Deep Learning and Neural Network Architectures – Pt. I

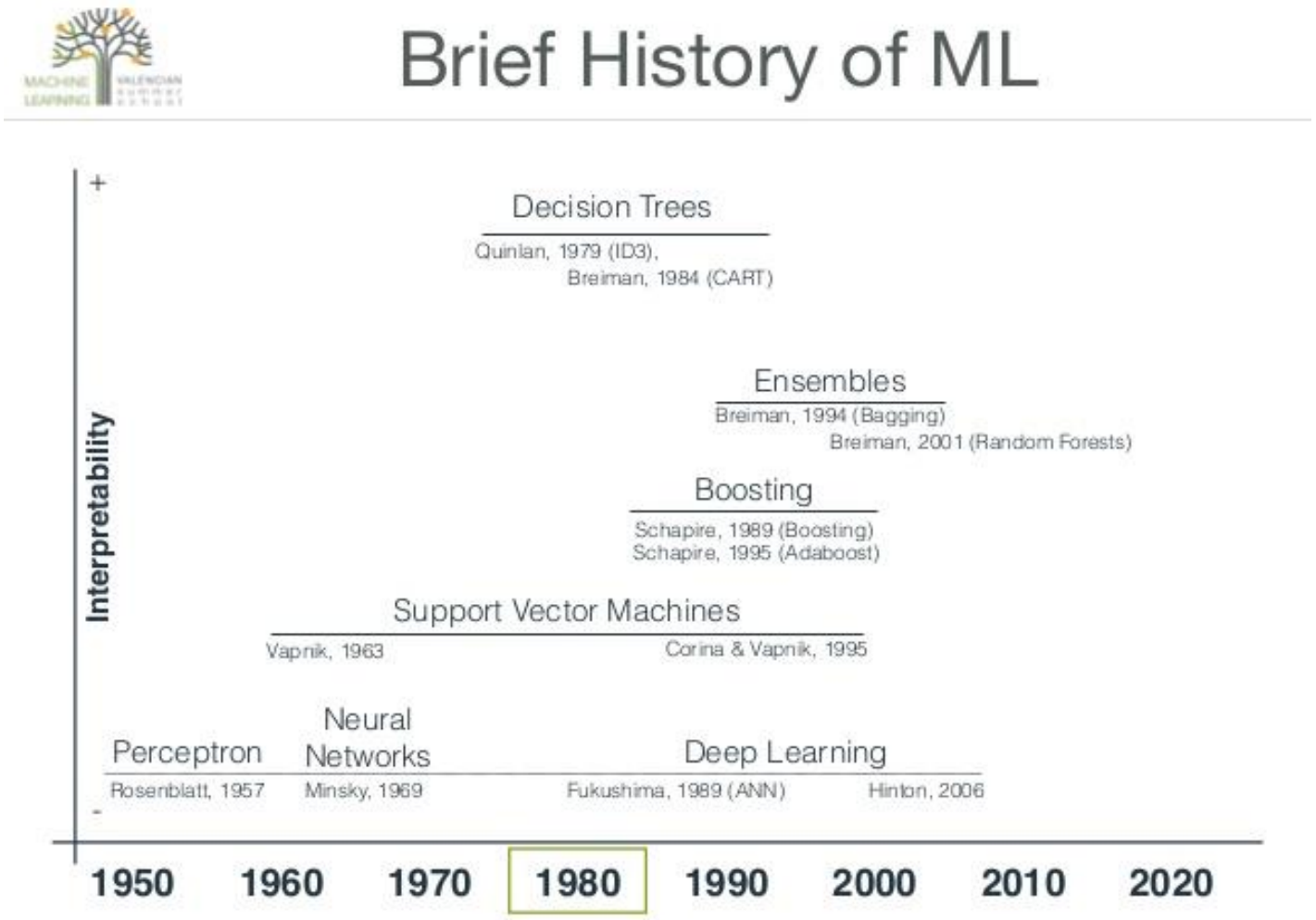
Prepared by Haihan – AQM 2018

Overview

- Learning Goals:
  - **Identify and describe the 3 major Machine Learning paradigms**
  - **Derive and implement Gradient Descent on simple neural net**
  - Understand the usage of different activation functions, loss functions and optimizers
  - Understand regularization, dropout and their effects on fitting
  - Understand variable selection and feature engineering on input data
  - Understand cross validation, bootstrapping, bagging and ensemble models
  - Implement Deep MLP network
  - Implement Convolutional Network
  - Implement LSTM Network
  - Understand the robustness and the pitfalls of these techniques
  - If time permits: SOMs, Word2vec, GANs, Deep Reinforcement Learning, TensorBoard, advanced ML paradigms

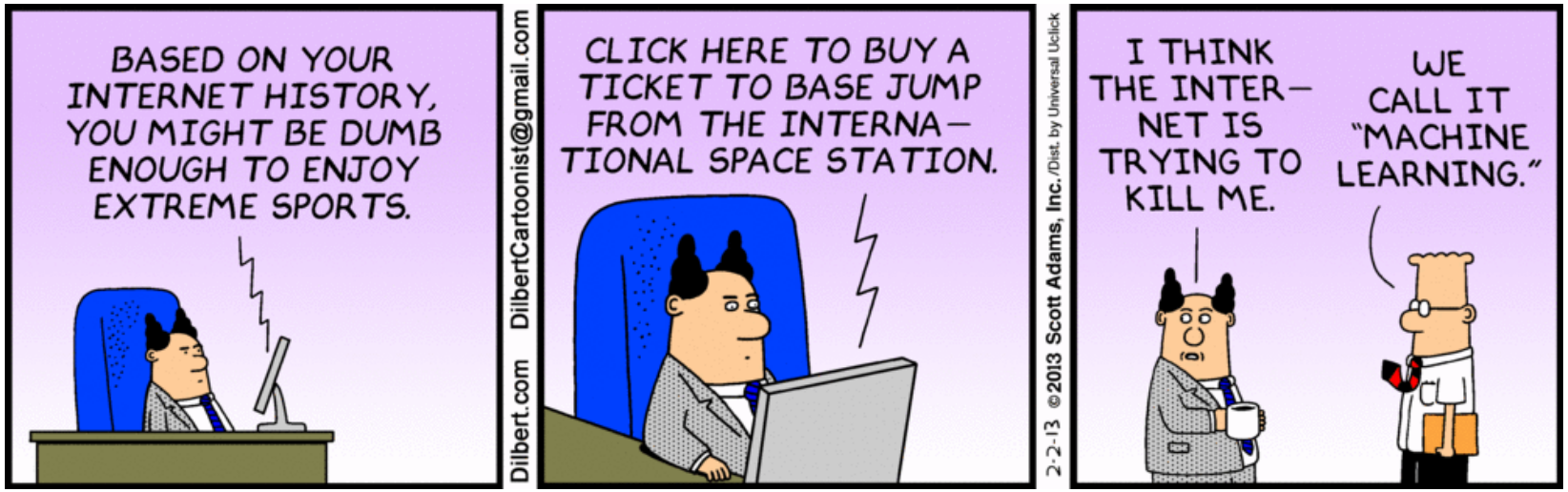
Introduction

- Also read http://www.erogol.com/brief-history-machine-learning/ if interested.



Introduction

- What Machine Learning ***isn't***:
  - A shortcut magic bullet, or a way to get 'free lunch' (https://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization)
  - Intelligent, in general
  - A rigid, static field of research where all the theorems have been proven already
  - Something you can use effectively without knowledge of fundamental theory and practical experience
  - Something your boss, manager, scrum product owner or the marketing team is likely to understand deeply

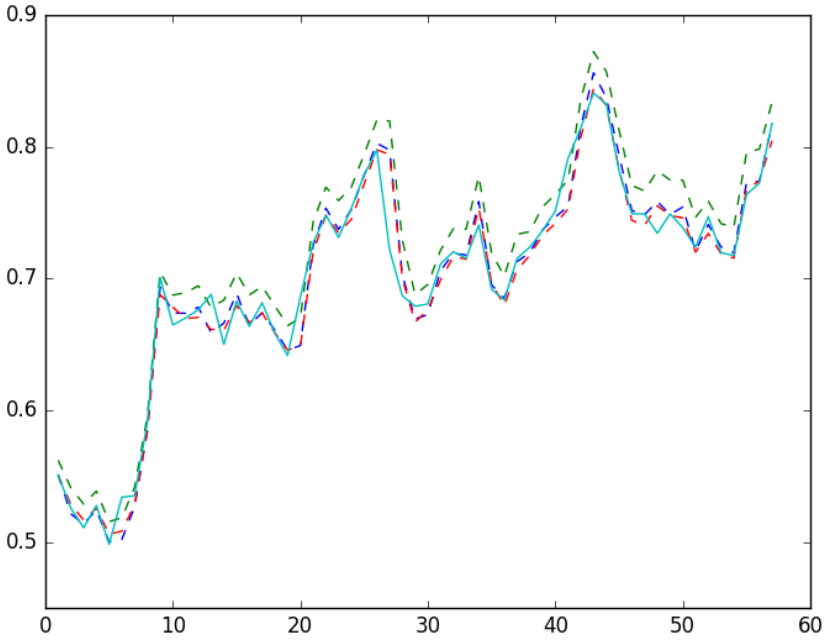


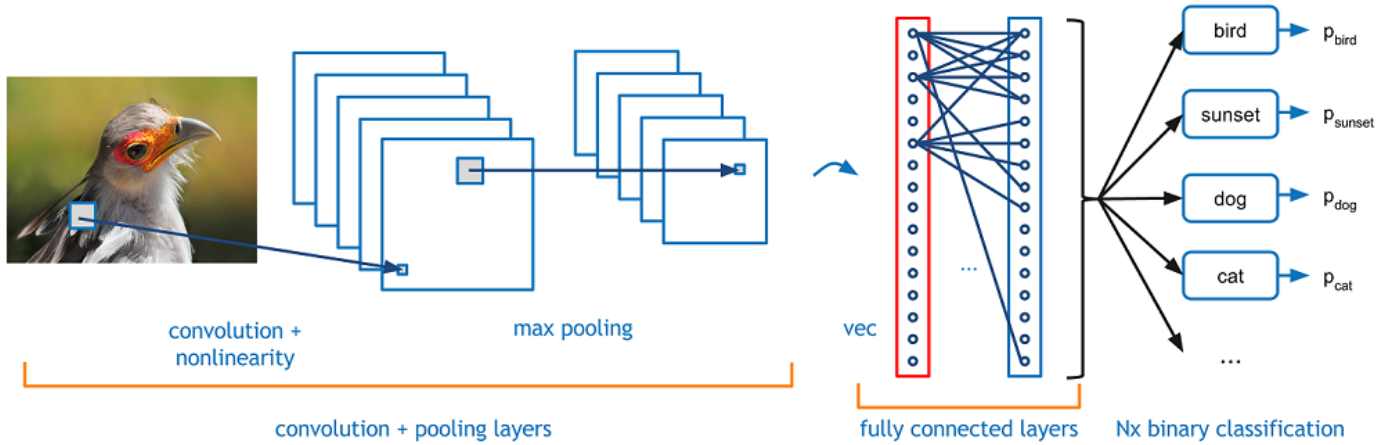
Supervised Learning

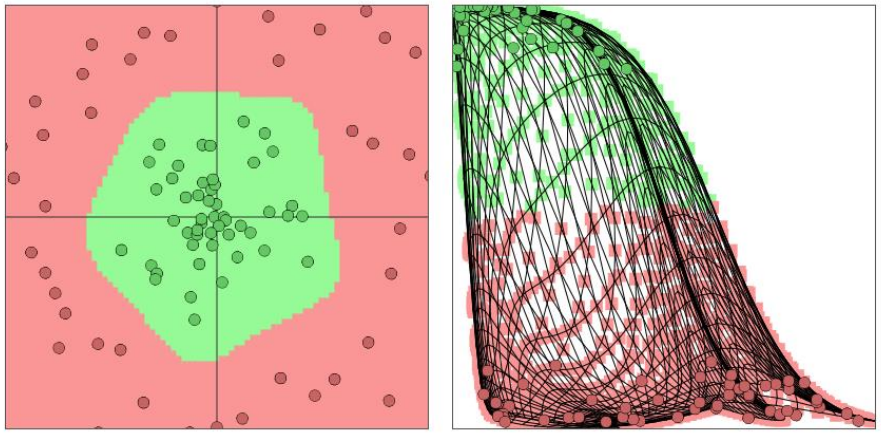
- Inferring or learning an unknown function *f(x)* from labelled data, or known input-output relationships
- Analogous to regression analysis in some aspects, but non-parametric in many cases
- Examples of supervised learning algorithms:
  - Support Vector Machines (blackbox)
  - Logistic Regression (interpretable)
  - Decision Trees and Random Forests (interpretable)
  - Perceptron Networks (blackbox)
  - Convolutional Networks (blackbox)
  - LSTM Networks (blackbox)
- In general neural net models when applied correctly can drastically outperform classical supervised methods on large and complex datasets
- Unlike regression analysis, generalization and extrapolation is important in Supervised ML

Supervised Learning

- Applications of supervised learning:





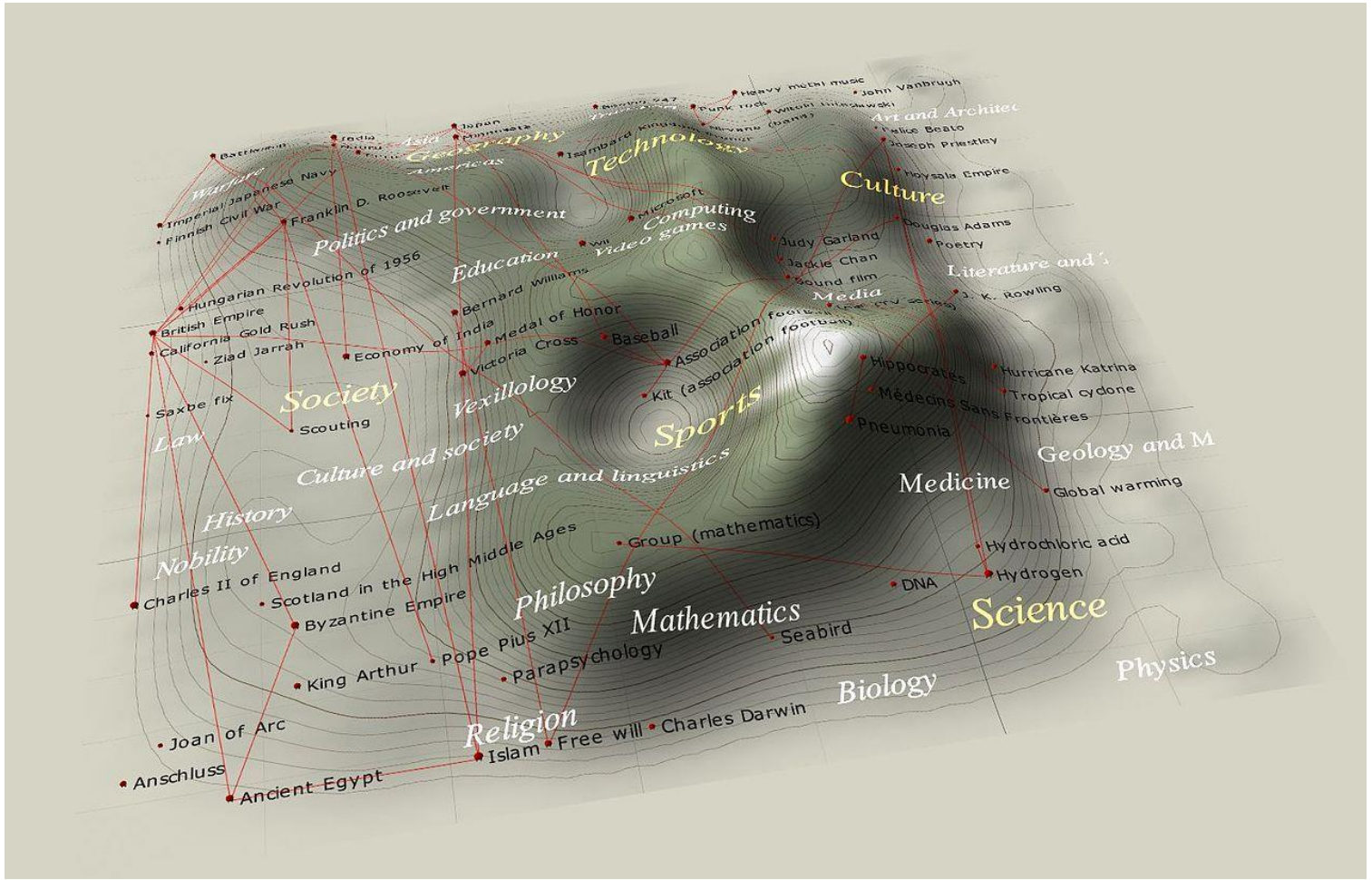


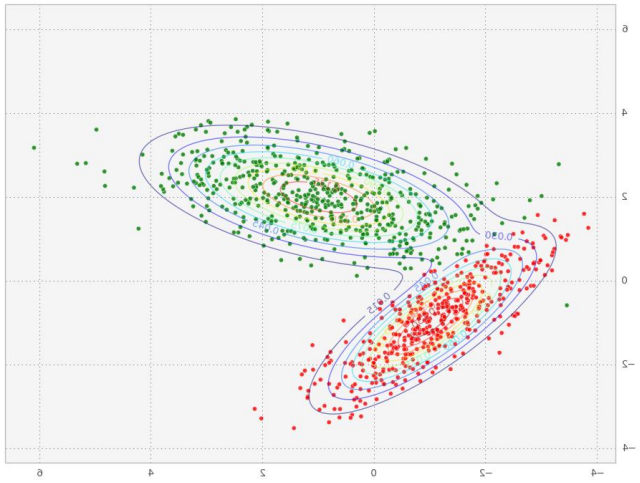
Unsupervised Learning

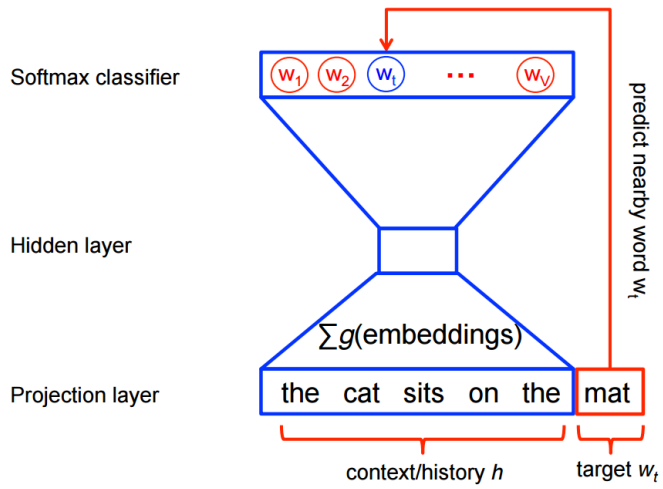
- Inferring or learning a clustering, structure, pattern or distribution from unlabeled data
- Clustering algorithms iteratively find clusters or groups in data using a specified similarity metric:
  - K-means
  - Gaussian Mixture Models
  - DBSCAN
  - Hierarchical Clustering
  - Self-organizing map (SOM)
- Other methods learn underlying structures and distributions of the data:
  - Autoencoder network
  - Restricted Boltzmann Machine and Deep Belief Networks
  - Word2vec
- Clusterings generated by unsupervised methods can be used to train supervised models

Unsupervised Learning

- Applications of unsupervised learning:





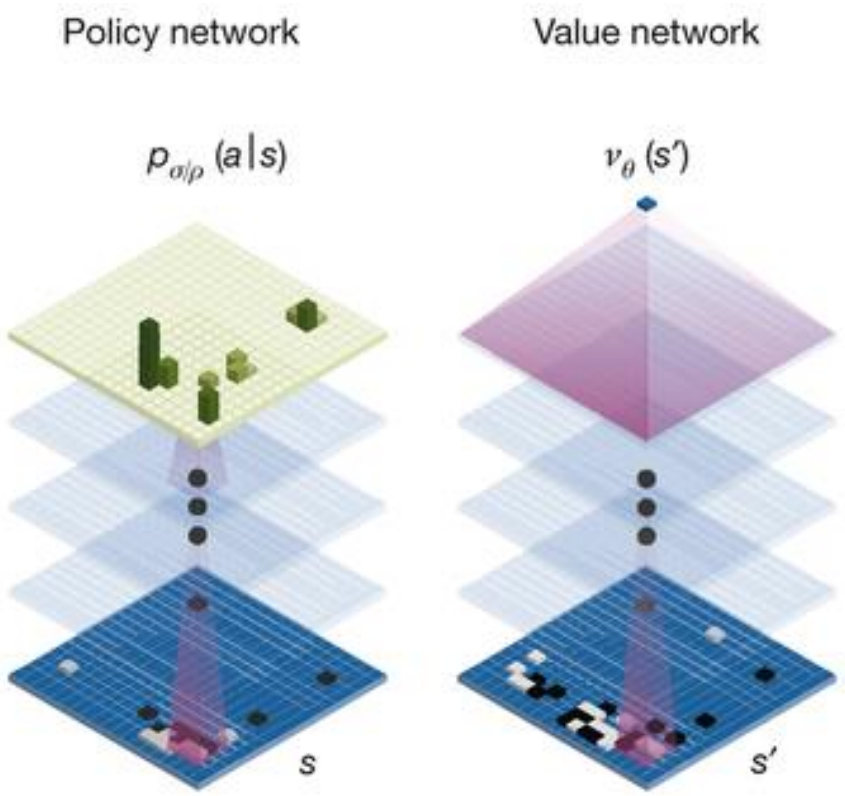


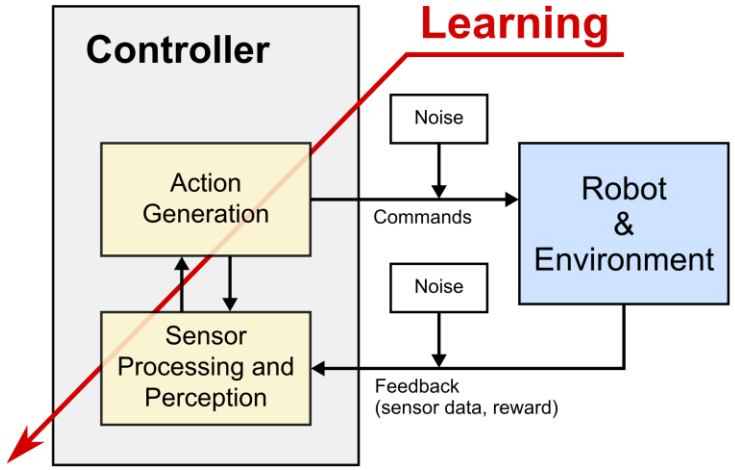
Reinforcement Learning

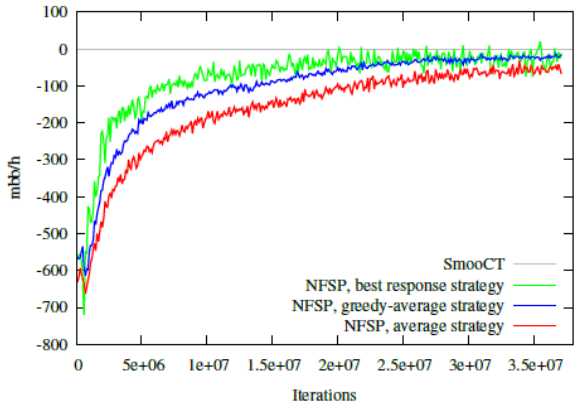
- Originally formulated as optimization problem with Markov Decision Process agents acting in an pre-defined Environment
- Objective is to iteratively visit state-action tuples and maximize estimated cumulative terminal reward
- RL is neither strictly supervised nor strictly unsupervised, can be combined with supervised methods to form Deep RL systems:
  - AlphaGo
  - Limit Texas Hold-’em
  - Controlling a robot to walk on mechanical legs
- RL can also be used to indirectly model other optimization problems, as long as the original optimization problem can be faithfully reformulated as an RL problem

Reinforcement Learning

- Applications of RL:



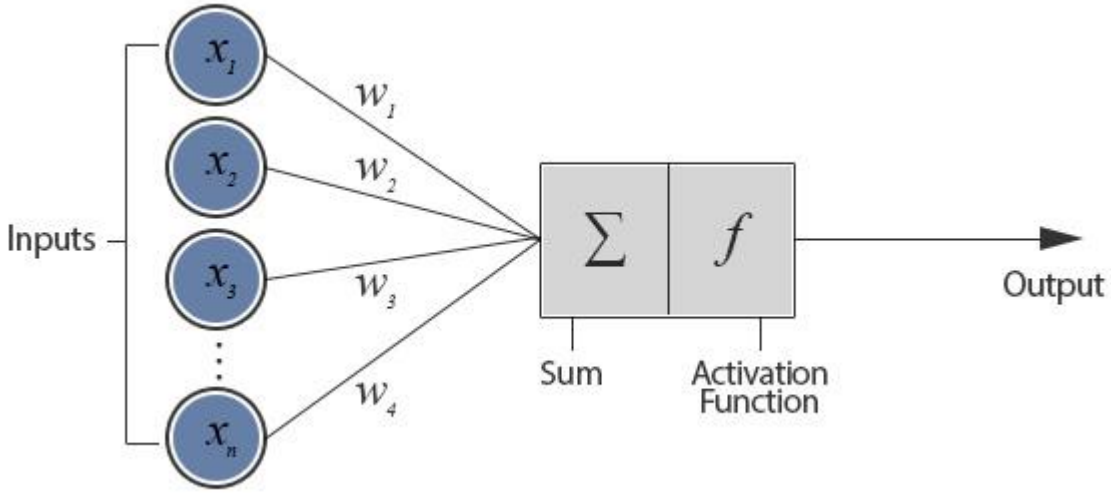




Match-up	Win rate (mbb/h)
escabeche	-52.1 ± 8.5
SmooCT	-17.4 ± 9.0
Hyperborean	-13.6 ± 9.2

The Basic Artificial Neuron

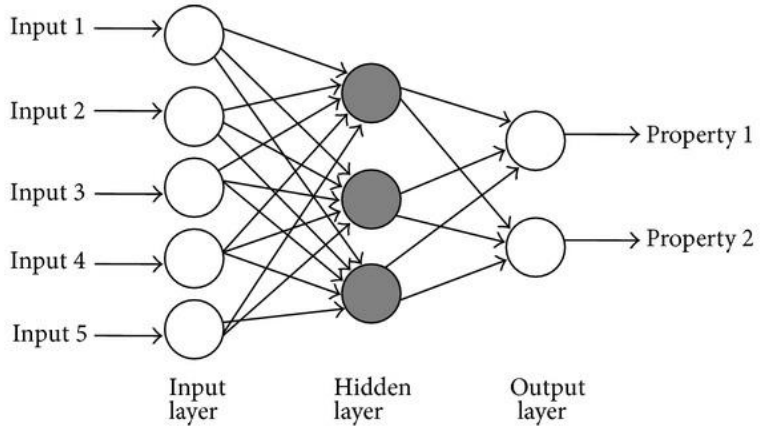
- The basic building block of any neural network is the artificial neuron. There are several different types, here we will cover them one of the most basic ones, the *Perceptron neuron*



- Symbolically, $y = \varphi(\mathbf{x} \cdot \mathbf{w} + \mathbf{b})$, where y is the numerical *output* of the neuron, $\mathbf{x}$ is the vector of *inputs*, $\mathbf{w}$ is the vector of the neuron's *weights*, $\mathbf{b}$ is the *bias* vector, and φ is the *activation function*

The Neural Network

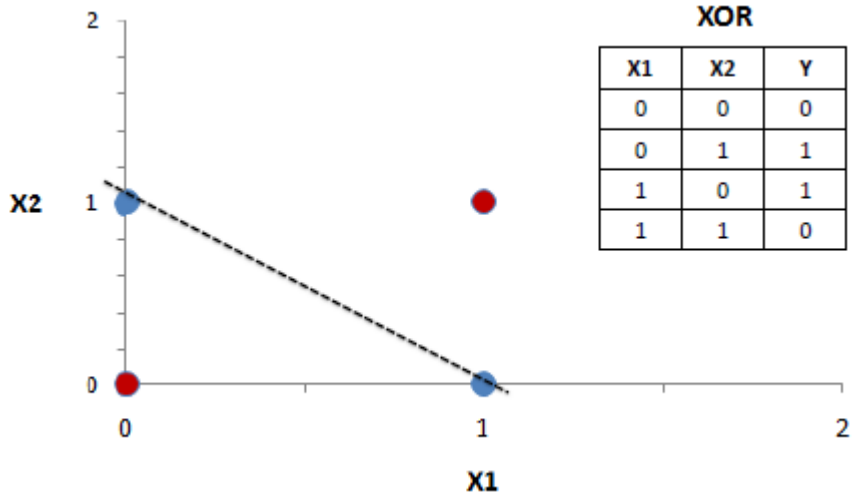
- Individual artificial neurons are connected in a specific way to form a network. One of the most basic neural networks is the Multi-layer Perceptron (MLP).



- In the network above, the input layer neurons are passthrough or dummy neurons, the hidden and output layer neurons apply the dot product and activation function as in the previous slide. For now, the connections are ‘dense’; each input is connected to every neuron in a layer.
- The MLP has some interesting theoretical properties, such as the ability to solve the XOR classification problem, and other nonlinear classification problems in general, as well the property of Universal Approximation.

The XOR problem

- The XOR problem is a simple learning problem widely used for benchmarking and theoretical understanding. The NN tries to learn the truth table of the XOR boolean function where X1 and X2 are inputs and Y is the output:



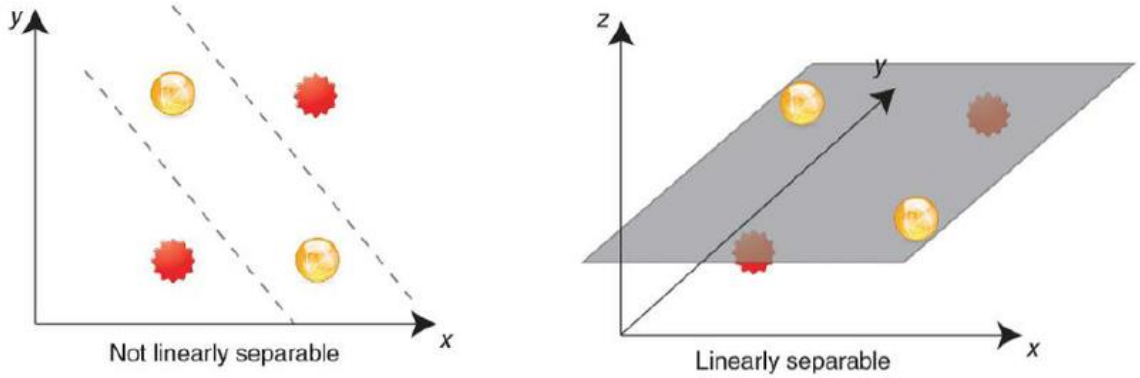
XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- You cannot draw a single line to completely separate the red dots and the blue dots into distinct regions. The XOR problem is *not linearly separable*.

The XOR problem

- What we can we do to solve this problem?
  1. Introduce another line, or in general a closed polygon shape of many lines to separate the regions
  2. Expand (embed) the problem to a higher dimension in order to separate the regions
  3. Apply a composition of linear followed by non-linear transforms until the problem becomes linearly separable. Somewhat like the Kernel trick for SVMs.



- In general, MLPs and Deep Networks (Networks with > 1 hidden layer) use a combination of all three methods. Neural nets are capable of highly complex *data representation*, and indeed this is what gives them their power.

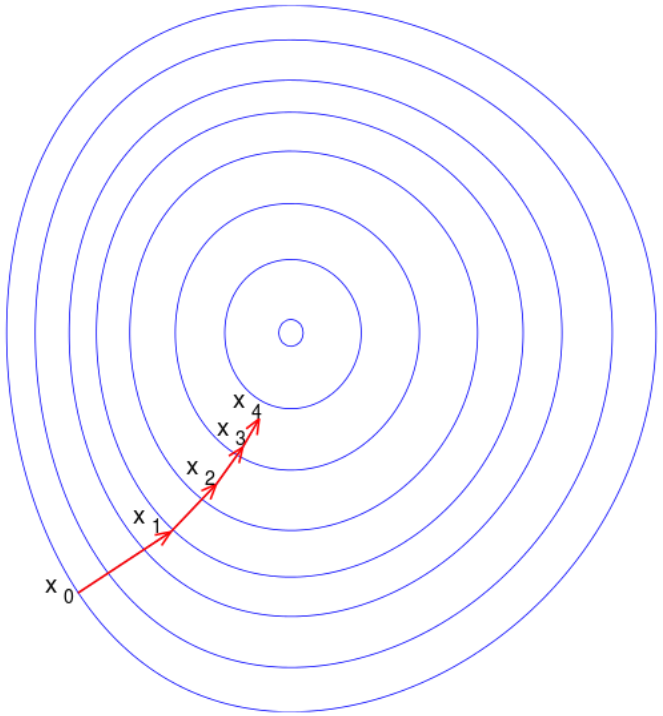
Training the Network

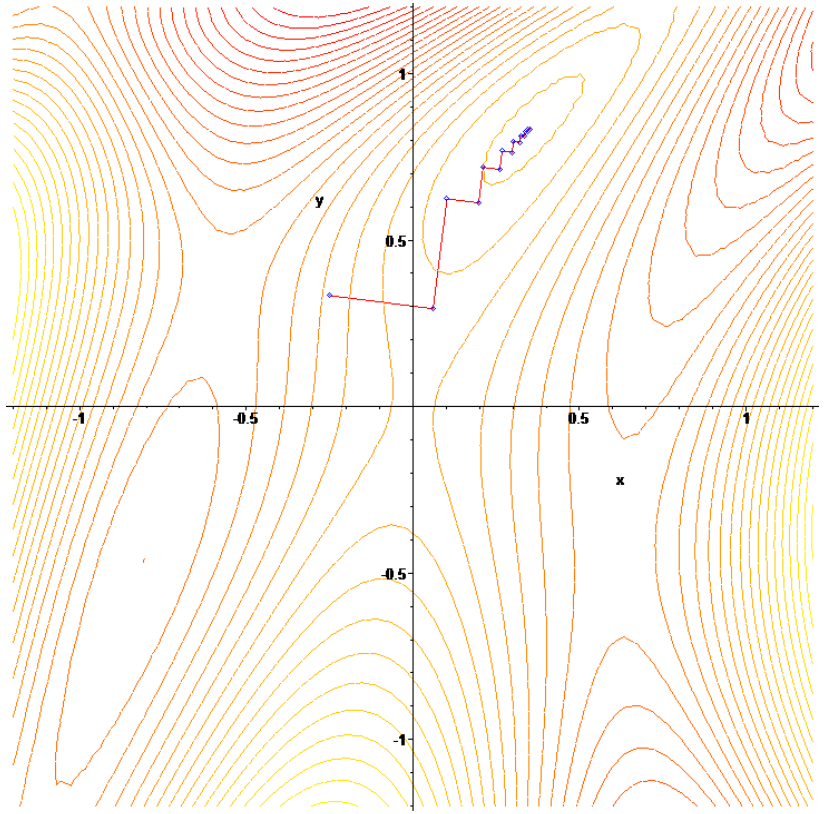
- Referring to the network in Slide 13, looking at the topmost output layer neuron:
  - $y_{o1} = \varphi(y_{h1}w_1 + y_{h2}w_2 + y_{h3}\,w_3 + b_1 + b_2 + b_3)$
  - This is just a simple composition of functions, where each $y = \varphi(\mathbf{x} \cdot \mathbf{w} + \mathbf{b})$ feeds into the next layer of neurons.
  - For each y, $\mathbf{x} \cdot \mathbf{w} + \mathbf{b}$ is a non-singular (why?) linear transform and φ is the subsequent non-linear transform.
  - The trainable parameters of the neural network (and the parameters that completely characterize it) are $\mathbf{w}$ and $\mathbf{b}$.
- Once a suitable *loss function* is specified, the gradient of each $\mathbf{w}$ and $\mathbf{b}$ w.r.t. the *error* can be found symbolically. Once these gradients are found, the error can be minimized. Finding the optimal or approximately optimal $\mathbf{w}^*$ and $\mathbf{b}^*$ is what training a neural net refers to.
- The error is given by the *loss function* (another term for objective function).

Training the Network

- One loss function that is typically used in both classification and regression is the Mean Squared Error (MSE) function:
  - $E_i = \frac{1}{2}(\mathbf{y}_i - \mathbf{t}_i)^2 \quad (1)$
  - here $\mathbf{y}_i$is the vector of the neural net's outputs at iteration i and $\mathbf{t}_i$ is the *target* vector at iteration i, E_i is the error at iteration i
  - The high dimensional 'shape' of the error surface given by the MSE loss is parabolic, but don't let this fool you. The Hessian Matrix of E w.r.t the parameters $\mathbf{w}$ and $\mathbf{b}$ will most likely be neither positive semi-definite nor negative semi-definite, implying neither non convex nor concave.
  - *Note: some sources define the MSE as $\frac{1}{2n}(y-t)^2$, where n is the number of training samples, but in Neural Net literature the definition given above is the one used. The factor $\frac{1}{2n}$ only scales the gradient and can be absorbed into the 'learning rate' hyperparameter for the same net effect*
- The objective is to *minimize* the error, or in general the difference between the outputs and the targets.
  - Minimize E w.r.t. the parameters $\mathbf{w}$ and $\mathbf{b}$
  - Do it the usual way: $\frac{\partial E}{\partial \mathbf{w}} = 0, \frac{\partial E}{\partial \mathbf{b}} = 0$
  - This method using the first derivatives to find a minimum is called *Gradient Descent*

Training the Network





Training the Network

- The update equations for $\mathbf{w_i}$ and $\mathbf{b_i}$ are:
  - $\mathbf{w_{i+1}} = \mathbf{w_i} - \alpha \frac{\partial E}{\partial \mathbf{w_i}}$ (2)
  - $\mathbf{b_{i+1}} = \mathbf{b_i} - \alpha \frac{\partial E}{\partial \mathbf{b_i}}$ (3)
  - Updates for the weights and biases are in the negative direction of the gradient (in the direction of the minimum) this is known as the *Delta rule* for updating the parameters.
- The *hyperparameter* α is a adjustable parameter that affects the learning rate of the network. Note α is not necessarily constant.
- Eqns (2), (3) and hyperparameter α are constitute the **Stochastic Gradient Descent (SGD)** method of training neural networks. Other update equations and hyperparameters are possible for different methods (called **optimizers** in Neural net parlance).
- The *Stochastic* property of SGD comes from the fact that we sample one data element at a time and perform a corresponding update on that one data element. Batch sampling and update are also possible (Slide 21)
- SGD is a general (first-order) method and can be applied to other models aside from Neural nets

Training the Network

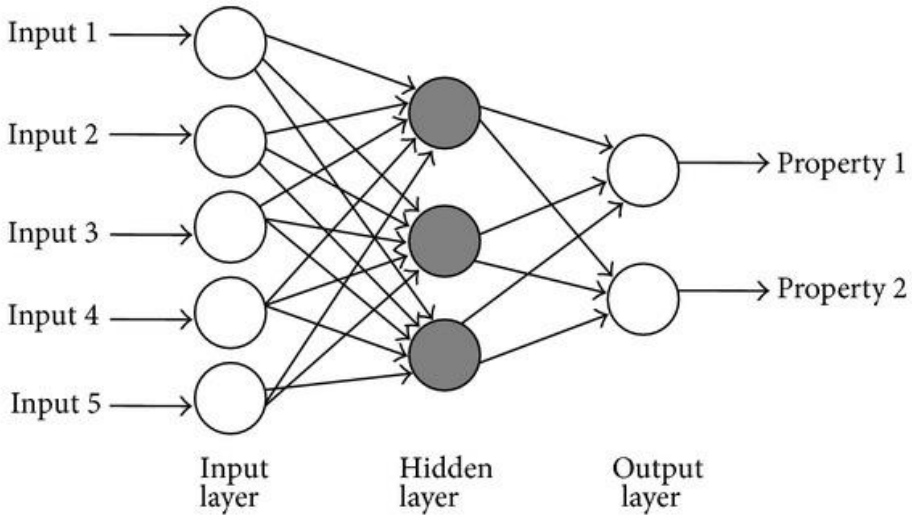
- General neural network training algorithm:

  Process and prepare data;
  While(convergence threshold not reached): [*epoch*]
    For(row i in dataset): [*iteration*]
      Feed row i to neural network;
      Obtain the network output; [*this and the previous step together are called 'feedforward'*]
      Apply optimizer on target and output;
      Update neural network parameters;
    End For
    Shuffle dataset;
  End While

- Usually the convergence threshold is a numeric value (such as 0.05) for E_i that is also numerically stable (i.e. settles to that value)

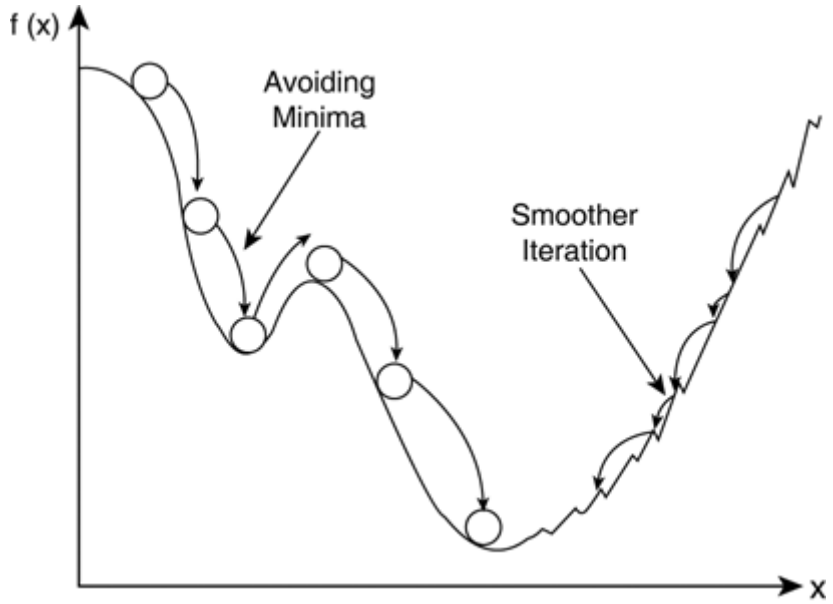
In Class Exercise

- Assuming MSE loss and the activation function φ is the unipolar logistic function $\varphi(x) = \frac{1}{1+e^{-x}}$
  - Find $\frac{\partial E}{\partial w}$ and $\frac{\partial E}{\partial b}$ where w and b are the first weight and first bias of the topmost output layer neuron in the network on the right-hand side
  - Repeat the above exercise where w and b are now for the topmost hidden layer neuron
  - You may find it useful to notate the w and b for the different neurons as w^{o1} or b^{h1}
  - Hint: $\frac{d\varphi(x)}{dx} = \varphi(x)(1-\varphi(x))$



Gradient Descent Momentum

- Most SGD implementations also include *momentum*, which helps filter out training noise and increase training performance. Momentum can be simply a 1st order low-pass filter applied to update equations (2) and (3), with its own momentum hyperparameter γ. The form of the 1st order low-pass is:
- $\mathbf{w}_{new} = (1-\gamma)\mathbf{w}_{old} - \gamma\alpha\frac{\partial E}{\partial \mathbf{w_i}}$ (this particular form is a discrete time 1-st order low pass filter, or an exponentially weighted moving average). Other more robust forms such as Nesterov Momentum are sometimes used.



Classification vs. Regression

- For regression problems, the number of output neurons is the same as the dimension of the target. The target can be a scalar, vector, matrix or tensor (a multi-dimensional array). The error of each output neuron is what is optimized over.
- For classification problems, categorical variables are one-hot encoded to an orthogonal basis.
  - Usually, something like {cat, dog, python} -> { [1 0 0], [0 1 0], [0 0 1] }
  - Training proceeds the same as with regression. When the network is used to predict, some decision function is used to map the output to a category eg. [0.1 0.2 0.93] -> {python}

Additional Notes

- Gradient descent works quickly when the objective function is simple and the dimensionality is low. However, depending on the dataset and neural net architecture, the objective function may be *non-convex*, noisy and high dimensional. Generally, the dimensionality of the objective function is dim($\mathbf{w}$) + dim($\mathbf{b}$).
- Updating weight and bias parameters after each single data row or element is called Stochastic Gradient Descent or SGD. There is a variant called *Batch SGD* where the parameter updates are average over n iterations and then applied as a single update.
- Regular SGD can use training noise to 'bump' the network out of a local minima traps and can make sometimes large changes to the parameters.
- Batch SGD tends to average out training noise when performing gradient descents and make more moderate updates to the weights. However, if n is too large, then the network may learn very slowly or completely diverge.
- $\mathbf{w}_{new} = \mathbf{w}_{old} - \frac{\alpha}{n}\sum_{i=0}^{n} \frac{\partial E}{\partial \mathbf{w}_i}$, here i represents each iteration in a batch and the updates are done at the end of each batch.
- Batch SGD where n is small offers a compromise between learning rate and performance. Usually problem spaces (dependent on dataset and neural net architecture) which are highly non-convex (i.e. many local extrema) are suitable for this 'mini-batch' approach.

Additional Reading

- Gradient descent and backpropagation, a more detailed explanation: http://neuralnetworksanddeeplearning.com/chap2.html
- A brief history of Artificial Neural Networks: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/
- Momentum and Learning Rate adaptation (variable learning rates) https://www.willamette.edu/~gorr/classes/cs449/momrate.html