Machine Learning in Healthcare

#C17 Neural Networks: Introduction

Technion-IIT, Haifa, Israel

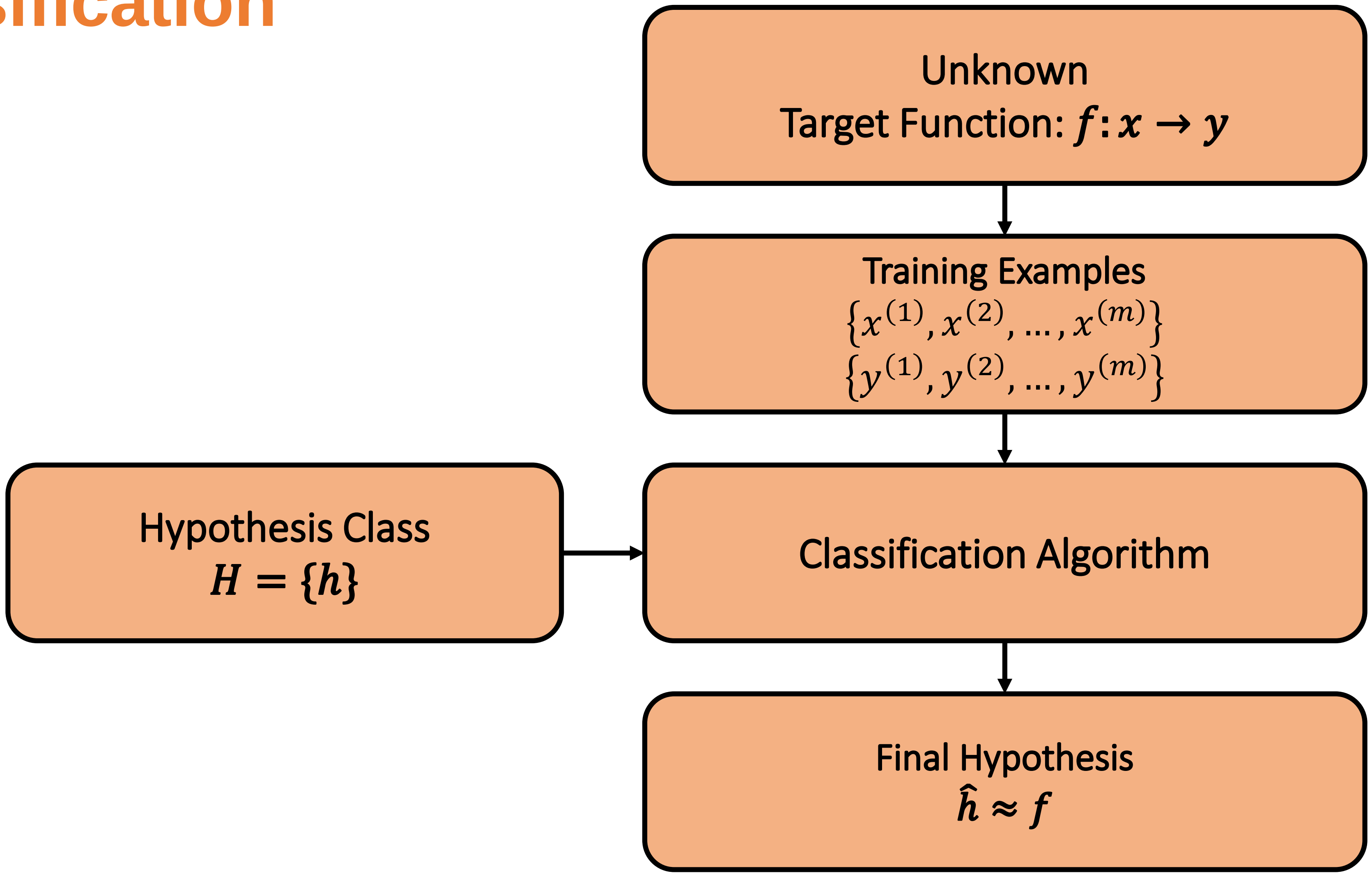
Assist. Prof. Joachim Behar
Biomedical Engineering Faculty
Technion-IIT

AIMLab.

Agenda

- Introduction to NN
  - LR and NN
  - Representation learning
- Forward propagation
  - Equations
  - Vectorization
- Backward propagation
  - Equations
- Activation functions
  - Types
  - Non-linearity
- Multiclass classification.

Classification

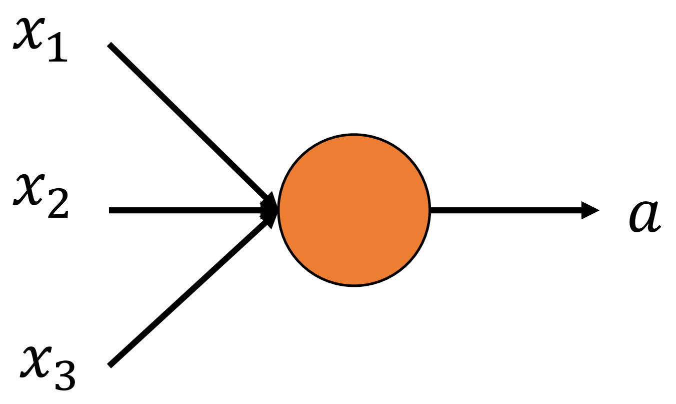

Unknown
Target Function: $\boldsymbol{f}: \boldsymbol{x} \rightarrow \boldsymbol{y}$
Training Examples
$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
$\{y^{(1)}, y^{(2)}, \dots, y^{(m)}\}$
Hypothesis Class
$\boldsymbol{H} = \{\boldsymbol{h}\}$
Classification Algorithm
Final Hypothesis
$\hat{\boldsymbol{h}} \approx \boldsymbol{f}$

Introduction to NN

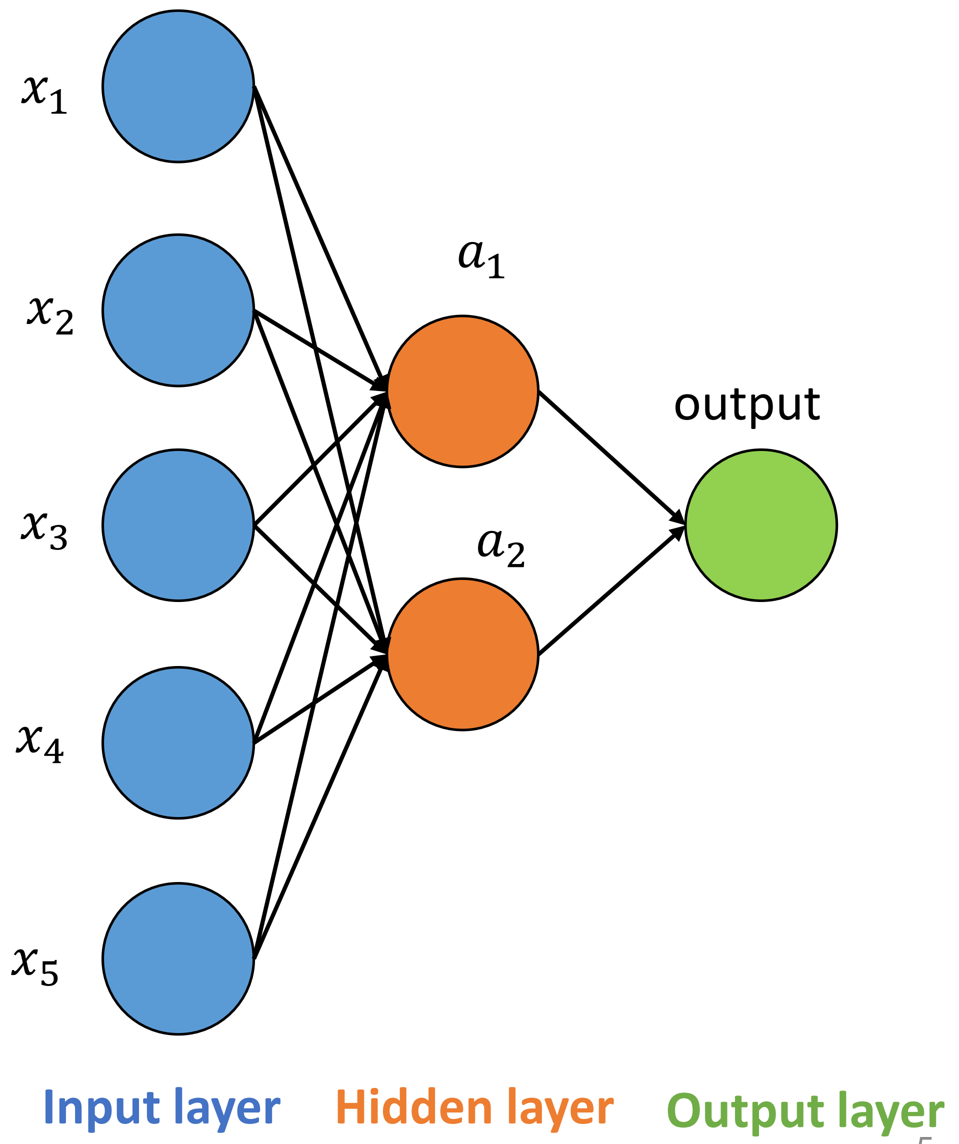
AIMLab.

Logistic regression and NN

- Logistic regression:



- NN as a superposition of multiple LR units:



History - Perceptron

Frank Rosenblatt

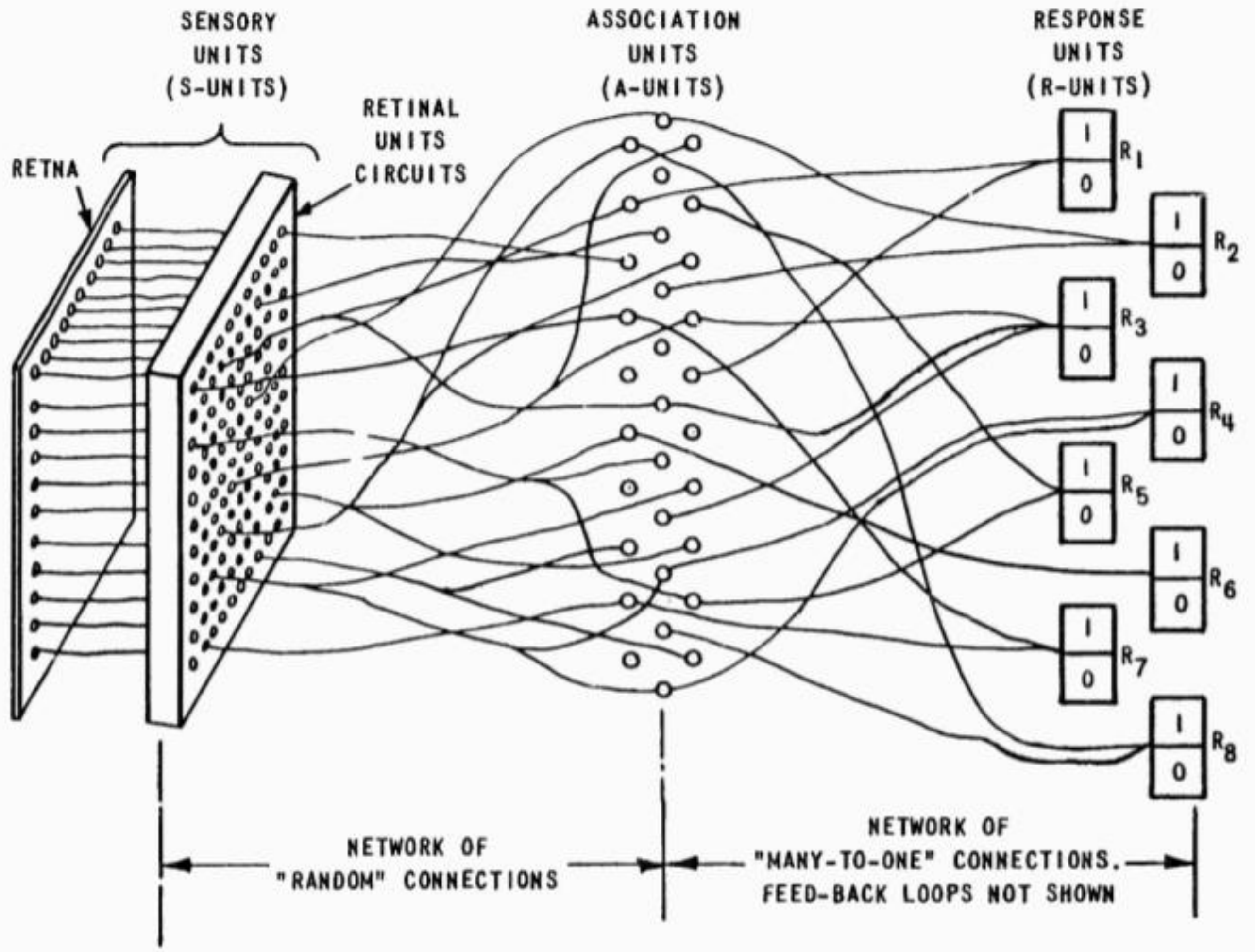
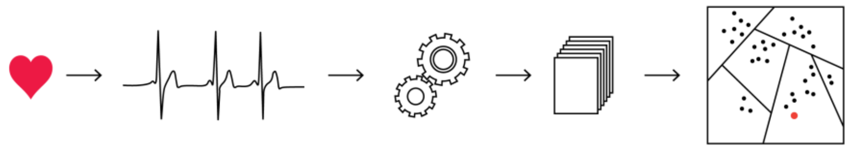

SENSORY UNITS (S-UNITS)
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
RETINAL UNITS CIRCUITS
RETNA
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8
1
0
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS. FEED-BACK LOOPS NOT SHOWN

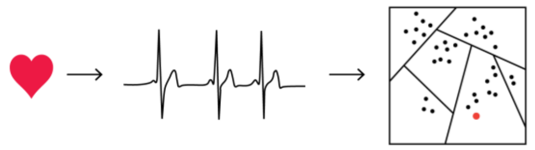
Figure I ORGANIZATION OF THE MARK I PERCEPTRON

Rosenblatt, Frank. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.
Image: Hay, John C., Ben E. Lynch, and David R. Smith. *Mark I perceptron operators' manual*. No. VG-1196-G-5. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1960.

Representation learning

- For many tasks it is difficult to know what features should be extracted.
  - E.g. detect a tumor on a CT scan. Might differ in their location, shapes etc.
- One solution is to use ML to **learn both the features and the mapping function** from the features to the output. This is called **representation learning**.
- Classic machine learning:
  - $f: x \rightarrow y$
  - x the features **we engineered**.
  - f mapping function **we want to learn**.
- Representation learning:
  - $f: x \rightarrow y$
  - x the "**raw data**".
  - f mapping function **we want to learn**.





Representation learning

- Different AI disciplines.
- Shaded box represent components that are able to learn from the data.

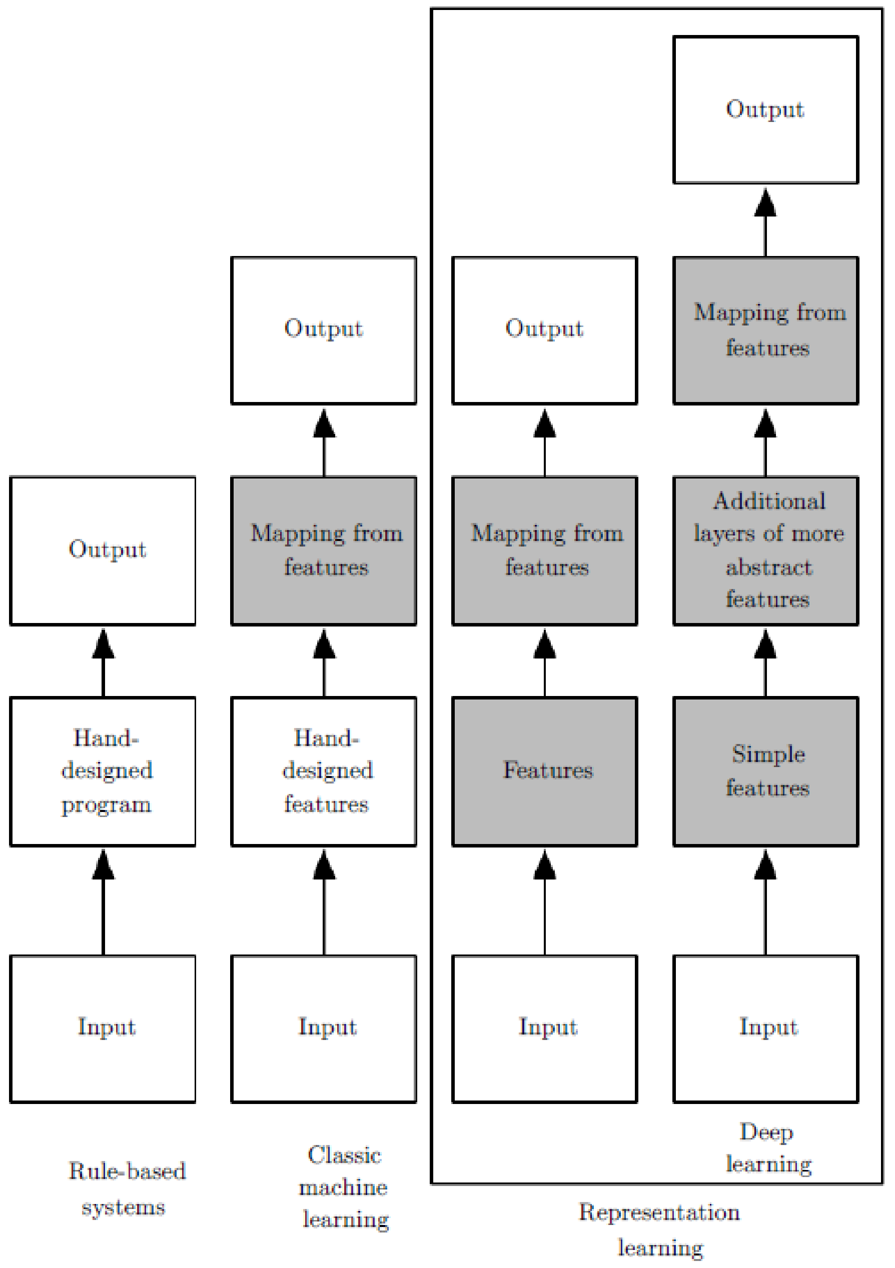


Figure 1.5 reproduced from Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Representation learning

- It might be challenging to directly learn high-level abstract features from raw data.
- Deep learning tackles this challenge in representation learning by introducing representations that are expressed in term of other, simpler representations.
- Rephrased: Deep learning enables to build complex concepts out of simpler ones.

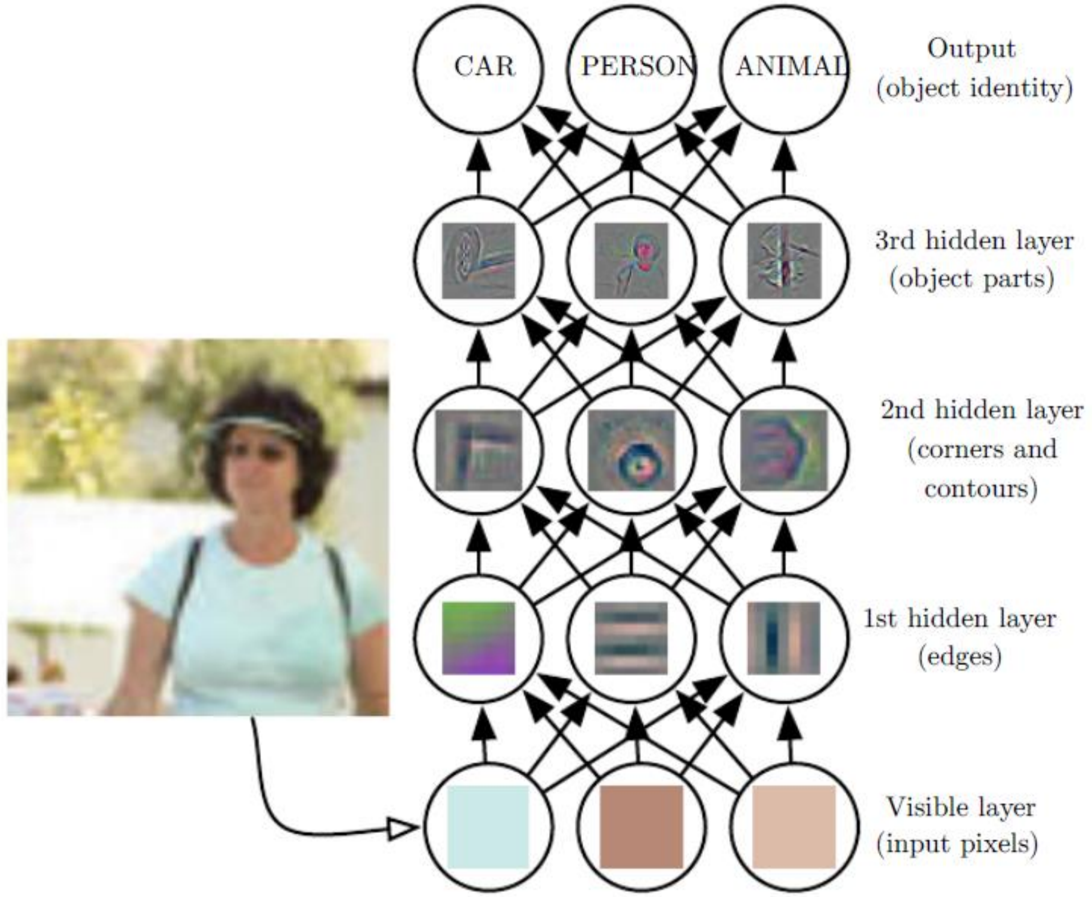


Image from Zeiler and Fergus (2014).

Representation learning

- Breaks a complex mapping into a series of nested simpler mappings.
- These nested simpler mapping are represented in the different layers.
- The networks learns increasingly more complex features as we get deeper and deeper in the layers.

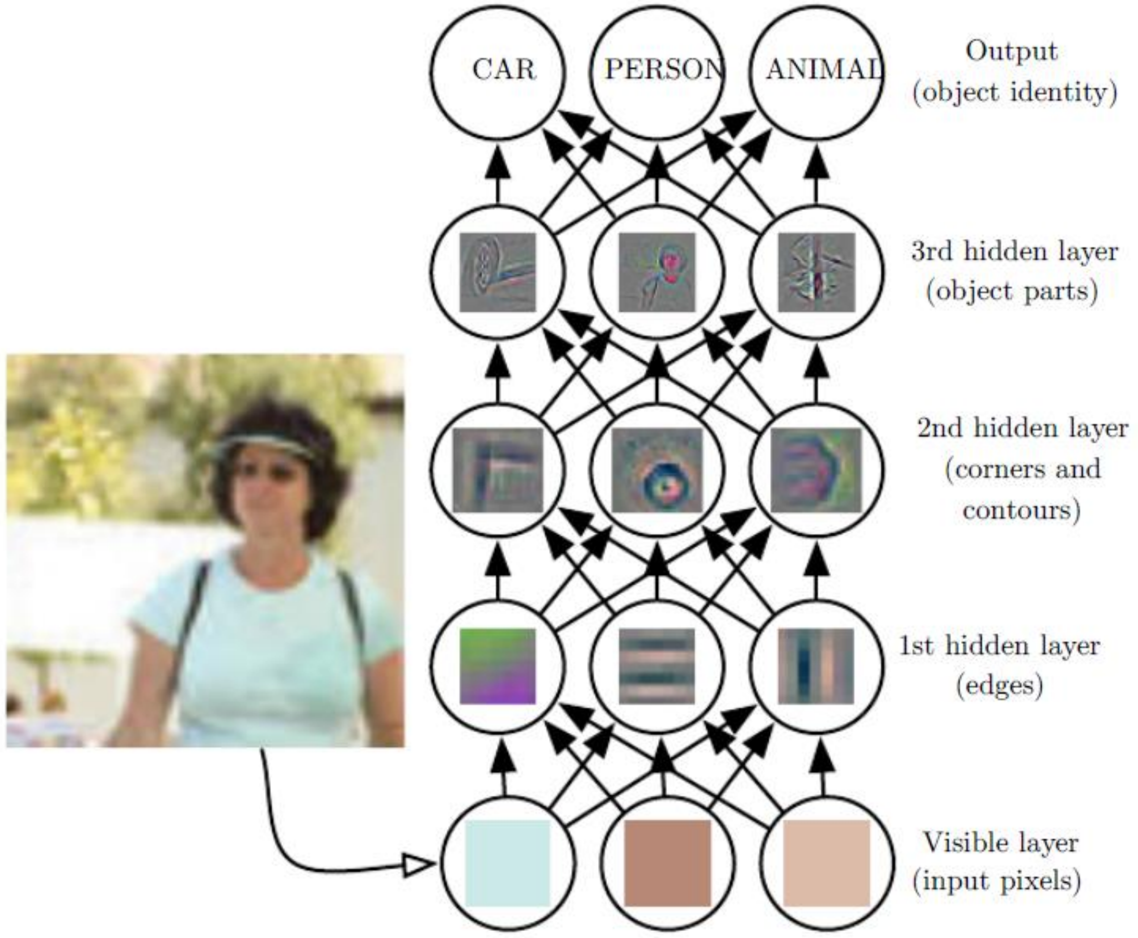
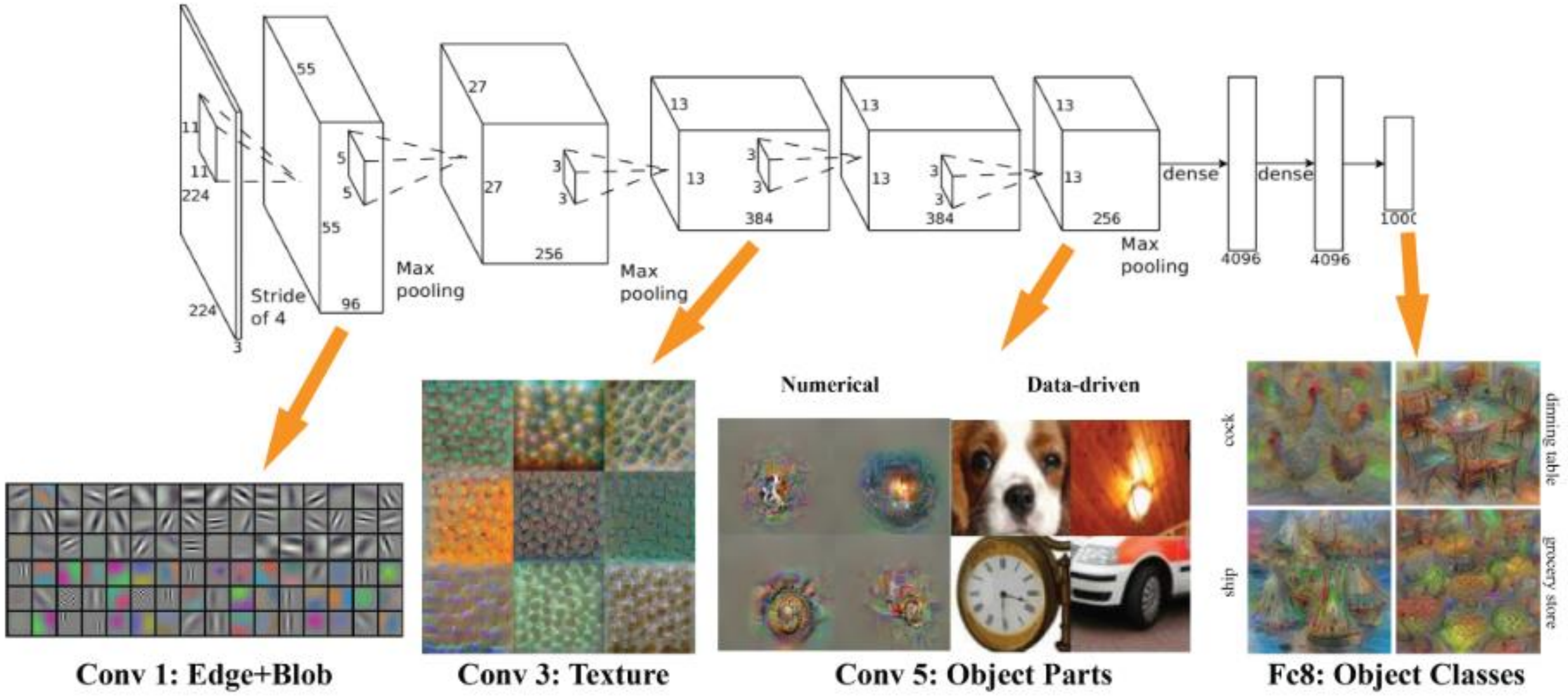

CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

Image from Zeiler and Fergus (2014).

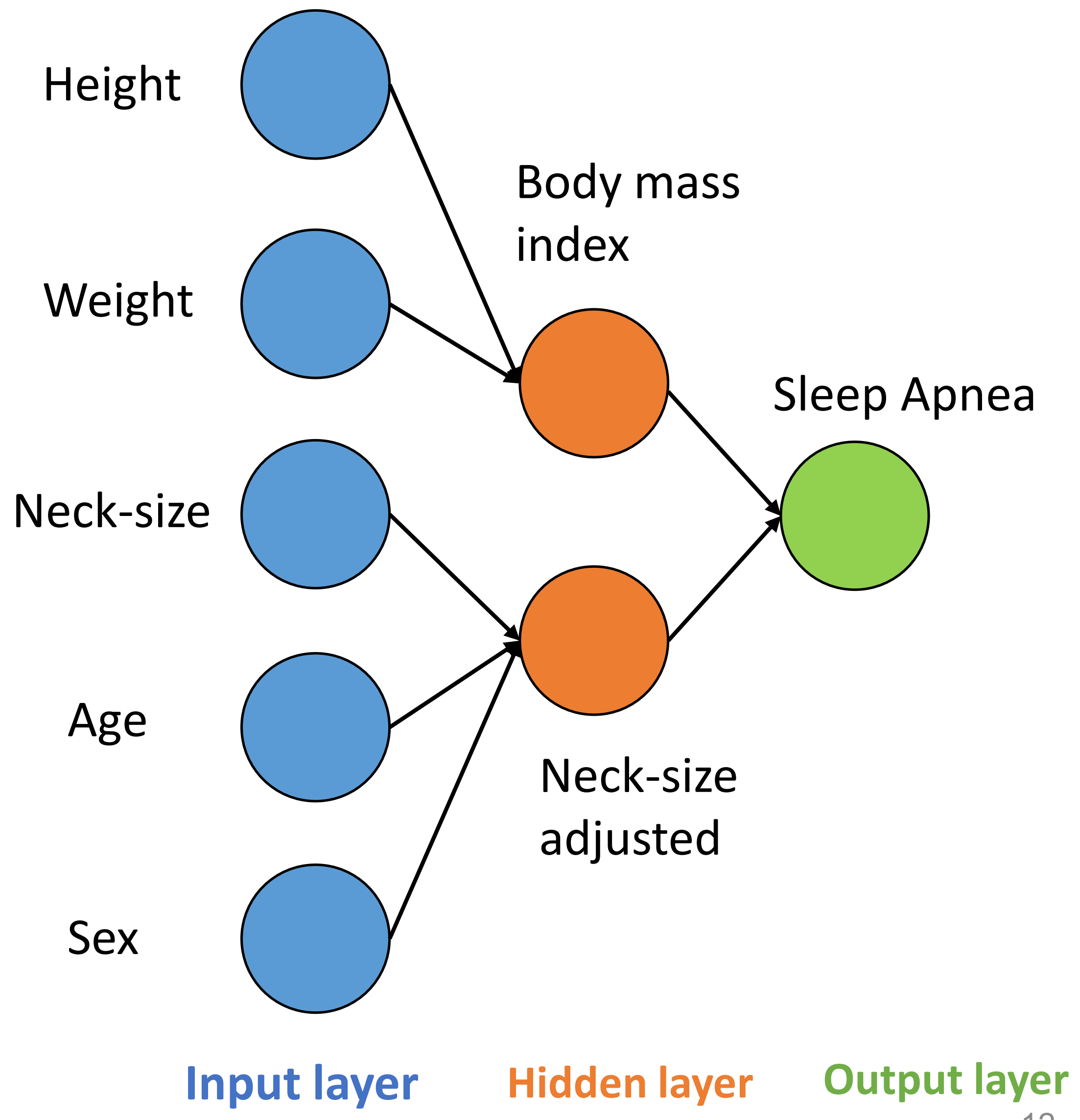
Representation learning



Mahendran, Aravindh, and Andrea Vedaldi. "Understanding deep image representations by inverting them." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
Code for visualize neurons trained from deep learning packages through backpropagation optimization : http://vision03.csail.mit.edu/cnn_art/index.html#v_single

Representation learning

- Let’s consider a set of demographic and anthropometric features we want to use to predict the likelihood of an individual having sleep apnea.
- The purpose being to design a simple questionnaire based model that can be used to screen for the condition.
- By adding more layers and units in each layer, the networks can represent mapping of increasing complexity.



Representation learning

- AI, machine learning and representation learning.
- <u>Representation</u> **learning**: learn features.
- <u>Deep</u> **learning**: Build complex concepts out of simpler ones.

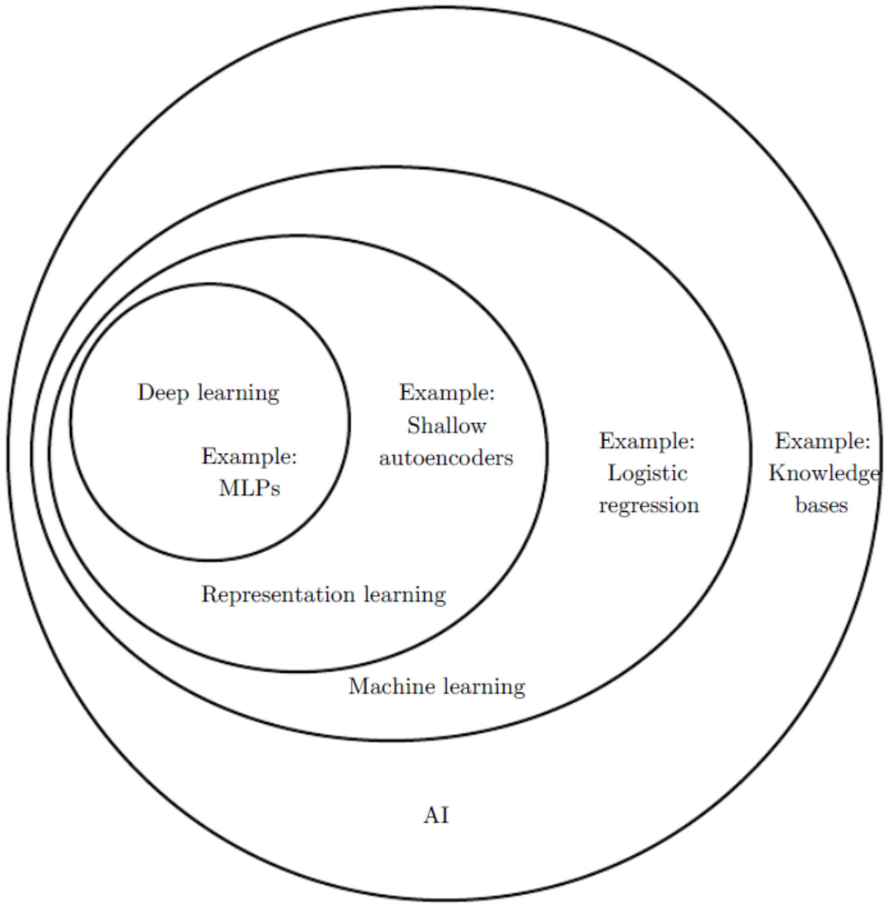
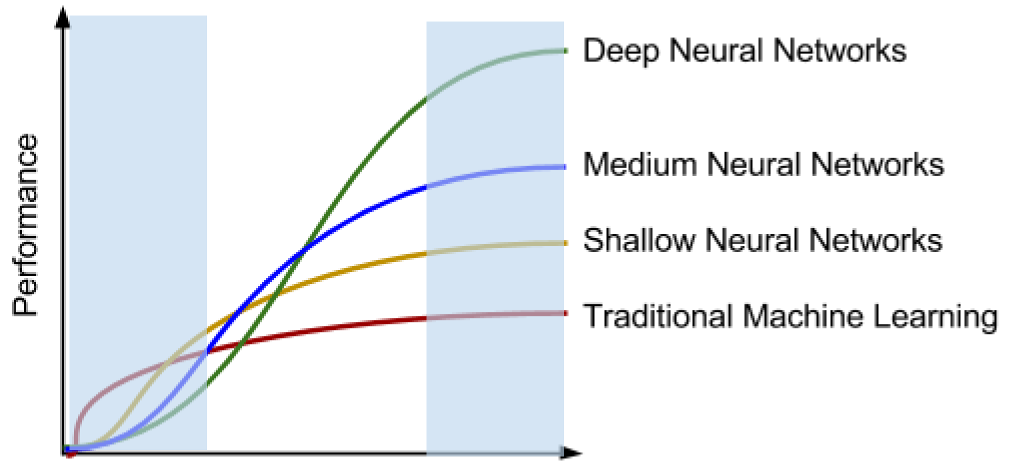


Figure 1.4 reproduced from Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Deep Learning

- What made Deep Learning so popular?
  - Amount of data that is available.
  - Computation and hardware.
  - Algorithms.
- Benchmark to other learning approaches?



The present
(if you have big data!)

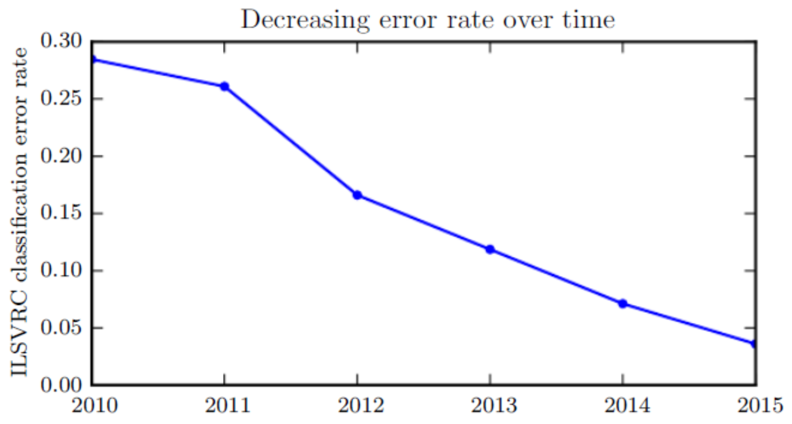
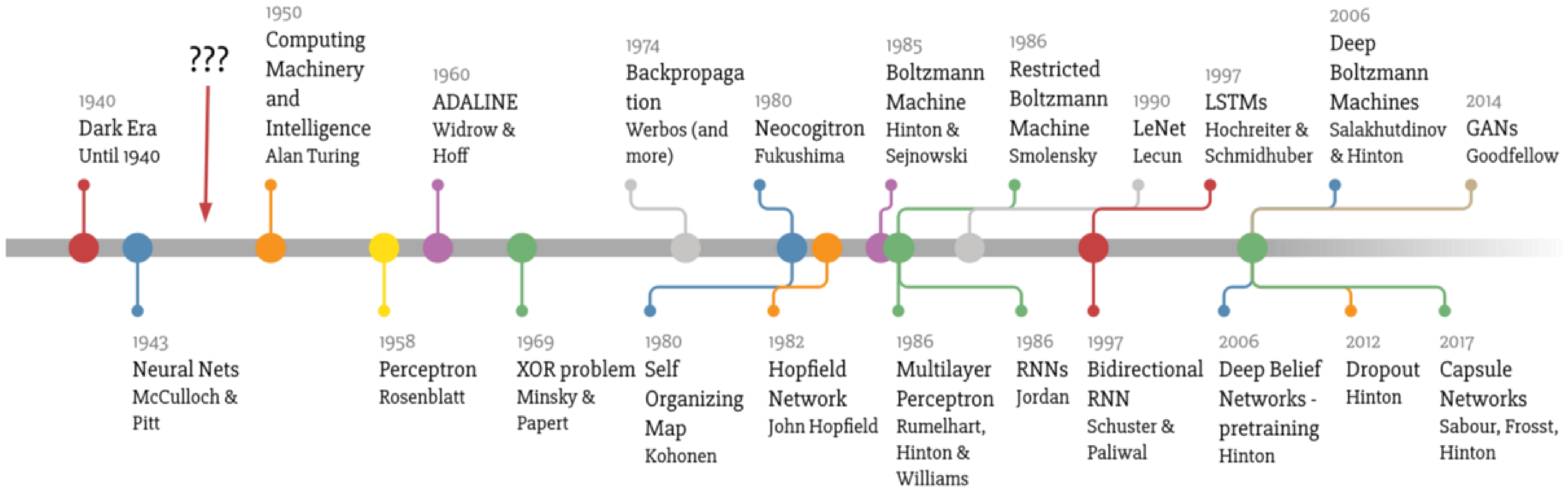


Figure 1.12: Since deep networks reached the scale necessary to compete in the ImageNet Large Scale Visual Recognition Challenge, they have consistently won the competition every year, and yielded lower and lower error rates each time. Data from Russakovsky *et al.* (2014b) and He *et al.* (2015).

Image adapted from: https://blog.easysol.net/building-ai-applications/
Figure 1.12 reproduced from Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

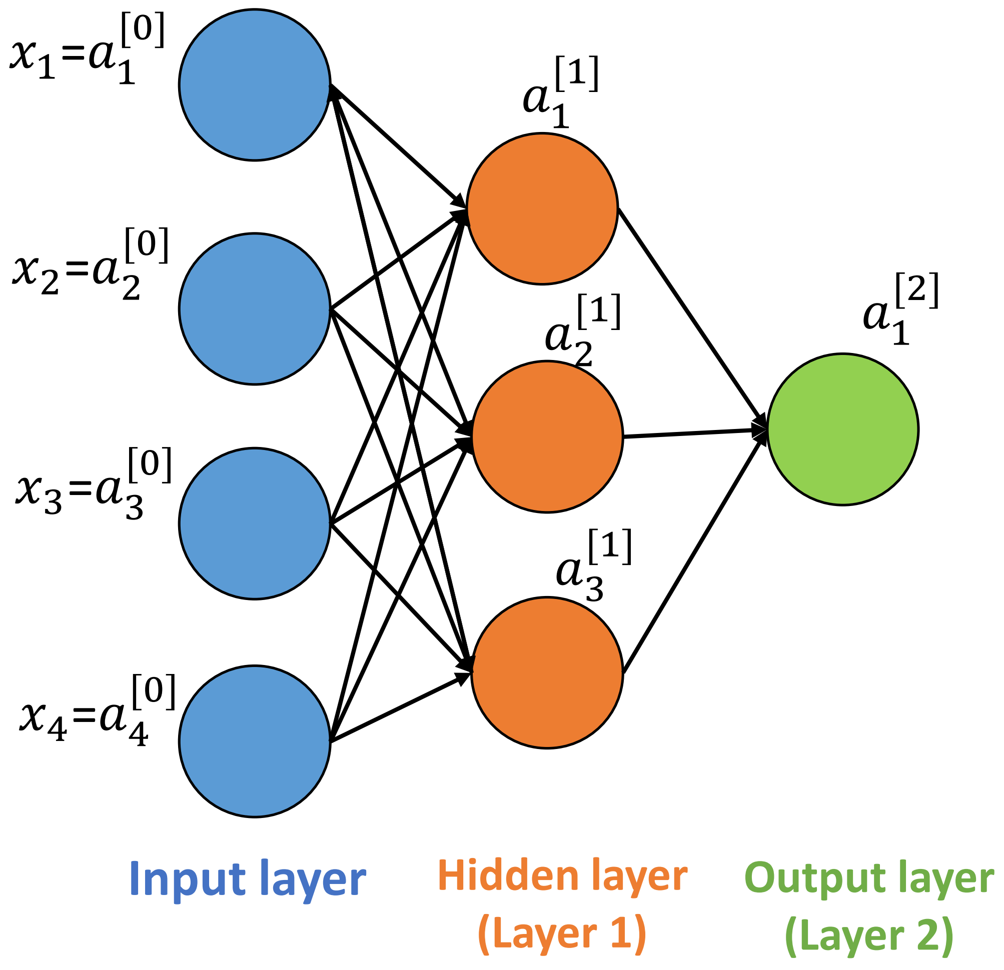
Deep Learning Timeline



Made by Favio Vázquez

Deep Learning

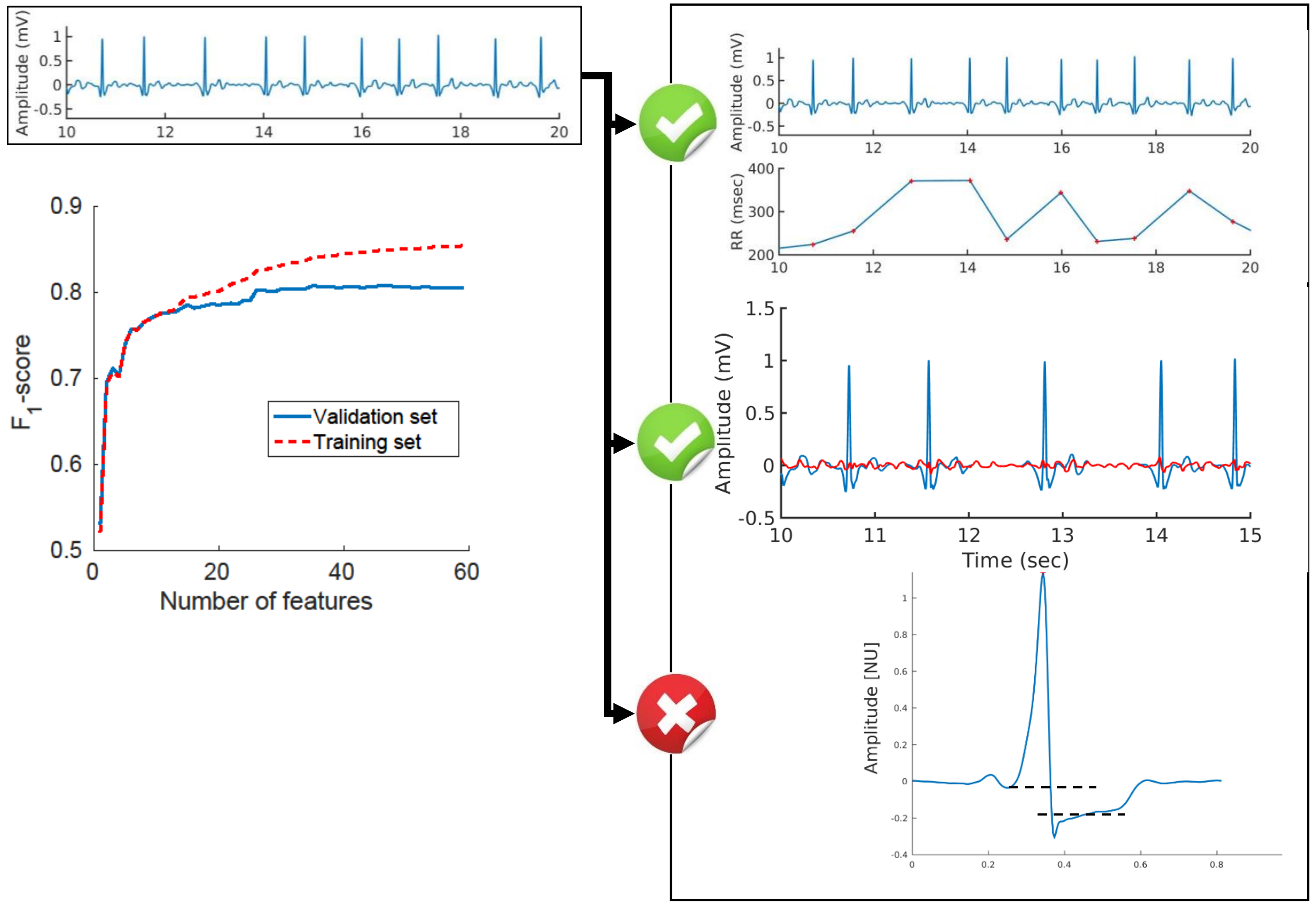
- You give to the neural network a series of m examples and their target class $\{x^{(i)}, y^{(i)}\}$, $\forall i \in [1, m]$.
- The NN will learn the weighs $W^{[l]}$ that connect between the layers.
- How do we make the network learn these weights?



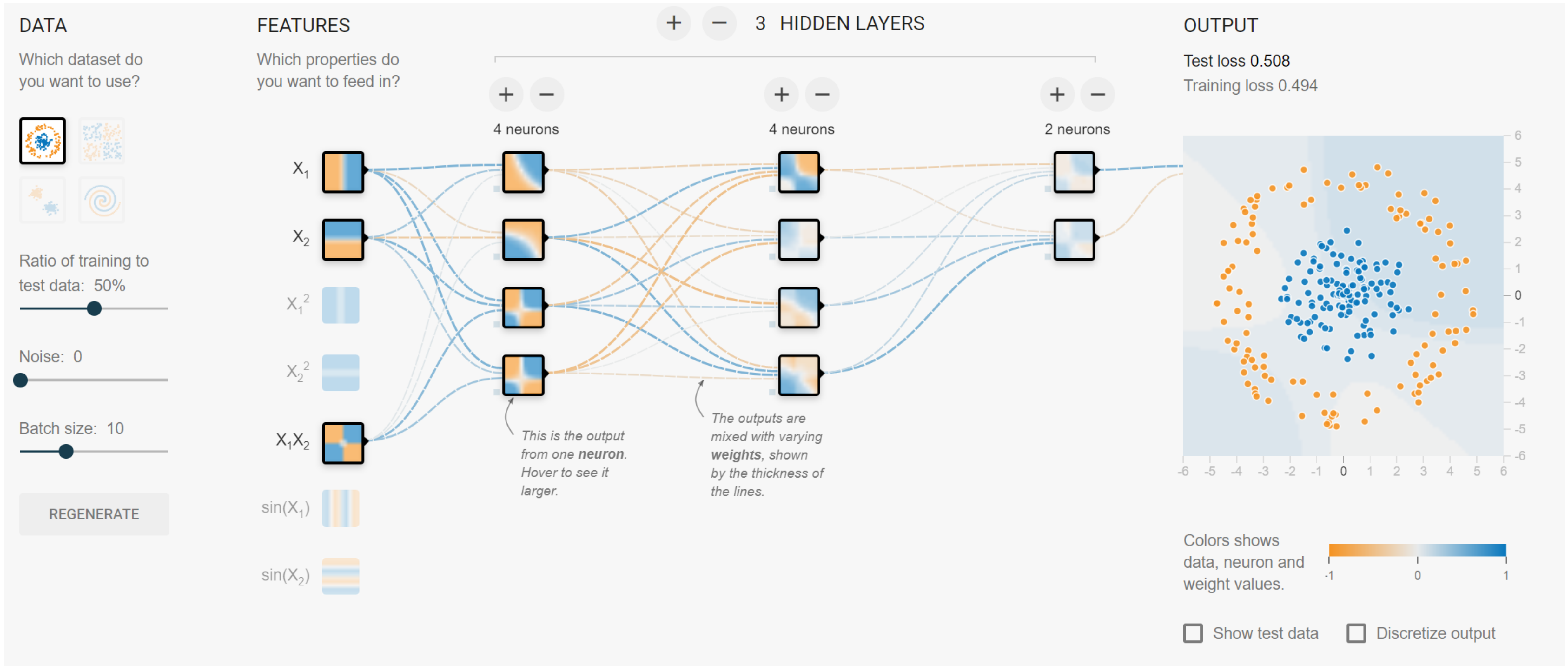
AIMLab.

Representation learning versus “traditional” learning

- Results will be more “black box” i.e. less interpretable with a Deep NN.
- However, a Deep NN might learn features that you would not have thought to engineer. In that sense it can perform better provided enough data.



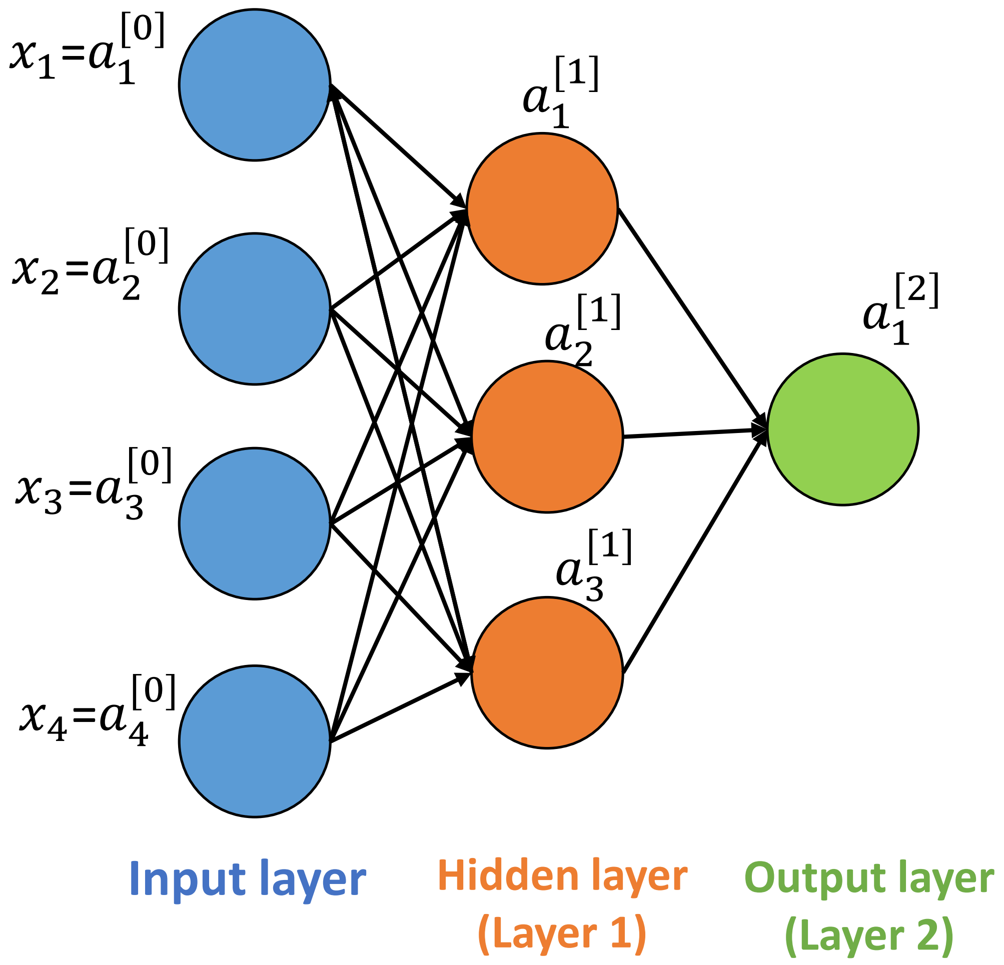
Behar et al., CinC 2017


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ − 3 HIDDEN LAYERS
4 neurons
4 neurons
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.508
Training loss 0.494
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,4,2&seed=0.75791&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=true&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

Vocabulary

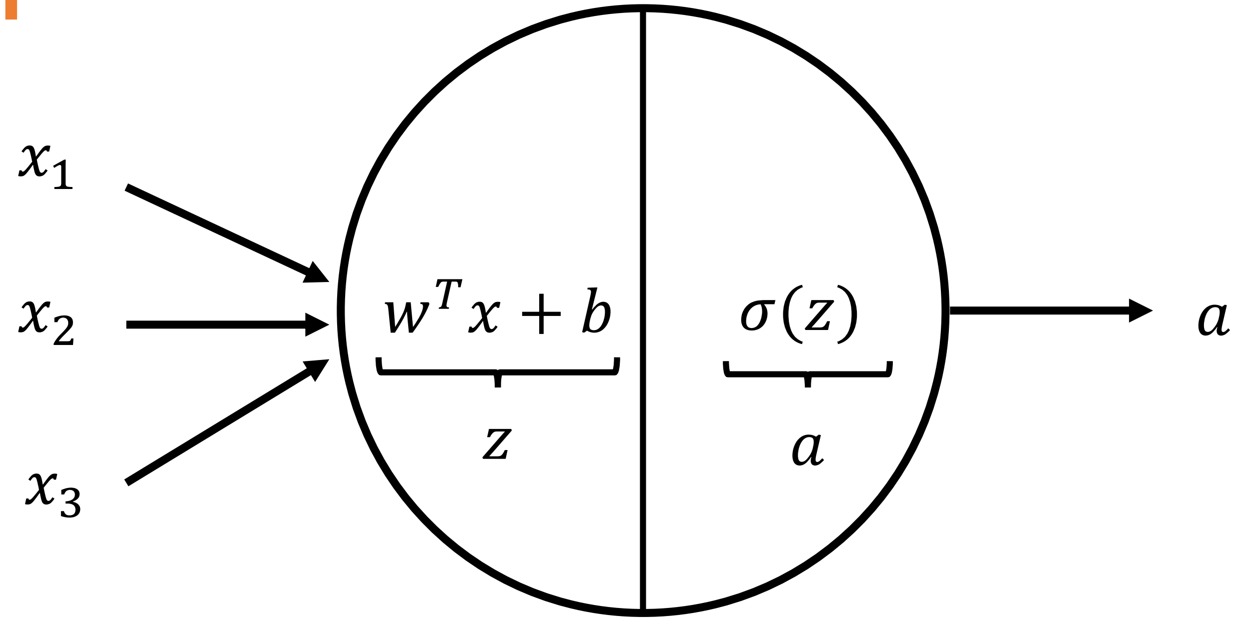
- Architecture: The specific arrangement of the layers and nodes in the network.
- Size: the number of nodes in the model.
- Width: the number of nodes in a specific layer.
- Depth: The number of layers in a neural network.
- Capacity: The type or structure of functions that can be learned by a network configuration. Sometimes called "representational capacity".
- Notation summarizes both the number of layers and the number of nodes in each layer: 4/3/1.



Forward propagation

Recall Logistic Regression

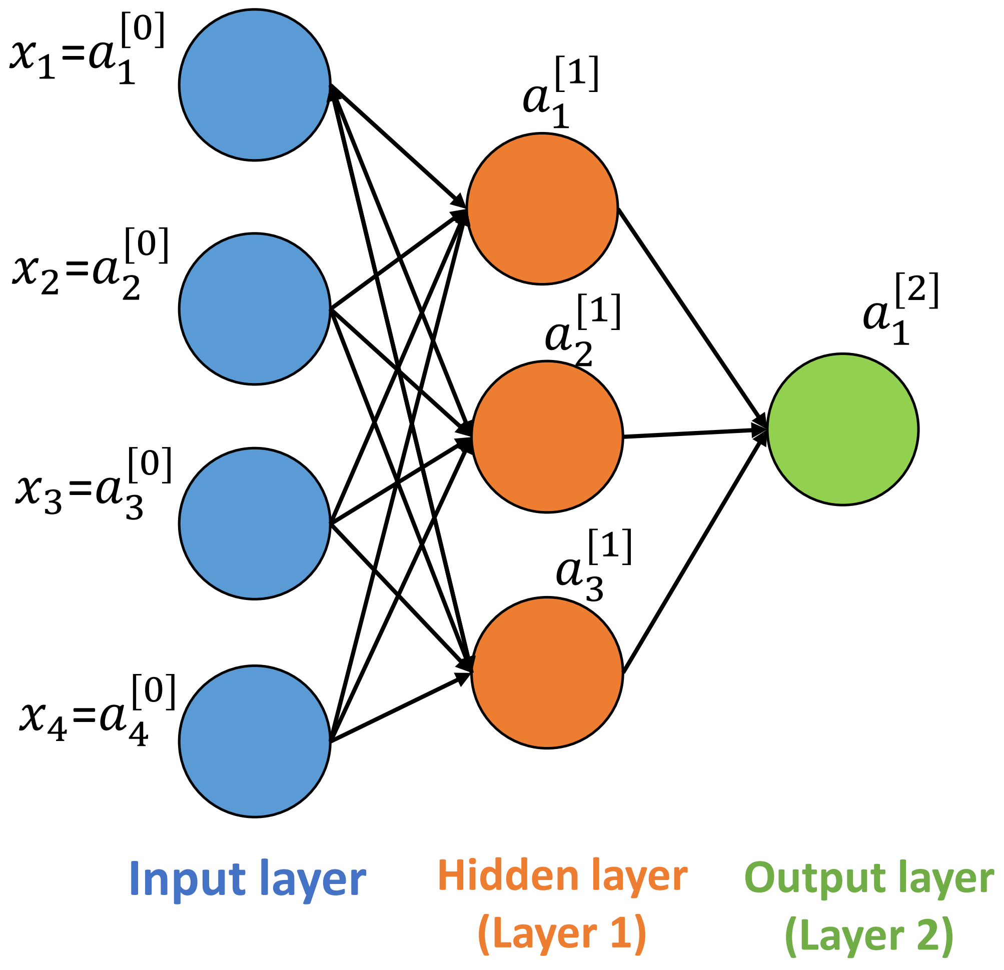
- Logistic Regression (LR):
  - $z = w^T x + b$
  - $a = \sigma(z)$
- Now consider m examples:
  - $z^{(i)} = w^T x^{(i)} + b, \forall i \in [1, m]$
  - $a^{(i)} = \sigma(z^{(i)}), \forall i \in [1, m]$



Single neuron.

Forward propagation, single example, explicit

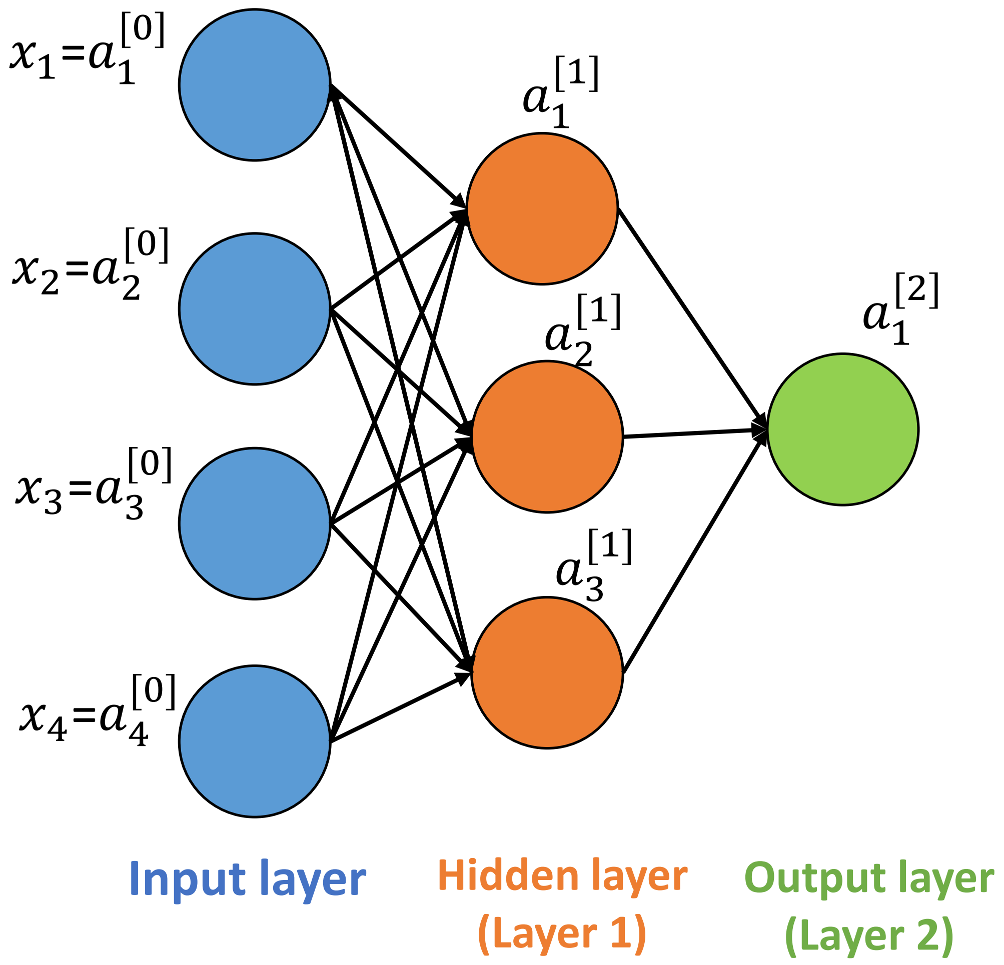
- Let’s consider a two layers NN:
  - $x = [x_1, x_2, x_3, x_4] = a^{[0]}$,
  - $a_j^{[l]}$: j^{th} activation at layer l,
  - **Model parameters**: $b_j^{[l]} \in \mathbb{R}$, $w_j^{[l]} \in \mathbb{R}^{n_h^{[l-1]}}$
    - Where $n_h^{[l]}$: number of units at layer l.
- Computation:
  - $z_1^{[1]} = w_1^{[1]T} \mathrm{x} + b_1^{[1]}$
  - $a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$
  - $z_2^{[1]} = w_2^{[1]T} \mathrm{x} + b_2^{[1]}$
  - $a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$
  - $z_3^{[1]} = w_3^{[1]T} \mathrm{x} + b_3^{[1]}$
  - $a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$.



AIMLab.

Forward propagation, single example, matrix

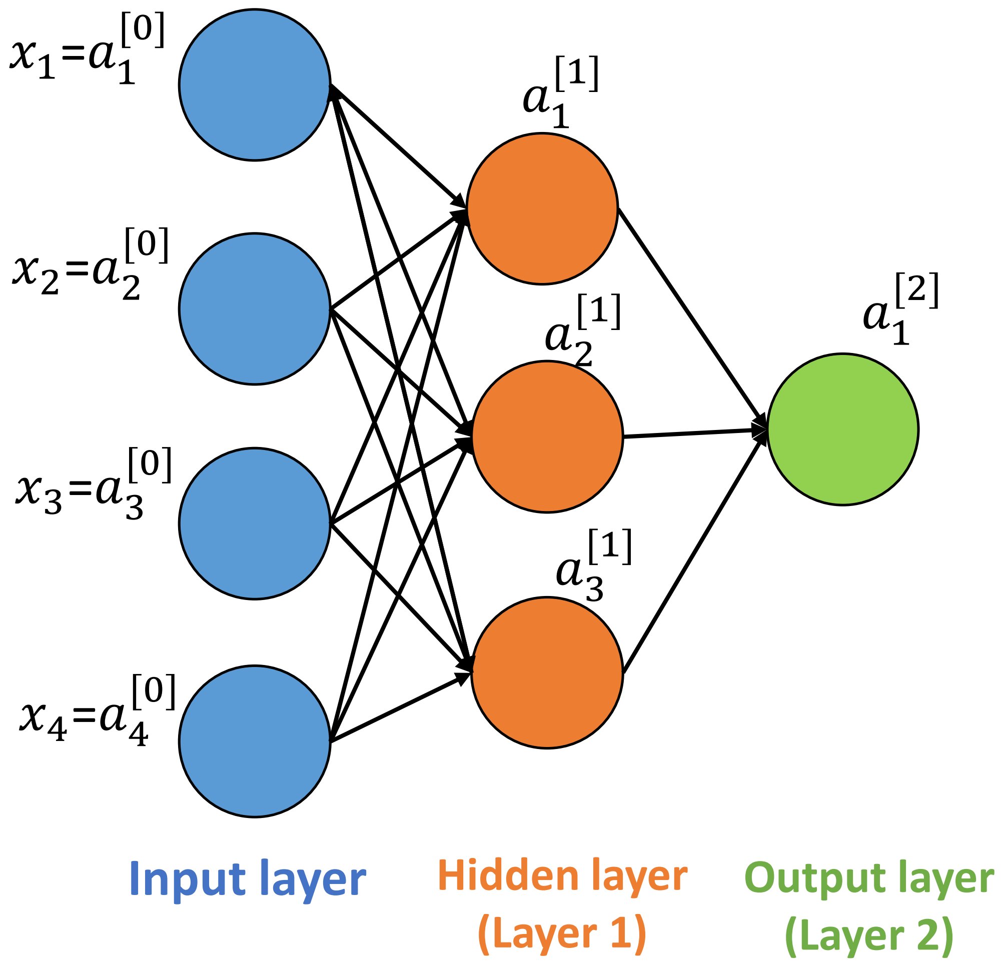
- Equations:
  - First layer:
  - $z_1^{[1]} = w_1^{[1]T}\mathrm{x} + b_1^{[1]}, a_1^{[1]} = \sigma\left(z_1^{[1]}\right),$
  - $z_2^{[1]} = w_2^{[1]T}\mathrm{x} + b_2^{[1]}, a_2^{[1]} = \sigma\left(z_2^{[1]}\right),$
  - $z_3^{[1]} = w_3^{[1]T}\mathrm{x} + b_3^{[1]}, a_3^{[1]} = \sigma\left(z_3^{[1]}\right),$
  - Second layer:
  - $z_1^{[2]} = w_1^{[2]T}a^{[1]} + b_1^{[2]},\ a_1^{[2]} = \sigma\left(z_3^{[2]}\right).$
- Matrix representation:
  - $z^{[1]} = W^{[1]}\mathrm{x} + b^{[1]}, a^{[1]} = \sigma(z^{[1]}),$
  - $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, a^{[2]} = \sigma(z^{[2]}).$
  - With **model parameters**: $W^{[l]} \in \mathbb{R}^{n_h^{[l]} \times n_h^{[l-1]}}, b^{[l]} \in \mathbb{R}^{n_h^{[l]}}$



Forward propagation, single example, matrix

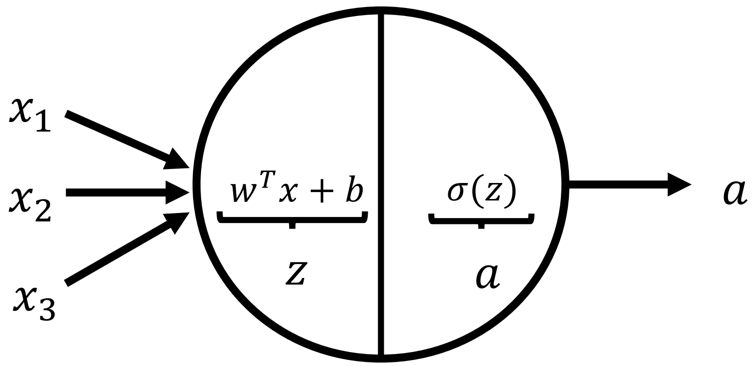
- Matrix:
  - $z^{[1]} = W^{[1]}\mathrm{x} + b^{[1]}, a^{[1]} = \sigma(z^{[1]})$,
  - $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, a^{[2]} = \sigma(z^{[2]})$.
- With:
  - $\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix} = \begin{bmatrix} \dots w_1^{[1]} \dots \\ \dots w_2^{[1]} \dots \\ \dots w_3^{[1]} \dots \\ \dots w_4^{[1]} \dots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$
  - $\begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} = \sigma\left(\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}\right)$.

$\updownarrow$ Number of units $n_h^{[l]}$.



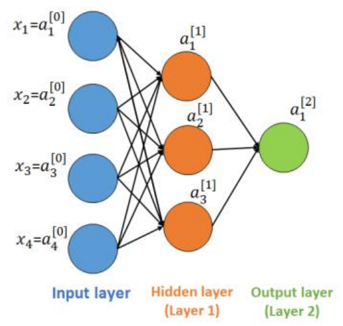
Parallel Between LR and NN

- Logistic Regression (LR):
  - $z = w^T x + b$
  - $a = \sigma(z)$
- Now consider m examples:
  - $z^{(i)} = w^T x^{(i)} + b, \forall i \in [1, m]$
  - $a^{(i)} = \sigma(z^{(i)}), \forall i \in [1, m]$



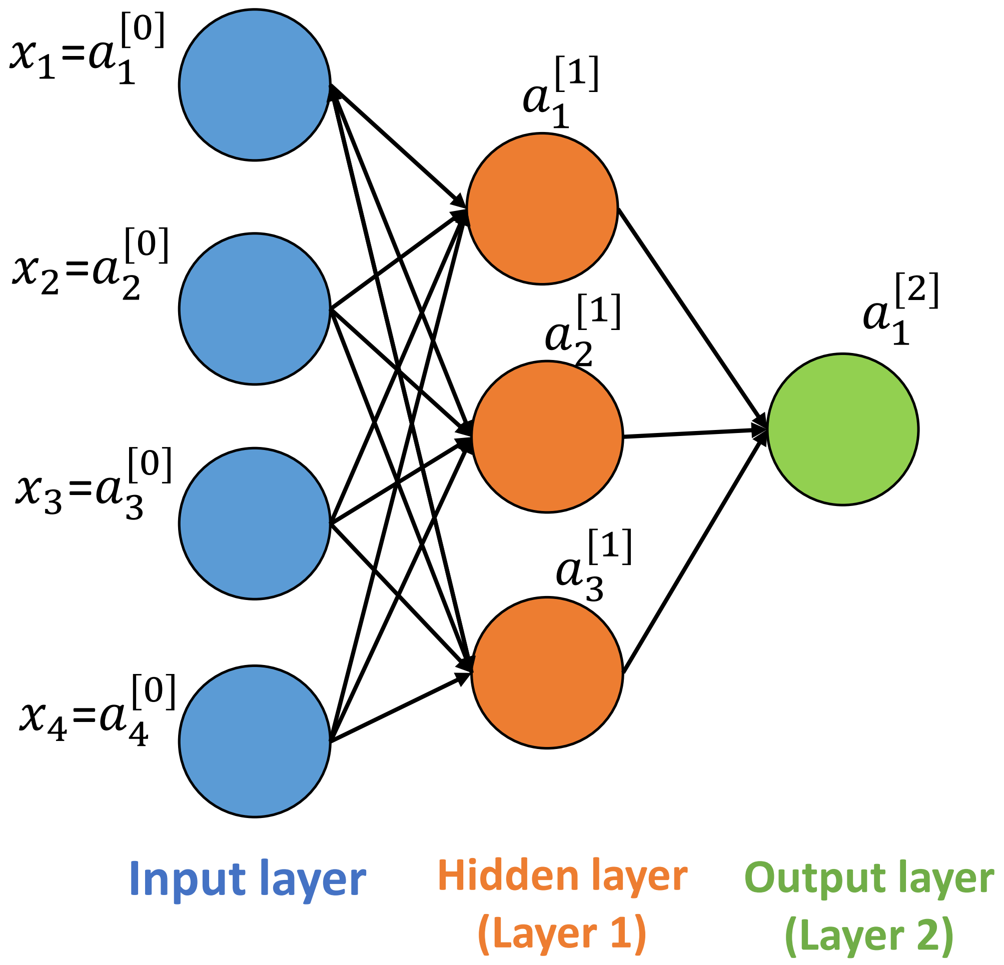
Single neuron.

- Neural network, layer l, one example:
  - $z^{[l]} = W^{[l]} x + b^{[l]}, \forall l \in [1, L]$
  - $a^{[l]} = \sigma\left(z^{[l]}\right), \forall l \in [1, L]$
  - Where l is the layer number.
- Neural network, layers l, **multiple examples** i:
  - $z^{[l](i)} = W^{[l]} x^{(i)} + b^{[l]}, \forall i \in [1, m]$
  - $a^{[l](i)} = \sigma\left(z^{[l](i)}\right), \forall i \in [1, m]$
  - Where i is the example number.



Forward propagation, multiple examples, matrix

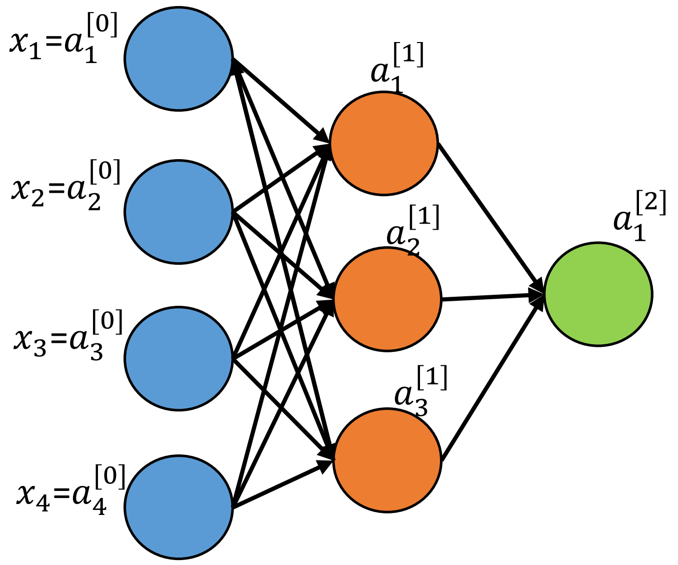
- Matrix formulation for a single example:
  - $z^{[1]} = W^{[1]}\mathrm{x} + b^{[1]}, a^{[1]} = \sigma\left(z^{[1]}\right),$
  - $z^{[2]} = W^{[2]}\mathrm{x} + b^{[2]}, a^{[2]} = \sigma\left(z^{[2]}\right).$
- Matrix formulation for multiple examples:
  - $Z^{[1]} = W^{[1]}\mathrm{X} + \tilde{b}^{[1]}, A^{[1]} = \sigma\left(Z^{[1]}\right),$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \tilde{b}^{[2]}, A^{[2]} = \sigma\left(Z^{[2]}\right).$
- With:
  - $Z^{[l]} = \begin{bmatrix} \vdots & \vdots & \\ z^{[l](1)} & z^{[l](2)} & \cdots \\ \vdots & \vdots & \end{bmatrix}, A^{[l]} = \begin{bmatrix} \vdots & \vdots & \\ a^{[l](1)} & a^{[l](2)} & \cdots \\ \vdots & \vdots & \end{bmatrix}.$
  - $\tilde{b}^{[l]} = \begin{bmatrix} \vdots & \vdots & \\ b^{[l]} & b^{[l]} & \cdots \\ \vdots & \vdots & \end{bmatrix}$ This is called **broadcasting**.
  - $X \in \mathbb{R}^{n_x \times m}, Z^{[l]}$ and $A^{[l]} \in \mathbb{R}^{n_h^{[l]} \times m}$
  - Model parameters: $W^{[l]} \in \mathbb{R}^{n_h^{[l]} \times n_h^{[l-1]}}, \tilde{b}^{[l]} \in \mathbb{R}^{n_h^{[l]} \times m}$



AIMLab.

Backward propagation

Backward propagation



- Cost function:
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
- Gradient descent:
  - $W^{[2]} \coloneqq W^{[2]} - \alpha \frac{\partial J}{\partial W^{[2]}}$; $b^{[2]} \coloneqq b^{[2]} - \alpha \frac{\partial J}{\partial b^{[2]}}$
  - $W^{[1]} \coloneqq W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}}$; $b^{[1]} \coloneqq b^{[1]} - \alpha \frac{\partial J}{\partial b^{[1]}}$
- We need to compute these derivatives. We provide the results in order to understand the importance of the choice of the activation function.
- Assuming a sigmoid activation function for the output layer and a quadratic cost function:
  - $\frac{\partial J}{\partial W^{[2]}} = \frac{1}{n}(A^{[2]} - Y)A^{[1]T}$
  - $\underbrace{\frac{\partial J}{\partial W^{[1]}}}_{(n^{[1]}, m)} = \frac{1}{n}\underbrace{W^{[2]T}\left(A^{[2]} - Y\right)}_{(n^{[1]}, m)} * \underbrace{\color{red}{\boldsymbol{g^{[1]'}}}(Z^{[1]})}_{(n^{[1]}, m)}$

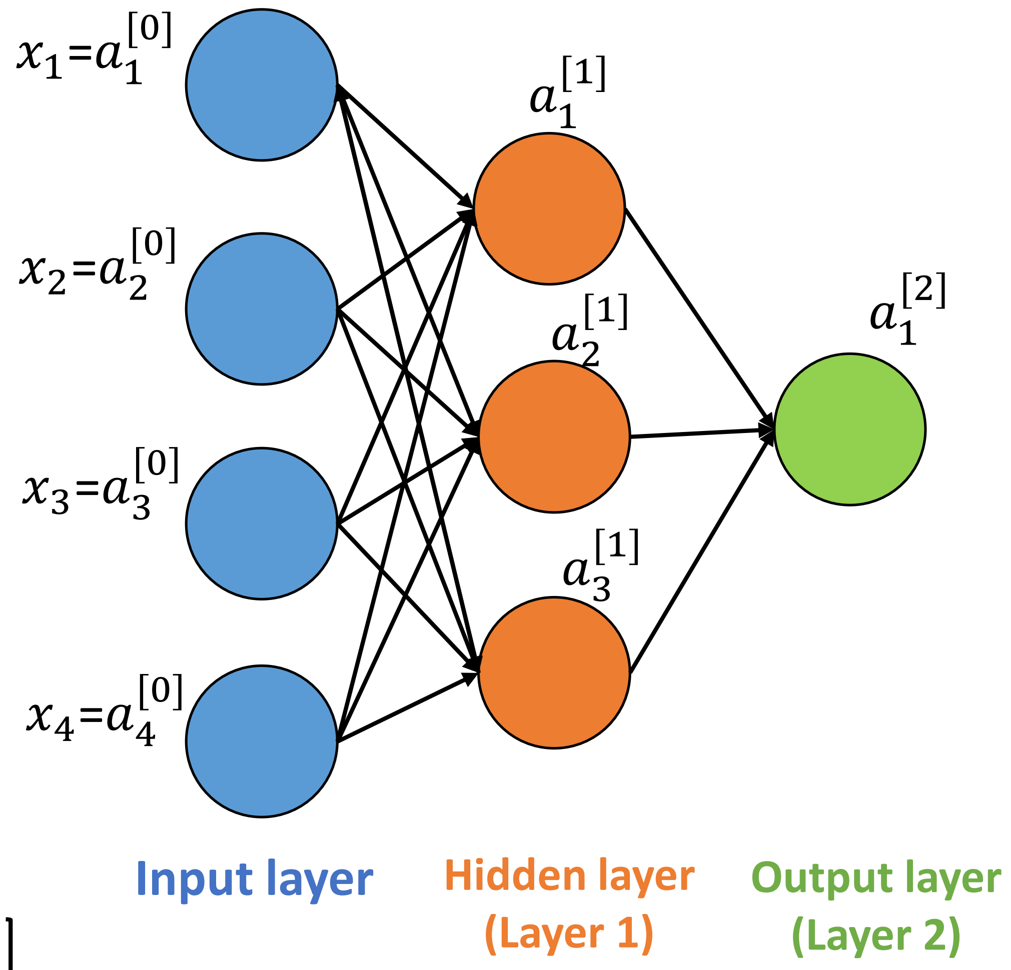
Summary on notations

Notation	Definition
n_x	Number of features or input samples (input size).
L	Number of layers in a neural network.
$n_h^{[l]}$	Number of hidden units of the l^{th} layer.
$x^{(i)} \in \mathbb{R}^{n_x}$	Column vector of the i^{th} example.
$x_j^{(i)}$	Scalar value of the j^{th} feature for example i^{th}.
$a_j^{[l]}$	j^{th} activation at layer l.
$y^{(i)} \in \mathbb{R}^{n_y}$	Target label for the i^{th} example.

Notation	Definition
$X \in \mathbb{R}^{n_x \times m}$	Input matrix i.e. matrix with input features n_x for all examples m.
$Y \in \mathbb{R}^{n_y \times m}$	Target matrix i.e. matrix with targets n_y for all examples m.
$W^{[l]} \in \mathbb{R}^{n_h^{[l]} \times n_h^{[l-1]}}$	Weight matrix for layer l.
$b^{[l]} \in \mathbb{R}^{[l]}$	Bias vector for layer l.

Model parameters we want to learn.

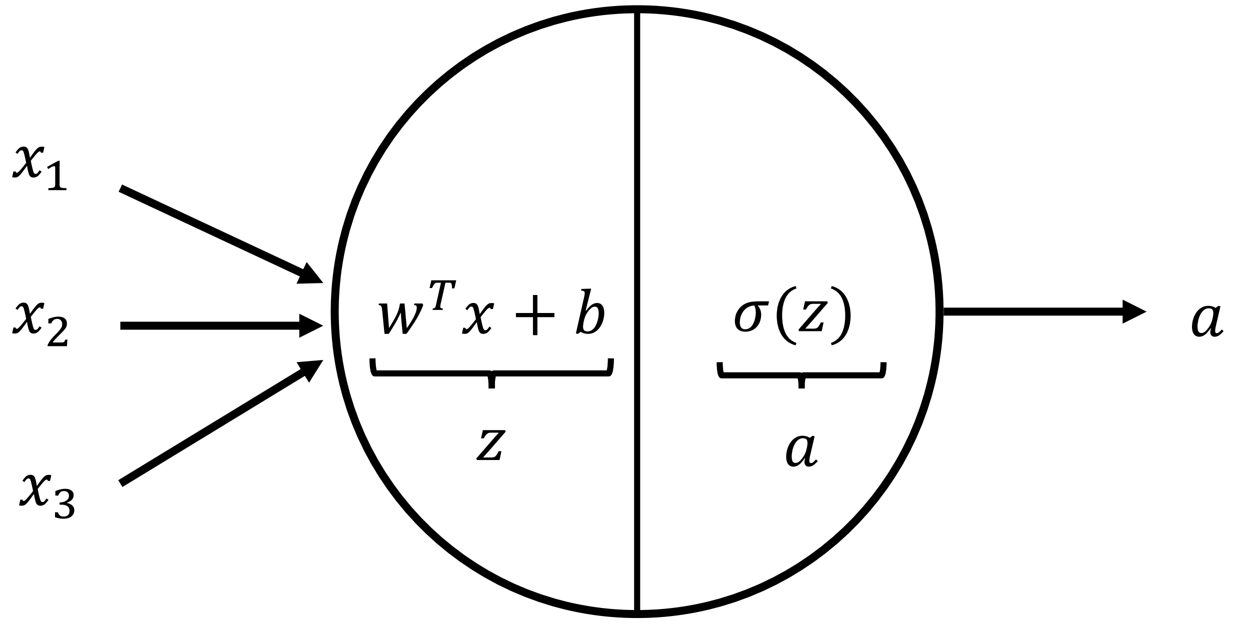
For a given example ***i***



Activation functions

Definition

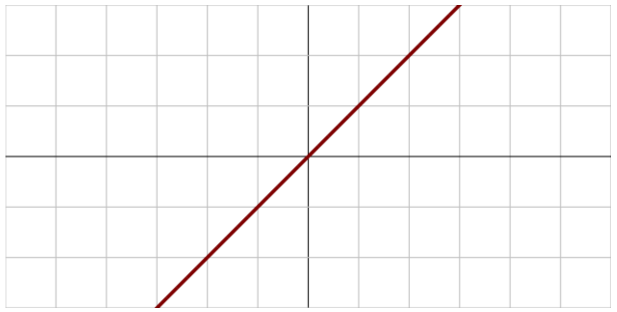
- The activation function transforms the weighted sum of the inputs into the activation of the node.
- Up to now we used the sigmoid function that we denoted σ.
- We will explore different types of these functions and the intuition behind them.

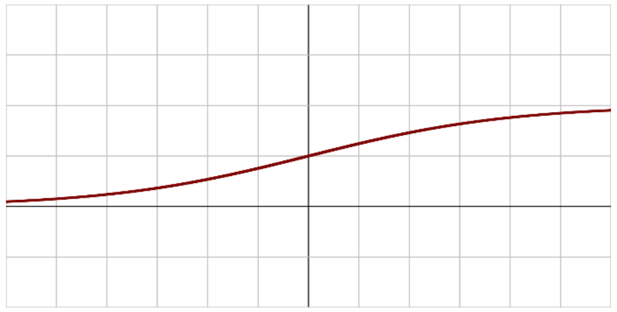


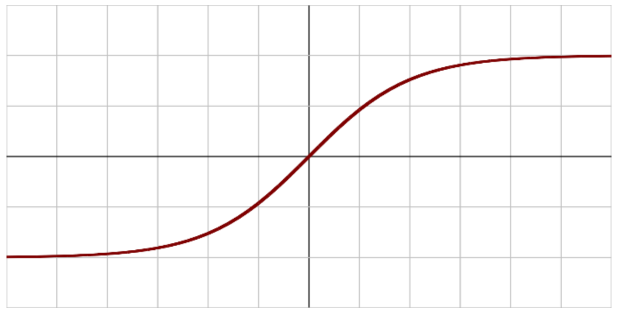
Single neuron.

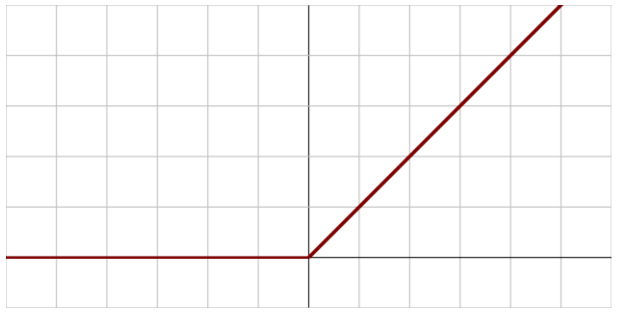
Activation functions

- Linear activation function: cannot learn complex mapping functions. However, it is still used for the output layer in regression problems.
- Nonlinear activation functions: can learn complex mapping functions:
  - Sigmoid: σ [used before the 90's]
  - Hyperbolic tangent: $tanh$ [used in the 90's]
  - Rectified linear unit: $ReLU$ [extensively used today.]
  - And many other flavors! A bit like the Fourier Window functions!


Linear


Logistic regression


Hyperbolic tangent


Rectified linear unit

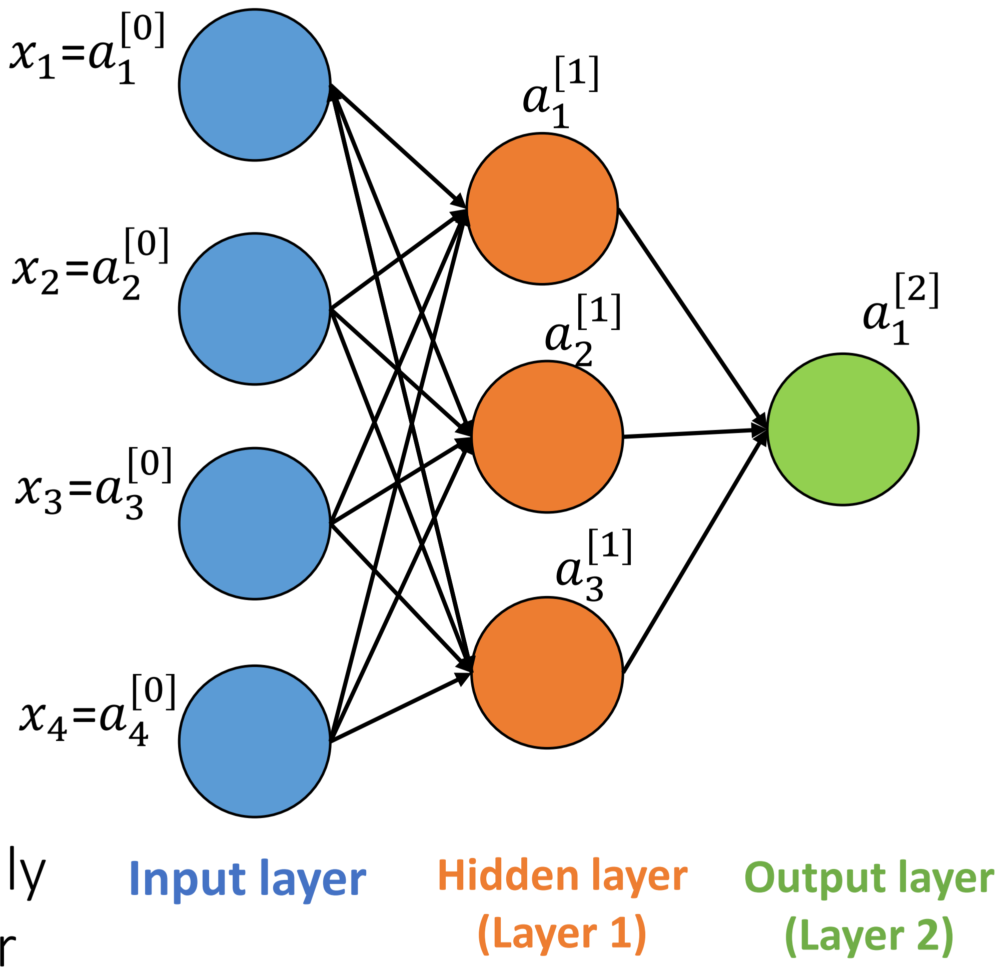
Images: By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920600

Activation functions: why nonlinear?

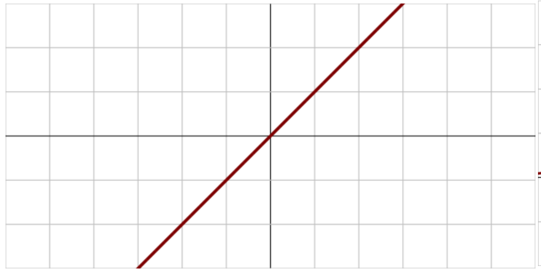
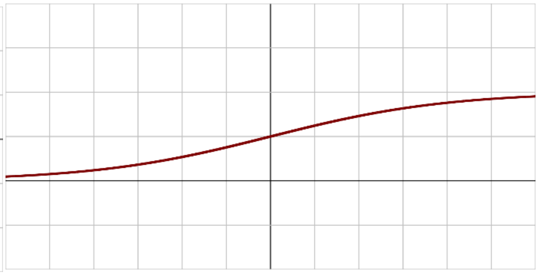
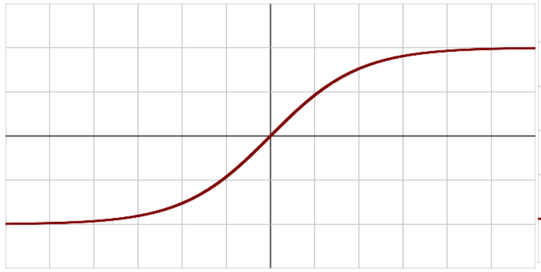
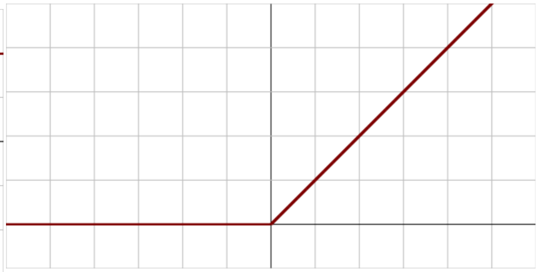
- Matrix formulation for a single example:
  - $z^{[1]} = W^{[1]T}\mathrm{x} + b^{[1]}, a^{[1]} = g\left(z^{[1]}\right),$
  - $z^{[2]} = W^{[2]T}\mathrm{x} + b^{[2]}, a^{[2]} = g\left(z^{[2]}\right).$
- If we assume $g = I$ i.e. a linear activation:
  - $a^{[1]} = z^{[1]} = W^{[1]}\mathrm{x} + b^{[1]}$
  - $a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
  - $a^{[2]} = W^{[2]}(W^{[1]}\mathrm{x} + b^{[1]}) + b^{[2]}$

$$= \underbrace{W^{[2]}W^{[1]}}_{\widetilde{W}}\mathrm{x} + \underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{\tilde{b}}$$

- Thus if the activation function is linear we can only learn a linear mapping function. This is not better than a simple LR model.



Derivatives of the activation functions

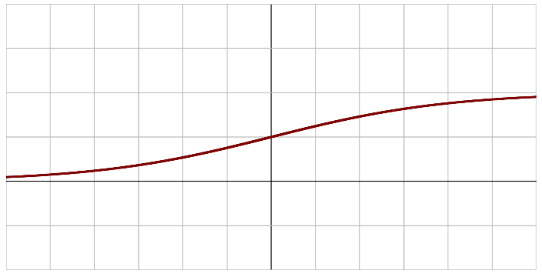





▪ Name	*Linear*	*Logistic regression*	*Hyperbolic tangent*	*Rectified linear unit*
▪ Equation	$a = g(z) = z$	$a = g(z) = \frac{1}{1+e^{-z}}$	$a = g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$a = g(z) = \max(0, z)$
▪ Derivative	$g'(z) = 1$	$g'(z) = a(1-a)$	$g'(z) = 1 - a^2$	$g'(z) = \begin{cases} 1\ if\ z < 0 \\ 0, if\ z > 0 \\ undef, if\ z = 0 \end{cases}$

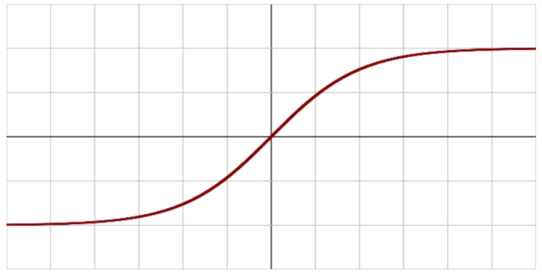
Images: By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920600

Activation functions

- The $tanh$ is better suited than σ because it has the effect of centering the data at zero and this makes learning easier for the next layer.



Logistic regression

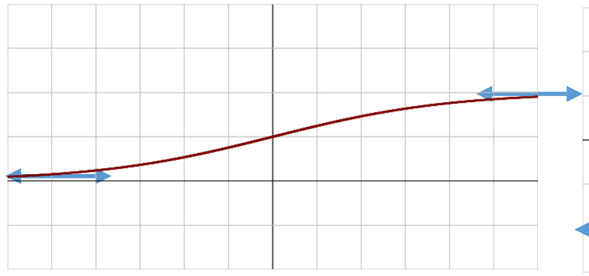


Hyperbolic tangent

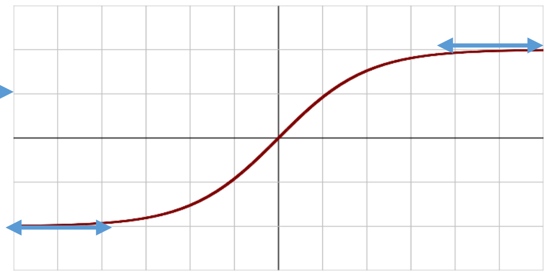
Images: By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920600

Activation functions

- A limit of $tanh$ and σ is that they **saturate** for very large value → 1 and for very small values → 0/1 (σ / $tanh$).
- When saturation happens, the learning algorithm has difficulties to continue learning because the derivative of the activation function will be zero.
- With the computational possibilities opened by GPU's hardware usage, people have been designing deeper and deeper networks and this shortcoming of $tanh$ and σ became an actual important limitation.



Logistic regression



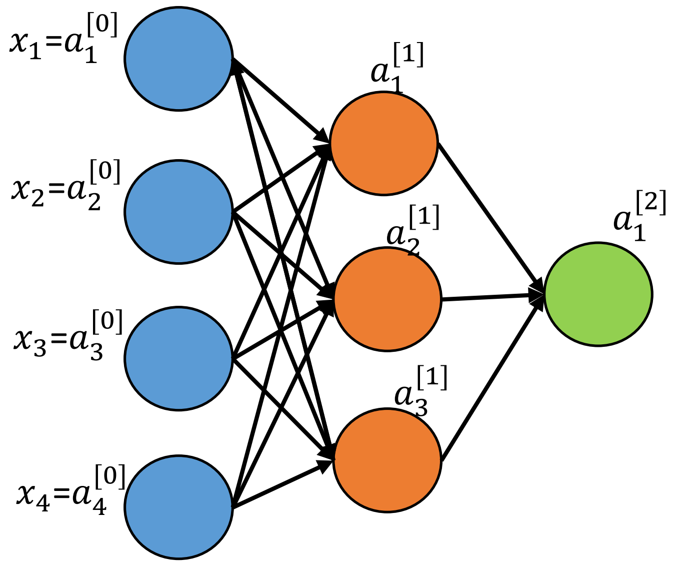
Hyperbolic tangent

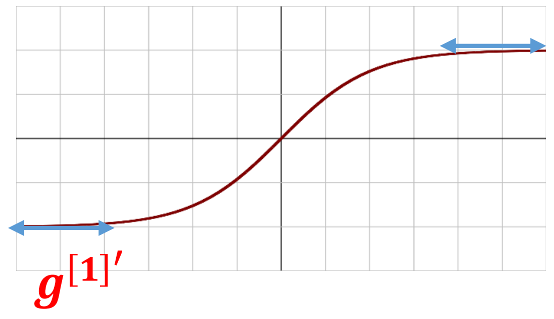
↔ *Derivative tends to zero.*

Images: By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920600

Activation functions

- Reminder backpropagation:
  - Cost function:
    - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
  - Gradient descent:
    - $W^{[2]} := W^{[2]} - \alpha \frac{\partial J}{\partial W^{[2]}};\ b^{[2]} := b^{[2]} - \alpha \frac{\partial J}{\partial b^{[2]}}$
    - $W^{[1]} := W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}};\ b^{[1]} := b^{[1]} - \alpha \frac{\partial J}{\partial b^{[1]}}$
  - Derivatives:
    - $\frac{\partial J}{\partial W^{[2]}} = \frac{1}{n}(A^{[2]} - Y)A^{[1]T}$
    - $\frac{\partial J}{\partial W^{[1]}} = \frac{1}{n}W^{[2]T}\left(A^{[2]} - Y\right) * \boldsymbol{g}^{[1]'}(Z^{[1]})$
- So when we enter the saturation regime the derivative tends to zero and the weights stop being updated in the backpropagation algorithm.



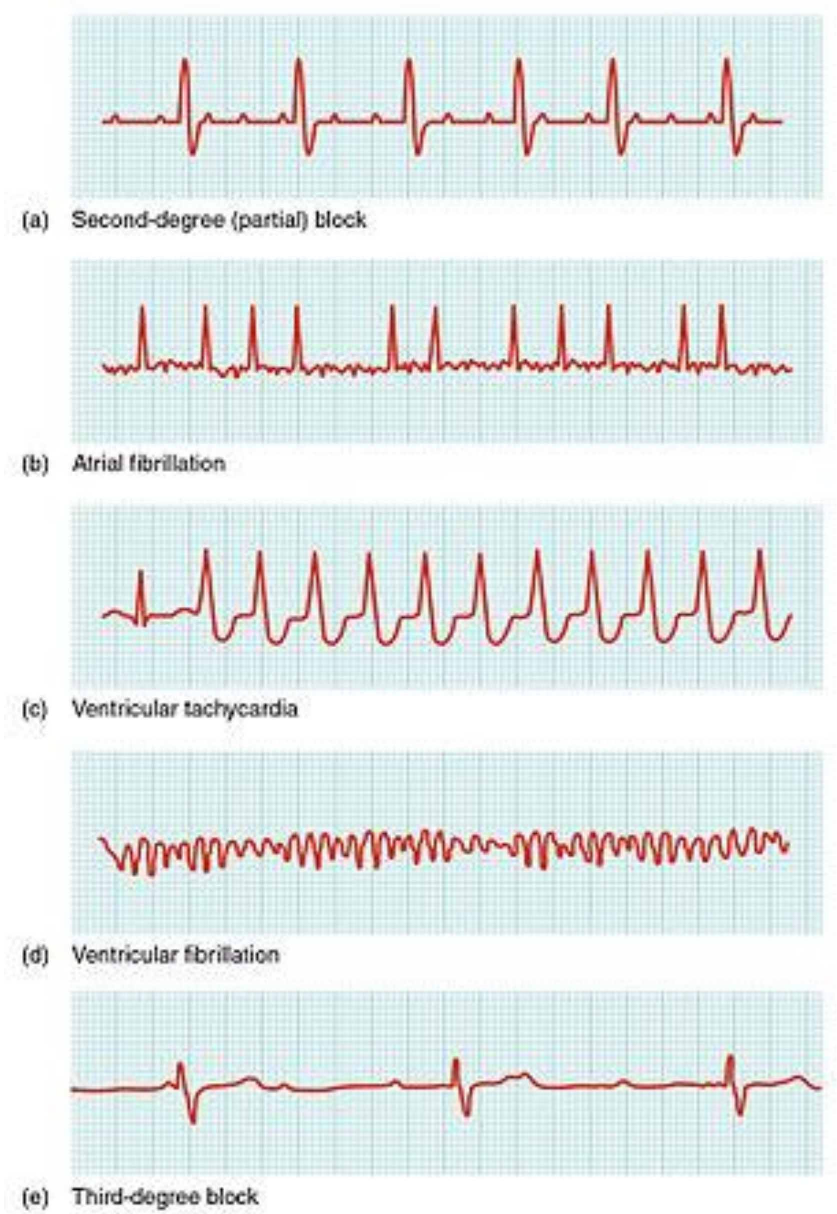


Hyperbolic tangent

Multiclass classification

Multiclass classification

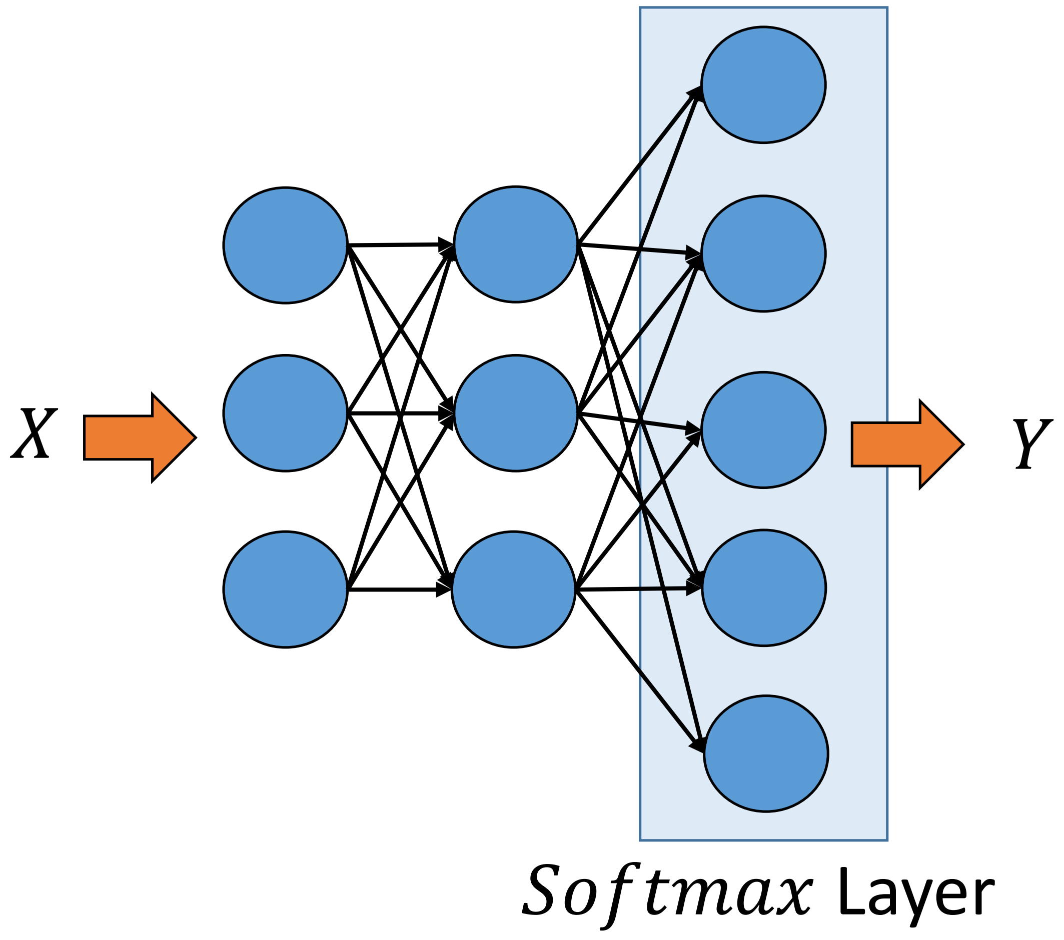
- What if we want to recognize multiple classes?
- We use a “softmax layer” for the output layer. It consists of a layer with a softmax activation function:
  - The usual: $z^{[L]} = w^{[L]}a^{[L-1]} + b^{[L]}$
  - Now activation function: $a^{[L]} = softmax\left(z^{[L]}\right)$
  - $a^{[L]} = e^{z^{[L]}} / \sum_{i=1}^{K} e^{z_i^{[L]}}$
- The softmax is also called normalized exponential function. It is a function that takes an input vector of size K and normalizes it into K probability distribution proportional to the exponentials of the input numbers.

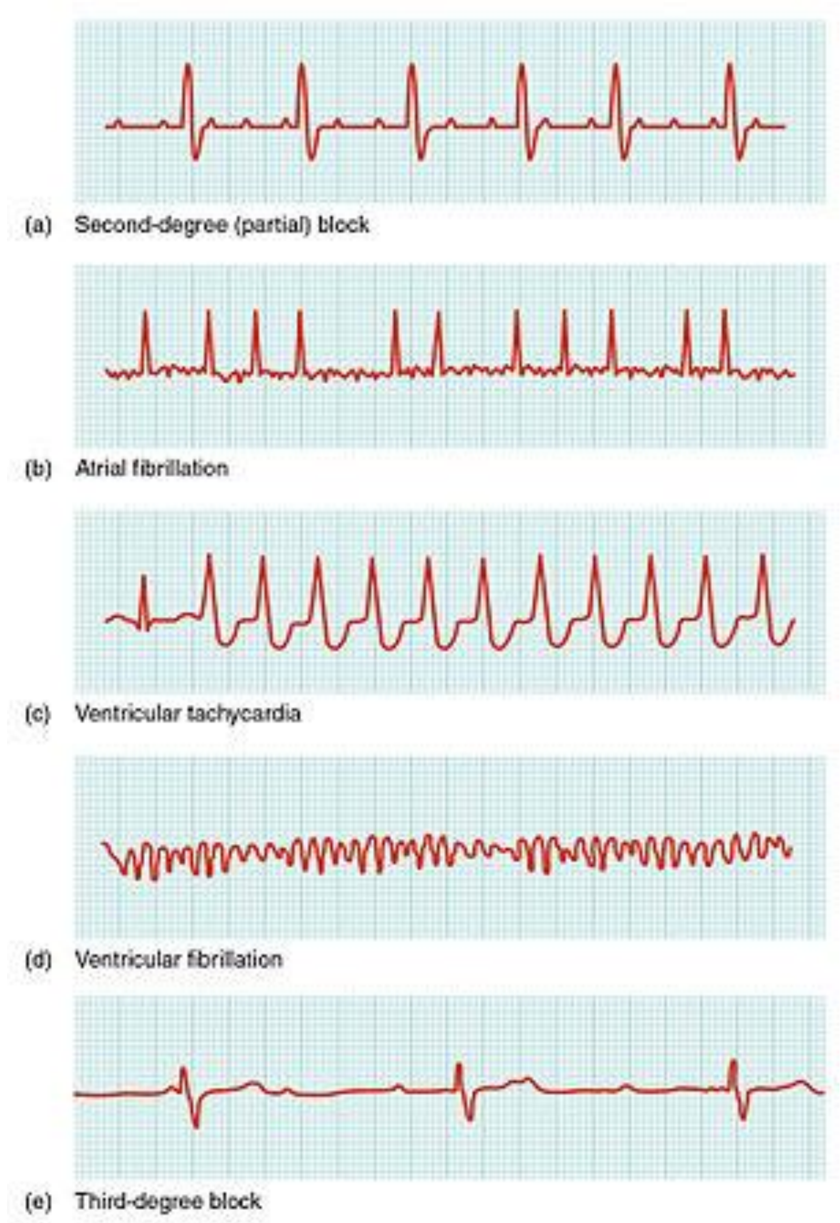

(a) Second-degree (partial) block
(b) Atrial fibrillation
(c) Ventricular tachycardia
(d) Ventricular fibrillation
(e) Third-degree block

https://simple.wikipedia.org/wiki/Cardiac_arrhythmia

Multiclass classification

- Example for arrhythmia classification:

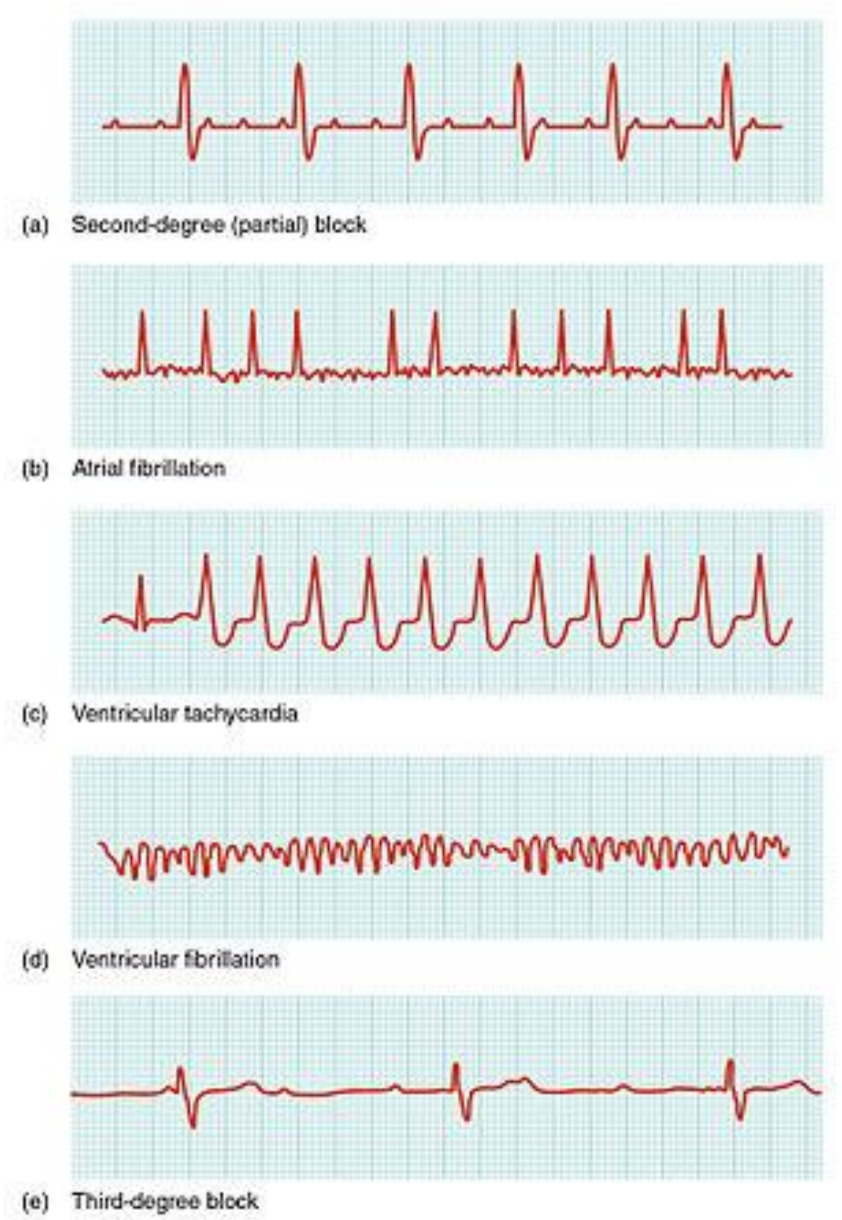



(a) Second-degree (partial) block
(b) Atrial fibrillation
(c) Ventricular tachycardia
(d) Ventricular fibrillation
(e) Third-degree block

https://simple.wikipedia.org/wiki/Cardiac_arrhythmia

Multiclass classification

- We use it for representing a categorical distribution that is, a probability distribution over K different possible outcomes.
- What’s the difference between the softmax and other activation function we have seen before?
  - It takes a vector as an input and returns a vector output whereas the sigmoid function (or others we had seen thus far) take a real number and outputs a real number.
  - It takes a vector as the input because it normalizes by the sum of the exponentials.


(a) Second-degree (partial) block
(b) Atrial fibrillation
(c) Ventricular tachycardia
(d) Ventricular fibrillation
(e) Third-degree block

https://simple.wikipedia.org/wiki/Cardiac_arrhythmia

Take home

- Representation learning:
  - Build complex concepts out of simpler ones.
  - In NN, the NN learns increasingly more complex features as we get deeper and deeper in the layers.
- Forward and backward propagation.
- Activation functions:
  - These day $ReLU$ is mostly used.
  - σ is often used for the output layer if we want to have an output in [0 - 1] range.
  - Use a linear activation function for the output layer for a regression problem.
  - The reason we use a non-linear activation function is because it is what enables us to learn complex mapping functions. If all activation functions are linear then the NN is a linear classifier equivalent to a simple LR model.
  - The $Softmax$ activation function is used for multiclass classification.

References

[1] Andrew Ng, Coursera, Neural Networks and Deep Learning. Coursera.

[2] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[3] Machine Learning Mastery.

A Gentle Introduction to the Rectified Linear Unit (ReLU)

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

How to Configure the Number of Layers and Nodes in a Neural Network

https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[4] Andrew Ng and Kian Katanforoosh. CS229 Lecture Notes on Deep Learning.

http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf

[5] Model, A. Single Neuron. "Lecture 4-Neural Networks.“ Lecture notes.