Deep Learning Techniques for Music Generation (2)

Jean-Pierre Briot

Jean-Pierre.Briot@lip6.fr

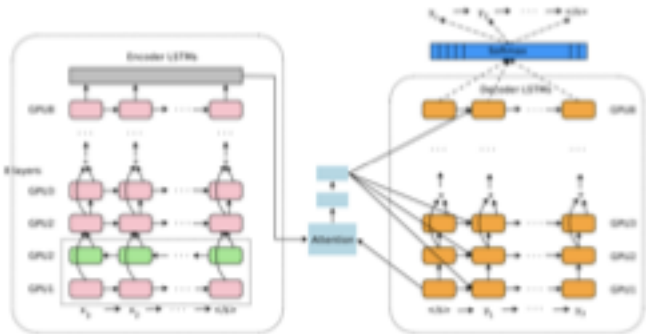
Laboratoire d'Informatique de Paris 6 (LIP6)
Sorbonne Université – CNRS

Programma de Pós-Graduação em Informática (PPGI)
UNIRIO

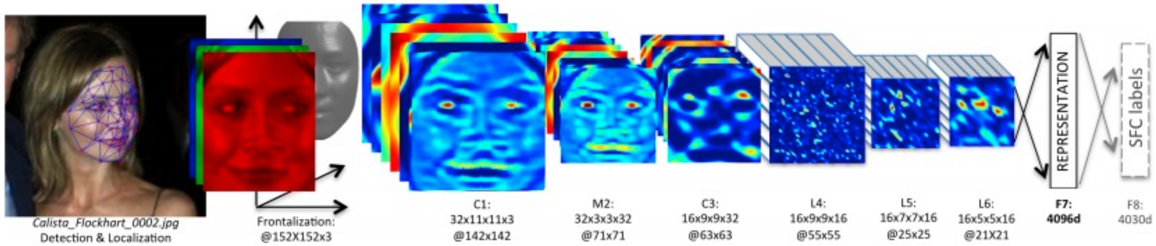
Deep Learning

Deep Learning – Already There

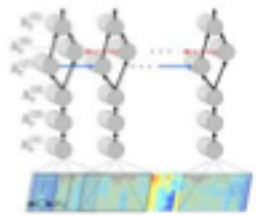
- Translation, ex: Google Translate



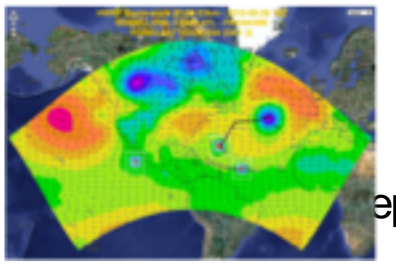
- Image Recognition, ex: Facebook



- Speech Recognition, ex: Apple Siri, Amazon Echo…

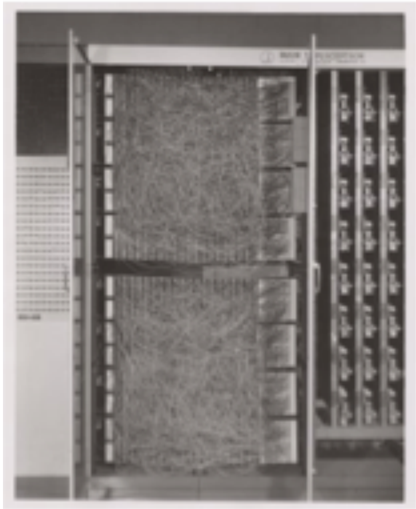


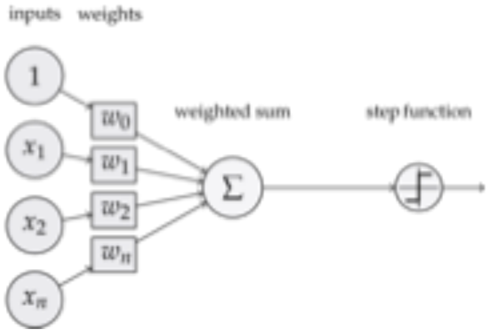
- Weather Prediction



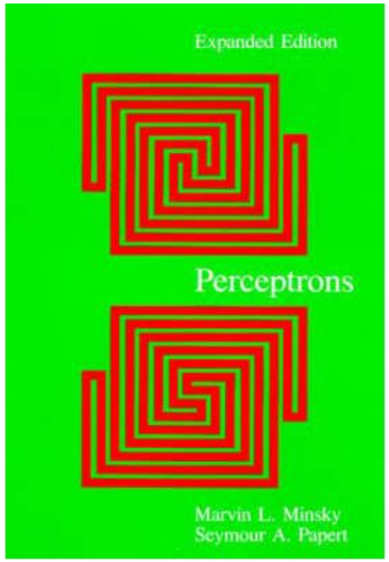
History

History: From Perceptron to Artificial Neural Networks to Deep Learning (1/4)



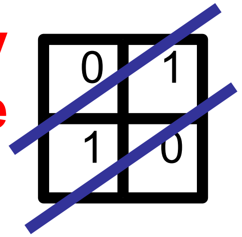


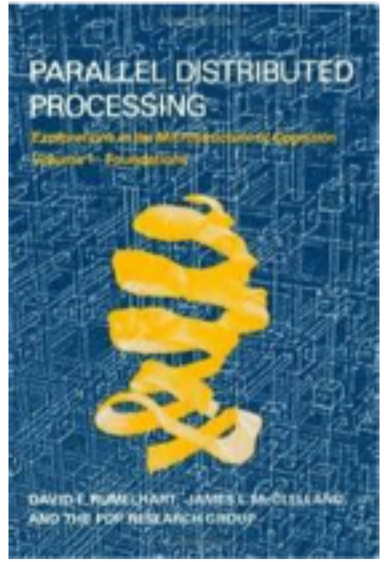
Perceptron [Rosenblatt 1957]



Perceptrons (Book) [Minsky & Papert 1969]

Linear Separable only
XOR counter example



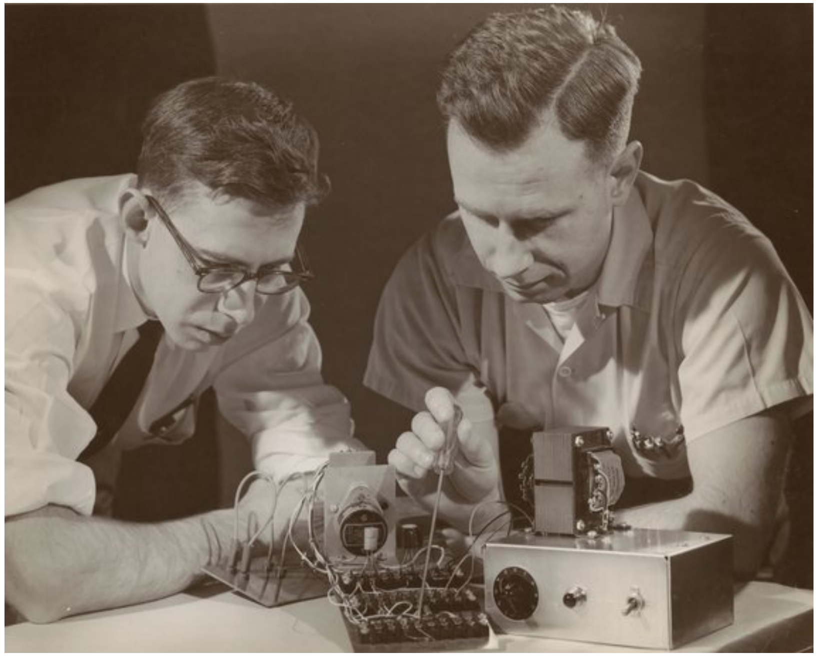


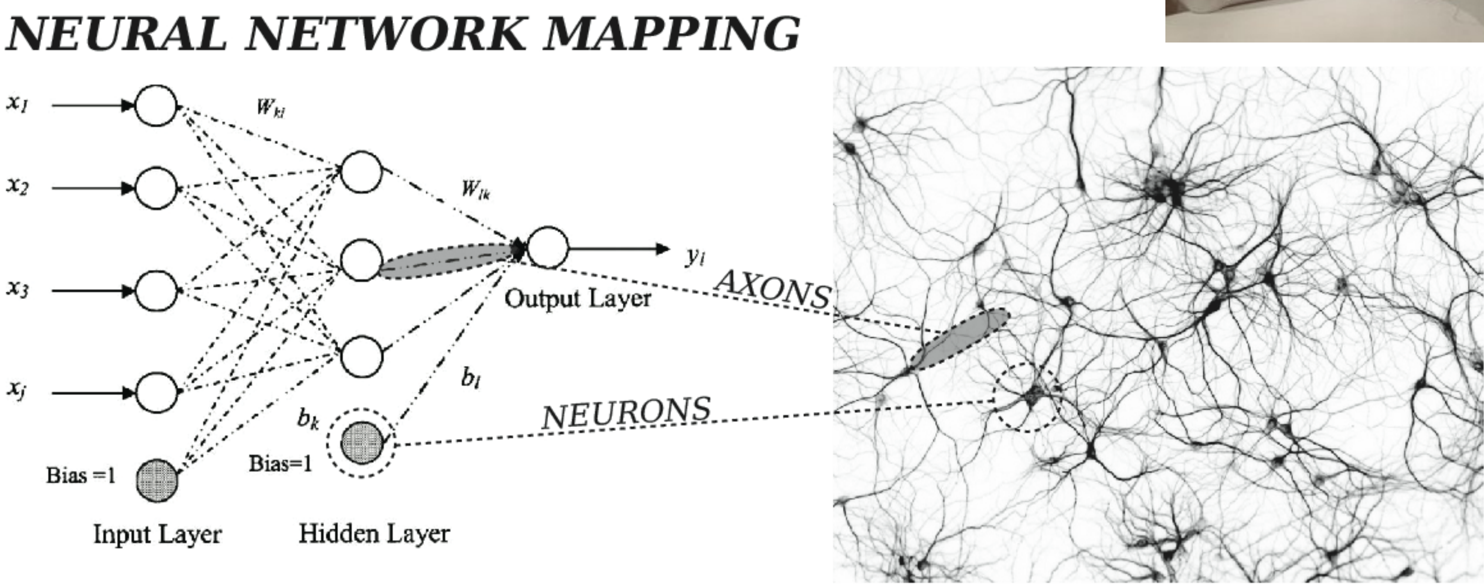
PDP (Books) [Rumelhart et al. 1986]

Multi-layer networks
Backpropagation

Deep Learning Bio-inspired or/and Regression-based ?

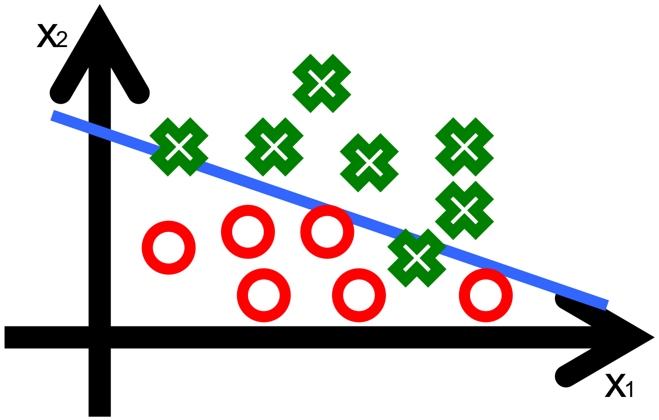
- Historically Perceptron bioinspired
- Aimed at Character Recognition



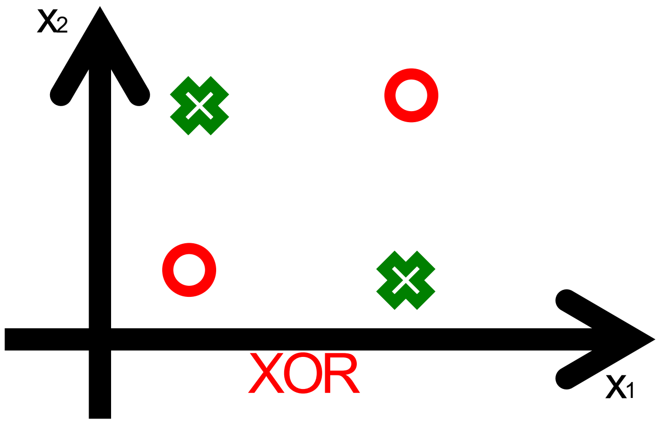


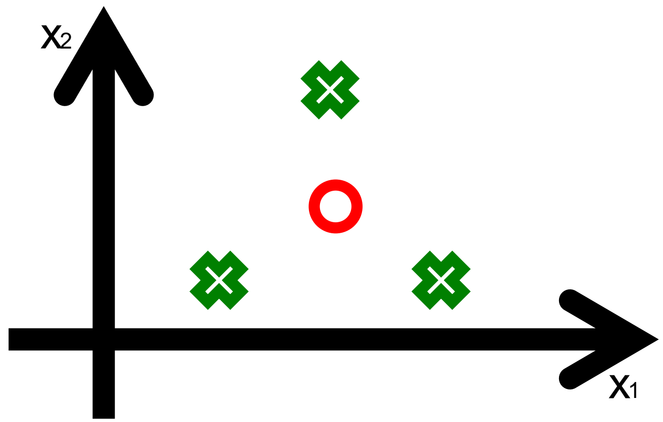
Linear vs Non Linear Decision Boundary

- Linear



- Non Linear

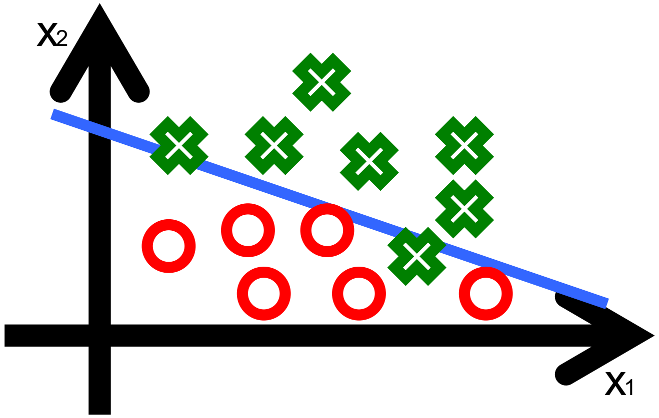




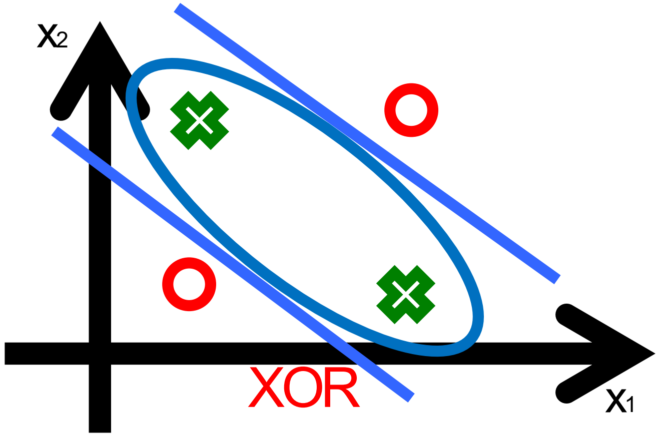
- Argument (XOR) used by [Minsky & Papert 1969] to criticize Perceptrons [Rosenblatt 1957] (and advocate Symbolic Artificial Intelligence)
- This stopped research on Perceptrons/Neural Networks for a long while
  - until Hidden Layers and Backpropagation or/and Kernel Trick (see later)

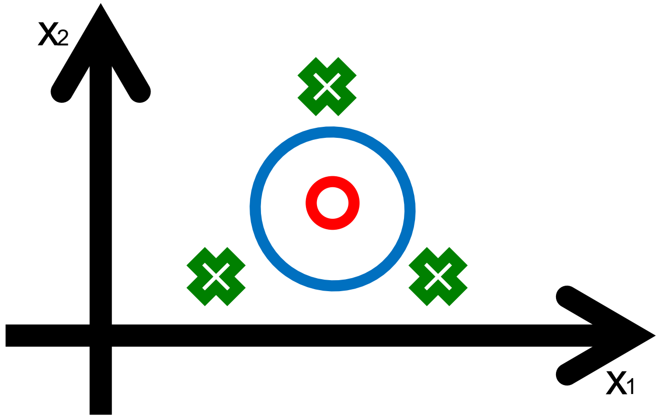
Linear vs Non Linear Decision Boundary

- Linear



- Non Linear

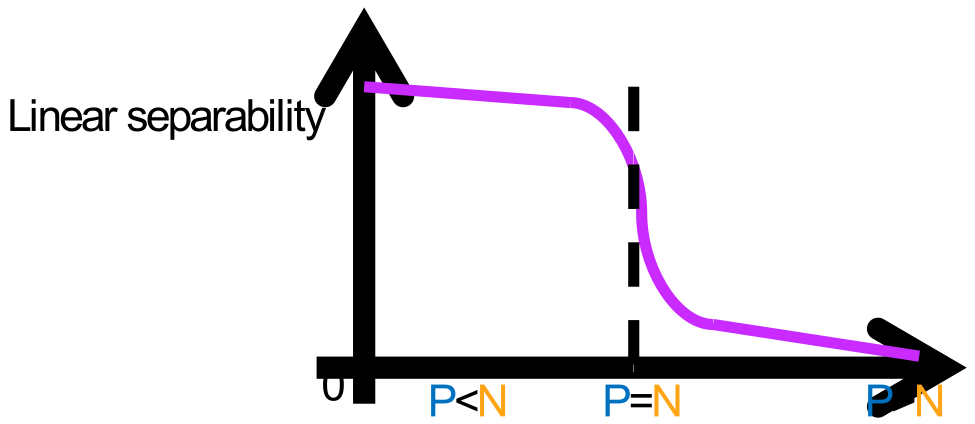




- Argument (XOR) used by [Minsky & Papert 1969] to criticize Perceptrons [Rosenblatt 1957] (and advocate Symbolic Artificial Intelligence)
- This stopped research on Perceptrons/Neural Networks for a long while
  - until Hidden Layers and Backpropagation or/and Kernel Trick (see later)

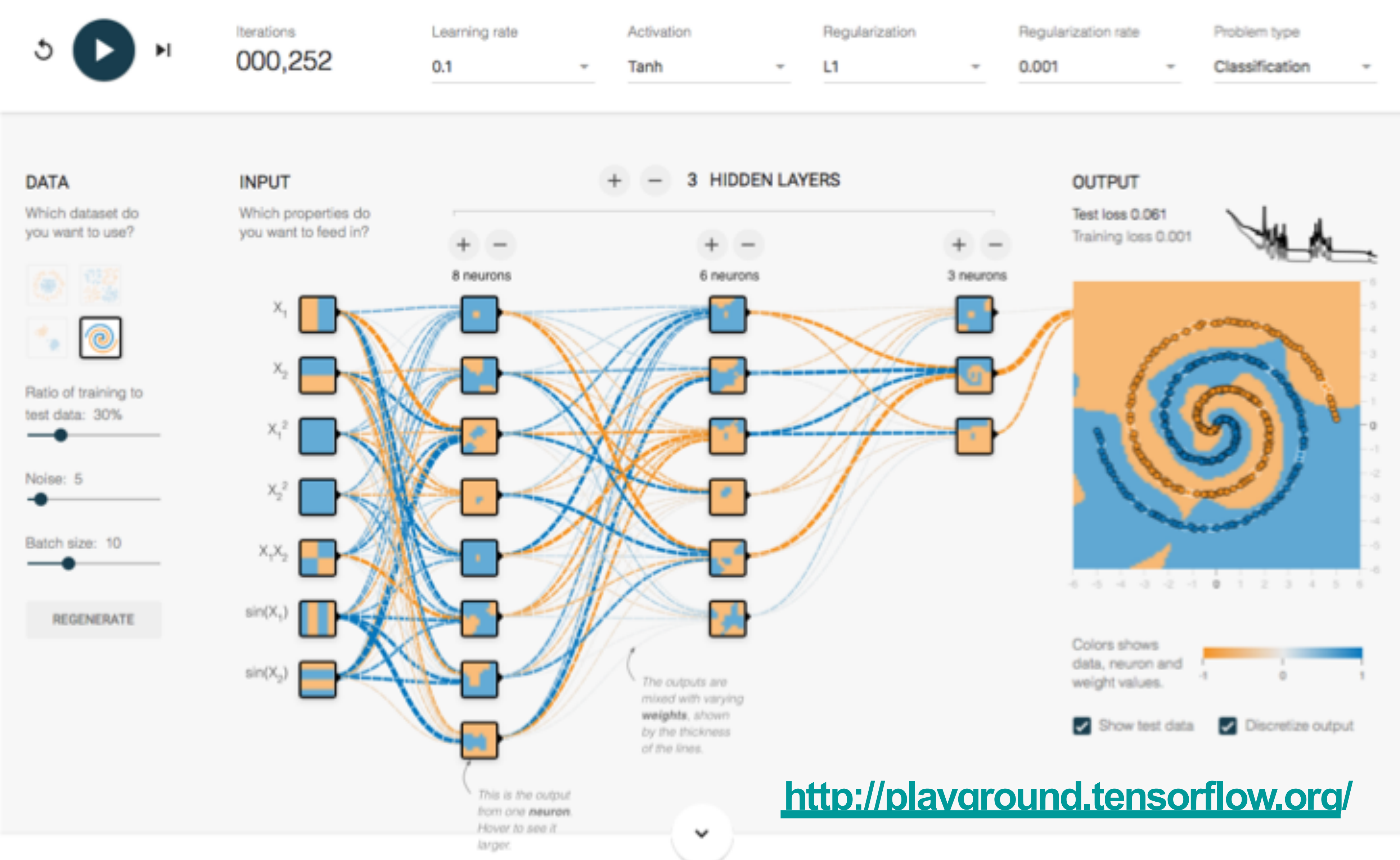
Linear vs Non Linear Decision Boundary

- A consequence of Cover's Theorem [Cover, 1966]
  - The probability that P samples (examples) of dimension N (number of parameters) are linearly separable goes to zero very quickly as P grows larger than N



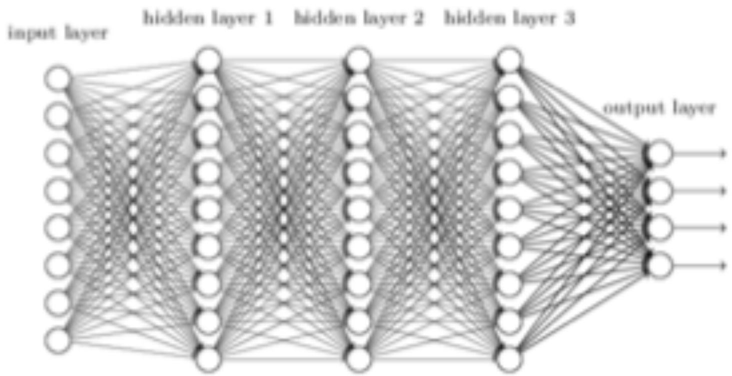
[LeCun 2016]

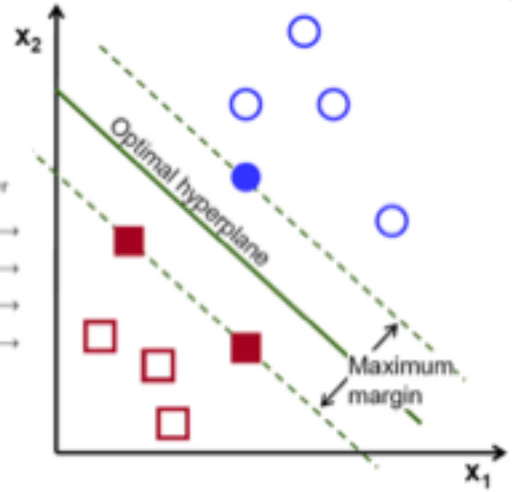
Example (TensorFlow PlayGround)

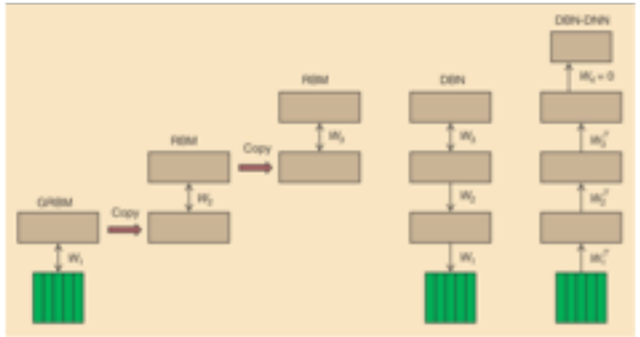


http://playground.tensorflow.org/

History: From Perceptron to Artificial Neural Networks to Deep Learning (2/4)







Pre-Training [Hinton et al. 2006]
Layer-Wise Self-Supervised Training/Initialization

Difficulty to Efficiently Train Networks with many Layers

Unstable Gradients

SVM [Vapnik 1963]
SVM + Kernel Trick [Vapnik et al. 1992]

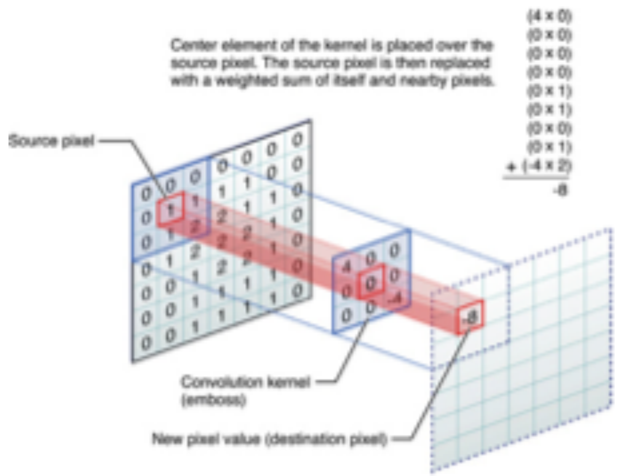
Nice Model and Optimized Implementation

Margin Optimization

Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

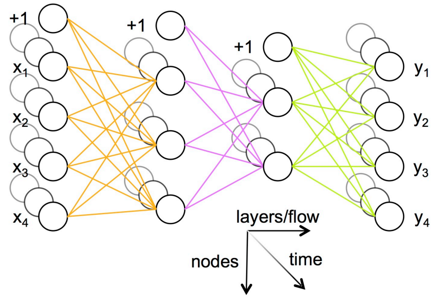
ImageNet 2012 Image Recognition Challenge Breakthrough

History: From Perceptron to Artificial Neural Networks to Deep Learning (3/4)

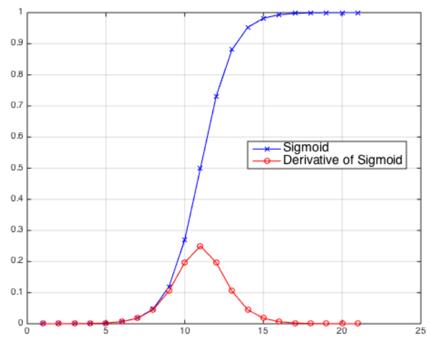


Convolutional Networks
[Le Cun et al. 1998]

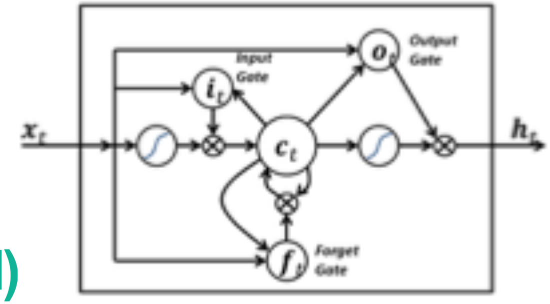
Equivariance (to translation)
& Invariance (to small transformations)



Recurrent Neural Networks (RNN)
(1986)
Temporal Invariance



Gradient Vanishing
or Explosion (1991)



Long Short-Term Memory
(LSTM)
[Hochreiter &
Schmidhüber 1997]

History: From Perceptron to Artificial Neural Networks to Deep Learning (4/4)

+

+

Massive Data Available

Efficient Implementation Platforms

Affordable Efficient Parallel Processing (Graphic Cards)

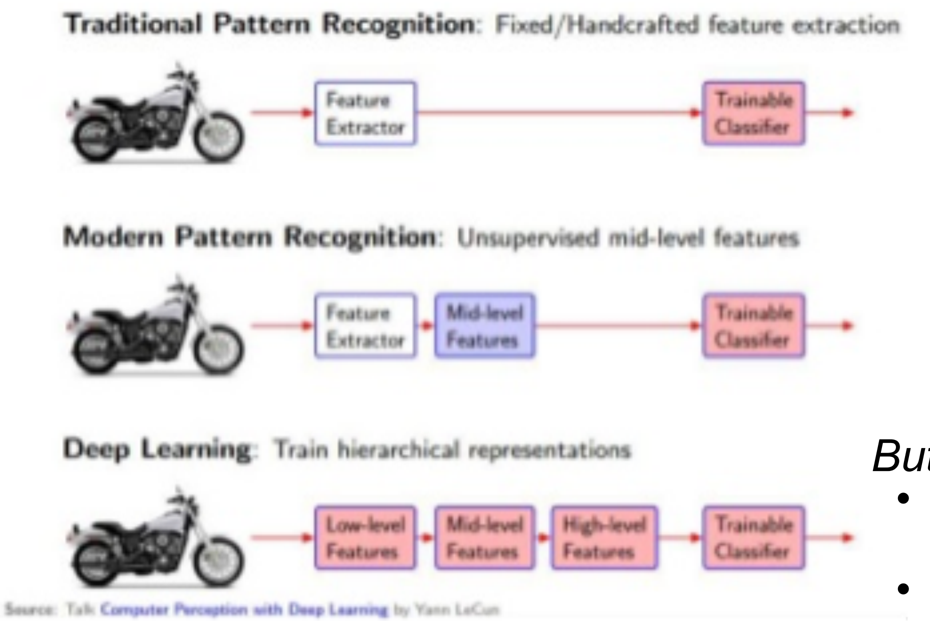
What and Why

Deep Learning

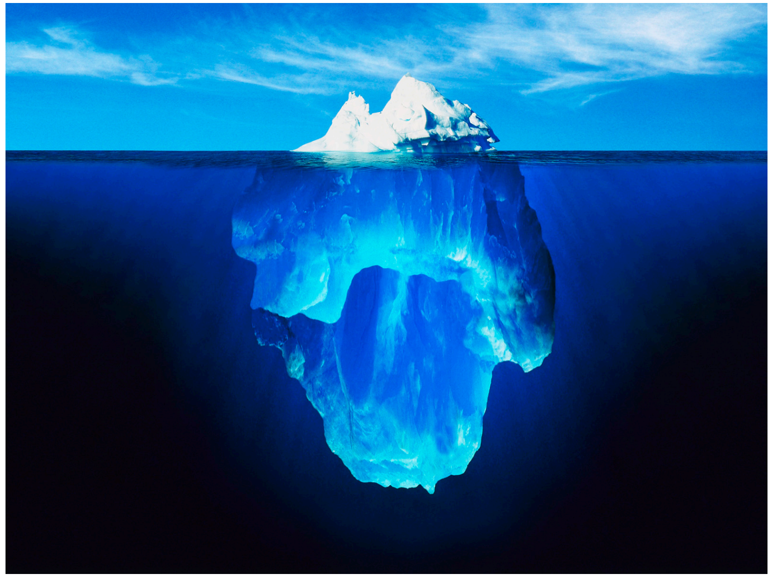
- Many Hiddden Layers
  - Deep Architecture
  - Deep Representation
- Various Types
  - Supervised learning
  - Unsupervised learning
  - Recurrent networks
  - Stochastic Neurons (ex: Restricted Boltzmann Machines)
- Various Objectives
  - *Classification*
  - *Prediction*
  - Feature extraction
  - Generation
    - Related Content (ex: Melody Accompaniment)
    - New Content (ex: new Melody)

Why Deep ?

- More Complex Models
- Learns better Complex Functions
- Hierarchical Features/Abstractions
- No Need for Handcrafted Features
  - (Automatically Extracted)



End-to-End Architecture

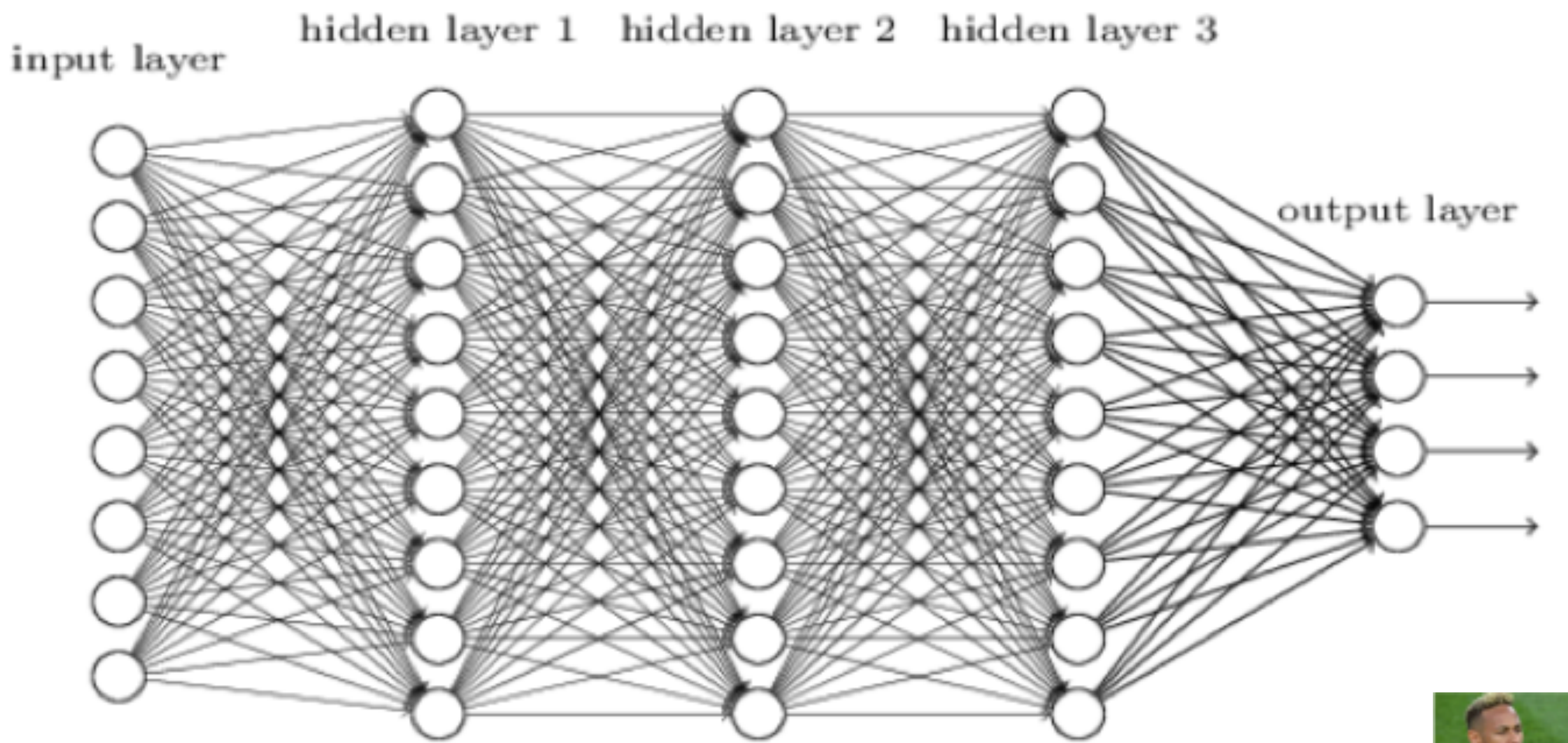


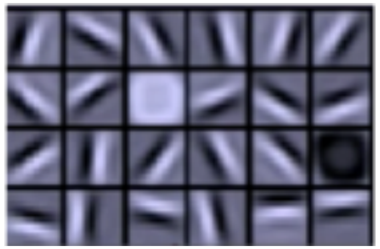
Distributed Representations

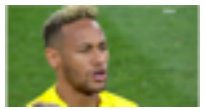
But the choice of the input representation is important

- Ex: for Audio:
  - Waveform or Spectrum
- Ex: For Symbolic representation
  - Additional information, ex: beat, fermata, enharmony (Cb =/= B)

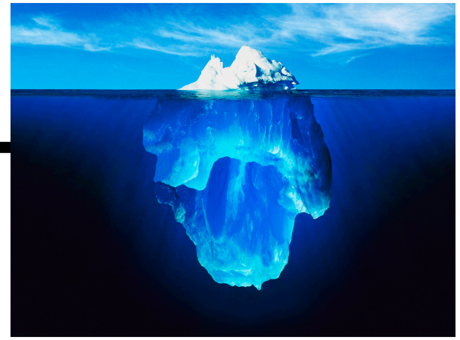
Successive Features/Abstractions Constructed







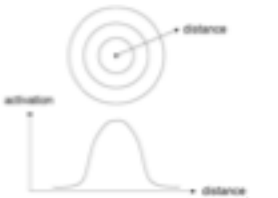
Why Deep ?

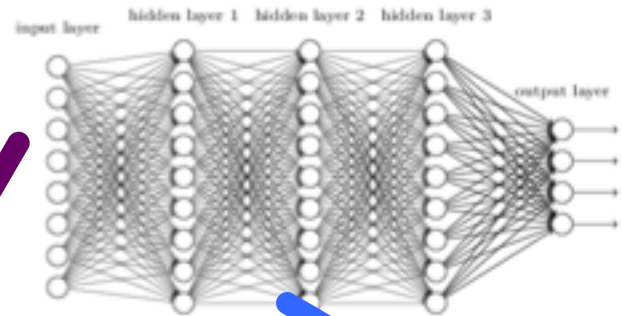


- Need for Deep ?! [Ba & Caruana 2014]
  - Train a Shallow Net to Mimic a Deep Net
  - Mimic = Train on the Logits (predicted values, before softmax/probabilities), not on the Original Output Data
  - Equivalent Accuracy
  - But for Convolutional Nets, Mimic Net Does Not Match the Initial Accuracy ! [Urban et al.& Caruna 2016]

- Counter Argument (Theorem) [Eldan & Shamir 2016]
  - There is a simple radial function on R^d, expressible by a 3-layer net, but which cannot be approximated by any 2-layer net to more than a constant accuracy unless its width is exponential on the dimension d
  - Depth vs/and Width

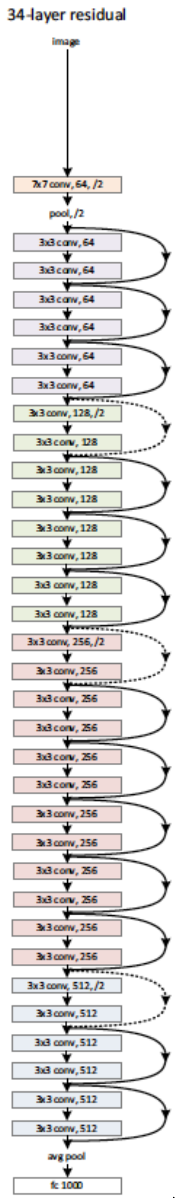
Radial function = Function whose value at each point depends only on distance between point and origin





Very Deep Learning

GoogLeNet (2014)

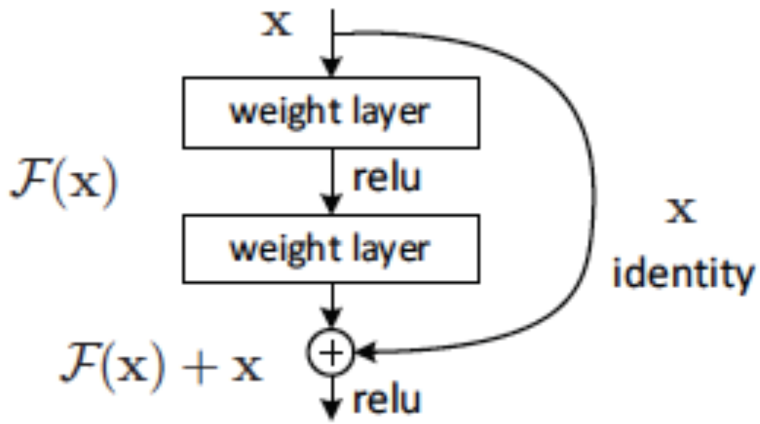


ResNet (2015)

(Very) Deep Networks

Upto 152 Layers !

New Techniques (Tricks ?!) e.g.,
Deep Residual Learning [He et al., 2015]
Replaced Pre-Training (less in vogue)



Linear Regression and Neural Networks

Machine Learning

- Why learning ?

- What is machine learning ?

- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [Mitchell, 1997]

Machine Learning Typology

- Learning = Using Experience/Memory

  to Infer Information/Behavior

  and to Improve Decision/Action

- Dynamicity
  - Off-Line (Training Phase vs Using/Production Phase)
    - » Ex: Supervised Learning
      - But also Incremental versions
  - On-Line/Incremental
    - » Ex: Reinforcement Learning
      - But also Off-line versions (ex: Deep Reinforcement Learning Replay Mechanism)

- Provided Information
  - Supervised (Correct Output)
  - Unsupervised (Ø)
  - Reward-Based (Quality assessment)

Machine Learning Typology (2)

- Data/Statistics-oriented
  - Supervised and Off-Line (mostly)
    - » Neural Networks
    - » Deep Networks
    - » Support Vector Machines
    - » Bayesian Networks
  - Unsupervised
    - » Clustering (ex: k-Means)
    - » Principal Component Analysis (PCA)
    - » Autoencoders
    - » Deep Networks

- Decision/Action-oriented
  - Fitness/Reward-based
    - » Evolutionary Algorithms
    - » Reinforcement Learning

- Knowledge/Rules-oriented
    - » Memory-based Reasoning and Case-Based Reasoning
    - » Inductive Logic Programming (ILP)

Machine Learning and Artificial Intelligence

- Machine Learning is Part of Artificial Intelligence Techniques

But also:

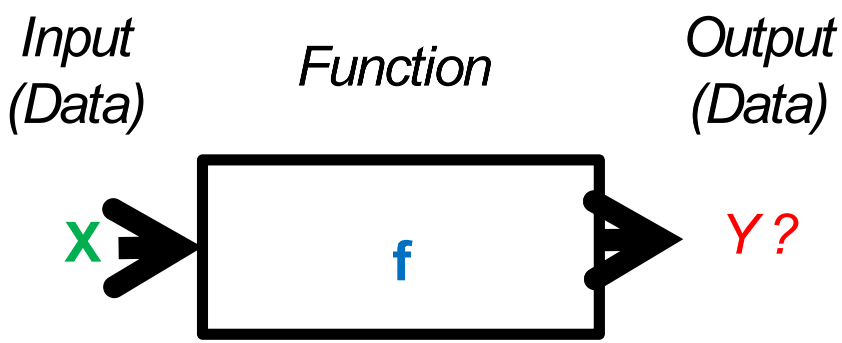
- Reasoning
- Planning
- Knowledge Representation
- User Modeling and Interaction
- Collaboration (Multi-Agent Systems)

- Natural Language Processing
- Dialogue
- Speech Processing

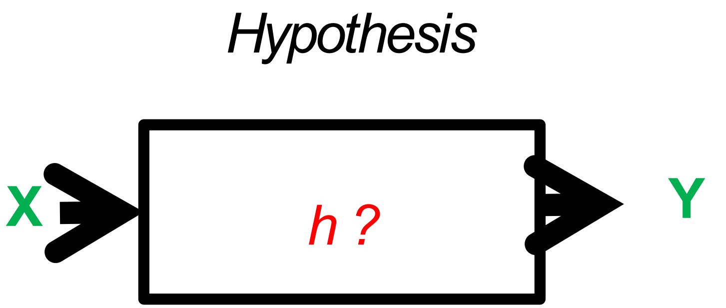
- Decision
- Game Theory
- Optimization
- Robotics

Machine Learning (Data Oriented/Supervised Learning)

- Traditional Programming



- Machine Learning (Supervised Learning)



Function/Module Configuration

(*Pre-Existing Representation Model*)

Machine Learning (Data Oriented)

- 3 Components [Domingos 2015]:

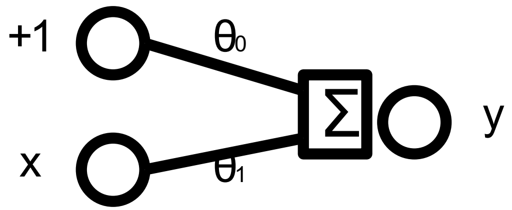
- Representation
  - To Model Predictors/Classifiers
  - e.g., Decision Tree, Rule, Neural Network, Graphical Model…

- Evaluation
  - of Predictors/Classifiers (Accuracy)
  - Cost, e.g., Squared Error, cross-entropy…
  - Precision, Recall

- Optimization
  - To Search among Predictors/Classifiers,
  - e.g., Batch Gradient Descent, Greedy Search, Stochastic Gradient…

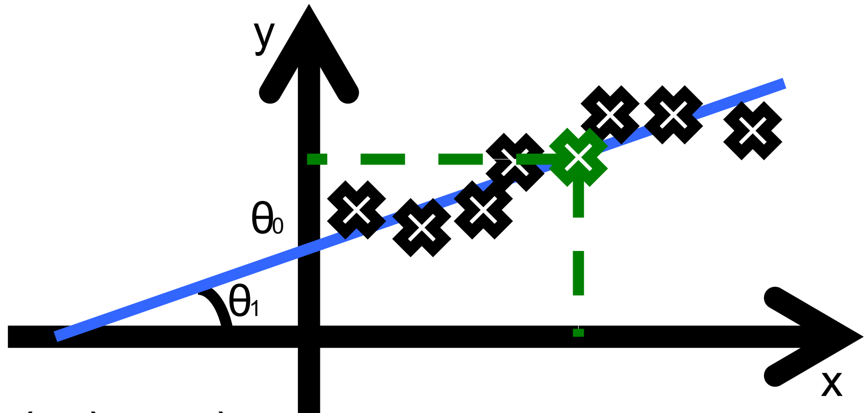
(Simple) Linear Regression (Prediction)

- Representation



Model (Hypothesis)

$$h(x) = \theta_0 + \theta_1 x$$



- Evaluation

Cost function : $J(\theta_0, \theta_1) = J_\theta = 1/2{*}m \; {}_{i=1}{}^{m}\Sigma \, (h(x^{(i)}) - y^{(i)})^2$

Distance between predicted and real

- Optimization (Training)

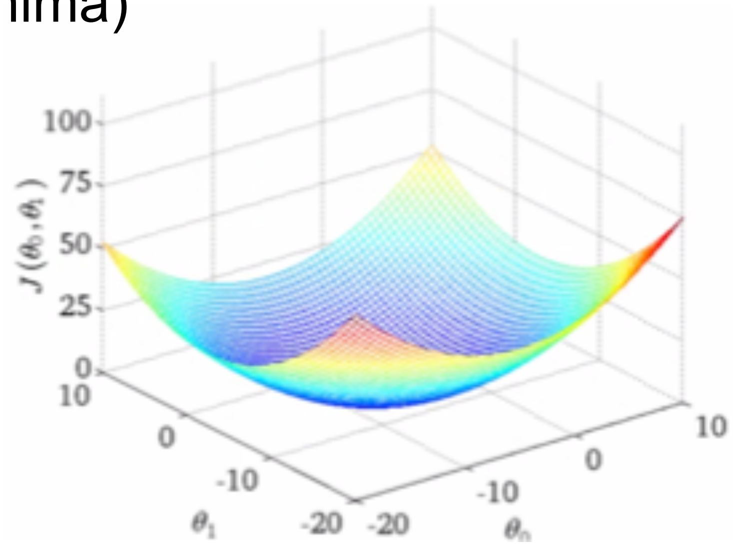
Objective: Find out "best" values of θ_0 and θ_1 to minimize Cost function J_θ

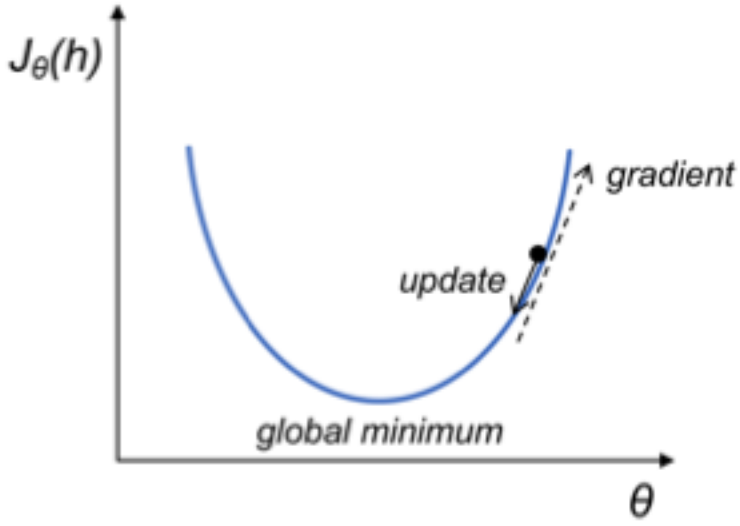
with:

	Example$^{(1)}$	Example$^{(2)}$		Example$^{(m)}$
Input Dataset : X :	$x^{(1)}$	$x^{(2)}$	…	$x^{(m)}$
and Output Dataset : y :	$y^{(1)}$	$y^{(2)}$	…	$y^{(m)}$

Optimization

- Objective: Minimization of Cost function J_θ
- Usual/simple algorithm: Gradient descent
- Therefore, we need to compute the Cost, but also the Gradients (partial derivatives respective to each θ (weight)) in order to guide gradient descent
- As this Cost function is *convex*, there is a global minimum (and no local minima)



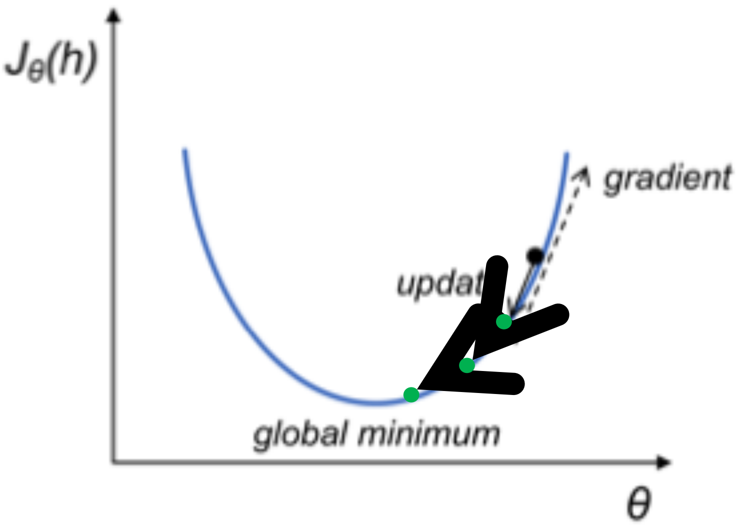


Learning rate and Stochastic Gradient Descent (SGD)

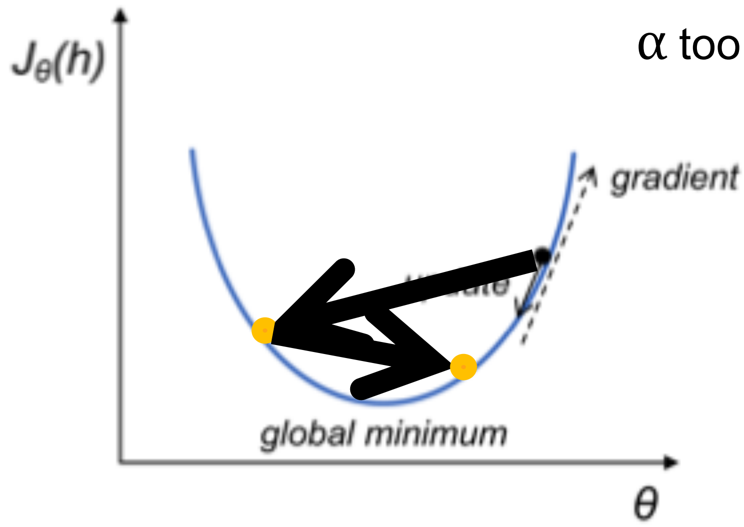
Update Rule $\theta_0 := \theta_0 - \alpha \, dJ(\theta_0, \theta_1)/d\theta_0$

$\theta_1 := \theta_1 - \alpha \, dJ(\theta_0, \theta_1)/d\theta_1$

α : Learning rate



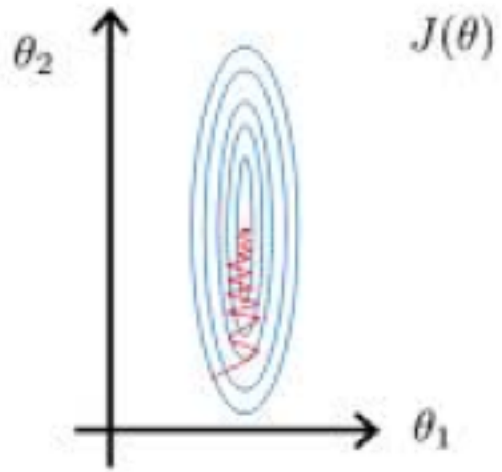
α too large

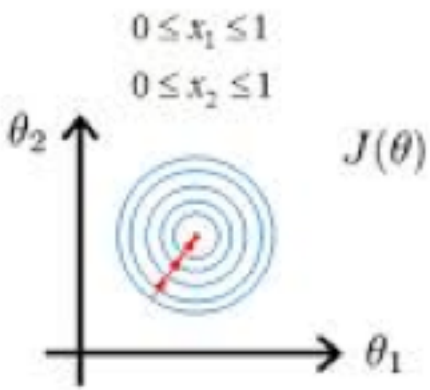


	Compute Cost and Gradients
• Gradient Descent (SGD):	for ALL examples
• Stochastic Gradient Descent (SGD):	for ONE example (randomly chosen)
• MiniBatch:	for a Batch (subset) randomly chosen

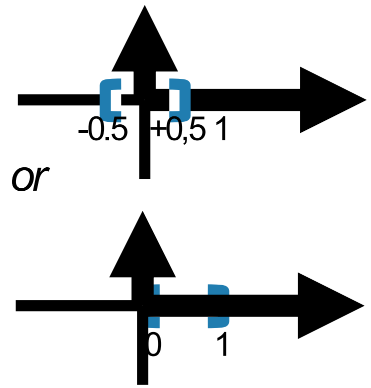
Scaling and Mean Normalization

- To improve learning convergence (faster), better to have features (x_1, x_2…) on a similar scale



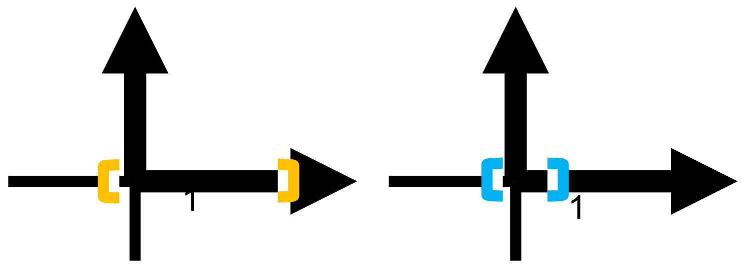


scaled and normalized



- Feature Scaling

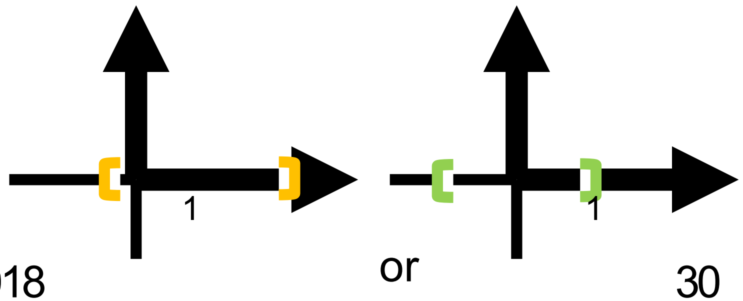
  x := x / range(x) -> Range of 1



(Could also apply)

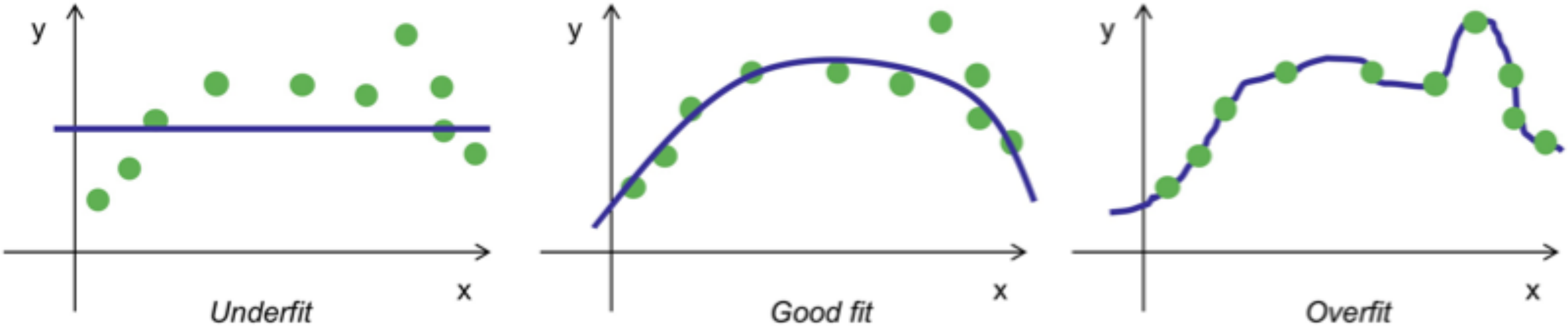
- Mean Normalization

  x := x – mean(x) -> Centered



Best Hypothesis

- Objective: Find out "best" hypothesis (values of θ_0 and θ_1)
- Best hypothesis means best fit to data
- And not just best fit only to Train data
- Otherwise the best strategy would be to exactly memorize every entry
- Best hypothesis means best fit for future/yet unknown data: Test data
- i.e., with good capacity for Generalization

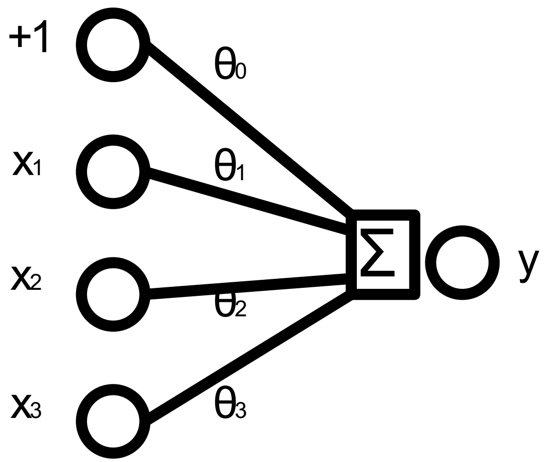


(High Bias)

(High Variance)
Low Generalization

- Regularization: technique for reducing overfitness
  – Weights decay, to penalize over-preponderant weights
  – Also: Dropout, Early stopping, Dataset augmentation

Multiple Linear Regression – More than 1 Variable

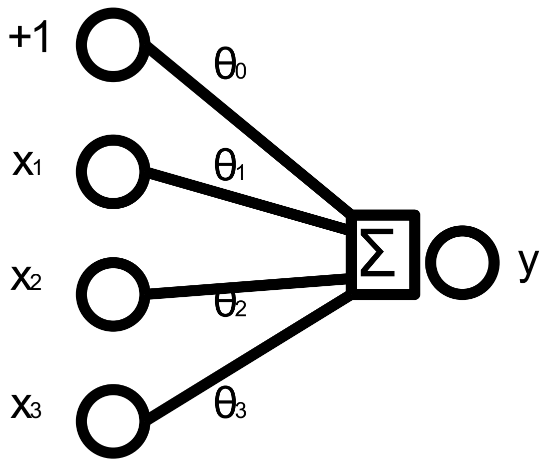


$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Input Dataset : X : $\boxed{\begin{matrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{matrix}}$ $\boxed{\begin{matrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{matrix}}$ … $\boxed{\begin{matrix} x_1^{(m)} \\ x_2^{(m)} \\ x_3^{(m)} \end{matrix}}$

Output Dataset : y : $\boxed{y^{(1)}}$ $\boxed{y^{(2)}}$ … $\boxed{y^{(m)}}$

Multiple Linear Regression Vectorization



$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$= {}_{i=0}{}^{n}\Sigma\ \theta_i\, x_i \qquad (\text{with } x_0 = 1)$$

$$= [\theta_0\ \theta_1\ \theta_2\ \theta_3] * \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

scalar(1x1) = *vector(1x4)* * *vector(4x1)*

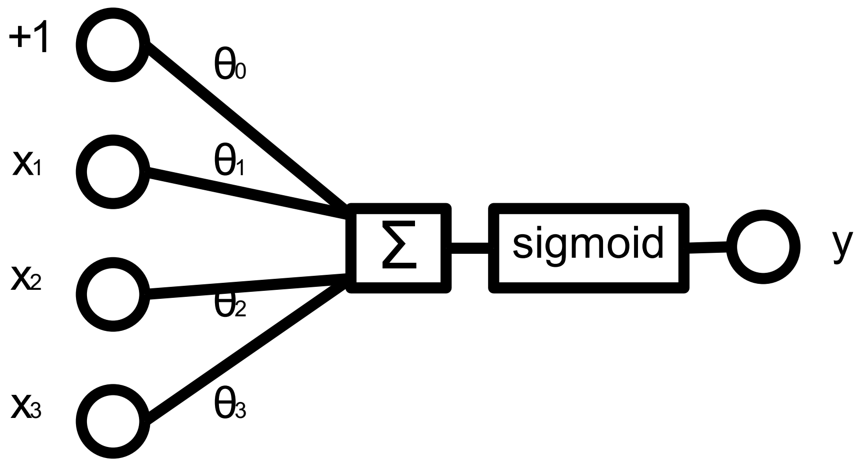
$$= \qquad \theta * x$$

Computing predictions for a set of examples X :

$$\theta * X = [\theta_0\ \theta_1\ \theta_2\ \theta_3] * \left[\begin{bmatrix} 1 \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \begin{matrix} \ldots \\ \ldots \\ \ldots \\ \ldots \end{matrix} \begin{bmatrix} 1 \\ x_1^{(m)} \\ x_2^{(m)} \\ x_3^{(m)} \end{bmatrix} \right] = [h^{(1)} \ldots h^{(m)}]$$

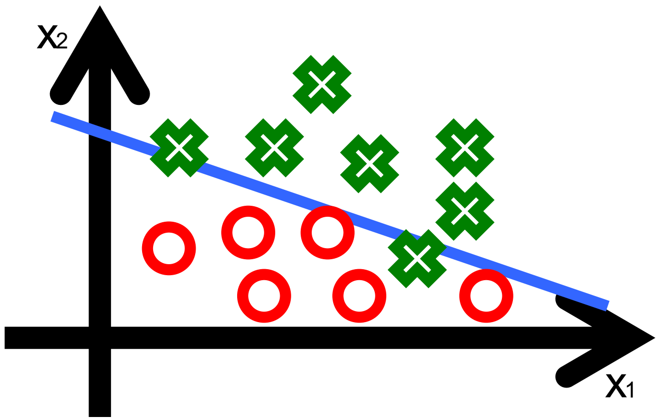
vector(1x4) * *matrix(4xm)* = *vector(1xm)*

Multiple Logistic Regression (Classification)

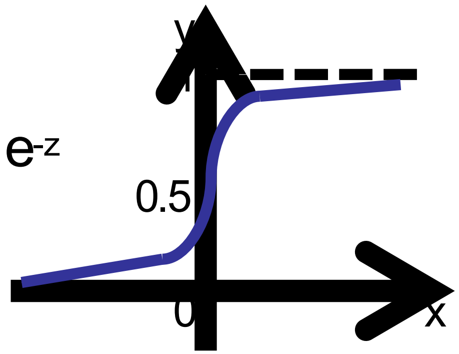


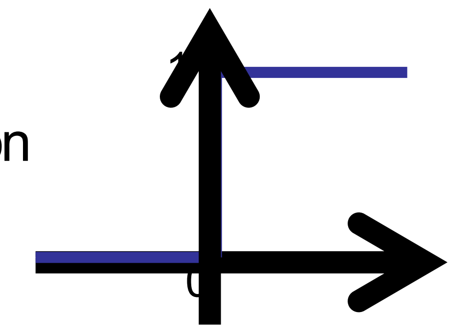
$h(x) € [0\ 1]$

$$h(x) = \text{sigmoid}(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)$$
$$= \text{sigmoid}(\theta * x)$$



$$\text{sigmoid}(z) = \sigma(z) = 1 \,/\, 1 + e^{-z}$$





Perceptron [Rosenblatt 1957] used unit step function

Current Neural Networks use Softmax (transforms into probabilities)

Classifier Evaluation Metrics

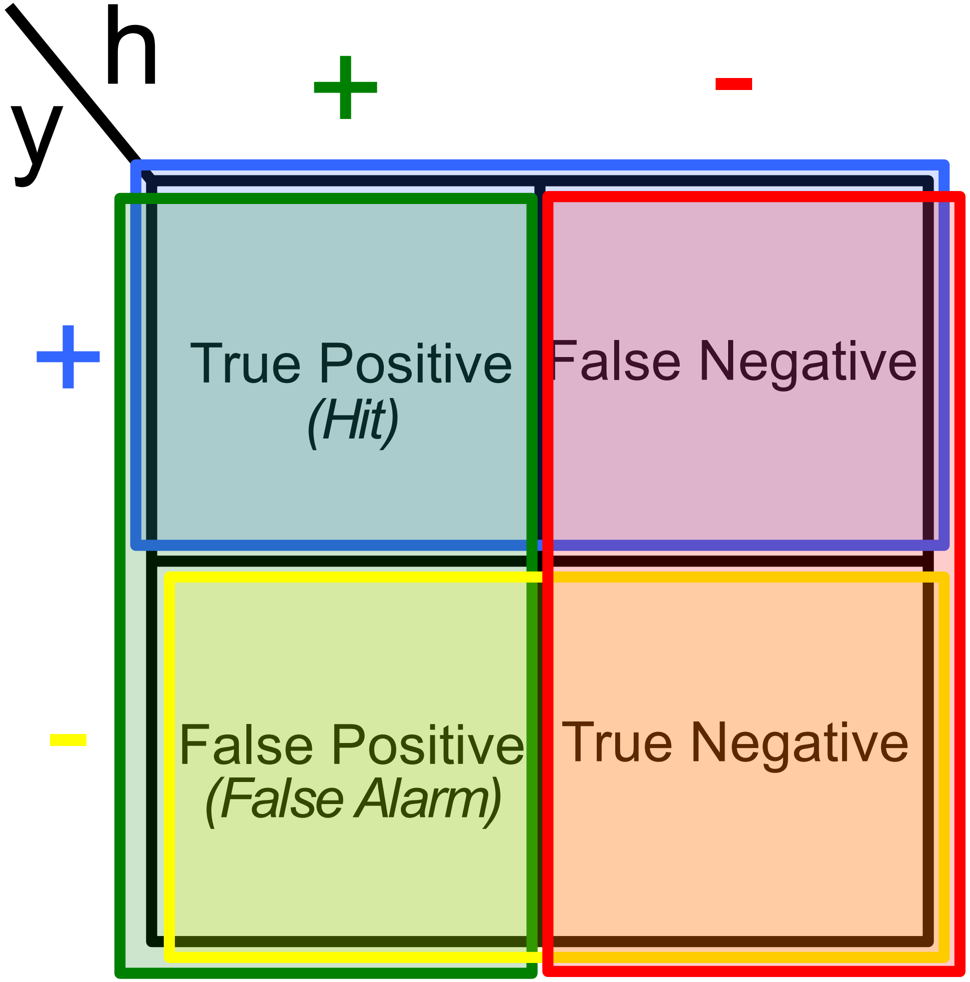
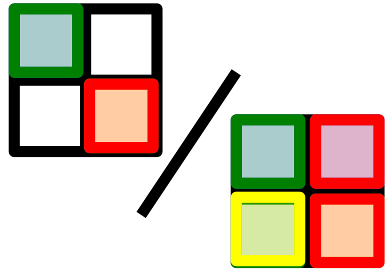
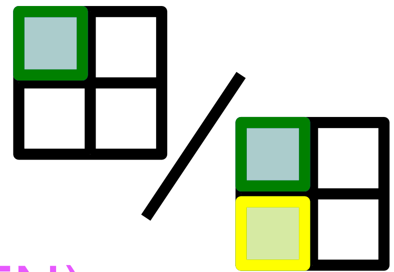


Table of Confusion
(Confusion Matrix)

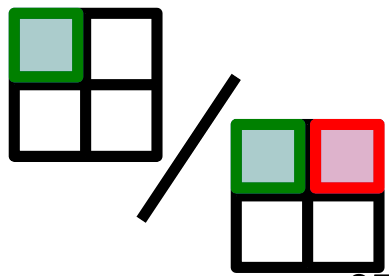
Accuracy = TP + TN / ∀
% correctly classified



Precision = TP / P (TP + FP)
true / classified positive
How useful the results



Recall = TP / P (TP + FN)
true / real positive
How complete the results



Skewed Classes

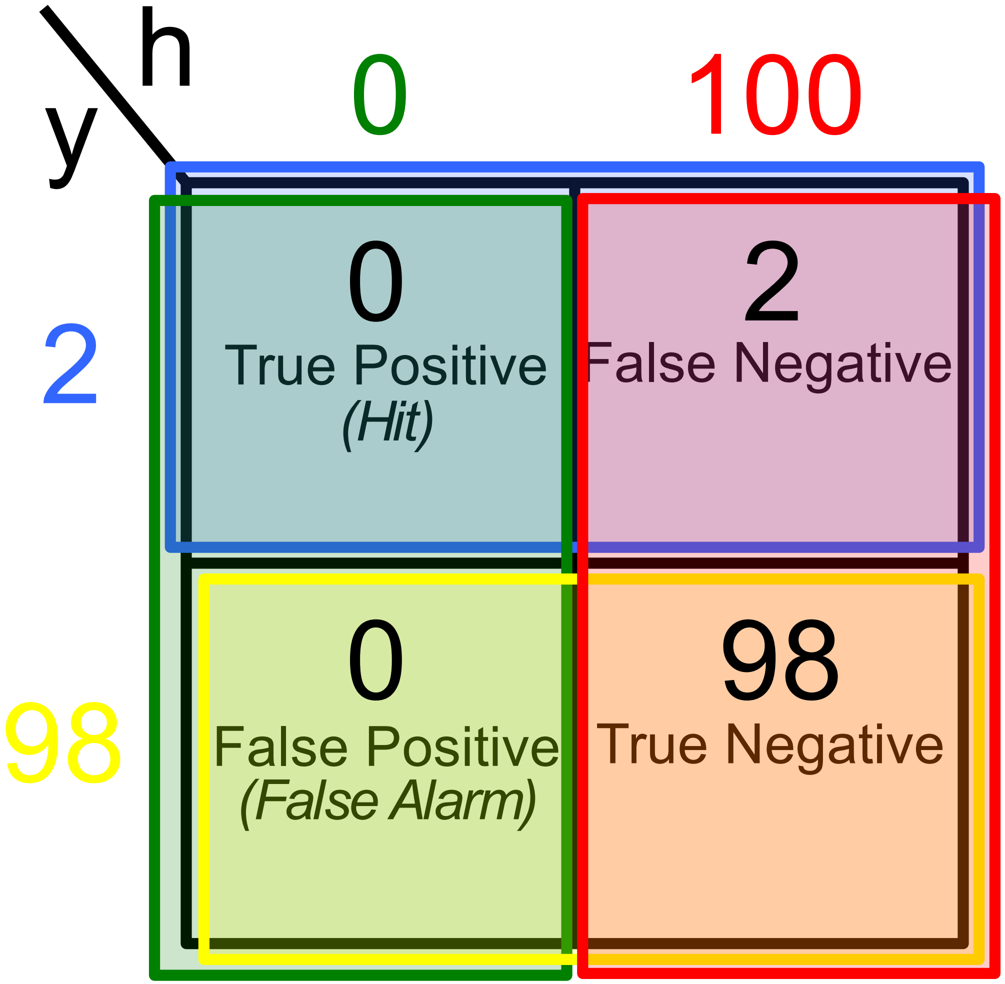
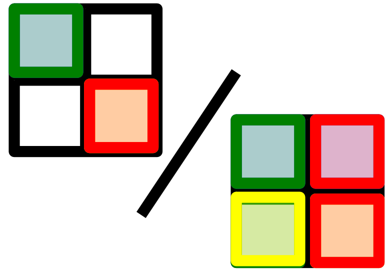
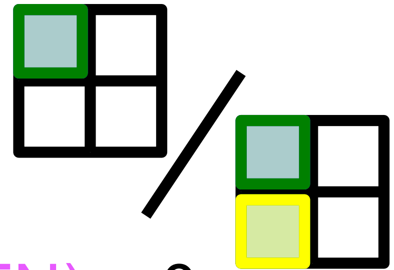


Table of Confusion
(Confusion Matrix)

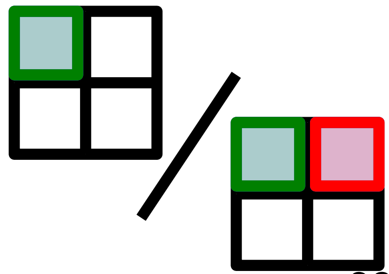
Accuracy = TP + TN / ∀ = 0.98
% correctly classified



Precision = TP / P (TP + FP) = 0
true / classified positive
How useful the results

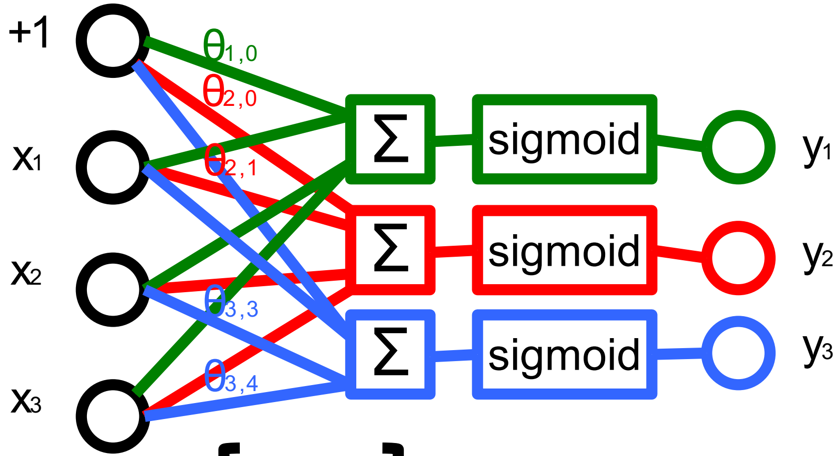


Recall = TP / P (TP + FN) = 0
true / real positive
How complete the results



Ex: Maligne Cancer Prediction: 2% Rate
Optimist Hypothese: Always (100%) False

Multivariate Multiple Logistic Regression (Classification)



$$h(x) = \text{sigmoid}(\theta * x)$$

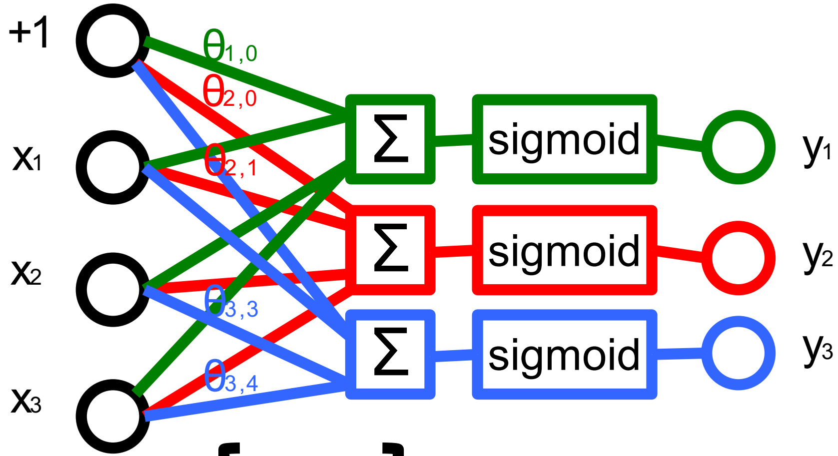
$$\theta = \begin{bmatrix} \cdots \ \theta_{i,j} \ \cdots \end{bmatrix}$$

$$X: \quad \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} \ \cdots \ \begin{bmatrix} x_1^{(m)} \\ x_2^{(m)} \\ x_3^{(m)} \end{bmatrix}$$

$$y: \quad \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} \ \cdots \ \begin{bmatrix} y_1^{(m)} \\ y_2^{(m)} \\ y_3^{(m)} \end{bmatrix}$$

Current Neural Networks use Softmax

Multivariate Multiple Logistic Regression (Classification)



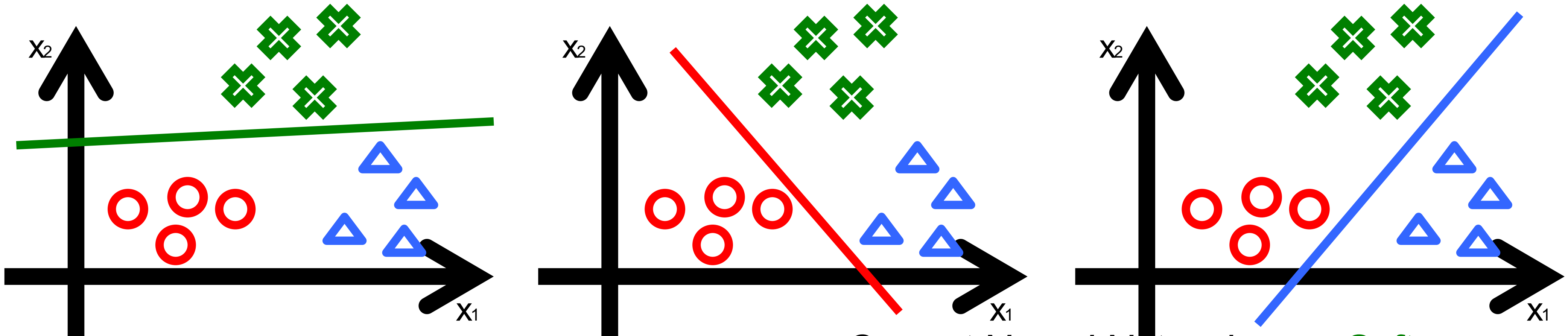
$$\theta = \begin{bmatrix} \cdots & \theta_{i,j} & \cdots \end{bmatrix}$$

$$h(x) = \mathrm{sigmoid}(\theta * x)$$

$$X : \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} \cdots \begin{bmatrix} x_1^{(m)} \\ x_2^{(m)} \\ x_3^{(m)} \end{bmatrix}$$

$$y : \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} \cdots \begin{bmatrix} y_1^{(m)} \\ y_2^{(m)} \\ y_3^{(m)} \end{bmatrix}$$

One vs All Algorithm: higher score = class recognized



Current Neural Networks use Softmax

Multivariate Multiple Classification Vectorization

$$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ h_3(x) \end{bmatrix} = \begin{bmatrix} \theta_{1,0} + \theta_{1,1}x_1 + \theta_{1,2}x_2 + \theta_{1,3}x_3 \\ \theta_{2,0} + \theta_{2,1}x_1 + \theta_{2,2}x_2 + \theta_{2,3}x_3 \\ \theta_{3,0} + \theta_{3,1}x_1 + \theta_{3,2}x_2 + \theta_{3,3}x_3 \end{bmatrix} = \theta * x = \begin{bmatrix} \theta_{1,0} & \theta_{1,1} & \theta_{1,2} & \theta_{1,3} \\ \theta_{2,0} & \theta_{2,1} & \theta_{2,2} & \theta_{2,3} \\ \theta_{3,0} & \theta_{3,1} & \theta_{3,2} & \theta_{3,3} \end{bmatrix} * \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

vector(3x1) = *matrix(3x4)* * *vector(4x1)*

Computing predictions for a set of examples X :

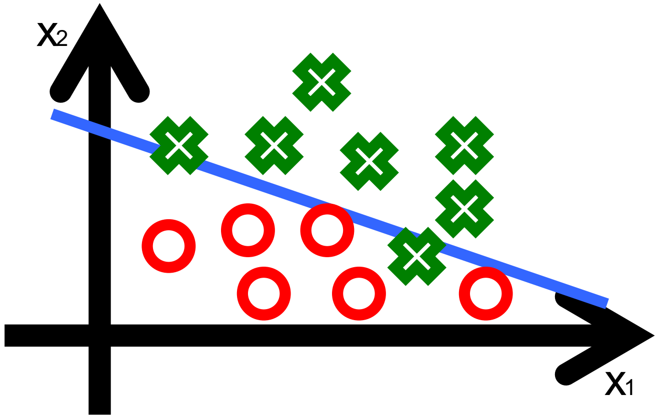
$$\theta * X = \begin{bmatrix} \theta_{1,0} & \theta_{1,1} & \theta_{1,2} & \theta_{1,3} \\ \theta_{2,0} & \theta_{2,1} & \theta_{2,2} & \theta_{2,3} \\ \theta_{3,0} & \theta_{3,1} & \theta_{3,2} & \theta_{3,3} \end{bmatrix} * \begin{bmatrix} 1 & \dots & 1 \\ x_1^{(1)} & \dots & x_1^{(m)} \\ x_2^{(1)} & \dots & x_2^{(m)} \\ x_3^{(1)} & \dots & x_3^{(m)} \end{bmatrix} = \begin{bmatrix} h_1^{(1)} & \dots & h_1^{(m)} \\ h_2^{(1)} & \dots & h_2^{(m)} \\ h_3^{(1)} & \dots & h_3^{(m)} \end{bmatrix} : [class^{(1)} \dots class^{(m)}]$$

matrix(3x4) * *matrix(4xm)* = *matrix(3xm)* *vector(1xm)*

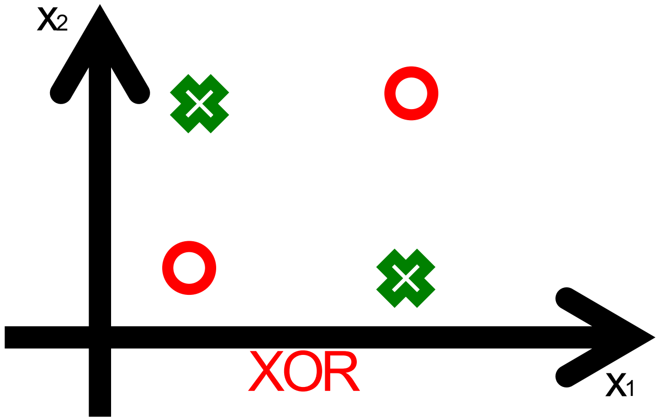
Non Linearity

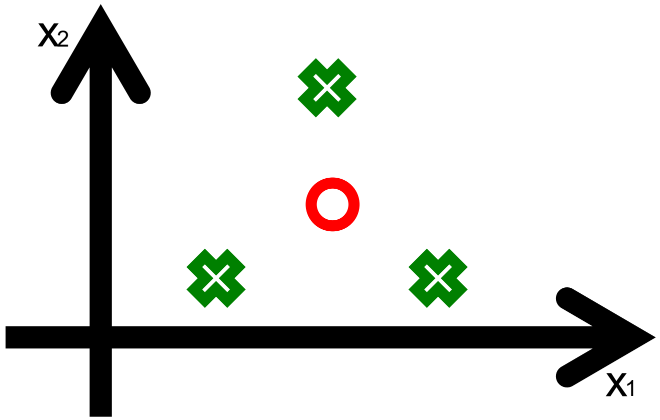
Linear vs Non Linear Decision Boundary

- Linear



- Non Linear

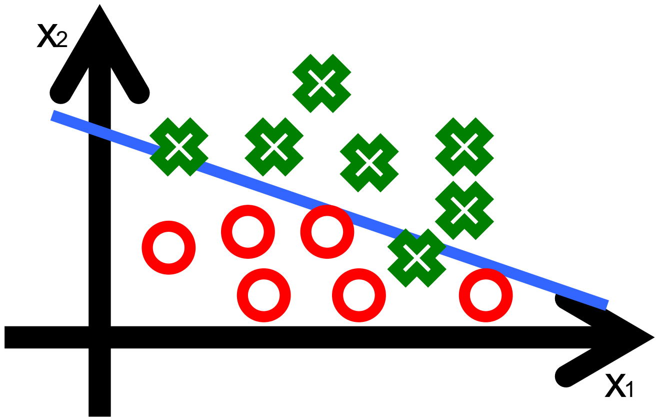




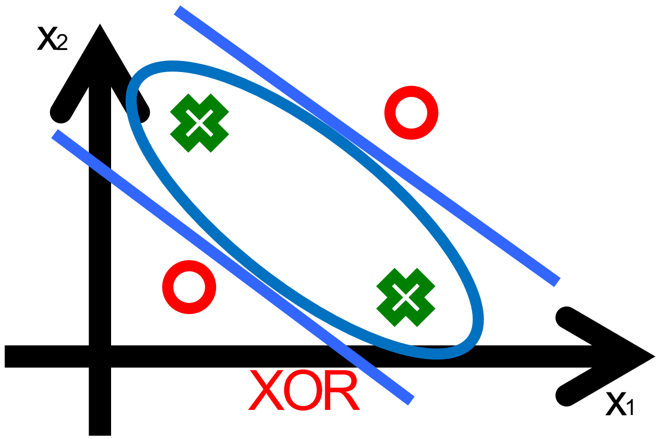
- Argument (XOR) used by [Minsky & Papert 1969] to criticize Perceptrons [Rosenblatt 1957] (and advocate Symbolic Artificial Intelligence)
- This stopped research on Perceptrons/Neural Networks for a long while
  - until Hidden Layers and Backpropagation or/and Kernel Trick (see later)

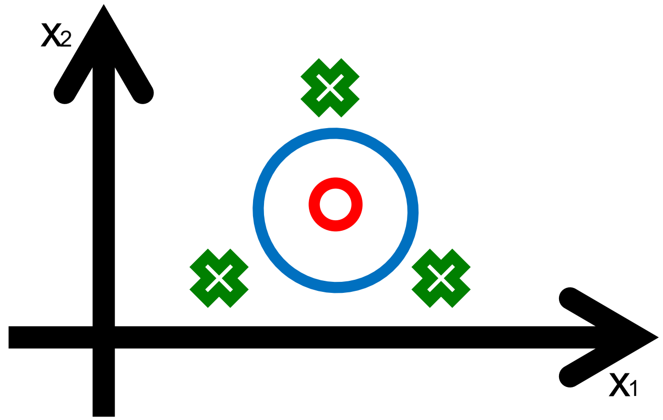
Linear vs Non Linear Decision Boundary

- Linear



- Non Linear

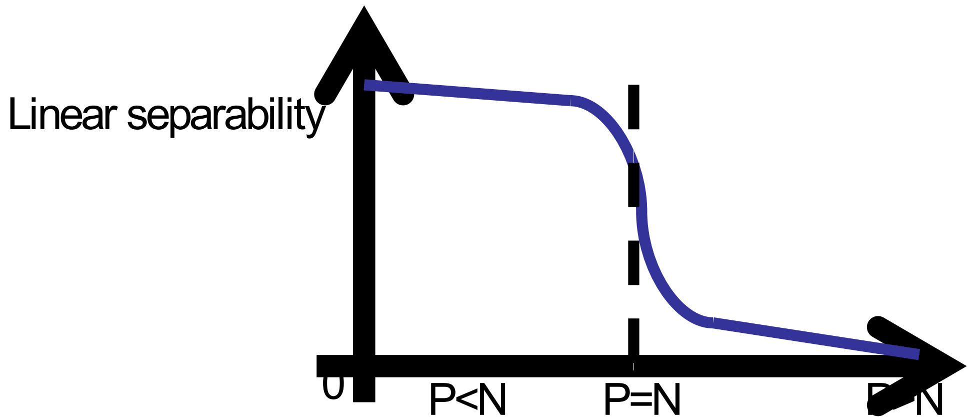




- Argument (XOR) used by [Minsky & Papert 1969] to criticize Perceptrons [Rosenblatt 1957] (and advocate Symbolic Artificial Intelligence)
- This stopped research on Perceptrons/Neural Networks for a long while
  - until Hidden Layers and Backpropagation or/and Kernel Trick (see later)

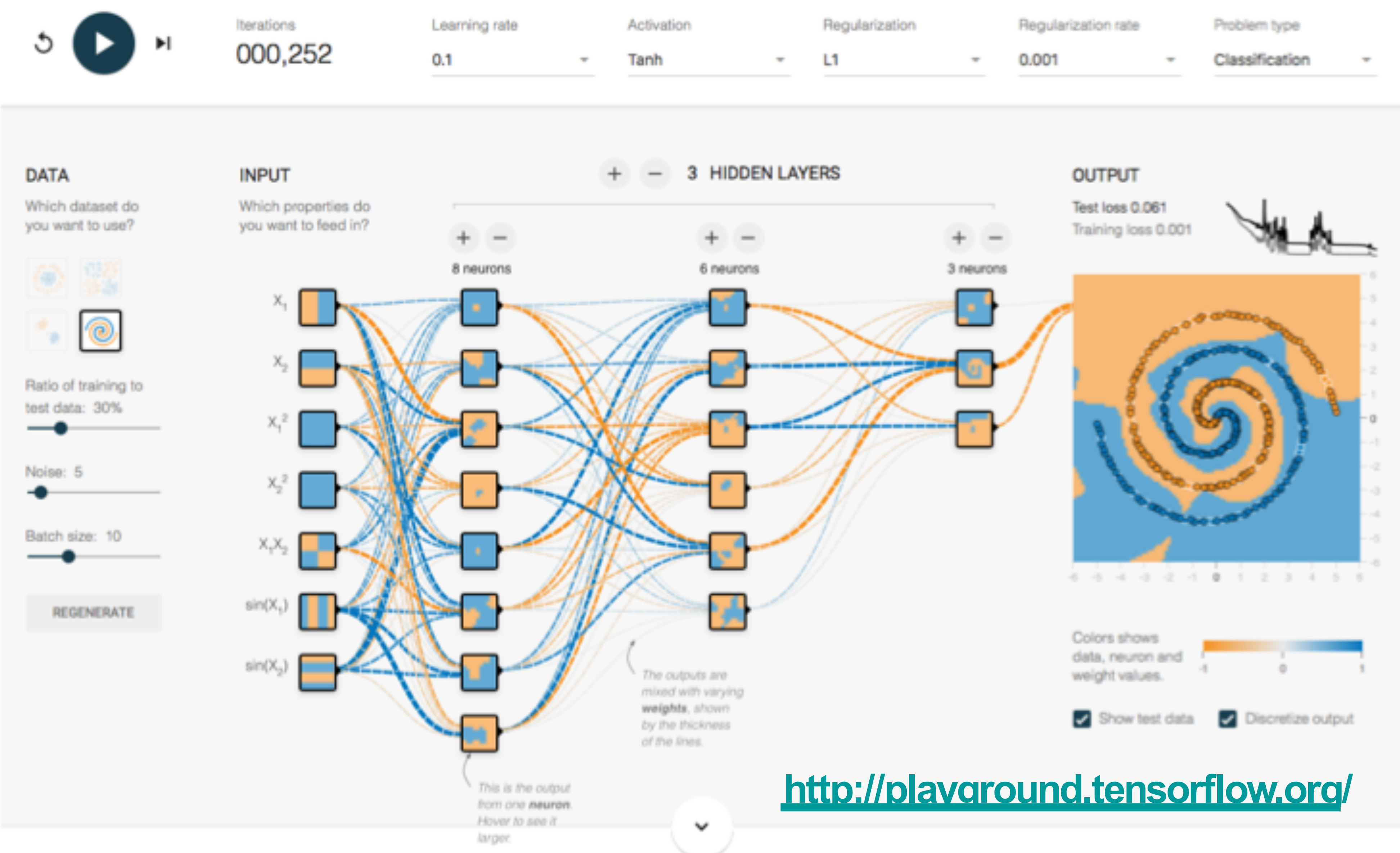
Linear vs Non Linear Decision Boundary

- A consequence of Cover's Theorem [Cover, 1966]
  - The probability that P samples (examples) of dimension N (number of parameters) are linearly separable goes to zero very quickly as P grows larger than N



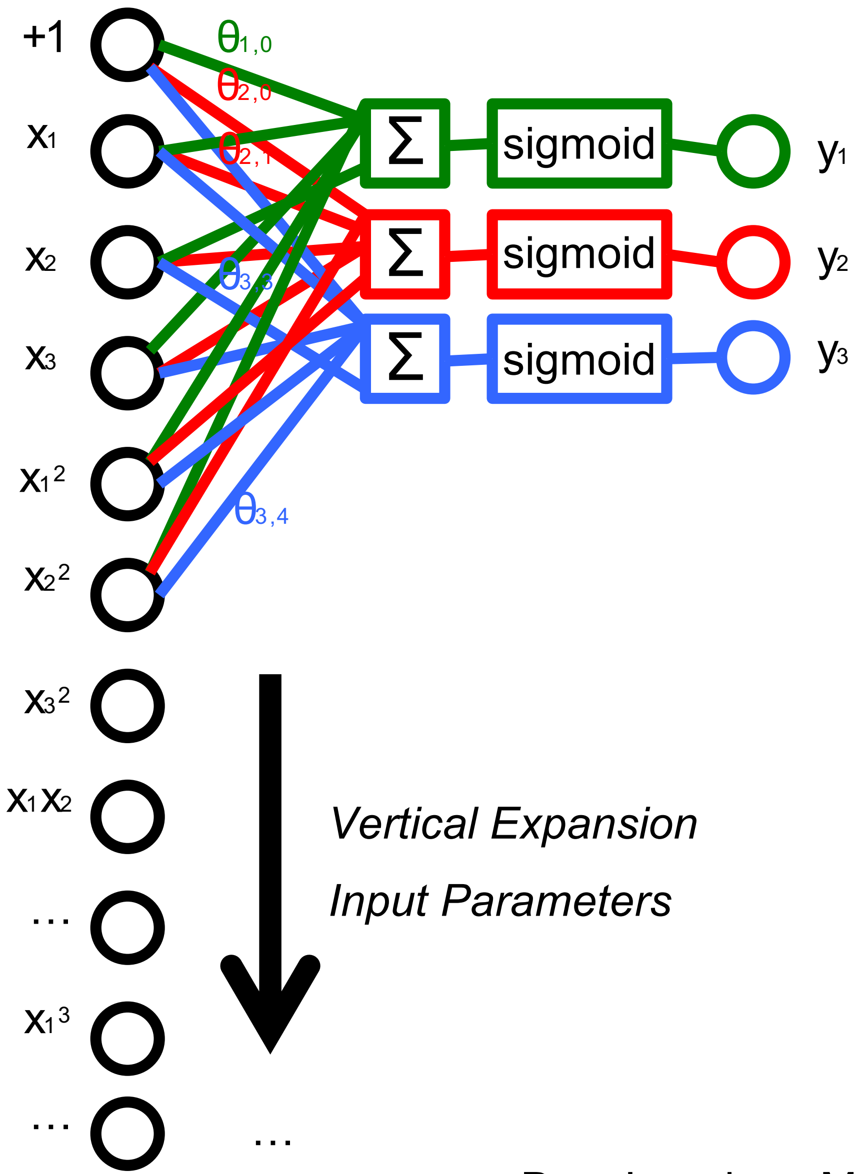
[LeCun 2016]

Example (TensorFlow PlayGround)



http://playground.tensorflow.org/

1st Approach: Polynomial Logistic Regression



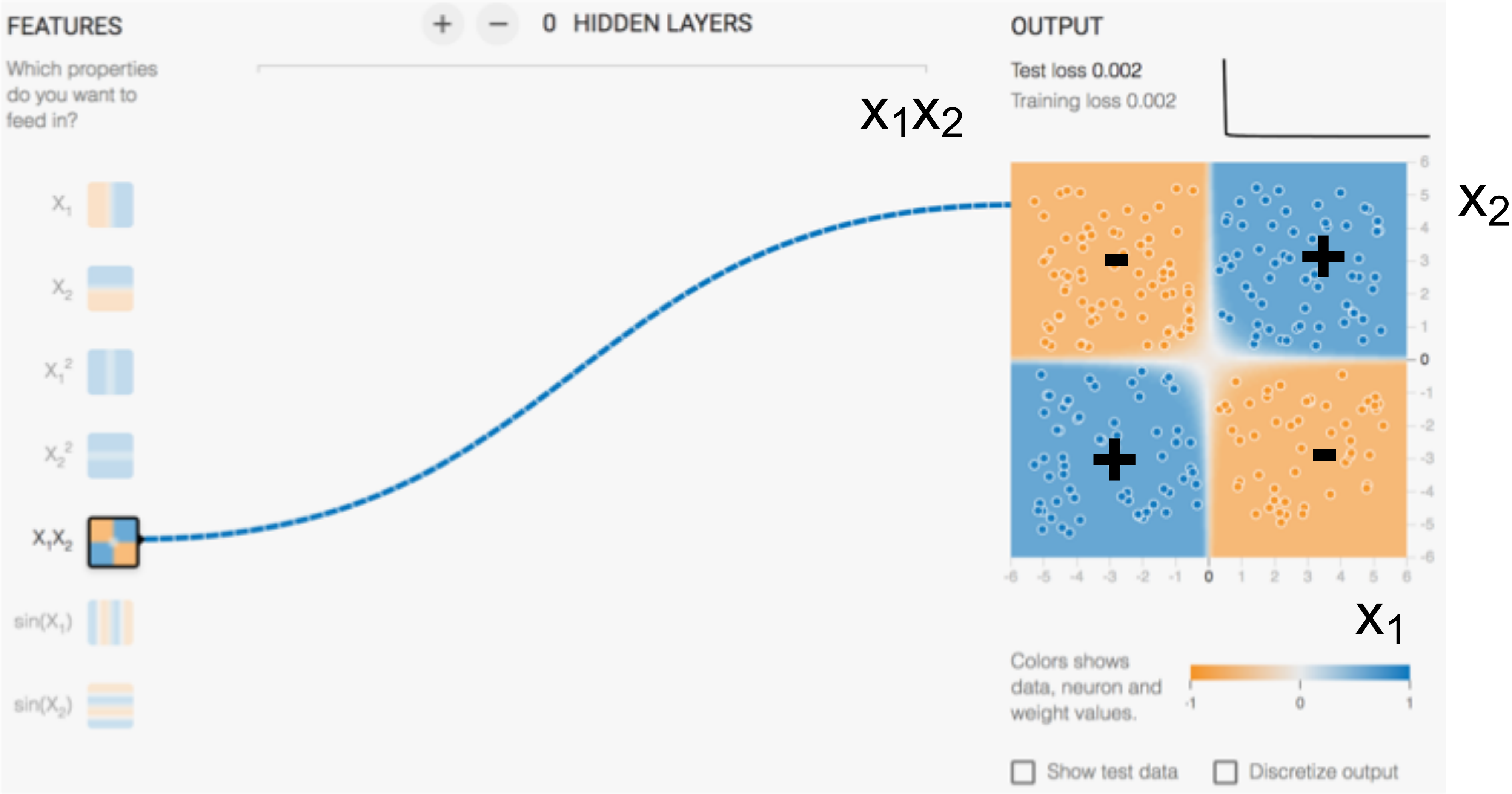
1st Approach: Polynomial Logistic Regression

- Idea:
  - Increase the number of Dimensions
  - By adding Polynomial Inputs
    - » Quadratic: x_1^2, x_2^2…
    - » Dot products: x_1x_2, x_1x_3…
    - » Cubic: x_1^3, x_2^3…
    - » Sinus: $\sin(x_1)$, $\sin(x_2)$...
    - » ...
- In order to find (explore) a higher Dimension Space in which could be found a Linear Decision Boundary

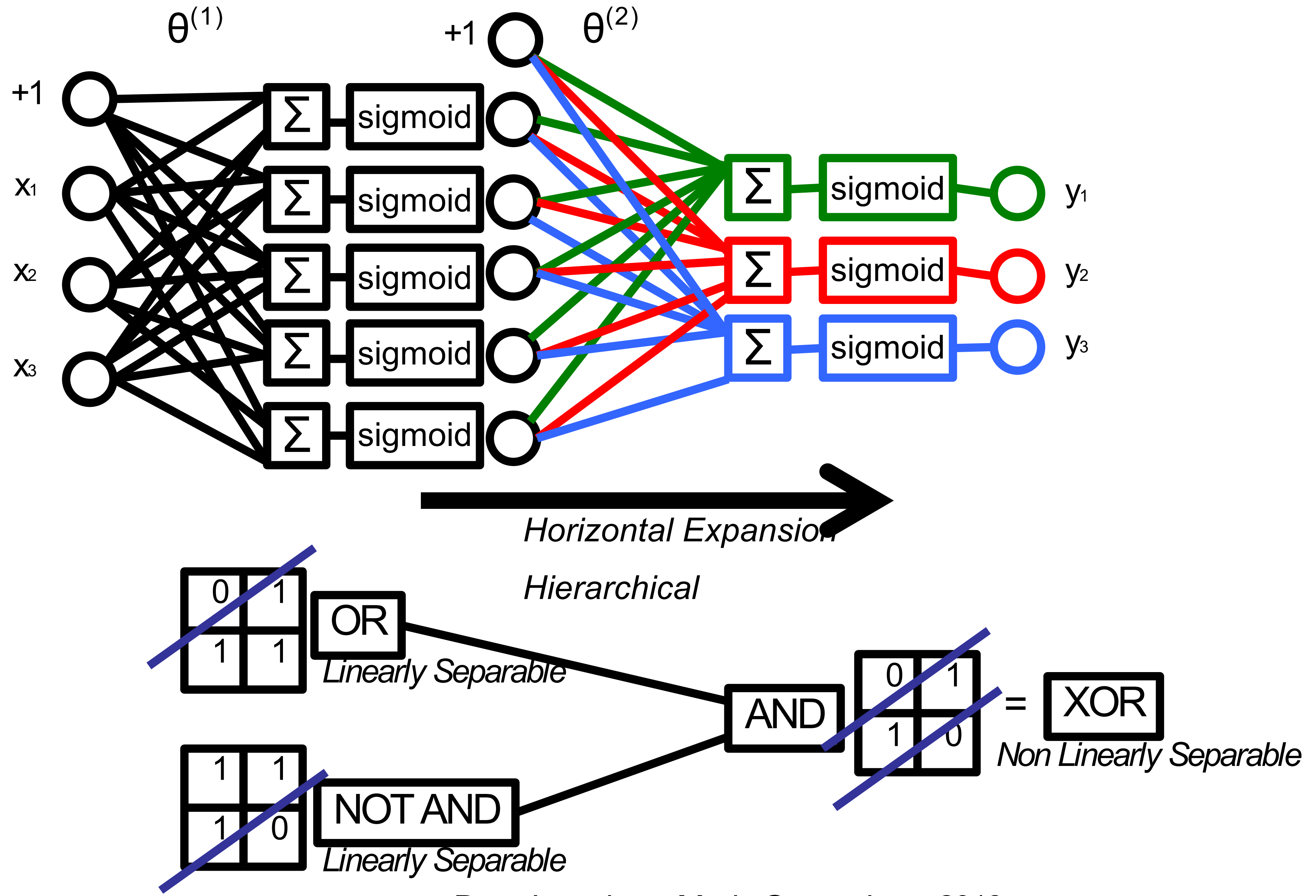
- Problems:
  - Select the Additional Inputs
  - Add them in a combinatorial way
  - The Number of Inputs could become Very Large
  - Computation and Data becomes more Heavy

Example: x_1x_2

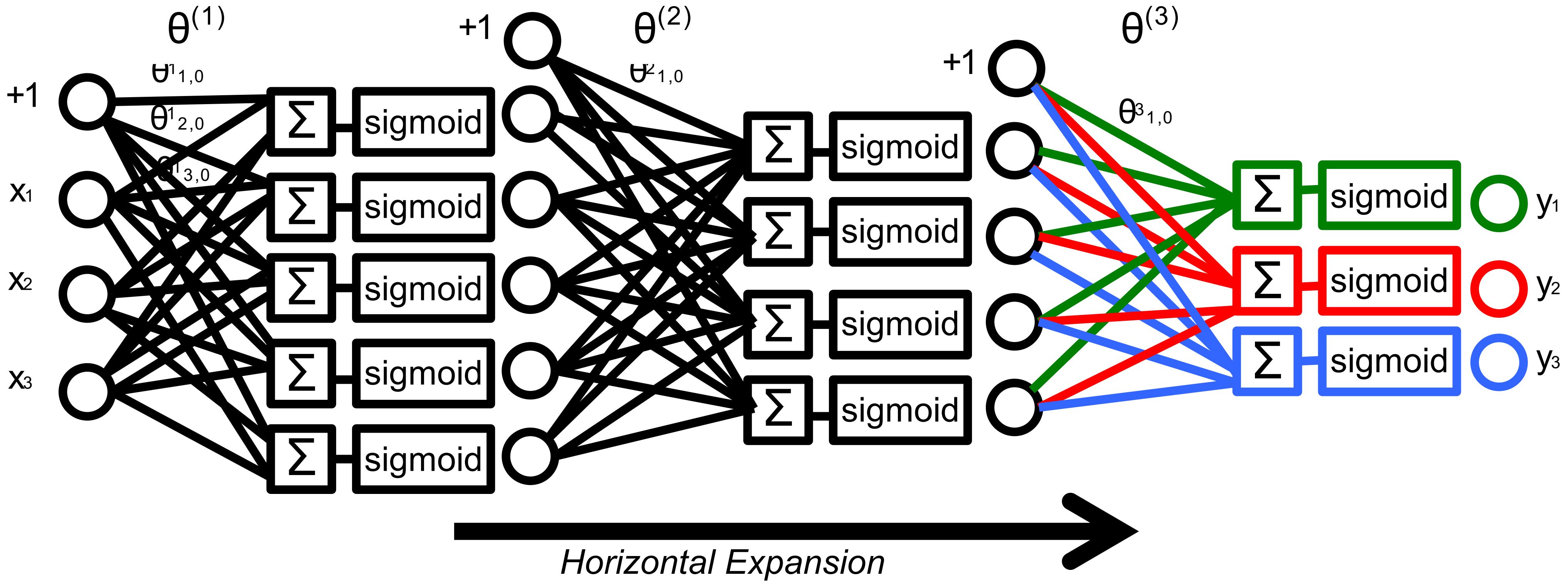
- Initial Representation $x_1 \times x_2$
- New Representation x_1x_2



2^{nd} Approach: Hidden Layers



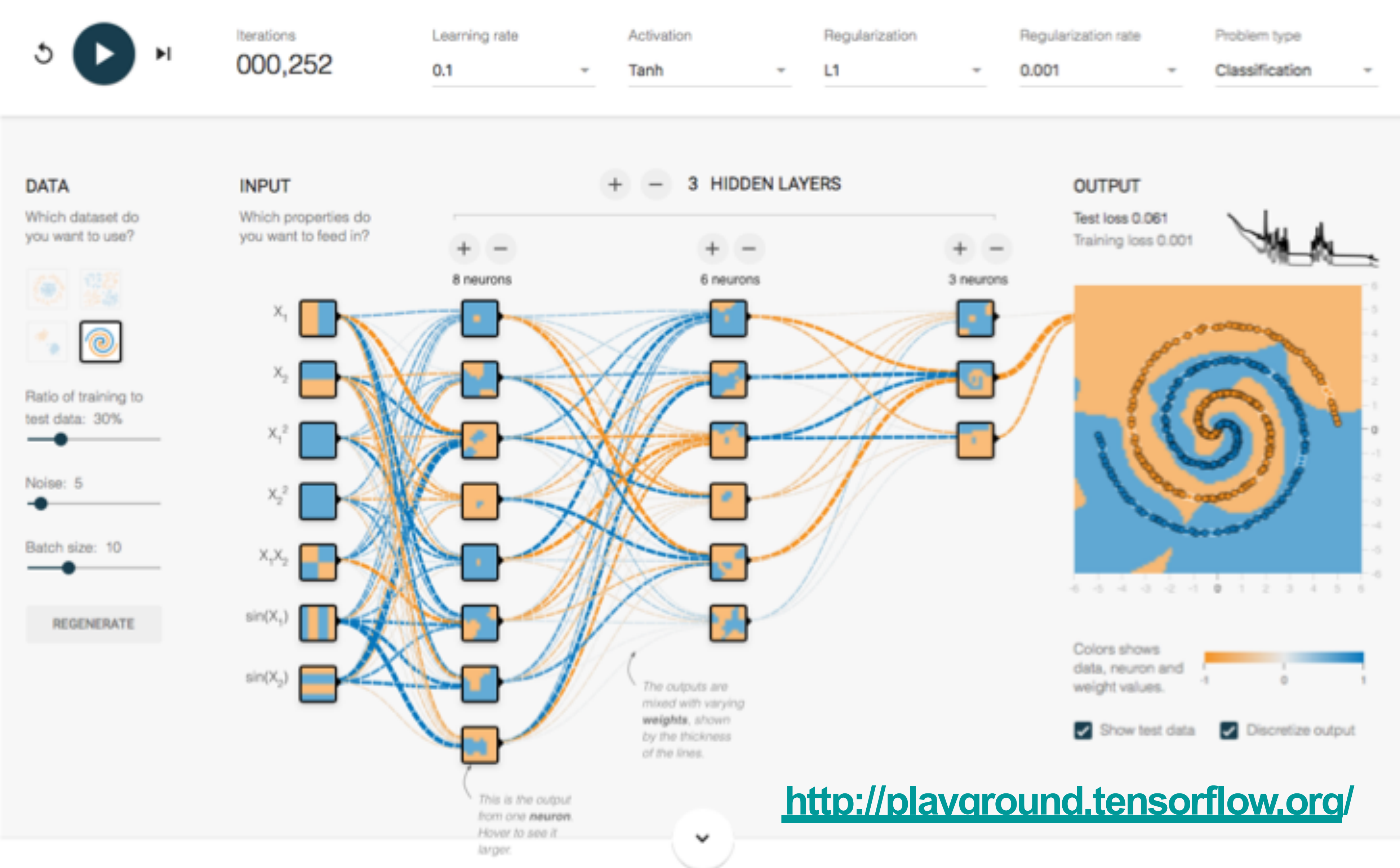
2nd Approach: Hidden Layers



Horizontal Expansion

Hierarchical

Example (TensorFlow PlayGround)



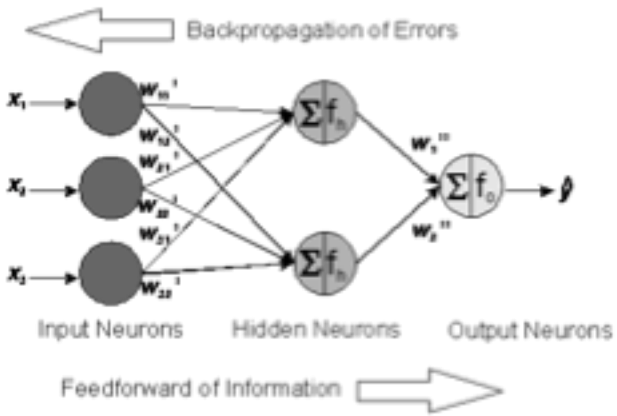
http://playground.tensorflow.org/

Backpropagation

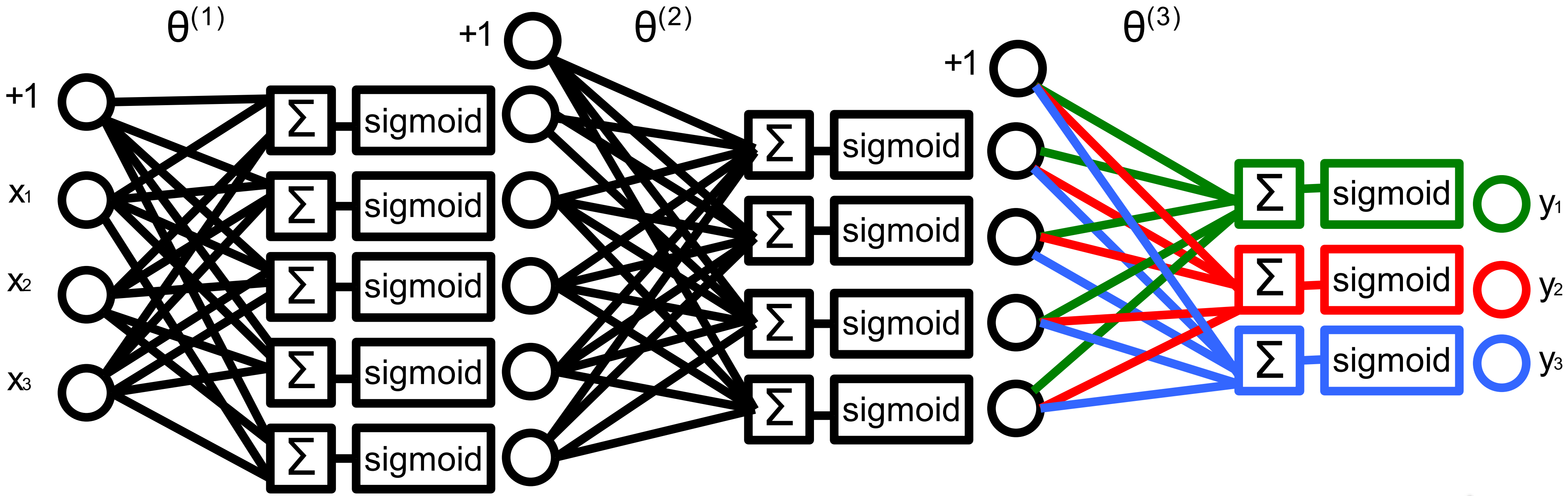
Issue:

- How to compute Gradients (partial derivatives respective to each $\theta^{(k)}_{i,j}$ (weight)) ?
- Now that there are several hidden layers…

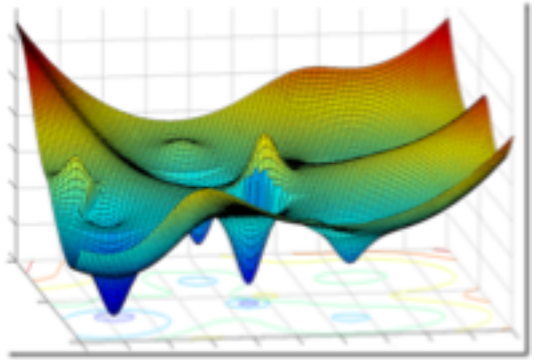
- Backpropagation (backward propagation of errors) algorithm [Rumelhart et al. 1986, initial ideas since 1960] estimates the gradients for each weight $\theta^{(k)}_{i,j}$ (in each layer k), through a chain rule approach:
  - Feedforward the network to compute the output
  - Compute output layer units errors (deltas between predicted and actual values)
  - Propagate backward the errors (deltas) to each unit of previous layer
  - And then compute (accumulate) the gradients (relative to each weight)



2^{nd} Approach: Hidden Layers – Limits



- Non Convex Optimization
  - Importance/Difficulty of Initialization of Weights ($\theta^{(n)}{}_{i,j}$)
  - Local Minimum



- Gradient Vanishing [Hochreiter 1991]
  - Backpropagation algorithm (chain rule based) to propagate gradients (from output errors)
  - Thinner Estimation (Increasing errors) of gradients while traversing layers

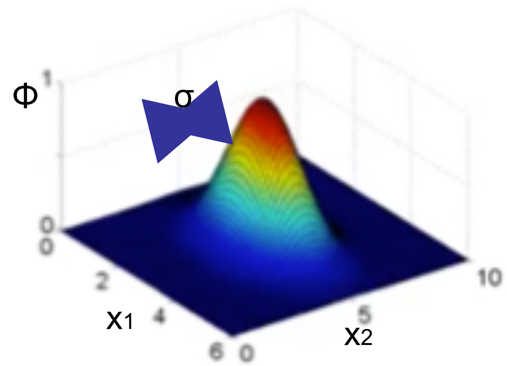
3^{rd} Approach: Kernel (Trick)

- Increase the number of Dimensions
- By a Transformation of Input Parameters into a Linearly Separable Higher Dimension Feature Space
- A Change of Representation

- This Transformation Function K is Named a *Kernel* Function

- A Kernel function $K(x^{(1)}, x^{(2)})$ represents the *Similarity* between two examples ($x^{(1)}$ and $x^{(2)}$)

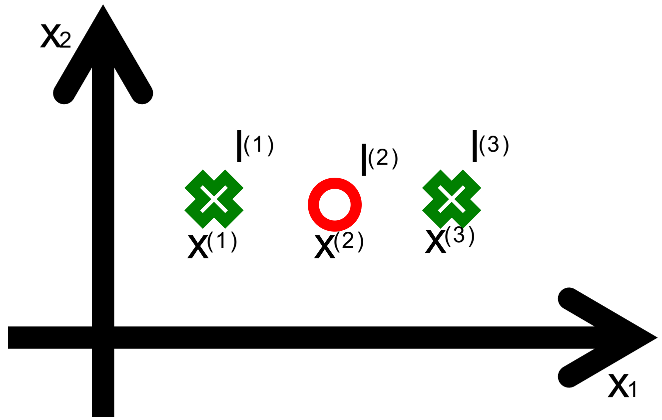
- Example of Kernel: Gaussian Kernel
  - also named Radial Basis Function (RBF) Kernel
  - $K(x^{(j)}, l^{(i)}) = \text{similarity}(x^{(j)}, l^{(i)}) = \exp(- ||x^{(j)} - l^{(i)}||^2 / 2\sigma^2)$

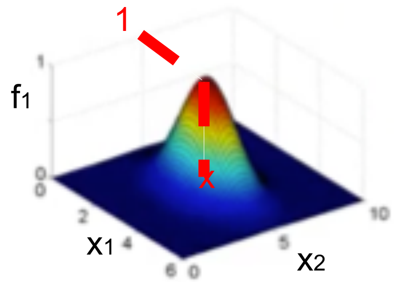


  - Other Kernels:
    - Linear (Without) Kernel, Polynomial Kernel, Sigmoid Kernel…
    - Homemade Kernel (must satisfy Mercer property [Mercer 1909])

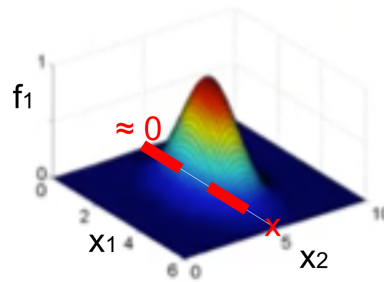
Kernel = Samples-Centered Feature Transformation

- Let's consider all (m = 3) examples $x^{(i)}$ as *Landmarks* $l^{(i)}$



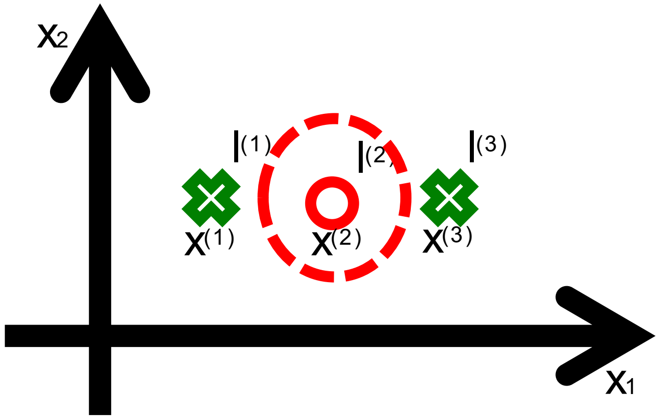


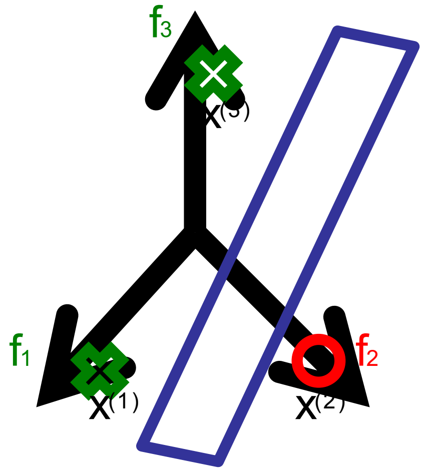
$f_1(x^{(1)}) = K(x^{(1)}, l^{(1)}) = 1$



$f_1(x^{(2)}) = K(x^{(2)}, l^{(1)}) \approx 0$

- Features are defined as following:
  - $f_1(x) = K(x, l^{(1)}) = \exp(- ||x - l^{(i)}||^2 / 2\sigma^2)$
  - $f_2(x) = K(x, l^{(2)})$
  - ...
  - $f_m(x) = K(x, l^{(m)})$





Kernel Trick

- Computing the Kernel is much easier (and more efficient) when it can be expressed as an inner product (in the feature space) of a kernel function f
- $K(x^{(1)}, x^{(2)}) = f(x^{(1)}).f(x^{(2)})$
- Ex: $K(x, y) = (\langle x_1, x_2\rangle \,.\, \langle y_1, y_2\rangle)^2$

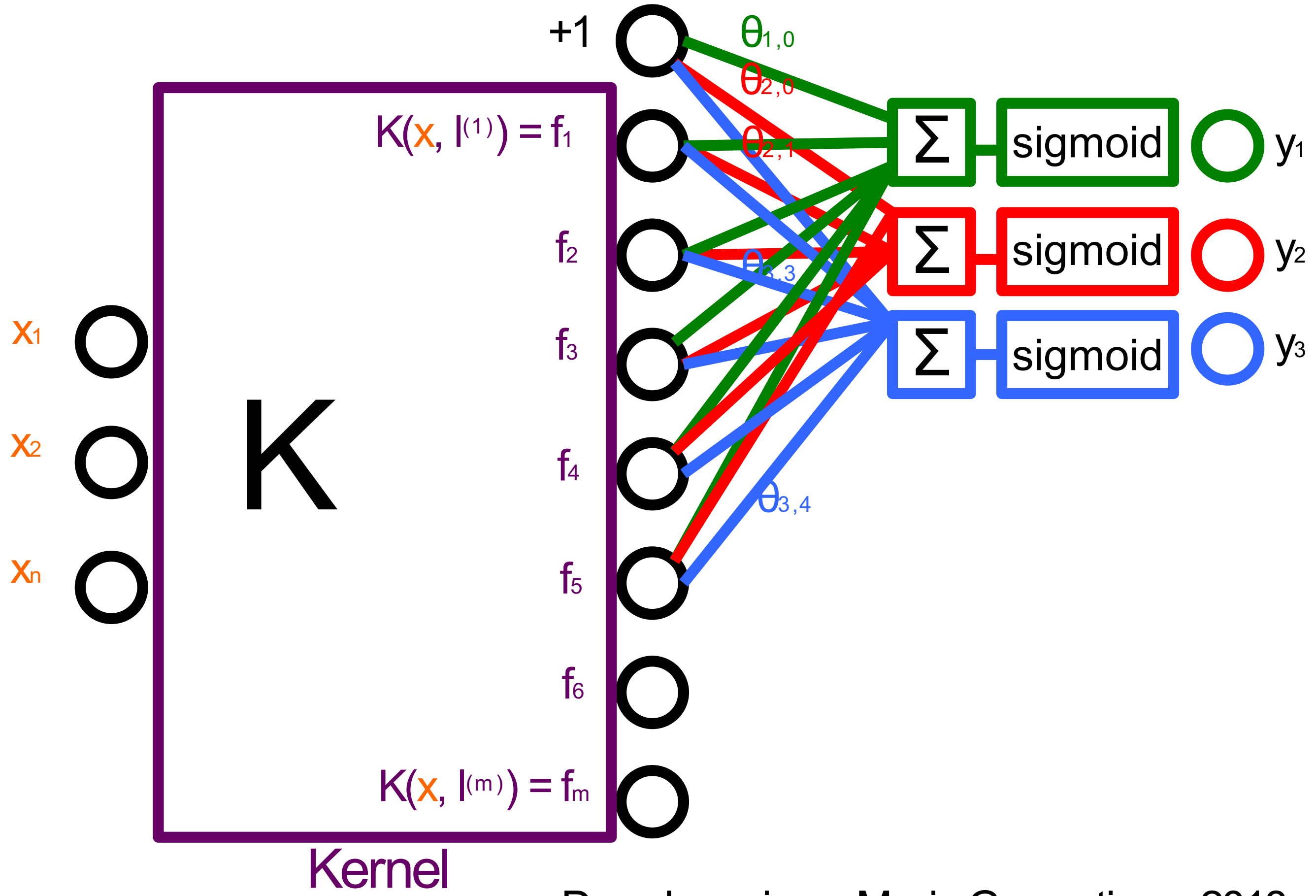
$$= (x_1y_1 + x_2y_2)^2$$
$$= x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2$$
$$= \langle x_1^2, \sqrt{2}\, x_1x_2, x_2^2\rangle \,.\, \langle y_1^2, \sqrt{2}\, y_1y_2, y_2^2\rangle$$
$$= f(x) \,.\, f(y)$$

with $f(x) = \langle x_1^2, \sqrt{2}\, x_1x_2, x_2^2\rangle$

Mercer's Theorem: Characterization of a function as a kernel function

Kernel (Trick)

- The initial n dimension space (number of parameters) is tranformed into a m (number of Training examples) dimension space of features (new parameters), linearly separable
- This new space is independent from the number of parameters (n) and so depending on the number of training examples (m)



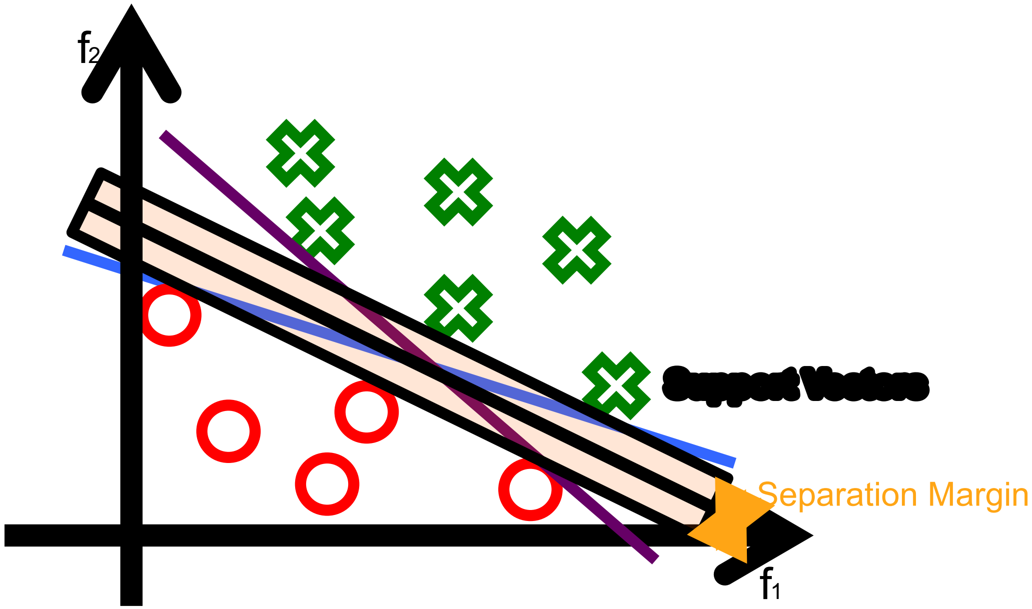
Note:

There are 2 phases:

- *Configuration of the Kernel component with Training examples – landmarks, $l^{(1)}, \ldots l^{(m)}$ are set and stored into the Kernel component*
- *Application of the Kernel component as a dimension transformation on examples (X -> F)*

Support Vector Machines

- The same Kernel Trick is used by Support Vector Machines
- Support Vector Machine [Boser et al. 1992] [Vapnik 1995]
  = Kernel [Mercer 1909] + Linear Support Vector Machine [Vapnik & Lerner 1963]
- A Linear Support Vector Machine is good at maximizing the Separation Margin



- From 2000's, SVMs outperformed Neural Networks and led to the decrease of research in Neural Networks
- Until the 2010's, when they returned through the idea of Deep Networks/Learning (see later)

MNIST Handwritten Character Recognition Test Case

Universal Approximator

- The combination of hidden layer and non linear activation function makes the neural network an universal approximator, able to overcome the non linear separability limitation

- The universal approximation theorem [Hornik, 1991] states that a feedforward network with a single hidden layer containing a finite number of neurons can approximate a wide variety of interesting functions when given appropriate parameters (weights)

- Meanwhile, there is no guarantee that the neural network will *learn* it !

No Free Lunch !

- The no free lunch theorem for machine learning [Wolpert, 1996] states that, averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.

- No machine learning is universally any better than any other.

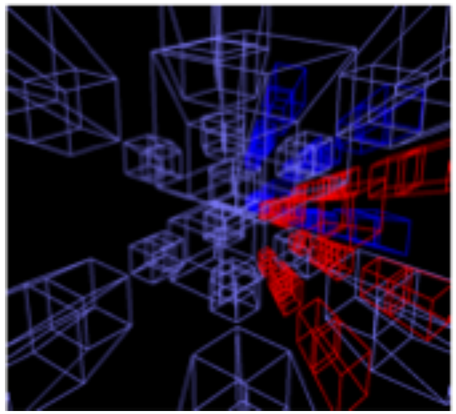
- The most sophisticated algorithm we can conceive has the same average performance, over all possible tasks, as merely predicting that every point belongs to the same class [Goodfellow *et al.*, 2016].

- But, these results hold when we average over all possible data-generating distributions.

- Real applications/data are specific, as well as related algorithms, and have some regularities and inner structure that an algoritm can exploit.

The Malediction of High Dimensionality [Mallat, 2018]

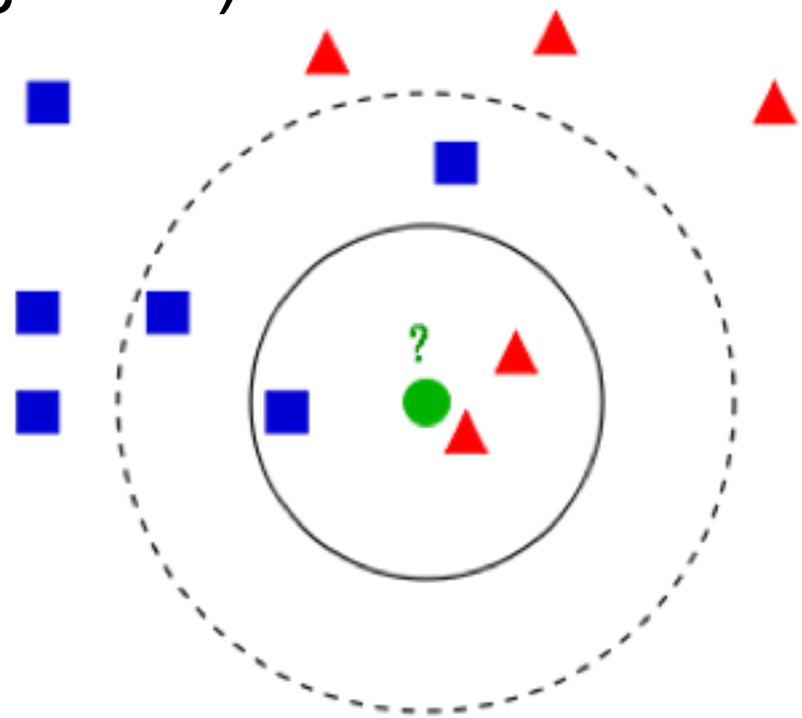
- Ex: Image with 2,000 x 1,000 pixels with color
- = 6.000.000 bits
- Space of dimension 6.000.000 !!



The Malediction of High Dimensionality [Mallat, 2018]

- Ex. of Task: Image recognition (Classification Task)

- kNN algorithm (k Nearest Neighbors)
- The class of an element =
  majority class
  of k nearest neighbors



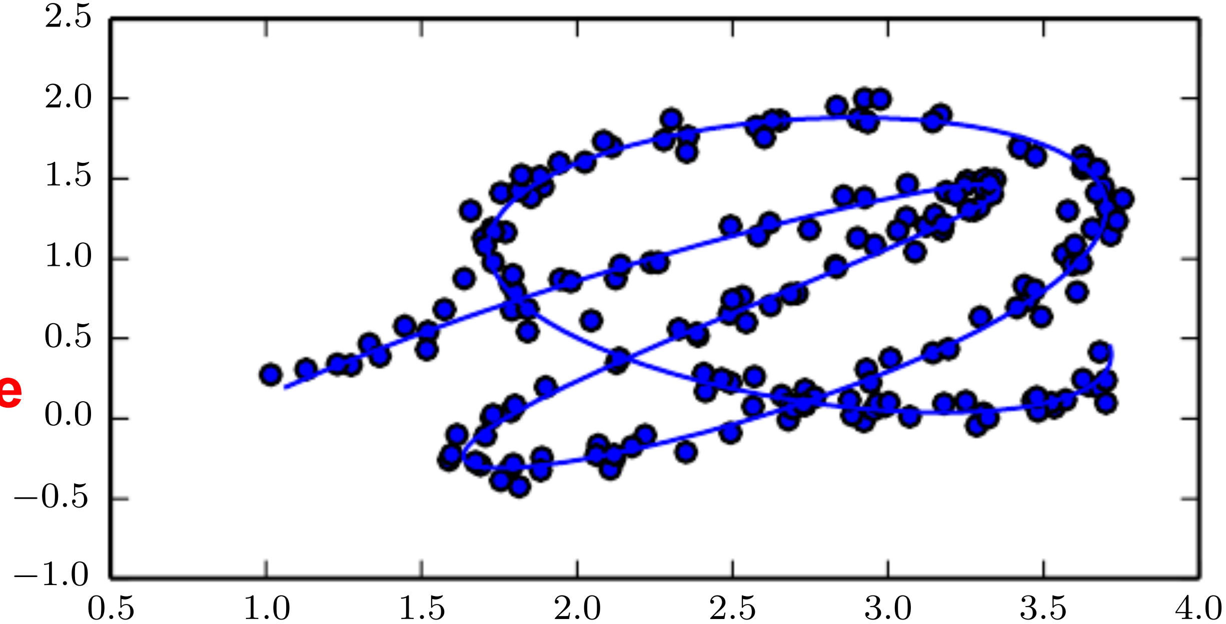
Problem:

- Euclidean distance is unhelpful in high dimensions because all vectors are almost equidistant to the query vector

Representation/Manifold Learning

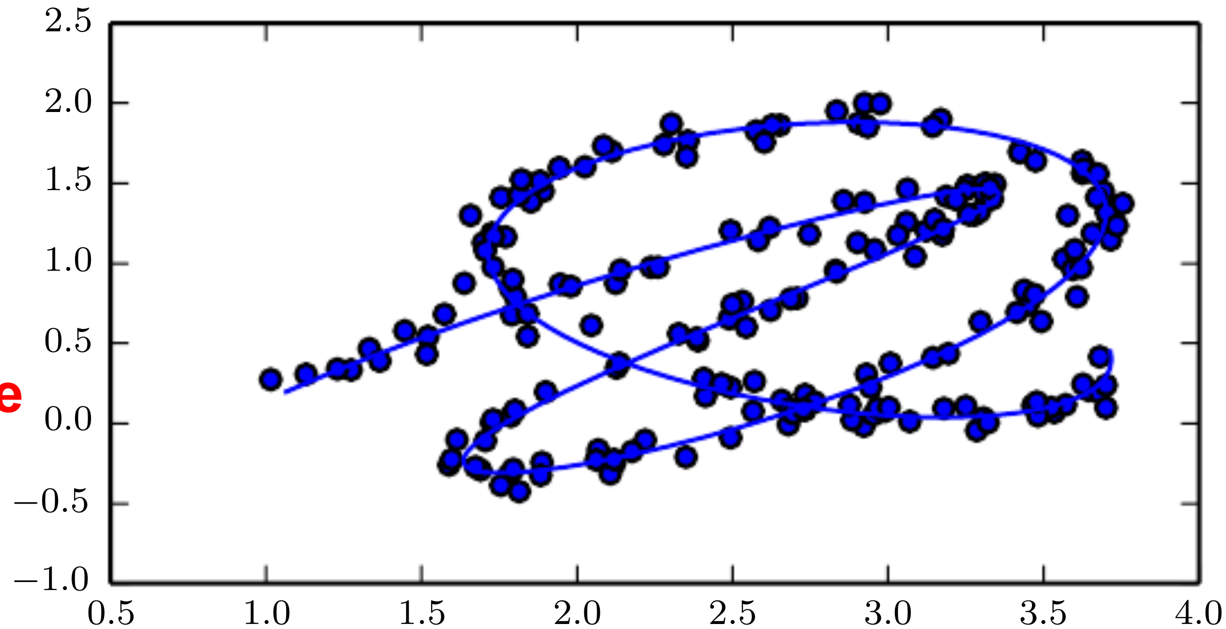
Manifold :
Set of connected points

In a high-dimensional space



[Goodfellow et al., 2016]

Representation/Manifold Learning



Manifold :
Set of connected points

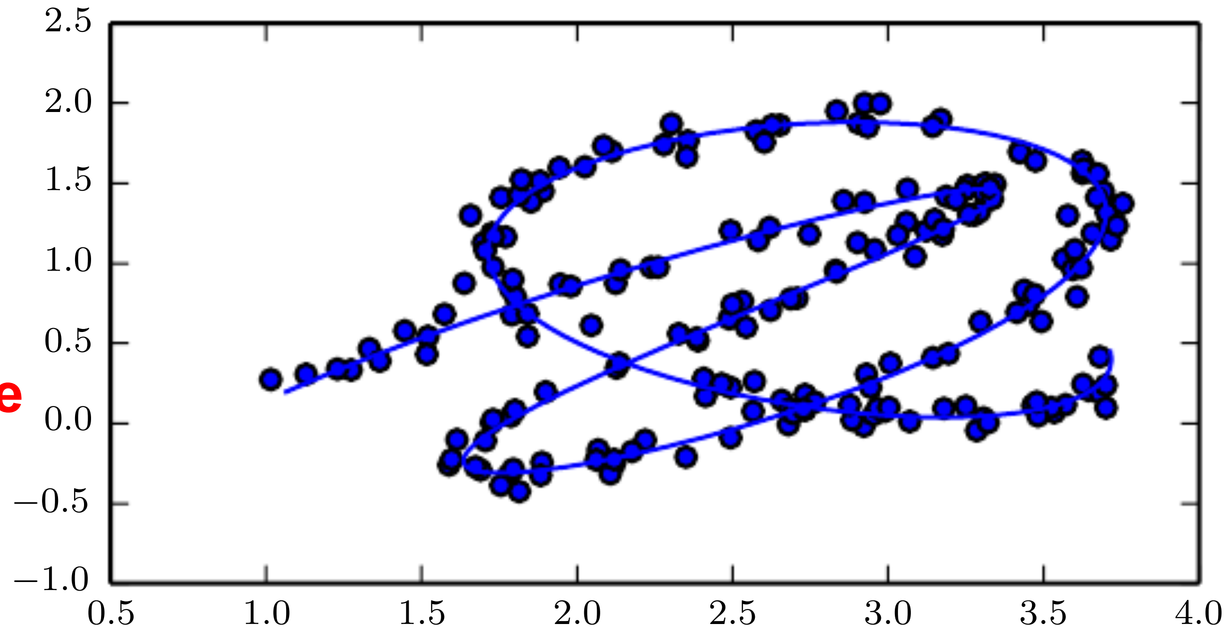
In a high-dimensional space

But can be approximated by
a smaller number of dimensions,
each dimension corresponding
to a local variation

Representation/Manifold Learning

Manifold :
Set of connected points

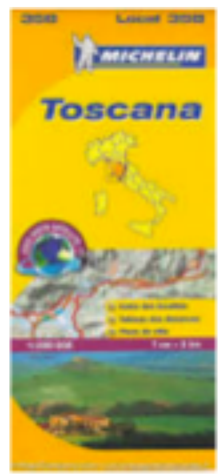
In a high-dimensional space



But can be approximated by a smaller number of dimensions, each dimension corresponding to a local variation

Analogy:

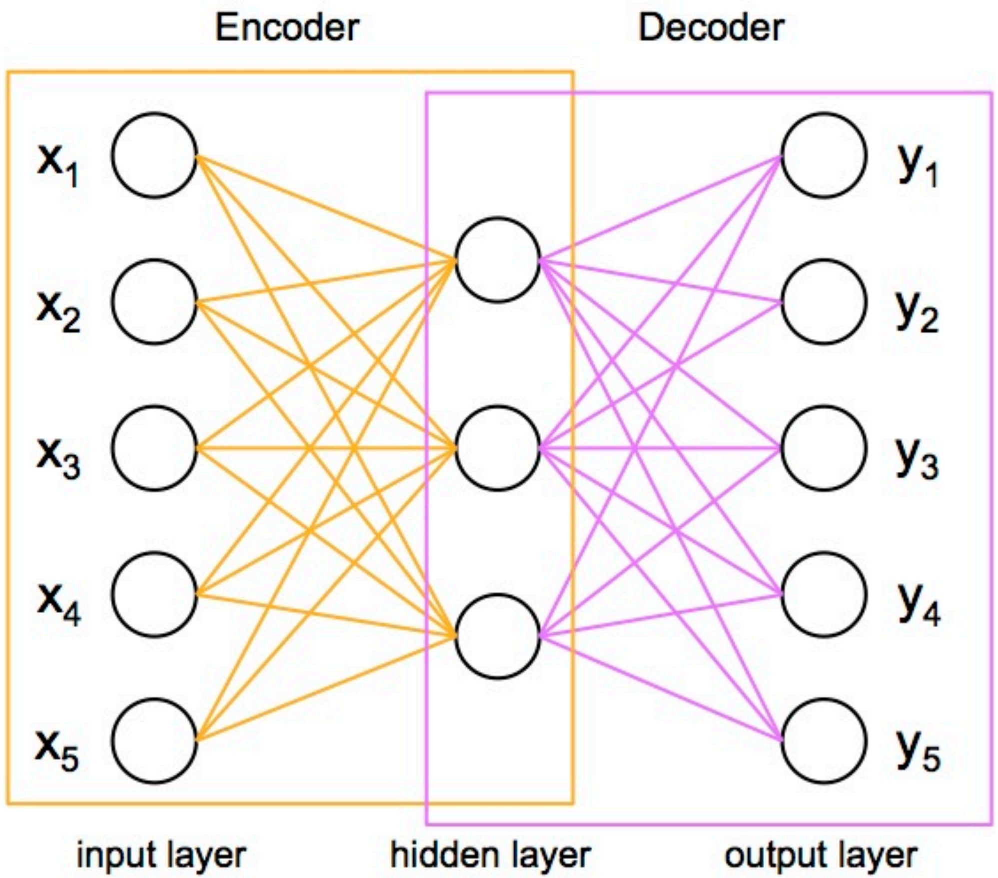
3D Earth



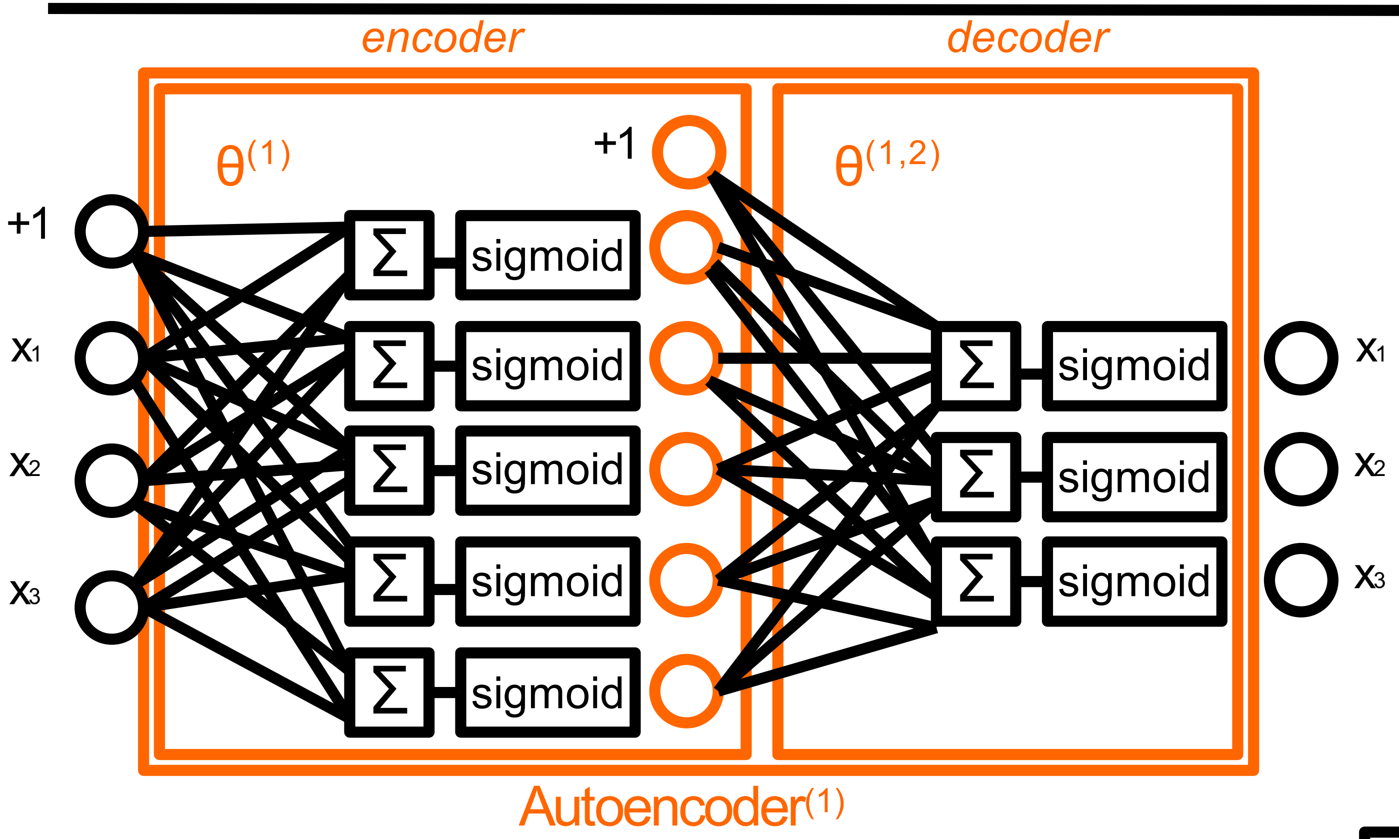
2D Map

Autoencoder

- Symmetric Neural Network
- Trained with examples as input and output
- Hidden Layer will Learn a Compressed Representation at the Hidden Layer (Latent Variables)



Autoencoder Self-Supervised Training

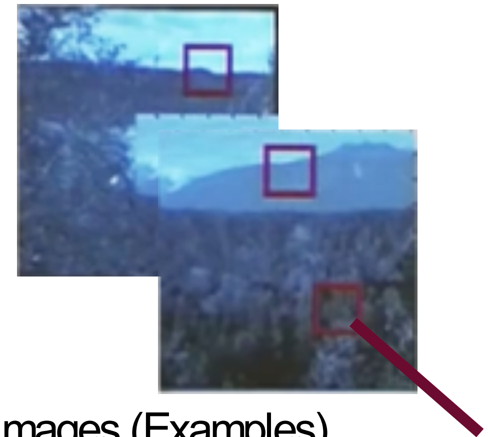


- Training (finding $\theta^{(1)}$ and $\theta^{(1,2)}$) on Input Dataset : X : $\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix}$ $\begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \end{bmatrix}$ … $\begin{bmatrix} x_1^m \\ x_2^m \\ x_3^m \end{bmatrix}$
- **Self-Supervised Training implemented through Supervised Training** with Output = Input : X : **Learn Identity with Sparsity Constraint** [Ng 2012]

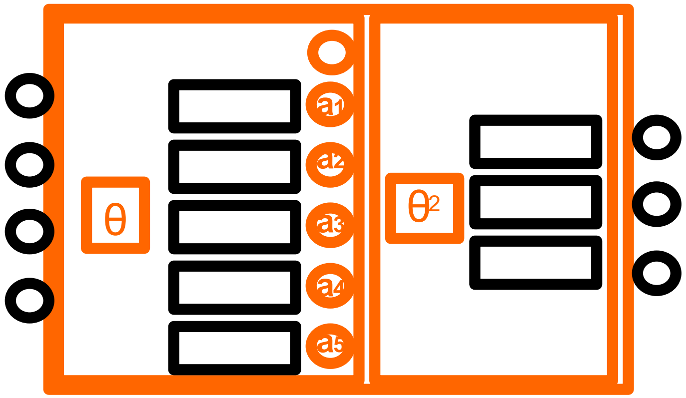
Sparse Autoencoder Learning Features

- Sparse Autoencoding [Olshausen & Field, 1996] [Ng, 2012]
- Originally developed to explain early visual processing in the brain (edge detection)
- Learns a Dictionary of bases $\Phi_1, \ldots \Phi_k$ so that each input can be approximately decomposed (recomposed) as: $x \approx {}_{j=1}{}^{k}\Sigma\ a_j\ \Phi_j$

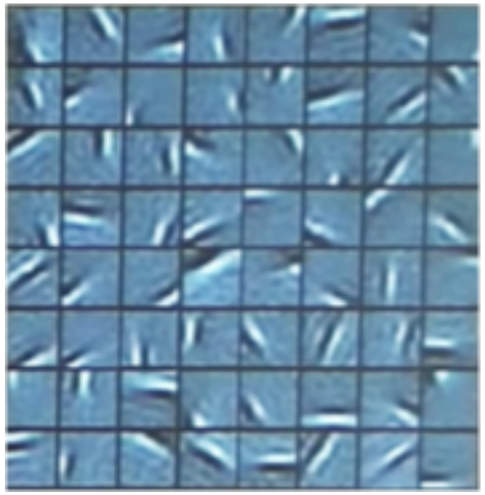
  such as a_j are mosty zero (sparsity constraint)



Images (Examples) x_i



Autoencoder

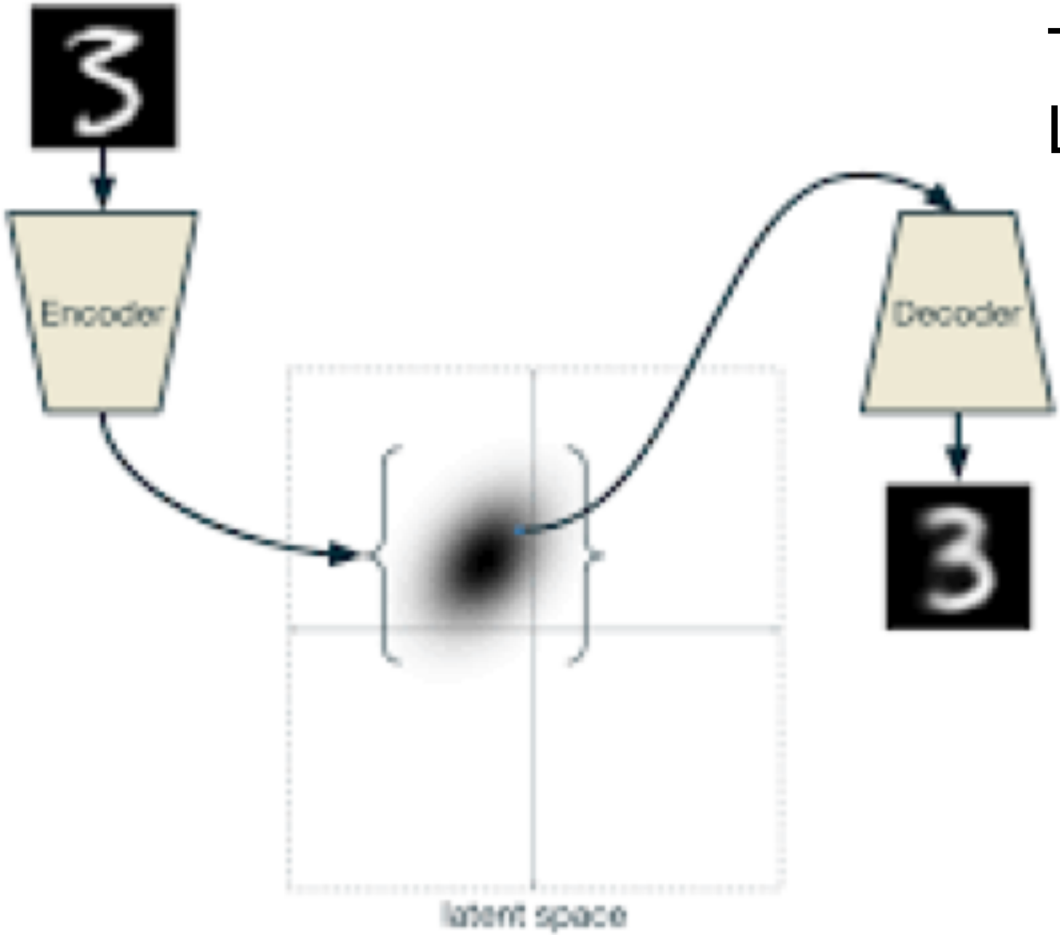


Features: $\Phi_1, \ldots \Phi_k$

[Ng, 2013]

- "Invents" (learns) higher level features (e.g., edges)
- Sparsity forces specialization (feature detector) of each unit
- Alternative Non supervised learning architectures, e.g., Restricted Boltzman Machines (RBM) [Smolensky 1986] [Hinton et al. 2006]

Variational Autoencoder

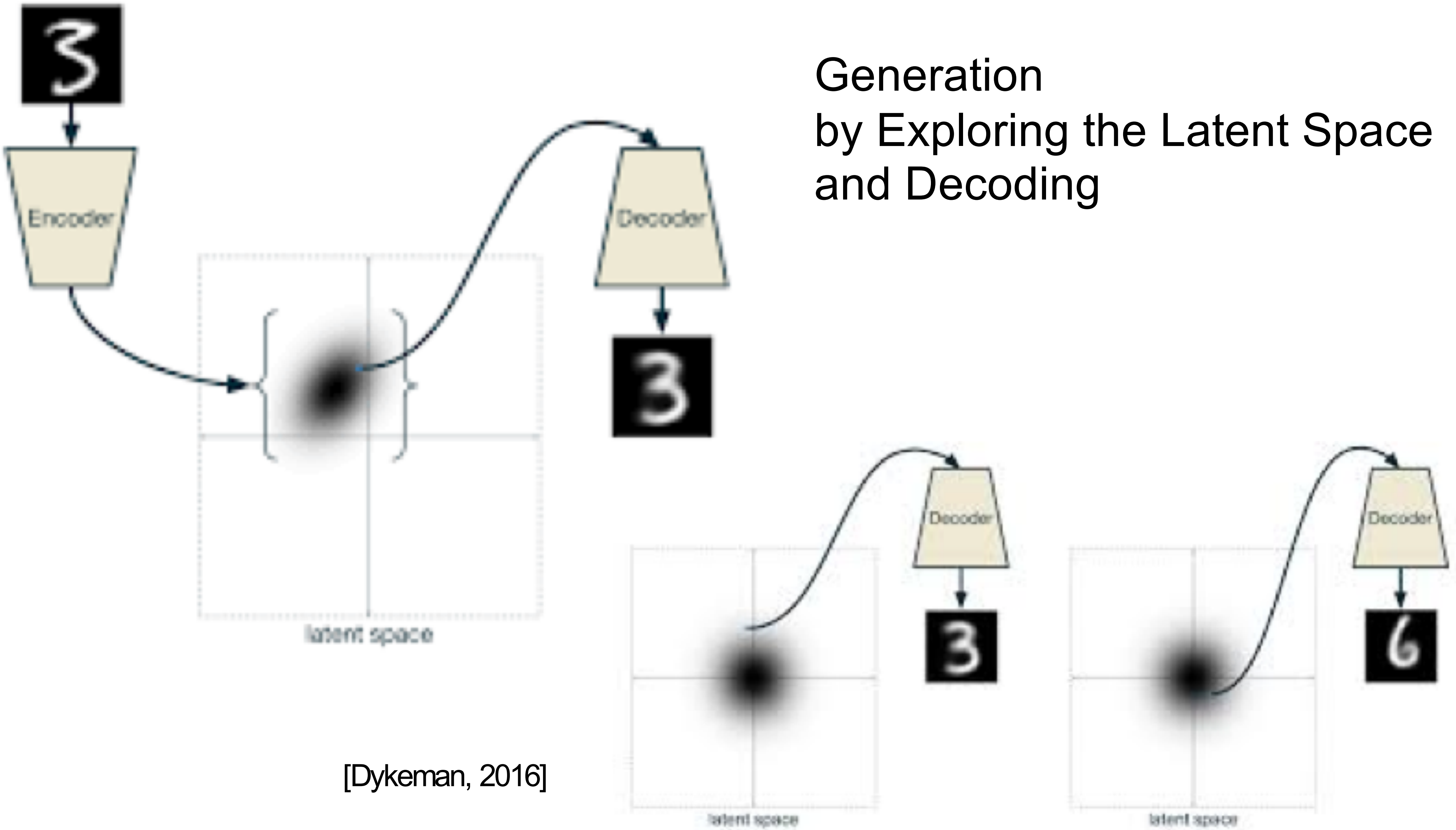


Training
Learning a Latent Representation

[Dykeman, 2016]

Variational Autoencoder

Generation
by Exploring the Latent Space
and Decoding

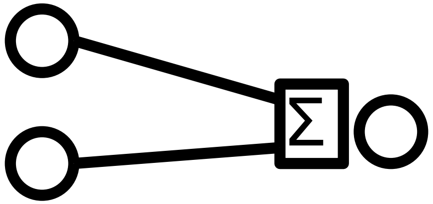


[Dykeman, 2016]

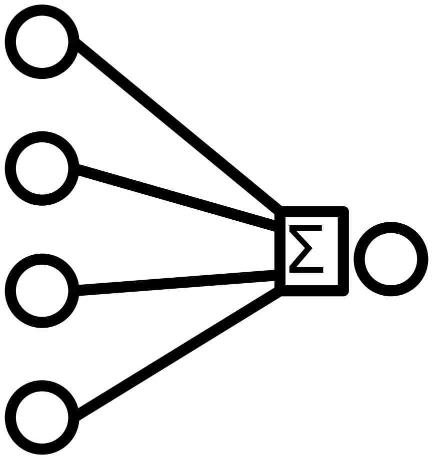
Variational Autoencoder

Deep Learning Bio-inspired or/and Regression-based ?

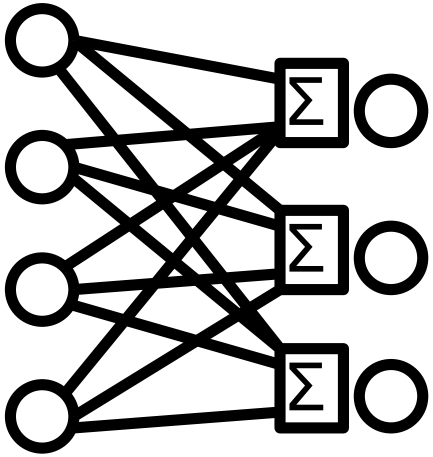
Simple Linear Regresssion



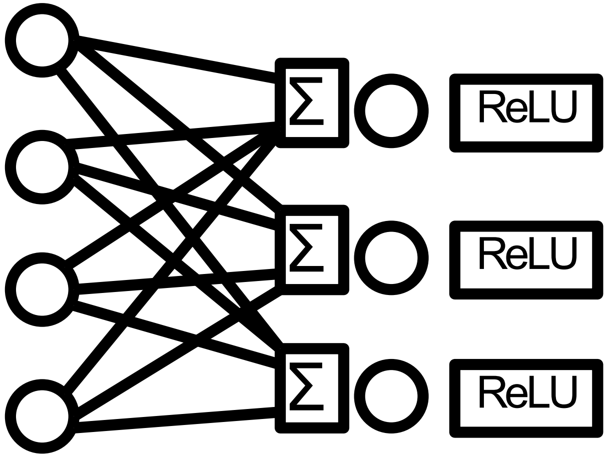
Multiple Linear Regression

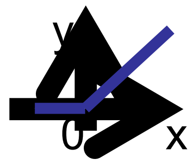


Multivariate Multiple Linear Regression



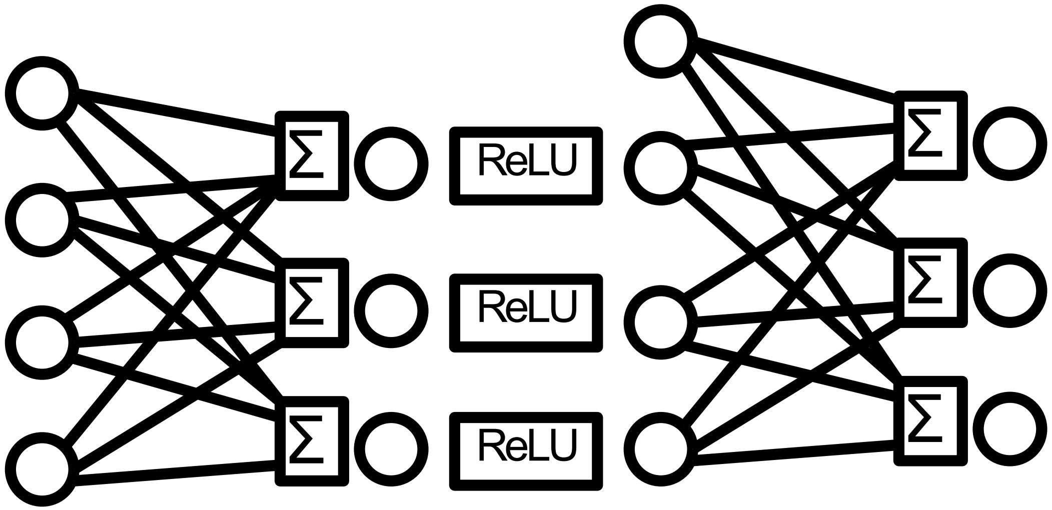
Non Linear Multivariate Multiple Linear Regression



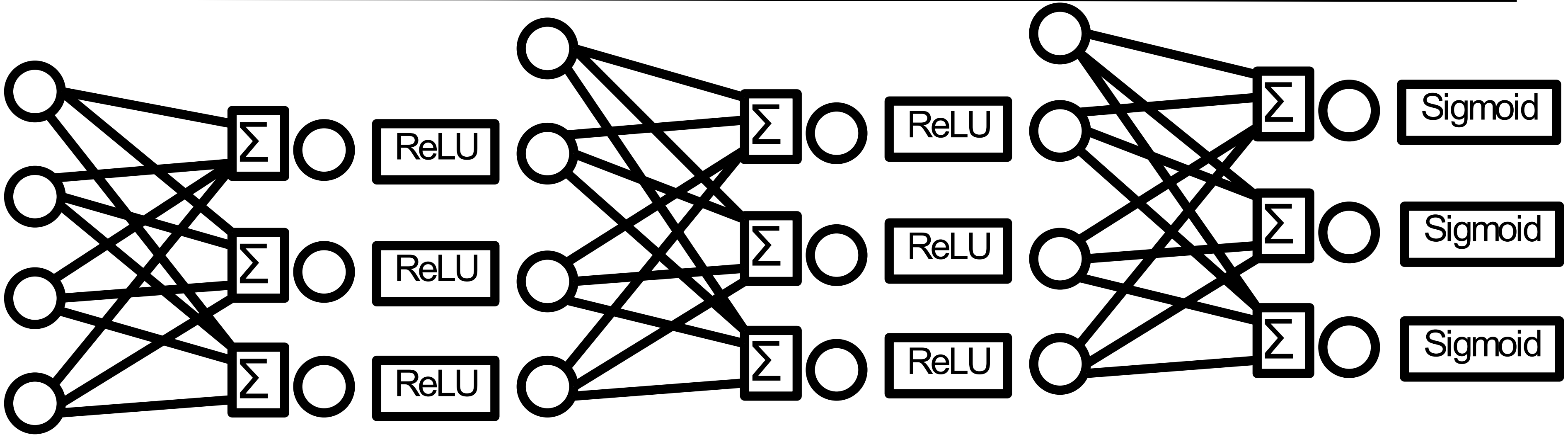


ReLU = max(x, 0)
Rectified Linear Unit

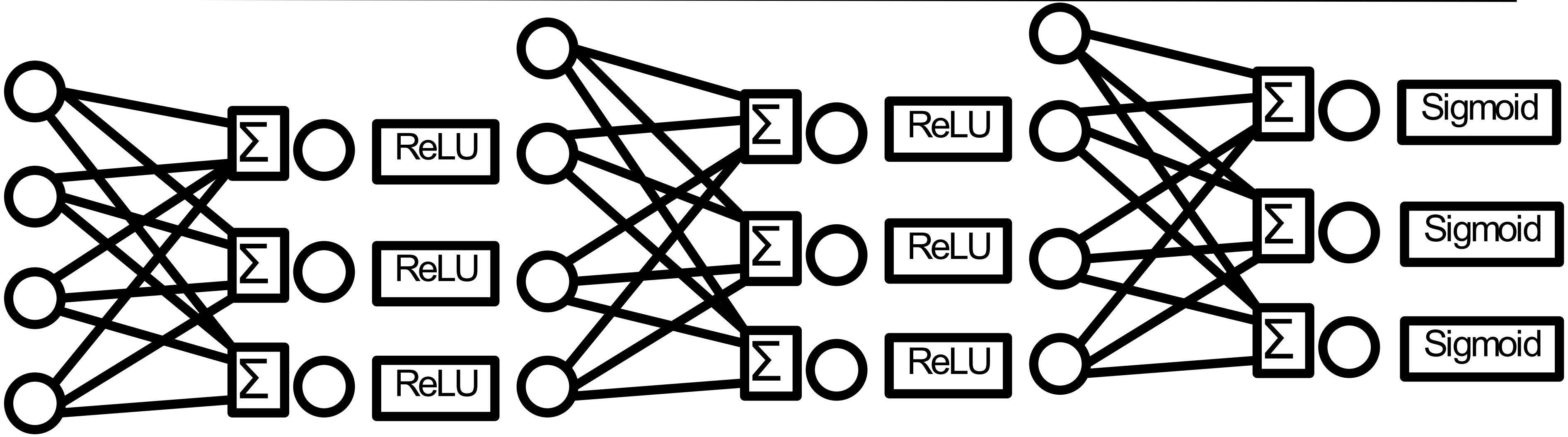
Multilayer Non Linear Multivariate Multiple Linear Regression



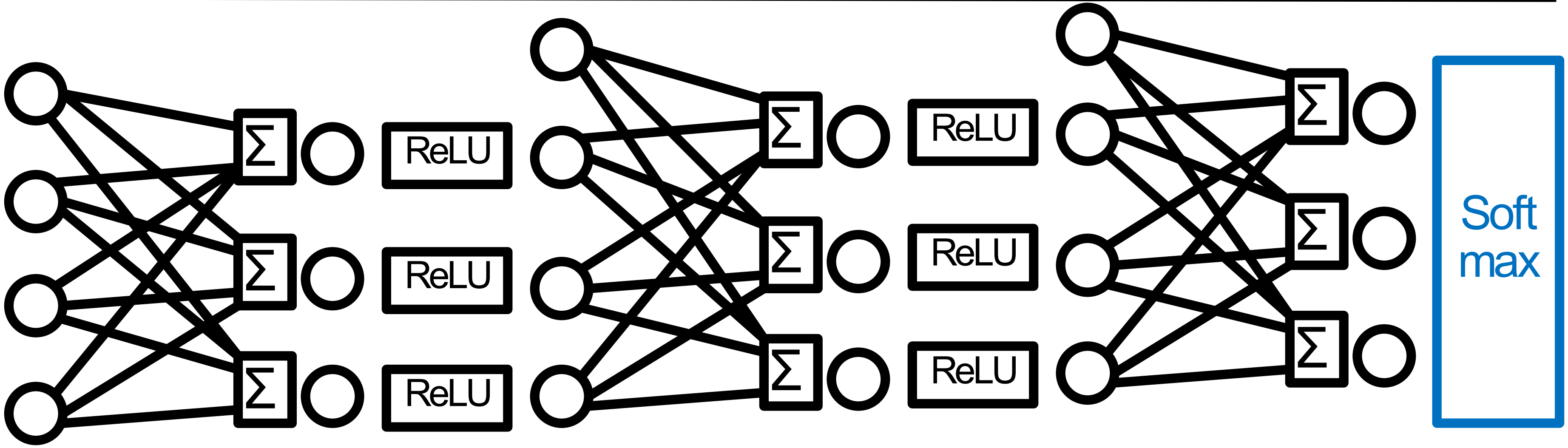
Multilayer Non Linear Multivariate Multiple Linear Regression



= Neural Network



Modern Neural Network



Softmax transforms values into probabilities

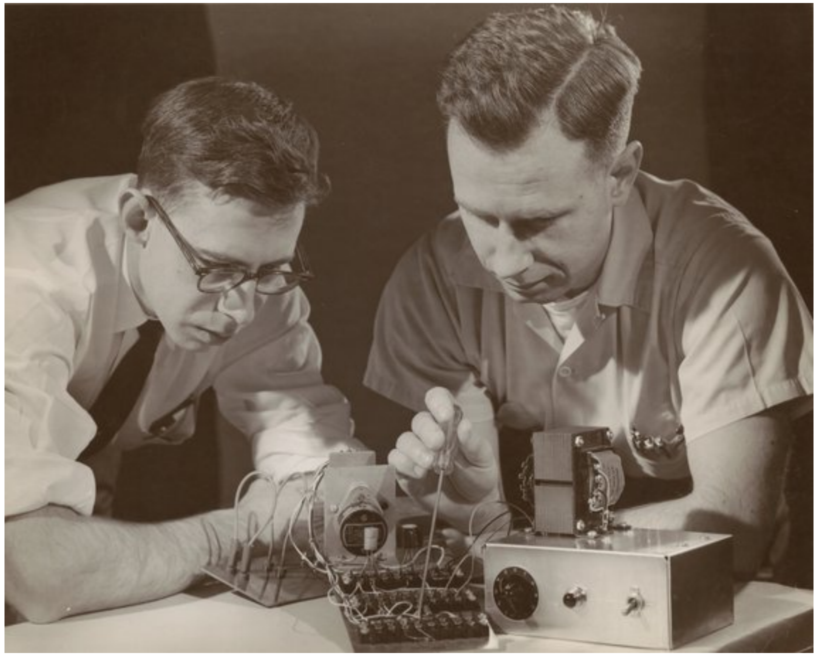
$$\sigma(z)_i = e^{z_i} / \Sigma\, e^{z_j}$$

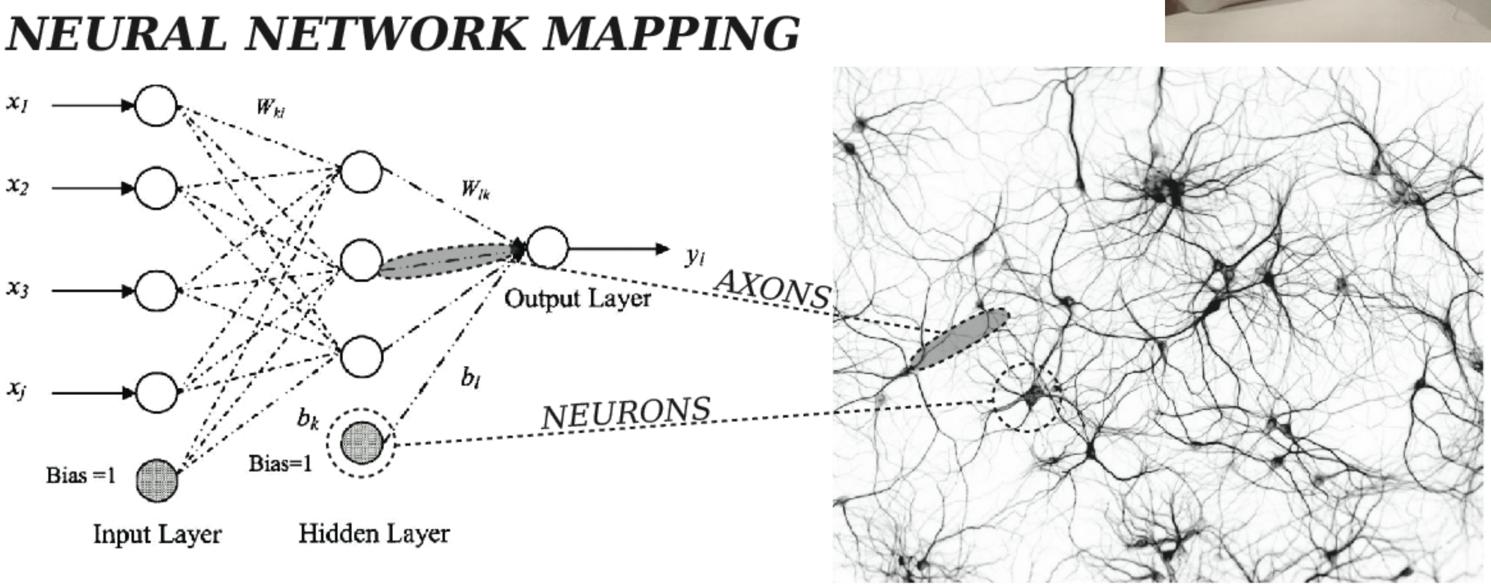
Generalization of sigmoid

Associated cost function is cross entropy

Deep Learning Bio-inspired or/and Regression-based ?

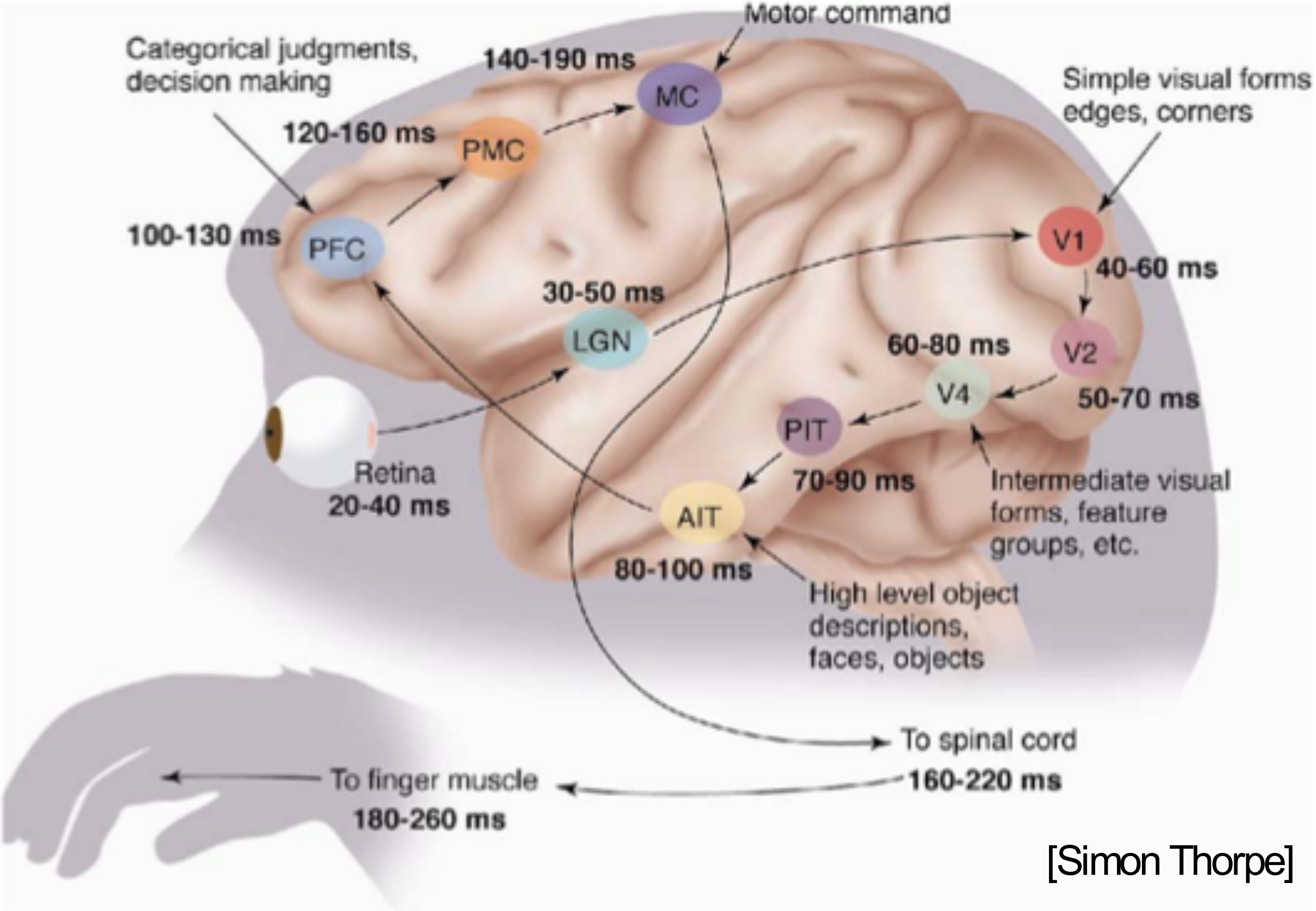
- Historically Perceptron bioinspired





Analogy with Mammalian Visual Cortex

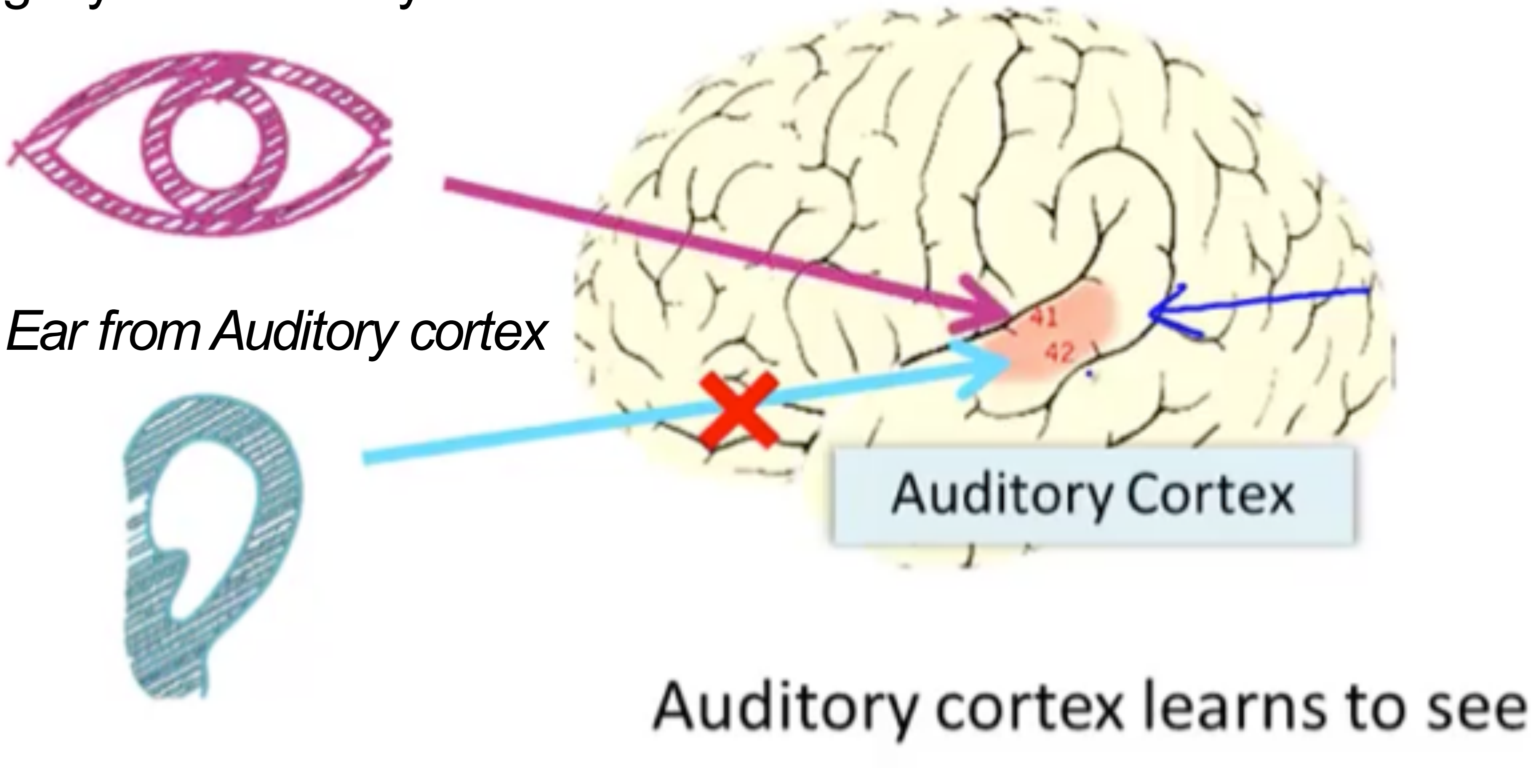
- Hierarchical
- Multiple Successive Stages
- Intermediate Representations (Features)



"Universal Learning" Biological Neural Network Hypothesis

Reconnecting Eye to Auditory cortex

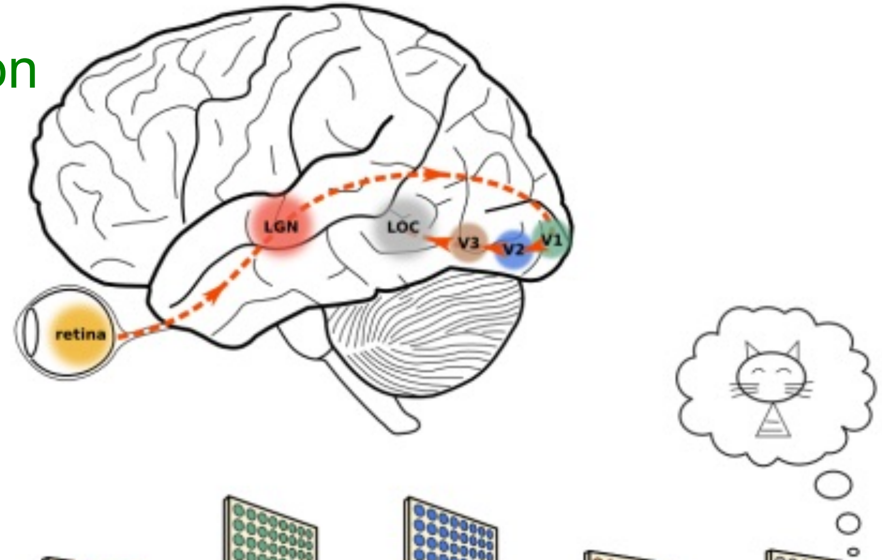
Disconnecting Ear from Auditory cortex

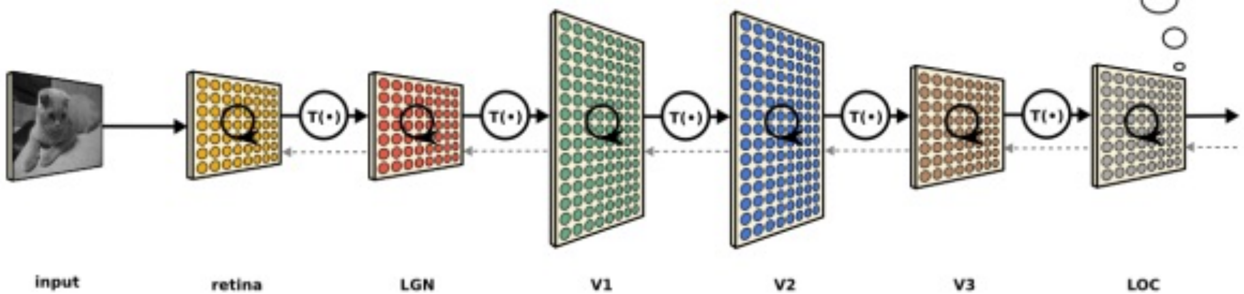


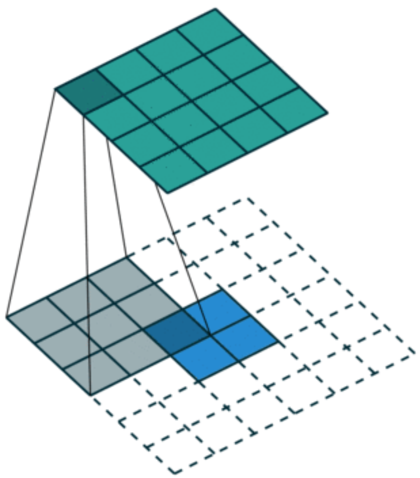
[Ng 2011]

Deep Learning Bio-inspired or/and Regression-based ?

- Convolutional Neural Networks (ConvNets [LeCun et al. 1989]) are bio-inspired
  - Inspired by animal visual cortex [Fukushima1980]
  - Complex neuron cells respond to stimuli in receptive fields (restricted region)
  - Receptive fields partially overlap
  - Allows tolerance to input image translation
    - » Weight sharing
    - » Translation invariance
    - » Subsumes (and thus avoids) segmentation
    - » For locally correlated data





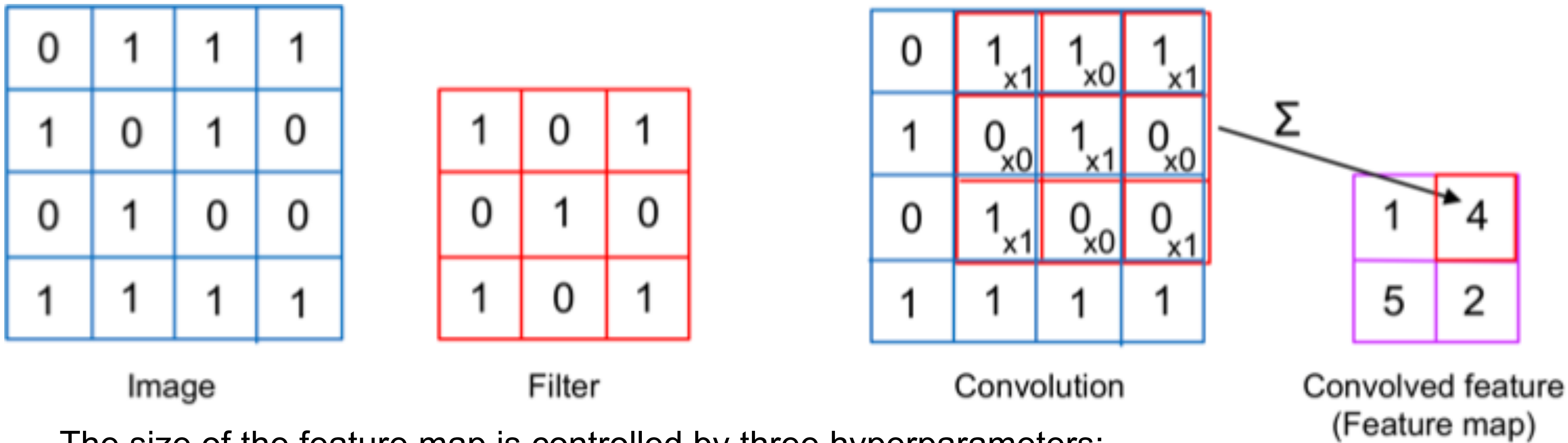


Jonas Kubilias

Convolution

Convolution

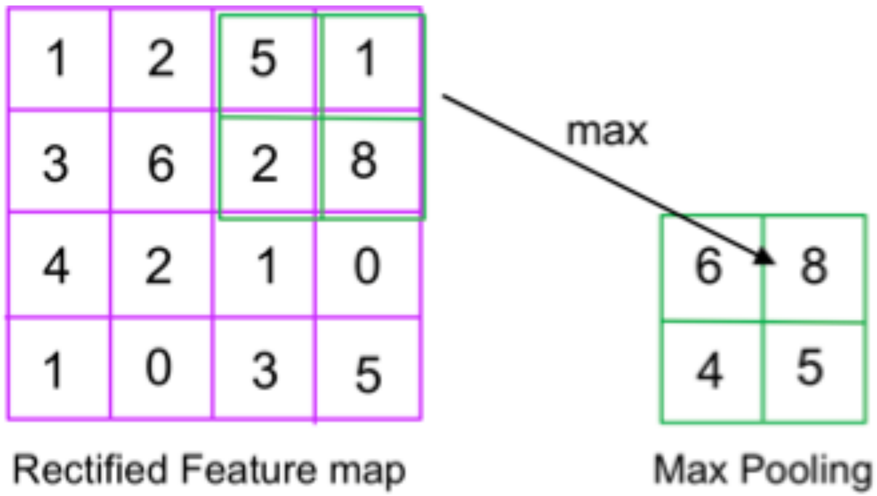
- Slide a matrix (named filter, kernel, or feature detector) through the entire image
- For each mapping position:
  - compute the dot product of the filter with each mapped portion of the image
  - then add all elements of the resulting matrix
- Resulting in a new matrix (named convolved feature, or feature map)



- The size of the feature map is controlled by three hyperparameters:
  - Depth: the number of filters used
  - Stride: the number of pixels by which we slide the filter matrix over the input matrix
  - Zero-padding: if the input matrix is padded with zeros around its border

Pooling

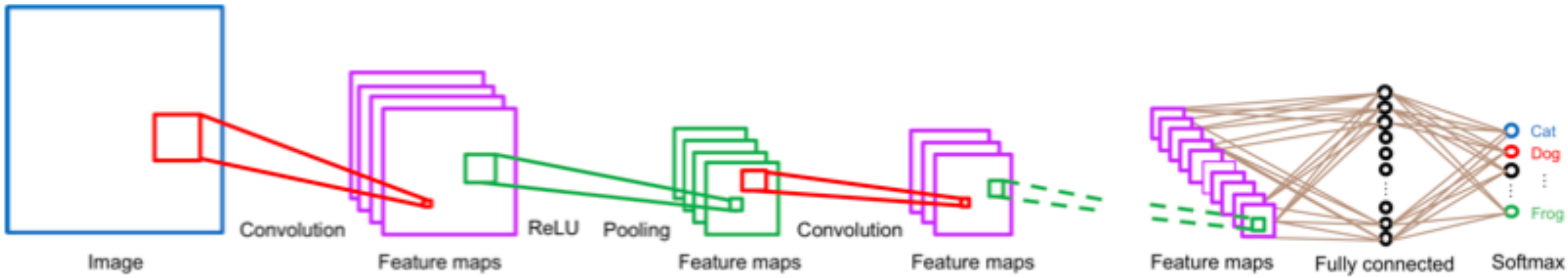
- Reduce the dimensionality of each feature map, while retaining significant information
- Standard operations:
  - max
  - average
  - sum



- Parameter sharing used by convolution (shared fixed filter) brings equivariance to translation: a motif in an image can be detected independently of its location
- Pooling brings invariance to small transformations, distortions and translations in the input image

Convolution

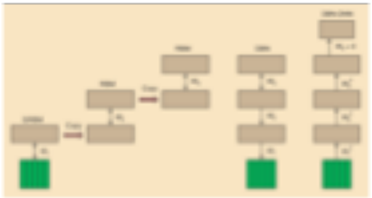
- Full Convolutional architecture:
  - Successive Layers, each Layer with 3 Stages:
    - » Convolution stage
    - » Non linear rectification (ReLU) stage
    - » Pooling stage



Techniques

Techniques

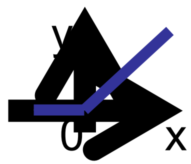
- Pre-Training



- Efficient Hardware (GPUs)

- Trading Space for Time (and Breadth for Depth) [Bengio & LeCun 2007]
  - More Layers, Cascade/Sequence

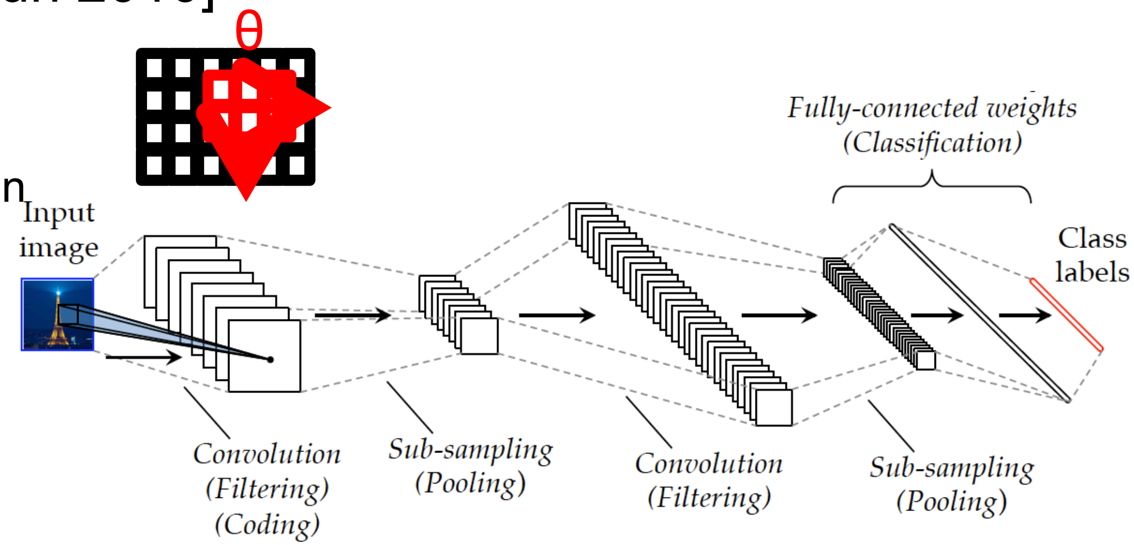
- Additional Techniques and Heuristics:
  - Convolution Neural Networks, Clustering, Pooling
  - Recurrent Networks... (See next slides)
  - Memory (LSTM, etc. See next slides)
  - Dropout (as a Regularization technique)
  - ReLU rather than Sigmoid in Hidden Layers
  - SoftMax final Layer
  - Restricted Boltzman Machines
    - » Contrastive Divergence Gradient estimation algorithm



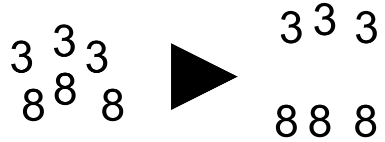
ReLU = max(x, 0)

Other Variations/Techniques

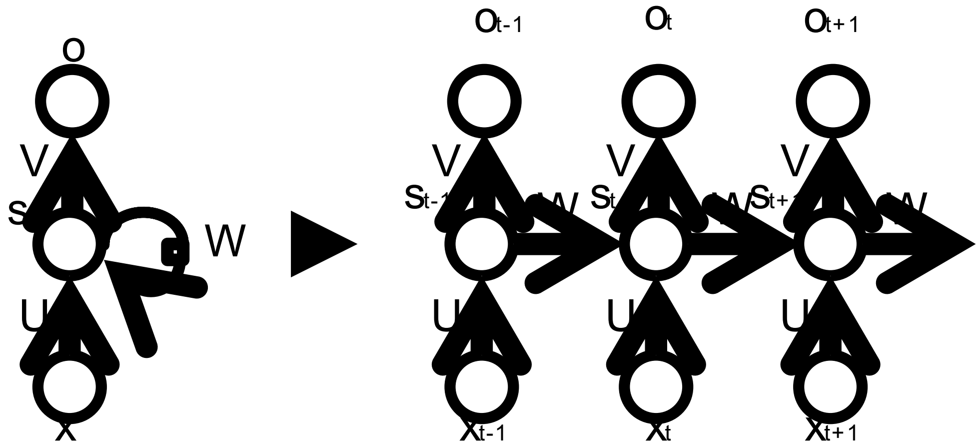
- Convolutional Neural Networks [LeCun 2010]
  - Weight sharing
  - Translation invariance
  - Subsumes (and thus avoids) segmentation
  - For locally correlated data



- Pooling (Compress/Factorize)
- Clustering (Group)



- Recurrent Networks
  - loop+delay/memory
  - for temporal or sequences of data
  - ex: Natural Language Processing

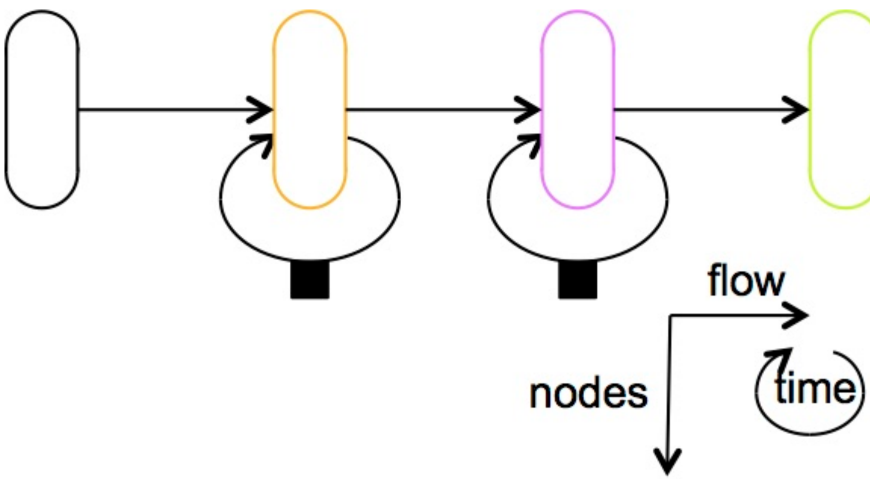


- Memory-Augmented Networks
  - LSTM, GRU, Neural Turing Machines
  - Memory access is differentiable thus trainable

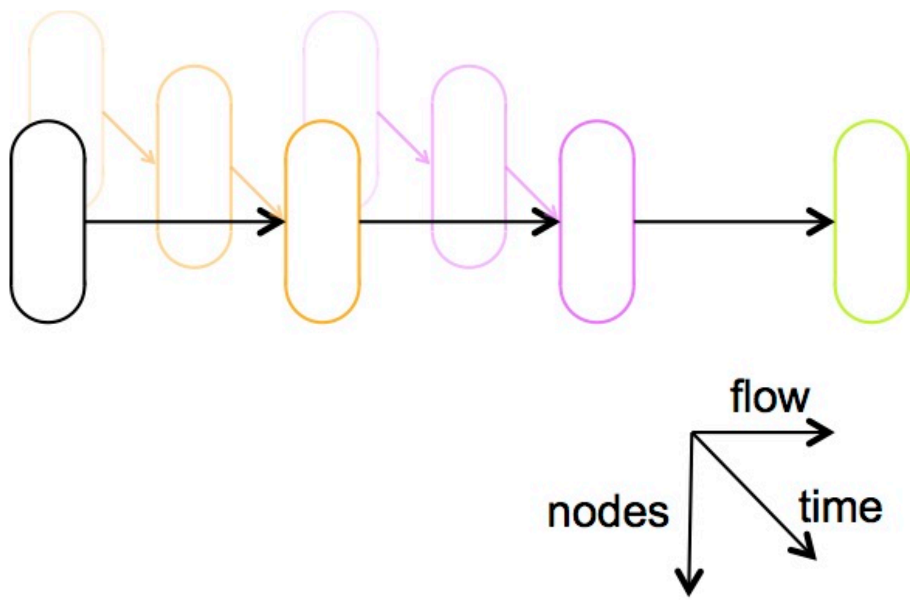
Recurrent Neural Networks (RNN)

- Recurrent connexion from a layer to itself
- Therefore, the layer learns also from its previous state
- It acts as a memory
- The RNN can learn sequences

Folded



Unfolded

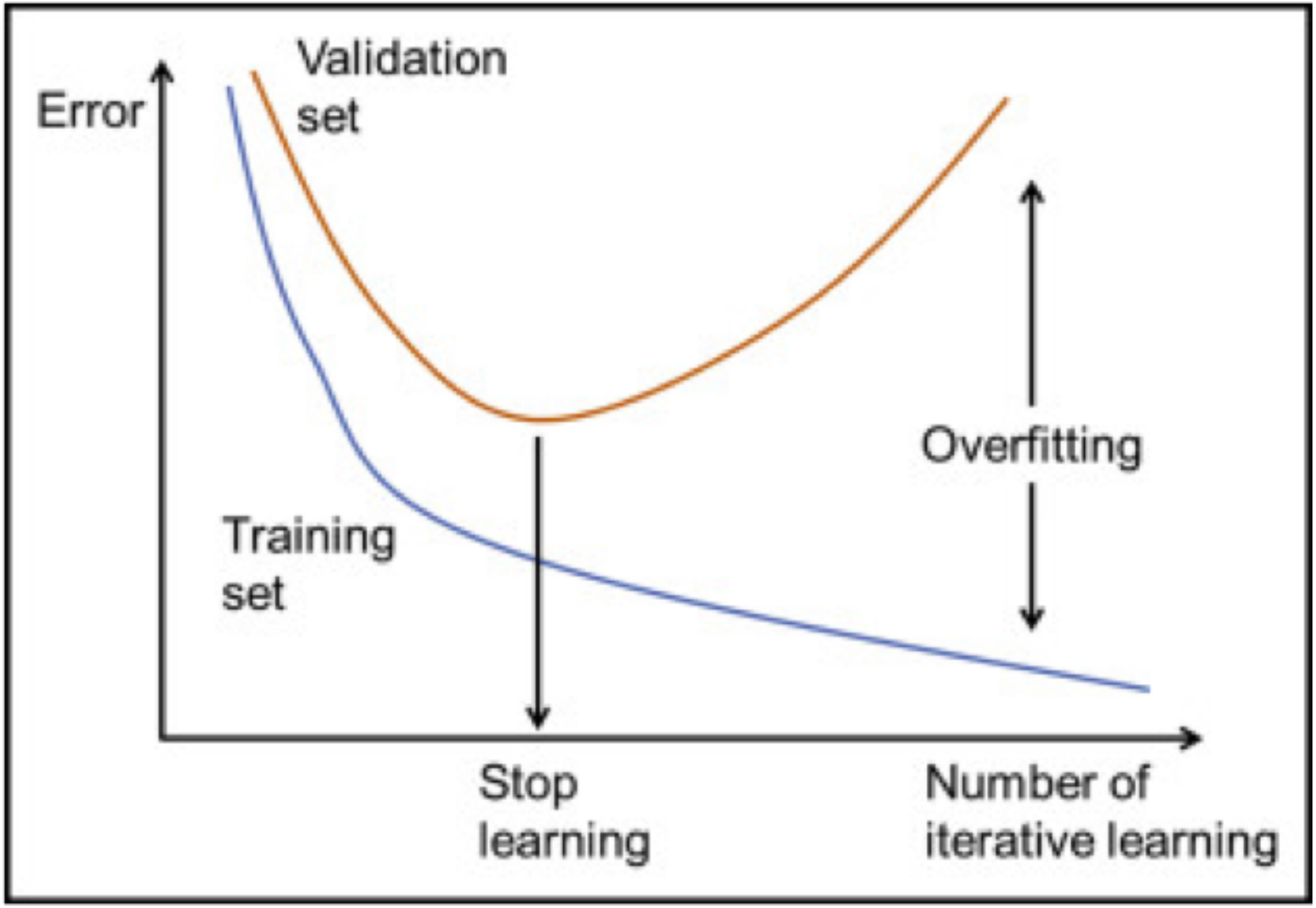


Engineering

Hyperparameters

- Fine Tuning of Hyperparameters of the Architecture/Model
- Architecture Structure
  - Number of layers
  - Number of units
  - Non linear Function
- Learning Heuristics
  - Cost (Prediction Error): Quadratic, Log, Cross Entropy...
  - Optimization: Gradient Descent, Stochastic, Mini-Batch, Adagrad, Adadelta...
  - Leaning rate
  - Regularization: L1, L2, Dropout...
- Convolution
  - Depth, Stride, Zero-padding
- LSTM
  - Input gate, Output gate, Forget gate biases
- Autoencoder
  - Sparsity
- Grid Seach for Exploring/finding good settings

Learning Curves



(Deep) Networks Platforms/Libraries

2 Levels:

- Modules: C, Matlab, Packages (Linear algebra…)
- Glue/Scripting/ADL: Architectural description for assembling modules (data flow graph) and system/application management

Examples:

- SN, Lush [LeCun & Bottou 1987]
  - Lisp-like (single language)
- Torch (Facebook, Google…)
  - LuaJIT, Python (PyTorch)
- TensorFlow (Google)
  - Python
- Theano (U. Montréal)
  - Python
- Keras
  - Superlayer of (both) TensorFlow and Theano
  - Python
- Azure (Microsoft)
  - Cloud
  - Drag and Drop