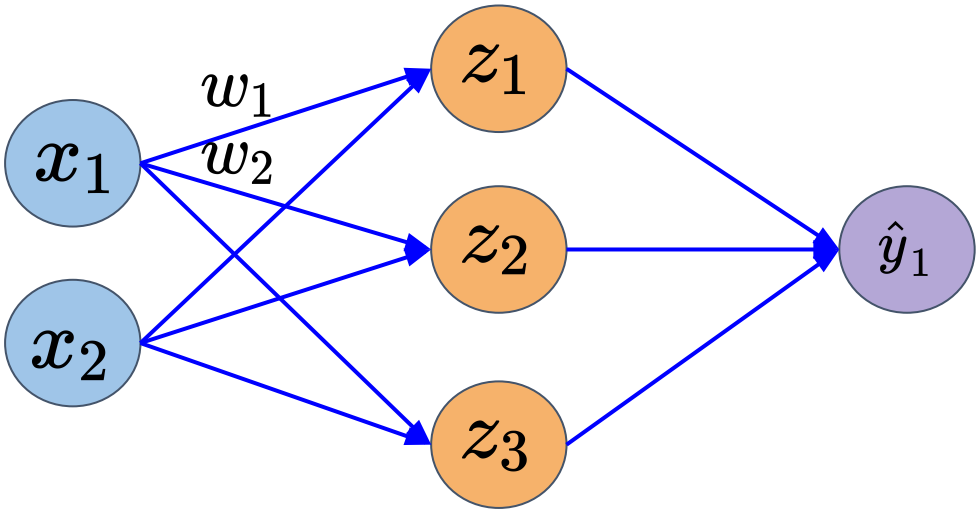
https://lecture-demo.ira.uka.de/neural-network-demo/

Neural Network Loss Functions

Quantifying Loss

The loss of a neural network measures the cost incurred from an incorrect prediction.
Measuring the error on a single training example.

$x^{(1)} = [1, 2]$



Predicted: 0.1
Actual: 1

$$\mathcal{L}(f(x^{(1)}), y^{(1)})$$

Prediction true label

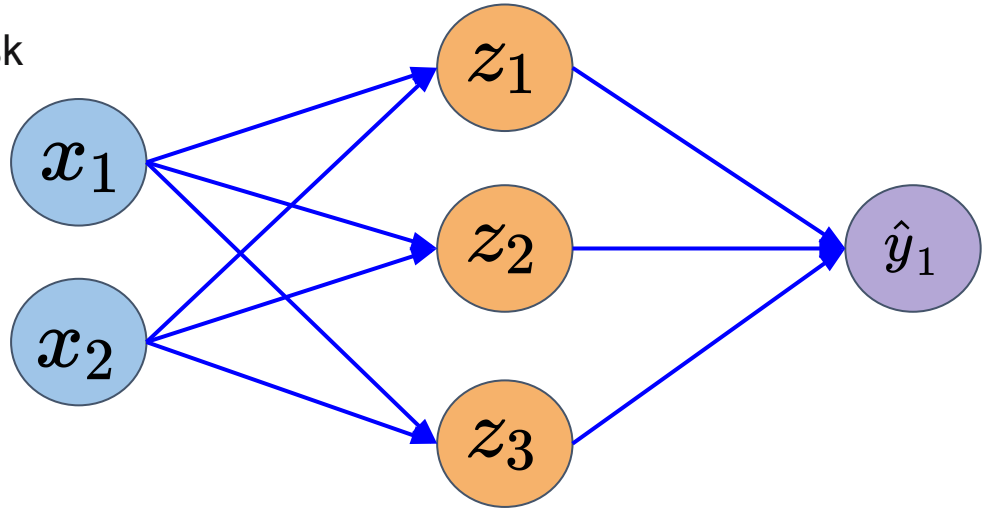
Source: http://introtodeeplearning.com/

Empirical Loss

The empirical loss measures the total loss over the entire dataset.
Usually computed as the mean of the losses.
Also sometimes referred to as:

- Objective function
- Cost function
- Empirical Risk

$$X = \begin{bmatrix} 1, & 2 \\ 4, & 5 \\ 3, & 2 \\ \dots, & \dots \end{bmatrix}$$



$$f(x) \quad \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \dots \end{bmatrix} \qquad y \quad \begin{bmatrix} 1 \\ 0 \\ 1 \\ \dots \end{bmatrix}$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}), y^{(i)})$$

Prediction true label

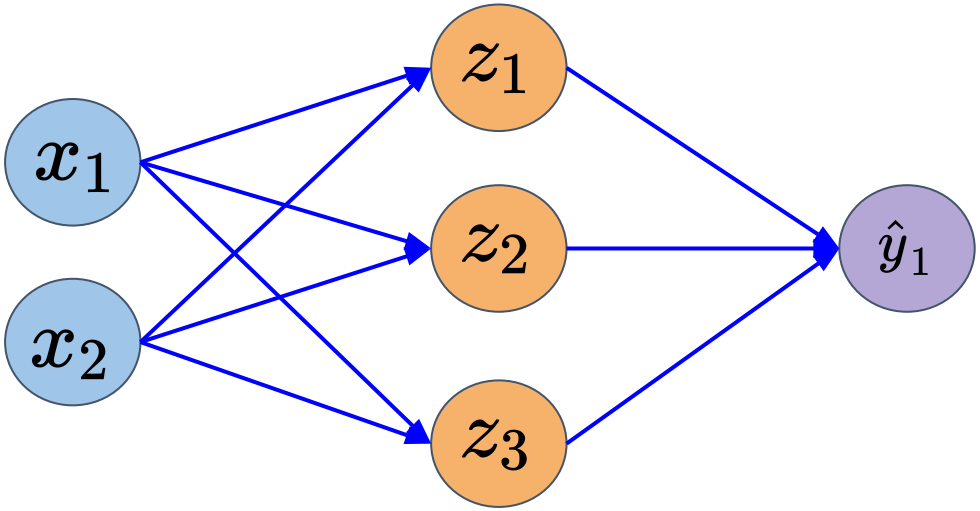
Source: http://introtodeeplearning.com/

Mean Squared Error Loss

The mean squared error loss can be used with regression models that output continuous real numbers.

Ex: final score prediction

$$X = \begin{bmatrix} 1, & 2 \\ 4, & 5 \\ 3, & 2 \\ \dots, & \dots \end{bmatrix}$$



$$f(x) \quad \begin{bmatrix} 30 \\ 80 \\ 85 \\ \dots \end{bmatrix} \qquad y \quad \begin{bmatrix} 90 \\ 20 \\ 95 \\ \dots \end{bmatrix}$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}))^2$$

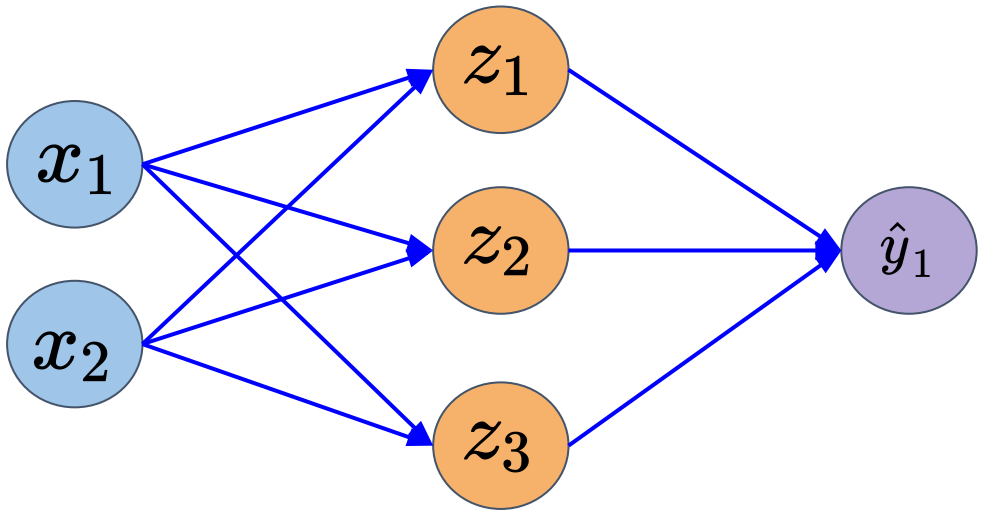
True label Prediction

Source: http://introtodeeplearning.com/

Binary Cross Entropy Loss

The cross entropy loss (as introduced with logistic regression) can be used with models that output a probability between 0 and 1.

$$X = \begin{bmatrix} 1, & 2 \\ 4, & 5 \\ 3, & 2 \\ \dots, & \dots \end{bmatrix}$$



Ex: passing probability

$$f(x) \quad \begin{bmatrix} 0.3 \\ 0.0 \\ 0.8 \\ \dots \end{bmatrix} \qquad y \quad \begin{bmatrix} 1 \\ 0 \\ 1 \\ \dots \end{bmatrix}$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} log(f(x^{(i)})) + (1 - y^{(i)}) log(1 - f(x^{(i)}))$$

True label Prediction True label Prediction

Source: http://introtodeeplearning.com/

Categorical Cross-entropy loss

Also often called **Softmax Loss**. It consist of a softmax activation plus the cross-entropy loss. This can be used to output a probability over *k* classes.

Softmax function also known as **softargmax** or **normalized exponential function:** Input *k* real numbers -> *k* probabilities proportional to the exponentials of the input numbers.
After softmax, all components will add up to 1.

$$softmax(z)_i = S(z)_i = \frac{exp(z_i)}{\sum_{j=1}^{k} exp(z_j)}$$

Notes to Softmax

The name “softmax” can be somewhat confusing. The function is more closely related to the arg max function than the max function. The term “soft” derives from the fact that the softmax function is continuous and differentiable.

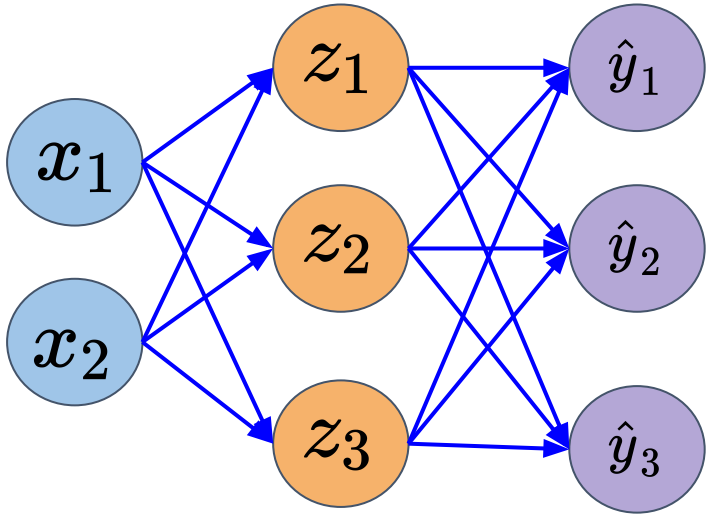
The arg max function, with its result represented as a one-hot vector, is not continuous or differentiable. The softmax function thus provides a “softened” version of the arg max.

It would perhaps be better to call the softmax function “**softargmax**,” but the current name is an entrenched convention.

Categorical Cross-entropy loss

Also often called **Softmax Loss**. It consist of a softmax activation plus the cross-entropy loss. This can be used to output a probability over *k* classes.

$$X = \begin{bmatrix} 1, & 2 \\ 4, & 5 \\ 3, & 2 \\ \dots, & \dots \end{bmatrix}$$



Ex: cat, dog, other?

$$f(x) \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.0 & 0.9 & 0.1 \\ 0.2 & 0.2 & 0.6 \\ \dots & \dots & \dots \end{bmatrix} \qquad y \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ \dots & \dots & \dots \end{bmatrix}$$

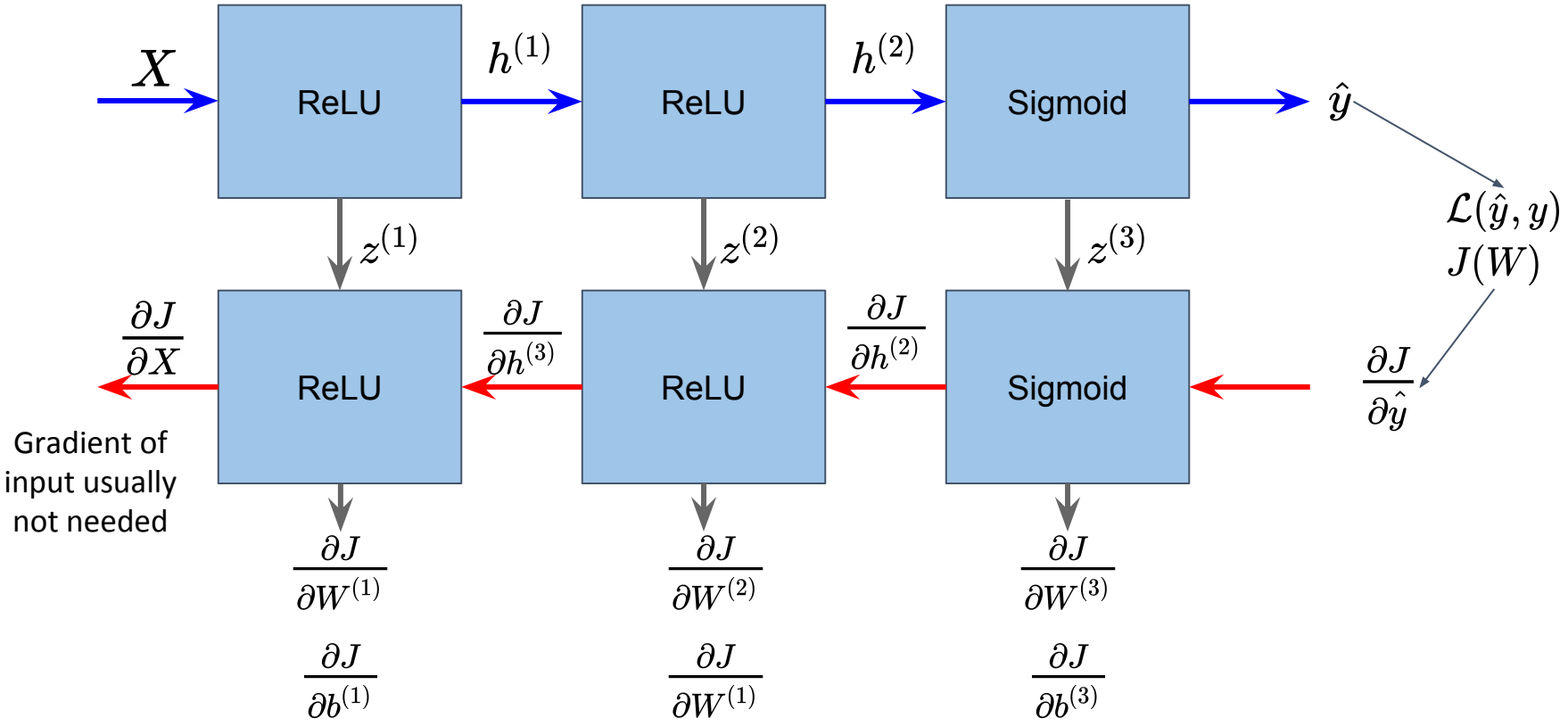
$$J(W) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} y^{(ij)} log(S(f(x^{(i)})) + (1 - y^{(ij)}) log(1 - S(f(x^{(i)}))$$

True label Prediction True label Prediction

Source: http://introtodeeplearning.com/

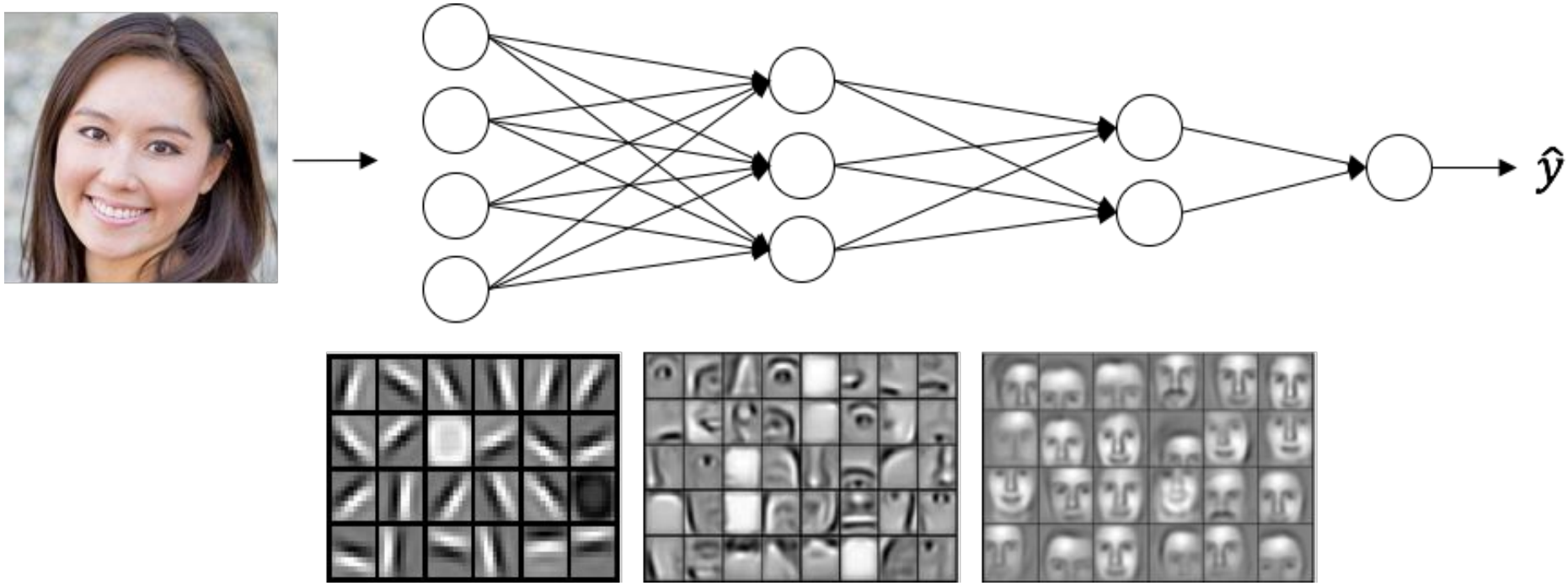
Forward and Backward Propagation

$$h^{(1)} = g^{(1)}(z^{(1)}) = g^{(1)}(W^{(1)T}x + b^{(1)})$$
$$h^{(2)} = g^{(2)}(z^{(2)}) = g^{(2)}(W^{(2)T}h^{(1)} + b^{(2)})$$
$$\hat{y} = g^{(3)}(z^{(3)}) = g^{(3)}(W^{(3)T}h^{(2)} + b^{(3)})$$



Source: https://www.coursera.org/learn/neural-networks-deep-learning

Intuition about deep representation



Source: https://www.coursera.org/learn/neural-networks-deep-learning

Universal approximation theorem

Universal approximation theorem

A feedforward network with a **single hidden layer** containing a finite number of neurons can approximate **any function**, but the layer may be unfeasibly large and may fail to learn and generalize correctly.

Using deeper models can reduce the number of units required to represent the function and reduce the generalization error.

See “Neural Networks and Deep Learning, Michael Nielsen” for an intuitive discussion on that topic.

History of Deep Learning

Modern feedforward networks is the culmination of centuries of progress.

- 17th century: Chain rule that underlies the back-propagation algorithm
- 19th century: Iteratively approximating the solution
- 1940s: Perceptron (linear)
- 1965: Multilayer perceptron
- 1989: LeNet - Handwritten character recognition (sigmoid activation functions)
- 2006: Deep-learning renaissance due to datasets and computation power
- 2009: ImageNet (14M labeled images)
- 2012: AlexNet

The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.

History of Deep Learning

Most improvements in neural networks performance from the 1980s to 2015 can be attributed to two factors:

- Larger datasets: reduced the degree to which statistical generalization is a challenge for a neural network.
- Larger architectures due to more powerful computers and better software infrastructure.

Additional important algorithmic changes:

- Replacement of mean squared error (popular in 1980s and 1990s) with the cross-entropy loss function.
- Replacement of sigmoid activation function in hidden units with piecewise linear hidden units, such as ReLU.

Deep Learning APIs

- Provide a high level API for learning neural networks (define models, load data, automated differentiation)
- Mostly python libraries (caffe is c++)
- For “standard” users these APIs have little difference in terms of what you can do with them

Automated differentiation with autograd in PyTorch

- Differentiating mathematical programs with *autograd*
- Automated differentiation is the basis for learning neural networks

Python Deep-Learning Notebooks

- PyTorch Introduction
- Linear regression in PyTorch
- Using a pre-trained network for image classification
- Transfer learning - Using a pre-trained network on another dataset

Summary

- A neural network is represented as a computational graph.
- Each artificial neuron can be modelled as a perceptron.
- The choice of activation function is important to be differentiable and small derivatives should be avoided (vanishing gradients).
- Many different cost functions exists - most used is cross entropy for classification tasks.
- With Deep Learning libraries we only need to specify the forward pass.

Credits

Books:

- https://www.deeplearningbook.org/
- http://neuralnetworksanddeeplearning.com/

Online Course from MIT:

- http://introtodeeplearning.com/

Online course from Stanford University:

- https://www.coursera.org/specializations/deep-learning?

Other

- cs231n.github.io
- appliedgo.net
- brohrer.github.io
- learnopencv.com

General:
- Motivation ? How should this be done?
- What is deep-learning (ai->ml->dl)?
- History of deep-learning
- NN basics broad overview
- NN basics technical
  - Computational graph
  - Loss functions (perceptron, logreg, …)
  - Activation functions
  - Weight initialisation
  - Overfitting:
    - Regularization (weight, dropout, early-stopping, data augmentation)
  - Normalization
  - Vanishing/exploding gradients
  - Optimization algorithms (GD, mini-batch GD, momentum, RMSprop, Adam)
    - Learning rate decay

- Evaluation metrics (how to compare performance?)

CNNs
  - Edge detection
  - Convolutions (vs cross correlation)
  - Padding, stride, etc.
  - Pooling layers
- Common CNN network types:
  - LeNet
  - AlexNet
  - VGG
  - ResNet
  - Inception

Popular datasets
GANs and autoencoders
Beyond Classification (localization/detection?)
Face verification/classification
Neural style transfer

PyTorch introduce:
- Autograd
- Simple logreg setup
- How to manually create a network (exercise)
- How to create convolutions (exercise)
- How to use existing architectures (pretrained)
- Transfer learning (retrain softmax)
-