CS-F407: Artificial Intelligence

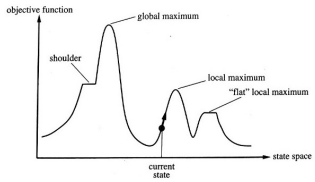
16 Spl Topics KNN, NN

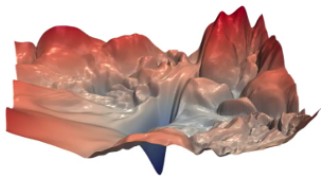
Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Nov 22, 2024 M/W/F - 3:00 pm 6101 @ BITS-Pilani [Aug-Dec 2024]

http://ktiwari.in/ai

Optimization





We essentially need **convexity**

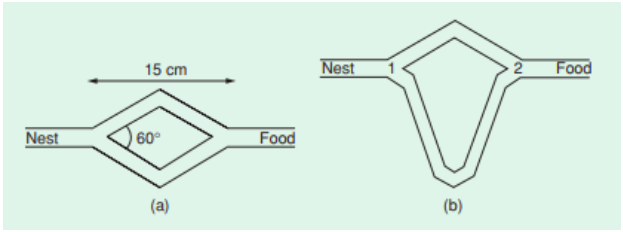
$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2)$$

for all $\alpha \in (0, 1)$

It is used in weight update as $w_i = w_i - \alpha \frac{\partial J}{\partial w_i}$

Ant Clony Optimization [1]

Swarm intelligence takes inspiration from the social behaviors of insects and of other animals for problem solving



Pheromone for probability,

[1] Ref-15420 Dorigo, Marco and Birattari, Mauro and Stutzle, Thomas. **Ant colony optimization**, IEEE computational intelligence magazine, 1(4), pp 28–39, 2006, IEEE

Particle Swarm Optimization [2]

Population based stochastic algorithm for optimization of nonlinear functions

1. Initialize particles
2. Calculate fitness of all particle and maintain best-till-now
3. Who has highest best-till-now
4. Update particle locations in direction of global fit

Conceptually, it seems to lie somewhere between genetic algorithms and evolutionary programming. It is highly dependent on stochastic processes, like evolutionary programming. The adjustment toward pbest and gbest by the particle swarm optimizer is conceptually similar to the crossover operation utilized by genetic algorithms. It uses the concept of fitness, as all evolutionary computation paradigms.

[2] Ref-74943 Kennedy, James and Eberhart, Russell. **Particle swarm optimization**, International conference on neural networks, vol 4, pp 1942–1948, 1995, IEEE

Fairness in Model

How to ensure that **data biases** and **model inaccuracies** do not treat individuals unfavorably on the basis of race, gender, disabilities, and sexual or political orientation?

- Fairness through unawareness
- Equalized odds

$$P(\hat{y}|a=0, Y=y) = P(\hat{y}|a=1, Y=y)$$

- Equalized opportunity

$$P(\hat{y}=1|a=0, Y=1) = P(\hat{y}=1|a=1, Y=1)$$

Correlation fallasy, overgeneraization, class imbalance.

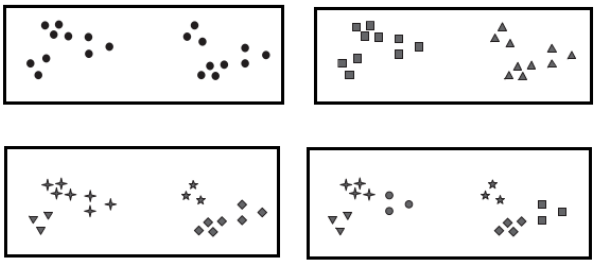
Interpretable/Explainable Model

Computers usually do not explain their predictions. How can we develop trust?

- We need causality, transferability, informativeness, fairness and ethical decision
- Transparency has three levels
  Simulatibility: one can do it on paper
  Decomposability: different parts
  Algorithmic transparency: convergence guarantees
- Decision tree and linear models

Clustering

Grouping data based on their homogeneity (similarity or closeness).



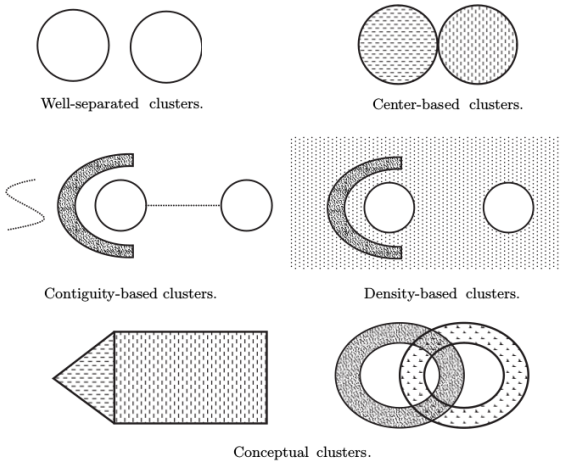
Objects within a group are similar (or related) and are different from the objects in other groups. When it is better?

Clustering

- **Unsupervised** in nature (i.e. right answers are not known)
- Clustering is useful to 1) Summarization, 2) Compression, and 3) Efficiently Finding Nearest Neighbors
- **Type:**
  - Hierarchical (nested) versus Partitional
  - Exclusive versus Overlapping versus Fuzzy
  - Complete versus Partial
- **K-means:** This is a prototype-based[3], partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.

[3] object is closer (more similar) to a prototype

Clustering Approaches



K-means Algorithm

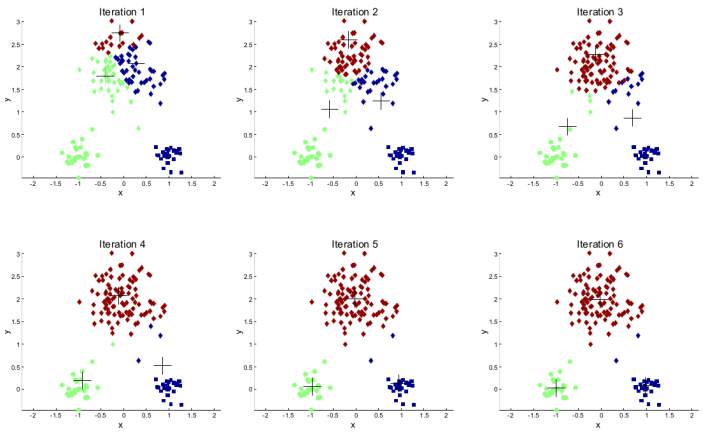
Number of clusters *i.e.* the value of K is provided by the user

Algorithm 1: K-means

1. Randomly select K points as centroids
2. **repeat**
3. **foreach** *datum point* d_i **do**
4. Assign d_i to one of the closest centroids (thereby forming K clusters)
5. Recompute centroid (mean) for each cluster
6. **until** *The centroids converge*;

Closeness is measured by **Euclidean distance**, cosine similarity, correlation, Bregman divergence *etc*

K-means in Action



Evaluation of K-means[4]

For a given data set $\{x_1, x_2, ..., x_n\}$, let K-means partitions it in $\{S_1, S_2, ..., S_K\}$ then the objective is

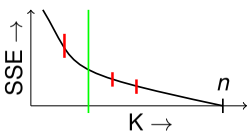
$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in S_i} dist^2(x, \mu_i)$$

where μ_i corresponds to i^{th} centroid. $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$

- Typical choice for *dist* function is Euclidean Distance

How to proceed?

- Choose a K (How?)
  - Run K-means algorithm multiple times
  - Choose clusters corresponding to the one that minimized sum of squared error (SSE)
- If $K == n$, no error.
- Good clustering has smaller K

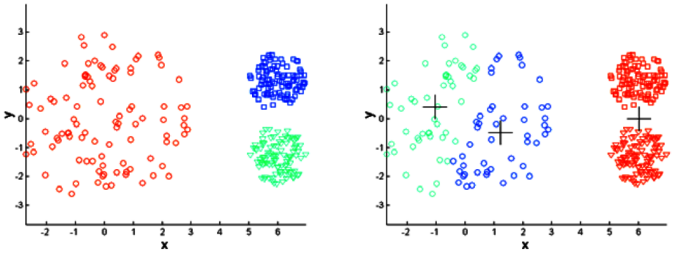


[4] Hamerly, Greg and Elkan, Charles, "Learning the k in k-means", pp 281–288, NIPS-2003

Evaluation of K-means

- **Choosing K:** 1) Domain Knowledge, 2) Preprocessing with another algorithm, 3) Iteration on K
- **Initialization of Centers:** 1) Random point in space, 2) Random point of data, 3) look for dense region, 4) Space uniformly in feature space
- **Cluster Quality:** 1) Diameter of cluster verses Inter-cluster distance, 2) Distance between members of a cluster and the cluster center, 3) Diameter of smallest sphere, 4) Ability to discover hidden patterns

Limitations of K-means



- Has problem when data has
  - Different size clusters
  - Different densities
  - Non-globular shape
- Handling Empty Clusters
- When there are outliers
- Updating Centroids Incrementally

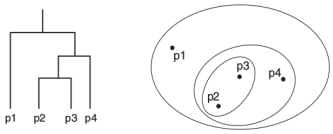
Important Note:

- K-Means and K-NN are different (K nearest neighbors)

K-NN is a **supervised** approach for **classification**

Other Clustering Approaches

- **K-Medoids:** chooses data point as center and minimizes a sum of pairwise dissimilarities. Resistance to noise and/or outliers
- **Agglomerative Hierarchical Clustering:** repeatedly merging the two closest clusters until a single (Single Link)



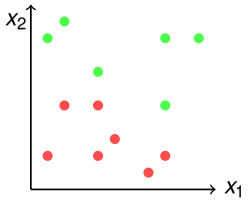
- **DBSCAN:** density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm.

Linear Classification

Consider Following data

x_1	x_2	y
1	9	green
10	9	green
4	7	green
4	5	red
5	3	red
8	9	green
4	2	red
2	5	red
7	1	red
2	10	green
8	5	green
1	2	red
8	2	red

Data is in 2D, so let us visualize



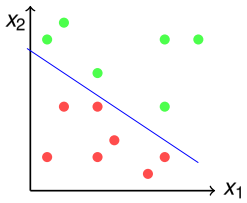
- Data looks linearly separable
- What is the decision boundary?

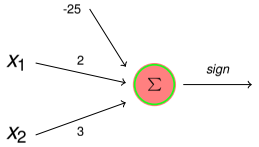
Many Possibilities, such as

if $(2x_1 + 3x_2 - 25 > 0)$ it is green otherwise red

What about this arrangement?

With chosen *decision boundary* $2x_1 + 3x_2 - 25 = 0$





- This illustration is called as **perceptron**
- Provides a graphical way to represent the linear boundary
- Values 3, 2, -25 are its parameters or weights

Given a data

"How to find appropriate parameters?" is an important issue

Perceptron Training Rule

Different algorithms may converge to different acceptable hypotheses

Algorithm 2: Perceptron training rule

```
Begin with random weights w
repeat
    for each misclassified example do
        w_i = w_i + η(t − o)x_i
until all training examples are correctly classified;
return w
```

- **Why would this strategy converge?**
  1. Weight does not change when classification is correct
  2. If perceptron outputs -1 when target is +1: weight increases ↑
  3. If perceptron outputs +1 when target is -1: weight decreases ↓

Conversion with perceptron training rule is subject to linear separability of training example and appropriate η

Example

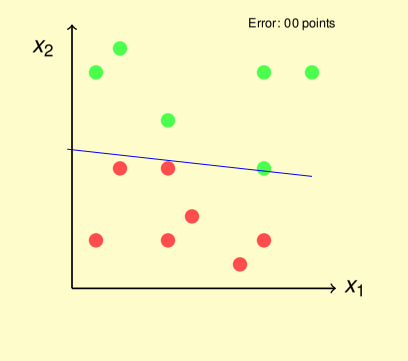
Consider the same data

x_1	x_2	y
1	9	green
10	9	green
4	7	green
4	5	red
5	3	red
8	9	green
4	2	red
2	5	red
7	1	red
2	10	green
8	5	green
1	2	red
8	2	red

$\eta = 0.01$

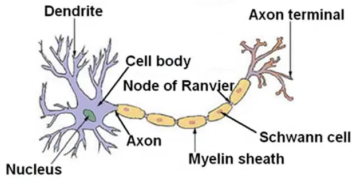
w0=0.500, w1=0.500, w2=0.500	err=7
w0=0.360, w1=-0.120, w2=0.100	err=6
w0=0.300, w1=-0.180, w2=0.060	err=5
w0=0.240, w1=-0.140, w2=0.140	err=4
w0=0.180, w1=-0.200, w2=0.100	err=5
w0=0.120, w1=-0.160, w2=0.180	err=4
w0=0.080, w1=-0.060, w2=0.180	err=5
w0=0.020, w1=-0.120, w2=0.140	err=4
w0=-0.040, w1=-0.180, w2=0.100	err=5
w0=-0.100, w1=-0.140, w2=0.180	err=4
w0=-0.140, w1=-0.040, w2=0.180	err=5
w0=-0.200, w1=-0.100, w2=0.140	err=3
w0=-0.260, w1=-0.160, w2=0.100	err=4
w0=-0.320, w1=-0.120, w2=0.180	err=3
w0=-0.360, w1=-0.020, w2=0.180	err=3
w0=-0.420, w1=-0.080, w2=0.140	err=2
w0=-0.420, w1=-0.080, w2=0.240	err=2
Fourteen more iterations	
w0=-0.900, w1=-0.020, w2=0.180	err=1
w0=-0.900, w1=-0.020, w2=0.240	err=2
w0=-0.920, w1=0.020, w2=0.220	err=2
w0=-0.960, w1=-0.020, w2=0.220	err=3
w0=-0.980, w1=0.020, w2=0.200	err=2
w0=-1.000, w1=0.060, w2=0.180	err=2
w0=-1.040, w1=0.020, w2=0.180	err=0

Visual Interpretation



- Conversion is not gradual. (Error is NOT reducing monotonically)
- It is difficult to decide when to stop if data is not linearly separable

Neural Network (NN)



- Cell, Axon, Synopses, Molicules, and Dendrites
- Humans have 10^{11} neurons, each connected to 10^4 others, switches in 10^{-3} sec

NN is biologically motivated learning model that mimic human brain

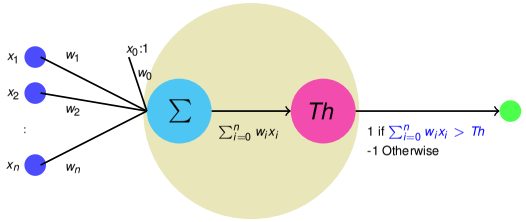
- Started by *W. McCulloch* study on working of neurons in 1943
- MADALINE (1959), an adaptive filter that eliminates echoes on phone lines was the first neural network
- Popularity of Neural Network diminished in 90's but, due to advances in **processing power** and availability of **large data** it again became state-of-the-art

Brief History

- 1872 Staining/Reticular Theory of Nervous Tissue
- 1943 McCulloch & Pitt (Neuron Model)
- 1947 Donald Hebb (Hebbian Learning)
- 1948 Nerbert (Cybernetics, optimal filter, feedback)
- 1954 Frank Rosenblatt (Perceptron)
- 1959 (MADALINE)
- 1962 Hubel & Wiesel (Visual Cortex Model)
- 1969 Minsky (Limitations of Perceptron)
- 1965 Frank Rosenblatt (MLP: Multi Layer Perceptron)
- 1986 (Backpropagation)
- 1989 Universal Approximation Theorem
- 1997 LSTM
- 1998 LeCun (ConvNet for MNIST digit)
- 2006 Unsupervised Pre Training
- 2010 Dahl. (Speech Recognition)
- 2012 AlexNet (ImageNet: Computer Vision) 26→16 ... →12% zfNet
- 2013 VGG →7.3
- 2014 Google LeNet →6.7
- 2015 ResNet →3.6
- 2017 DenseNet
- 2018 Transformers, U-Net segmentation

A Single Perceptron

Perceptron representation

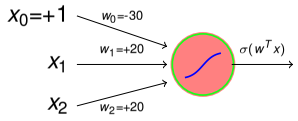


- A single perceptron can represent many boolean functions
- Any *m*-of-*n* function (at least *m* of the *n* inputs must be true) can be represented by perceptron. OR (*m*=1) and AND (*m*=*n*)

Two layer NN can represent any boolean function (Consider SOP)

An Example

Consider a perceptron with output 0/1 as below



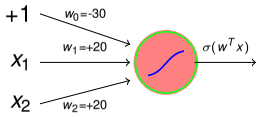
x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

This perceptron computes logical AND

- w_0=-10 gives logical OR
- w_0=10, w_1=-20 with single input gives logical NOT
- XOR is not possible (MLP[5] can do it)

[5] MLP (multi layer perceptron) with one-hidden-layer is *universal boolean function*, capable of expressing any truth table

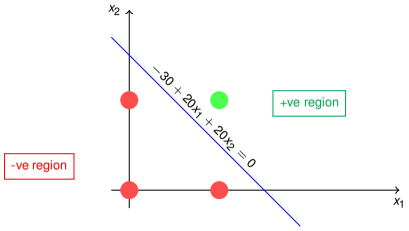
Essentially it Represents A Decision Boundary



Provides positive classification if

$$-30 + 20x_1 + 20x_2 \geq 0$$

Represents a linear decision boundary

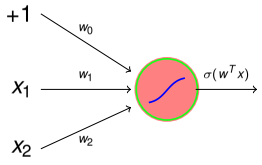


An Example

Design a **perceptron** for

x_1	x_2	Classification
0	0	0
0	1	0
1	0	1
1	1	0

Let us assume following



We have following four equations

$$w_0 + w_1 \times (0) + w_2 \times (0) < 0 \quad (1)$$
$$w_0 + w_1 \times (0) + w_2 \times (1) < 0 \quad (2)$$
$$w_0 + w_1 \times (1) + w_2 \times (0) \geq 0 \quad (3)$$
$$w_0 + w_1 \times (1) + w_2 \times (1) < 0 \quad (4)$$

By (1) $w_0 < 0$ so let $w_0 = -1$
By (2) $w_0+w_2<0$ so let $w_2 = -1$
By (3) $w_0+w_1\geq 0$ so let $w_1 = 1.5$
By (4) $w_0+w_1+w_2<0$ that is valid

So $(w_0, w_1, w_2) = (-1, -1, 1.5)$

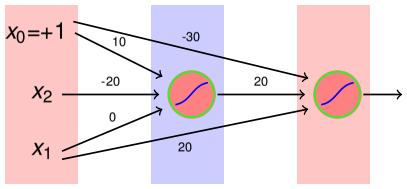
Other possibilities are also there

An Example

Design a **neural network** for

x_1	x_2	Classification
0	0	0
0	1	0
1	0	1
1	1	0

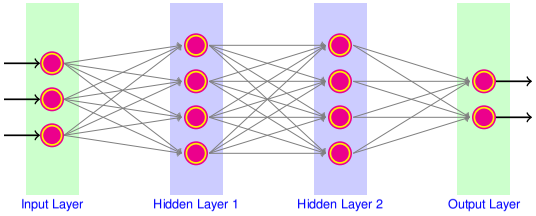
x_1	x_2	$\bar{x_2}$	$(x_1)AND(\bar{x_2})$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0



This arrangement is mostly avoided, as training is very challenging

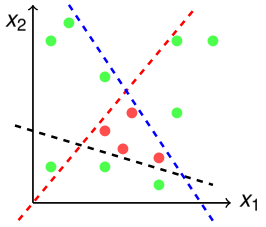
Neural Network

When neurons are interconnected in layers



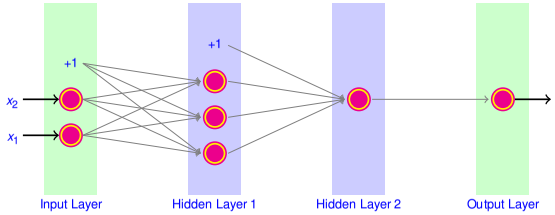
- Number of layers may differ
- Nodes in each intermediate layers may also differ
- Multiple output neurons are used for different class
- **Two levels deep** NN can represent any boolean function

More Example: Design NN for the following data



Whether it is green?

Red-line	Blue-line	Black-line	Color
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Note: Weights and activation in subsequent layers add power to the model in terms of non linearity.

Neural Network Applications

NN is appropriate for problems with the following characteristics:

- Instances are provided by many attribute-value pairs (more data)
- The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
- The training examples may contain errors
- Long training times are acceptable
- Fast evaluation of the target function may be required
- The ability of humans to understand the learned target function is not important

Perceptron Training (delta rule)

When data is not linearly-separable, error fluctuates with parameter update so, it becomes difficult to decide when to stop

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider unthresholded perceptron, $o(\vec{x}) = \vec{w}.\vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

- Gradient would specify direction of steepest increase $\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right]$
- Weights can be learned as $w_i = w_i - \eta\frac{\partial E}{\partial w_i}$
- It can be seen that $\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$

Perceptron Training (delta rule)

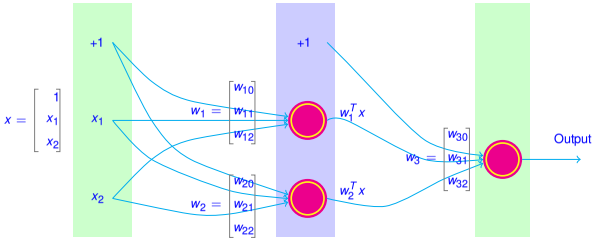
Algorithm 3: Gradient Descent (D, η)

```
Initialize w_i with random weights
repeat
    For each w_i, initialize △w_i = 0
    for each training example d ∈ D do
        Compute output o using model for d whose target is t
        For each w_i, update △w_i = △w_i + η(t−o)x_i
    For each w_i, set w_i = w_i + △w_i
until termination condition is met;
return w
```

- A date item $d \in D$, is supposed to be multidimensional $d = (x_1, x_2, ..., x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

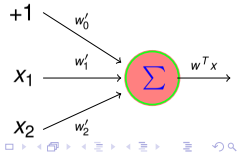
Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



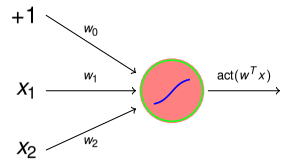
$$\begin{aligned}
Output &= w_{30}\times 1 + w_{31}\times(w_1^T x) + w_{32}\times(w_2^T x)\\
&= w_{30}\times 1 + w_{31}\times[w_{10}\times 1 + w_{11}\times x_1 + w_{12}\times x_2]\\
&\quad + w_{32}\times[w_{20}\times 1 + w_{21}\times x_1 + w_{22}\times x_2]\\
&= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21})\times x_1\\
&\quad + (w_{31}w_{12} + w_{32}w_{22})\times x_2\\
&= w_0' + w_1'\times x_1 + w_2'\times x_2
\end{aligned}$$

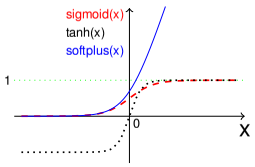
Expression of single perceptron

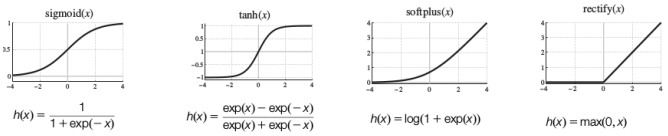


Neuron

Neuron uses nonlinear activation functions (*sigmoid*, *tanh*, *ReLU*, *softplus etc.*) at the place of thresholding







$h(x) = \frac{1}{1+\exp(-x)}$ $h(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ $h(x) = \log(1+\exp(x))$ $h(x) = \max(0, x)$

Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not ~~Perceptron~~)

$$o(\vec{x}) = \sigma(\vec{w}.\vec{x})$$

where $\sigma(y) = \frac{1}{1+e^{-y}}$

- Backpropagation algorithm learns the weights for a fixed set of units and interconnections
- It employs gradient descent to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$$

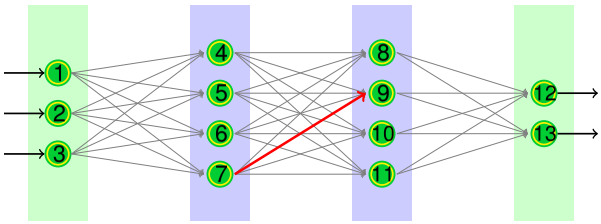
Backpropagation (for 2 layers)

Algorithm 4: Backpropagation(D, η, n_{in}, n_{out}, n_{hidden})

Create the feedforward network with n_{in}, n_{out}, n_{hidden} layers
Randomly initialize weights to small values $\in [-0.05, +0.05]$
repeat
  for *each* $<\vec{x},\vec{t}> \in D$ **do**
    $o_u =$ get output from network $\forall$ unit u
    $\delta_k = o_k(1-o_k)(t_k - o_k)$ for all output unit k
    $\delta_h = o_h(1-o_h)\sum_{k \in outputs}(w_{kh}\delta_k)$ for all hidden unit h
    $w_{ji} = w_{ji} + \triangle w_{ji}$ where $\triangle w_{ji} = \eta \delta_j x_{ji}$
until *converge*;

- Recall error function is $E(\vec{w}) = \frac{1}{2}\sum_{d \in D}\sum_{k \in outputs}(t_{kd} - o_{kd})^2$
- For a single training example $E_d(\vec{w}) = \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$
- Weight w_{ji} is updated by adding $\triangle w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$

Conventions Over The Network



x_{ji}	ith input to unit j (x_{94} is highlighted)
w_{ji}	weight associated with ith input to unit j
net_j	be $\sum_i w_{ji}x_{ji}$ the weighted sum of input for unit j
o_j	output computed by unit j. Consider it as $\sigma(net_j)$
t_j	target output for unit j
$outputs$	set of units in final layer ($\{12, 13\}$ in our case)
$Downstream(j)$	units whose immediate input is the output of unit j

We are interested in $\frac{\partial E_d}{\partial w_{ji}}$ it is $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$ and therefore, $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

Value of $\frac{\partial E_d}{\partial net_j}$ for output units

- $\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \times \frac{\partial o_j}{\partial net_j}$
- $\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2 = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2 = -(t_j - o_j)$
- Note that $o_j = \sigma(net_j)$ therefore $\frac{\partial o_j}{\partial net_j}$ is derivative of sigmoid

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\frac{1}{1+e^{-x}} = (-1)(1+e^{-x})^{-2}\frac{d}{dx}(1+e^{-x}) \\ &= (-1)(1+e^{-x})^{-2}(0-e^{-x}) \\ &= \frac{1}{1+e^{-x}} \times \frac{e^{-x}+1-1}{1+e^{-x}} = \sigma(x)(1-\sigma(x))\end{aligned}$$

- As a result $\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = \sigma(net_j)(1-\sigma(net_j)) = o_j(1-o_j)$
- $\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1-o_j)$
- Term $(t_j - o_j)o_j(1-o_j)$ is treated as δ_j

Therefore, $\triangle w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} \times x_{ji} = \boxed{\eta(t_j - o_j)o_j(1-o_j)x_{ji}}$

Value of $\frac{\partial E_d}{\partial net_j}$ for hidden units

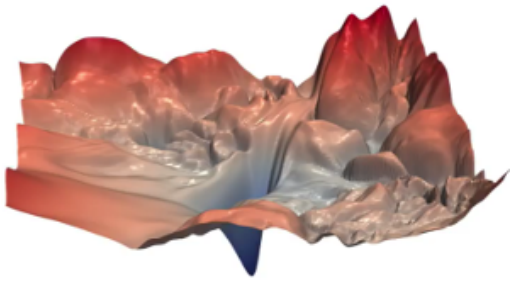
$$\begin{aligned}\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \times \frac{\partial net_k}{\partial net_j} \\ &= \sum_{k \in Downstream(j)} -\delta_k \times \frac{\partial net_k}{\partial net_j} \\ &= \sum_{k \in Downstream(j)} -\delta_k \times \frac{\partial net_k}{\partial o_j} \times \frac{\partial o_j}{\partial net_j} \\ &= \sum_{k \in Downstream(j)} -\delta_k \times w_{kj} \times o_j(1-o_j)\end{aligned}$$

- δ_j being $-\frac{\partial E_d}{\partial net_j} = o_j(1-o_j)\sum_{k \in Downstream(j)}\delta_k \times w_{kj}$

Therefore, $\triangle w_{ji} = \eta \delta_j x_{ji} = \boxed{\eta(o_j(1-o_j)\sum_{k \in Downstream(j)}\delta_k \times w_{kj})x_{ji}}$

Note: $net_k = w_{k1}x_{k1} + w_{k2}x_{k2} + \ldots + w_{kj}x_{kj} + \ldots$; as o_j is input to kth unit so $o_j = x_{kj}$; so $\frac{\partial net_k}{\partial o_j} = w_{kj}$

Loss Landscape [6]



Backpropagation over multilayer networks converge to a local minimum, NOT necessarily to the global minima

[6] Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, 2018

Backpropagation

- **Adding Momentum:** weight update during n^{th} iteration depend partially on the update that occurred during the $(n-1)^{th}$ iteration

$$\triangle w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \triangle w_{ji}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth, δ_r value for a unit in hidden layer is determined as

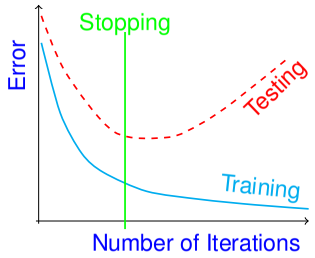
$$\delta_r = o_r(1-o_r)\sum_{s \in Downstream(r)} w_{sr} \times \delta_s$$

Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)
- Every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units
- Any function can be approximated to arbitrary accuracy by a network with three layers of units (Cybenko 1988)

Generalization, Overfitting, and Stopping Criterion

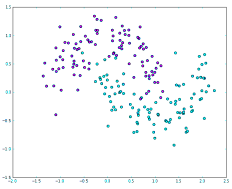
Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy

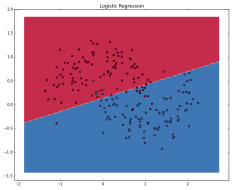


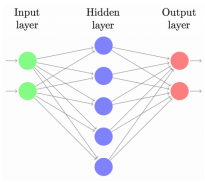
- Weight decay or use of validation set (k-fold ?) is suggested
- Input or output encoding can be used

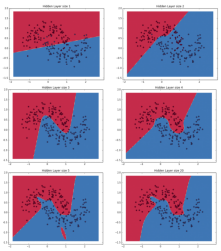
Coding Example [7]

Consider two class data
Logistic Regression
Neural Network
Decision Boundary
Bigger hidden layer









[7] https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb

An Example Code [1/4] [8]

```
import matplotlib.pyplot as plt
import numpy as np
import sklearn
import sklearn.datasets
import sklearn.linear_model
import matplotlib

matplotlib.rcParams['figure.figsize'] = (10.0, 8.0)
np.random.seed(0)
X, y = sklearn.datasets.make_moons(200, noise=0.20)
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)

def plot_decision_boundary(pred_func):
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

plot_decision_boundary(lambda x: clf.predict(x))
plt.title("Logistic Regression")
plt.show()
```

[8] https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb

An Example Code [2/4] [9]

```
#### Implementing NN ####
num_examples = len(X) # training set size
nn_input_dim = 2 # input layer dimensionality
nn_output_dim = 2 # output layer dimensionality

epsilon = 0.01 # learning rate for gradient descent
reg_lambda = 0.01 # regularization strength

def calculate_loss(model):
    W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
    z1 = X.dot(W1) + b1
    a1 = np.tanh(z1)
    z2 = a1.dot(W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    corect_logprobs = -np.log(probs[range(num_examples), y])
    data_loss = np.sum(corect_logprobs)
    data_loss += reg_lambda/2 * (np.sum(np.square(W1)) + np.sum(np.square(W2)))
    return 1./num_examples * data_loss

def predict(model, x):
    W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
    z1 = x.dot(W1) + b1
    a1 = np.tanh(z1)
    z2 = a1.dot(W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    return np.argmax(probs, axis=1)
```

[9] https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb

An Example Code [3/4] [10]

```
def build_model(nn_hdim, num_passes=20000, print_loss=False):
    np.random.seed(0)
    W1 = np.random.randn(nn_input_dim, nn_hdim) / np.sqrt(nn_input_dim)
    b1 = np.zeros((1, nn_hdim))
    W2 = np.random.randn(nn_hdim, nn_output_dim) / np.sqrt(nn_hdim)
    b2 = np.zeros((1, nn_output_dim))
    model = {}
    for i in range(0, num_passes):
        z1 = X.dot(W1) + b1
        a1 = np.tanh(z1)
        z2 = a1.dot(W2) + b2
        exp_scores = np.exp(z2)
        probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
        delta3 = probs
        delta3[range(num_examples), y] -= 1
        dW2 = (a1.T).dot(delta3)
        db2 = np.sum(delta3, axis=0, keepdims=True)
        delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
        dW1 = np.dot(X.T, delta2)
        db1 = np.sum(delta2, axis=0)
        dW2 += reg_lambda * W2
        dW1 += reg_lambda * W1
        W1 += -epsilon * dW1
        b1 += -epsilon * db1
        W2 += -epsilon * dW2
        b2 += -epsilon * db2
        model = { 'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}
    return model
```

[10] https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb

An Example Code [4/4] [11]

```
# Build a model with a 3-dimensional hidden layer
model = build_model(3, print_loss=True)

# Plot the decision boundary
plot_decision_boundary(lambda x: predict(model, x))
plt.title("Decision Boundary for hidden layer size 3")

plt.show()

######### Varying the number of hidden layers ##########
plt.figure(figsize=(16, 32))
hidden_layer_dimensions = [1, 2, 3, 4, 5, 20, 50]
for i, nn_hdim in enumerate(hidden_layer_dimensions):
    plt.subplot(5, 2, i+1)
    plt.title('Hidden Layer size %d' % nn_hdim)
    model = build_model(nn_hdim)
    plot_decision_boundary(lambda x: predict(model, x))
plt.show()
```

[11] https://github.com/dennybritz/nn-from-scratch/blob/master/nn-from-scratch.ipynb

Thank You!

Thank you very much for your attention!

Queries ?