Natural Language Processing

Neural Networks I: From Data to a Decision

Ramaseshan Ramachandran

MODEL

- Quantitative expression of the real world observations
- Mathematical Models translate the real world observations into mathematical expressions
- A machine learning model is a mathematical construct trained to recognize patterns in data
- It makes predictions or classifications based on those patterns
- Helps in understanding and interpreting information for decision making/classification
- Data driven decisions under uncertainty
- Model Parameters store vast amount of information captured from the data
  - Lexical, linguistic and semantic knowledge in the case of NLP
- Access the *knowledge* (learned from the data) of the Model by using the context

AN EXAMPLE OF A MODEL

Modeling is about describing the object using a set of mathematical relationships

1.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix}$$

$$\implies x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \cdots \\ a_{n1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ \cdots \\ a_{n2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ a_{3n} \\ \cdots \\ a_{nn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix}$$

$$Ax = \sum_{j=1}^{n} a_{ij} x_j$$

2. Another example $\langle w_i, \tilde{w}_k \rangle \approx \log(X_{ik})$

DISCRIMINATIVE MODEL

- A class of models that focuses on learning the decision boundary between different classes or outcomes
- It models the conditional probability distribution of the target variable given the input variables, $\mathrm{p}(\mathrm{y}|\mathrm{x})$.
- In the case of classification, this is $\mathrm{p}(\mathrm{C_k}|\mathrm{x})$
- OR, Learn a function, *discriminant function*, that maps inputs x directly into decisions

GENERATIVE MODEL

- If a function models input and output into probability distributions, then it is in general known as a generative model
- Given a training data, generate new samples similar to the training samples from the same distribution
  - Language model - generating the next word given the context
  - It is possible to generate synthetic data points (or new sentences) using these distributions

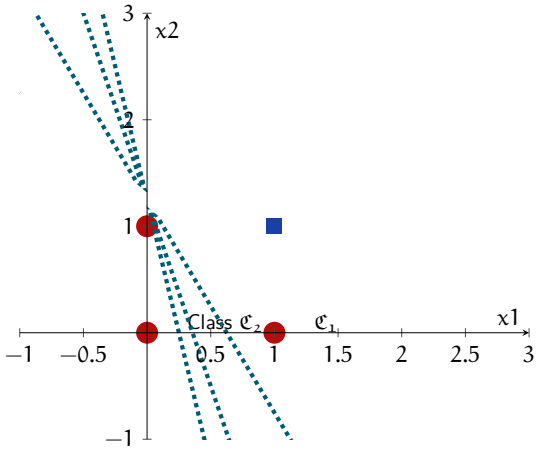
An algorithm is a sequence of instructions to solve a problem

- The steps to solve problems are well defined
- Steps are coded in some ordered sequence to transform the input from one form to another
- Rules are unambiguous
- Sufficient Knowledge is available to fully solve the problem

HUMAN/MACHINE LEARNING

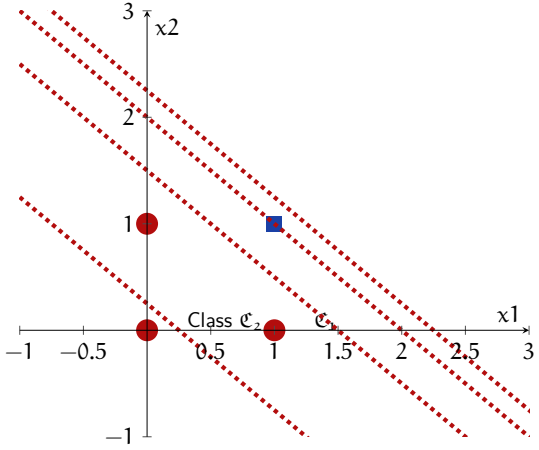
- There are problems whose solutions cannot be formulated using standard rule-based algorithms
- Problems that require subtle inputs cannot be solved using standard algorithmic approach - face recognition, speech recognition, hand-written character recognition, etc.
- Finding Examples and using experience gained in similar situations are useful
- Examples provide certain underlying patterns
- Patterns give the ability to predict some outcome or help in constructing an approximate model
- **Learning** is the key to the ambiguous world

DECISION BOUNDARY- VARIATION OF w_j

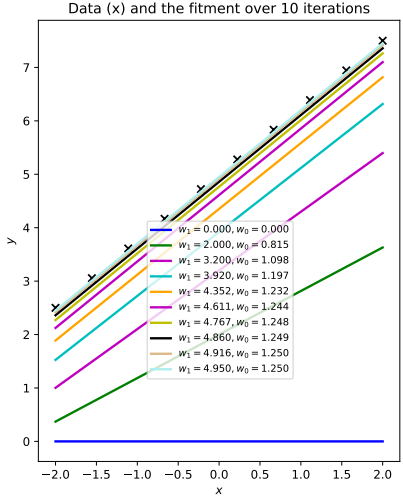


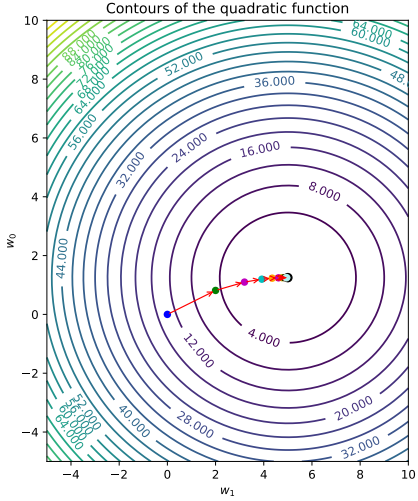
DECISION BOUNDARY - VARIATION OF BIAS

The contribution of bias to the the creation of the decision boundary

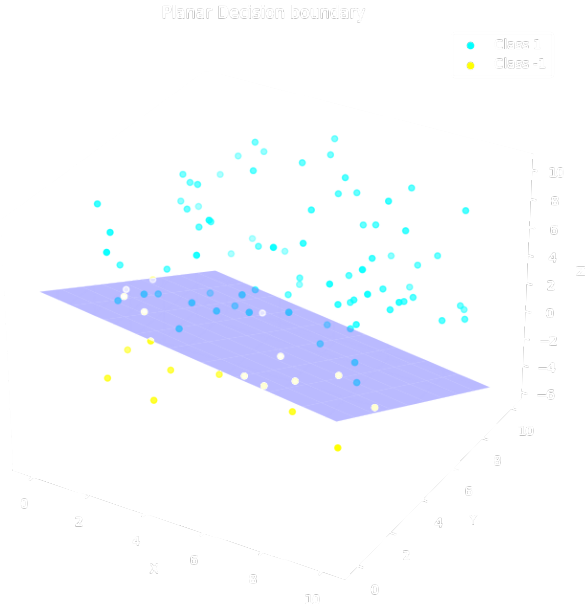


DECISION BOUNDARY AND GRADIENT DESCENT

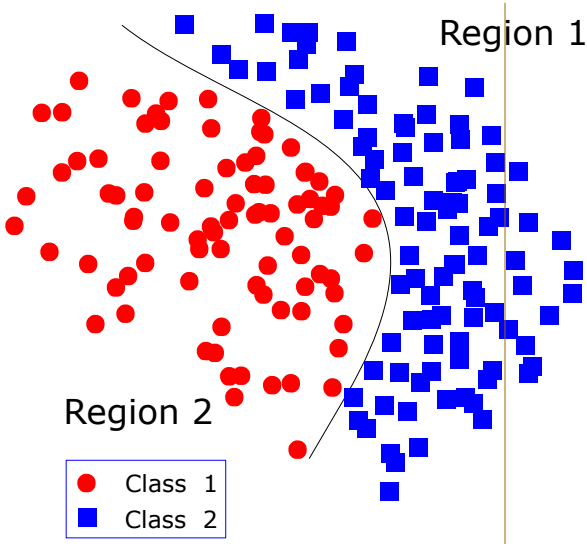




DECISION SURFACE FOR A 3-D VECTOR



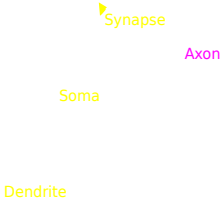
LINEARLY SEPARABLE?



Is this separable?

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

NEURAL NETWORK



- Each individual neuron can form thousands of links with other neurons.
- A typical brain has well over 100 trillion synapses
- Functionally related neurons connect to each other to form neural networks

- The electro-chemical connections between neurons are not static
- The more signals sent between two neurons, the stronger the connection grows and with each new experience and each remembered event, the brain slightly re-wires its physical structure.
- Our brains form a million new connections for every second of our lives

LAWS OF ASSOCIATION

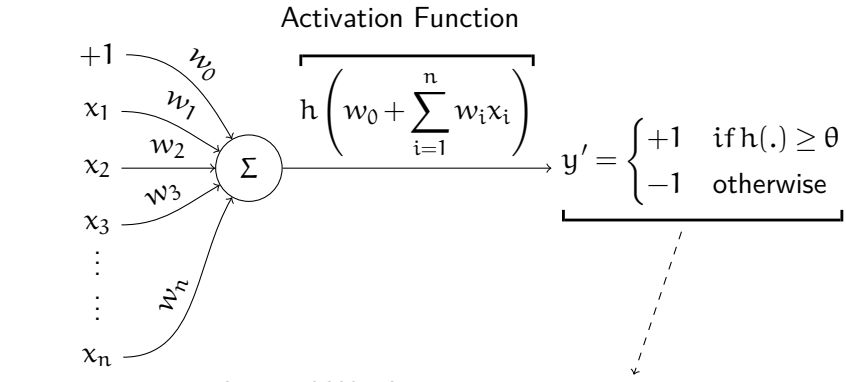
Aristotle's attempts on fundamental laws of learning and memory

The law of similarity	If two things are similar, the thought of one will tend to trigger the thought of the other - word2vec If you recollect one birthday, you may find yourself thinking about others as well
The law of contrast	Seeing or recalling something may also trigger the recollection of something completely opposite
The law of contiguity	Things or events that occur close to each other in space or time tend to get linked together in the mind If you found a snake in the corner of the street, every time you cross the corner, you tend to look for one. Events are conditioned based on the time and space
The law of frequency	The more often two things or events are linked, the more powerful will be that association - think of next word prediction - strength of the association decides who is the probable candidate

PERCEPTRON

Neuron	**Perceptron**
Biological	A mathematical model of a biological neuron
Dendrites receive electrical signals	Perceptron receives mathematical values as input
Electro-chemical signals between Dendrites and axons	The weighted sum represents the total strength of the signal
The electro-chemical signals are not static	Weights change during the training process

PERCEPTRON


Activation Function
+1
x1
x2
x3
xn
w0
w1
w2
w3
wn
Σ
h (w0 + Σ_{i=1}^{n} wi xi)
y′ = +1 if h(.) ≥ θ
−1 otherwise
Feature Vectors
x1, x2, x3, ...xn
Learned Weights
w0, w1, w2, w3, ...wn
Decision Function

PERCEPTRON LEARNING

- Perceptron learns the weights
- They are adjusted until the output is consistent with the target output in the training examples
- $w^{(k+1)} \propto (y - \hat{y})$
- The weights are updated as below

$$w_j^{(k+1)} = w_j^{(k)} - \eta(y_i - \hat{y}^{(k)})x_{ij}$$

where $w^{(k)}$ is the weight parameter associated with the i^{th} input at k^{th} iteration

η is the learning parameter and x_{ij} is the j^{th} attribute of the i^{th} training sample

- If $(y - \hat{y}) \approx 0$, no prediction error
- During the training the weights contributing most to the error require adjustments

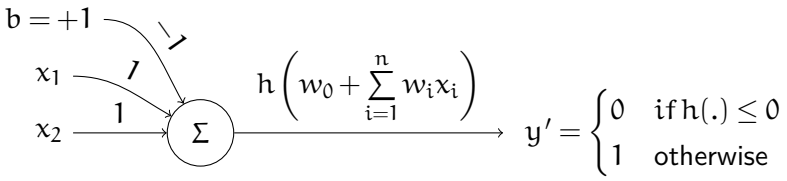
ALGORITHM FOR PERCEPTRON LEARNING

1: Total number of input vectors $= k$
2: Total number of features $= n$
3: Learning parameter $\eta = 0.01$, where $0 < \eta < 1$
4: epoch[1] count $t = 1, j = 1$
5: Initialize weights w_i with random numbers
6: Initialize the input layer with $\vec{x_j}$
7: Calculate the output using $\Sigma w_i x_i + w_0$
8: Calculate the error $(y - \hat{y})$.
9: Update the weights $w_j(t+1) = w_j - \eta(y - \hat{y})x_j$
10: Repeat steps 7 and 9 until: the error is less than θ or a predetermined number of epochs have been completed.

To provide a stable weight update for this step, $w_j(t+1) = w_j - \eta(y - \hat{y})x_j$, we require a small η. This results in slow learning. Bigger η would be good for fast learning. What are the problems? . What is the compromise?

[1] An epoch is one complete presentation of the data set to be learned to a learning machine.

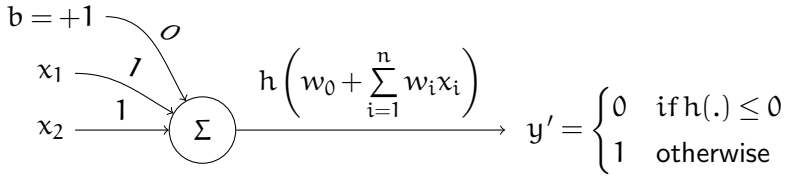
LOGICAL AND



Input x_1	Input x_2	$x_1.w_1 + x_2.w_2 + b.w_b$	output-y
0	0	0.1+0.1-1	0
0	1	0.1+1.1-1	0
1	0	1.1+0.1-1	0
1	1	1.1+1.1-1	1

Here, the perceptron is already trained and the learned weights are shown in the diagram

LOGICAL OR



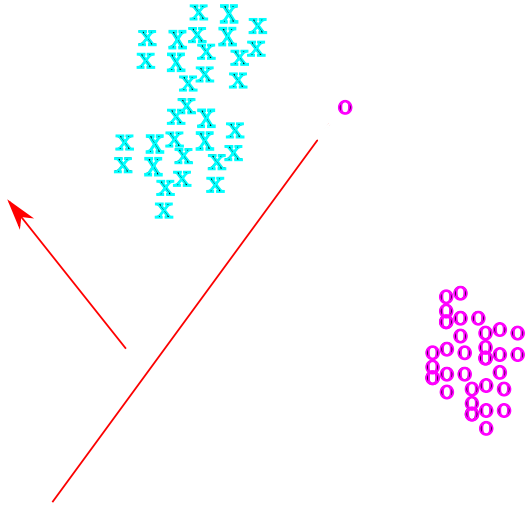
Input x_1	Input x_2	$x_1.w_1 + x_2.w_2 + b.w_b$	output-y

SENTIMENT ANALYSIS - USING PERCEPTRON

- Ability to classify reviews as positive or negative
- Positive and negative words for training
- Glove word embedding as features - input
  - 50 element word embedding[2]
  - Training Data generated using the intersection of the sentiment word list and word embedding from Glove

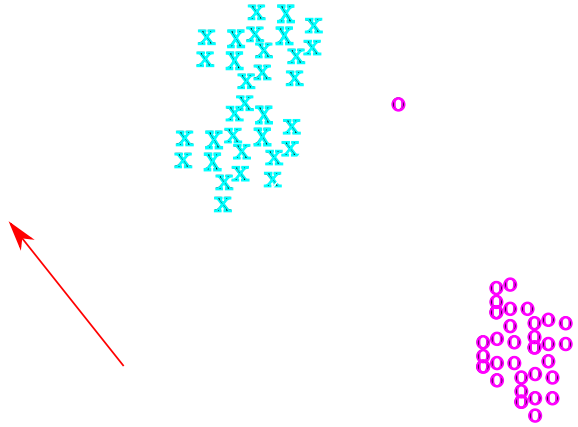
[2] data from https://nlp.stanford.edu/projects/glove/

PERCEPTRON LEARNING

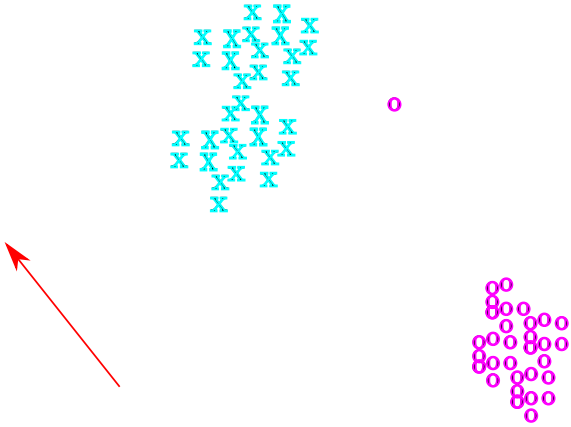


Figure

PERCEPTRON LEARNING



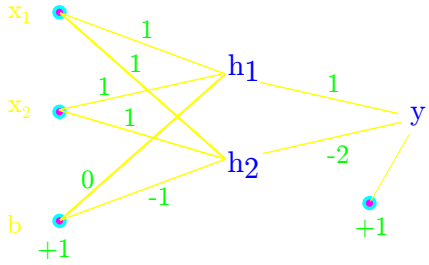
PERCEPTRON LEARNING



PERCEPTRON LIMITATIONS

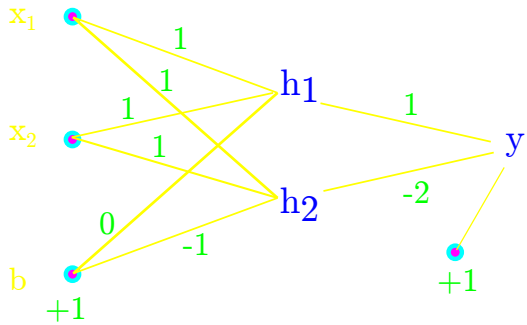
- It is based on the linear combination of fixed basis functions
- Updates the model only based on misclassification
- Documents that are linearly separable are classified

LOGICAL XOR



Input x_1	Input x_2	output-y
0	0	0
0	1	1
1	0	1
1	1	0

LOGICAL XOR



Input x_1	Input x_2	b	h_1	h_2	output-y
0	0	1	$f(0) = 0$	$f(-1) = 0$	0
0	1	1	$f(1) = 1$	$f(0) = 0$	1
1	0	1	$f(1) = 1$	$(f0) = 0$	1
1	1	1	$f(2) = 2$	$(f1) = 1$	0

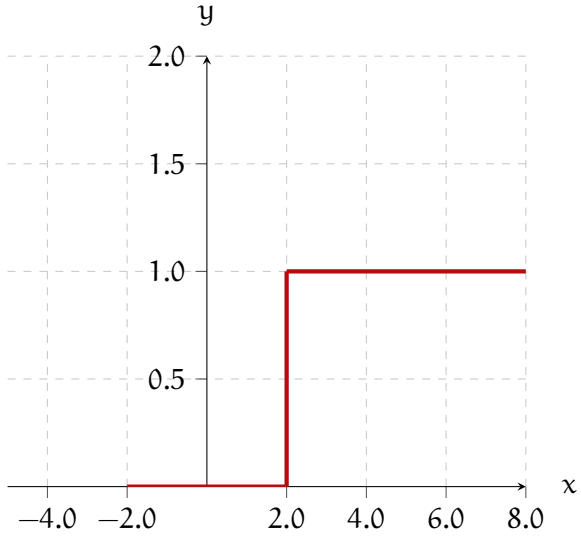
INTUITION

- Input space is transformed into hidden space
- Hidden layer represents the input layer
- Learns automatically the input representation and patterns
- $(0,1)$ and $(1,0)$ are merged into one in the h-space
- Patterns yielding similar results are merged into one
- Dimensionality reduction
- Learns to extract meaningful/latent features or representations from the input data
- Transforms the input information and create a representation in the new space
- Uses activation functions (e.g., sigmoid, tanh, or ReLU) to introduce non-linearity into the network, allowing it to capture complex relationships between input and output
- Are hidden layer neurons joining piecewise linear representations to create non-linear boundaries?

ACTIVATION FUNCTIONS

- Hard threshold
- Sigmoid
- Tanh
- ReLu - Rectified Linear Unit
- Leaky ReLu
- Softmax

HARD THRESHOLD



SIGMOID 1/3

Let us consider two classes c_1 and c_2. The posterior probability for c_1 using Bayes theorem is

$$p(c_1|w) = \frac{p(w|c_1)p(c_1)}{p(w|c_1)p(c_1) + p(w|c_2)p(c_2)}$$
$$= \frac{1}{1 + \frac{p(w|c_2)p(c_2)}{p(w|c_1)p(c_1)}}$$

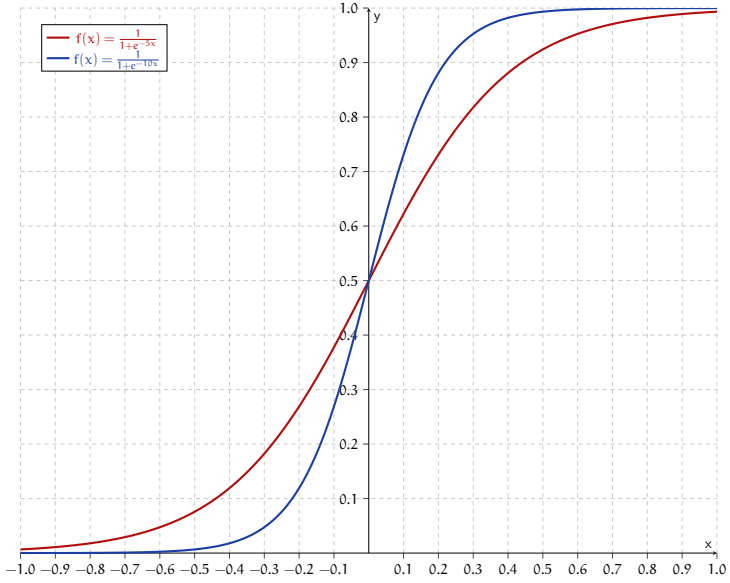
or

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \Rightarrow \text{Sigmoid}$$
$$\sigma(-x) = 1 - \sigma(x)$$
$$\text{where } x = \log\left(\frac{p(w|c_1)p(c_1)}{p(w|c_2)p(c_2)}\right)$$

SIGMOID 2/3

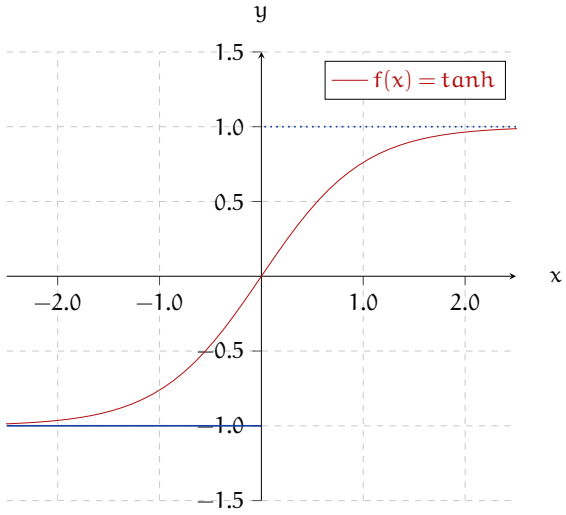
- The sigmoid is a non-linear function
- Better than hard threshold function as it squashes the net output into the range $[0, 1]$
- The values closer to the tails become 0 or 1
- In some cases, the values quickly saturate at 0 or 1
- At the bottom tail, most values become zero during the training and hence the most important aspect of learning of neural network is inhibited
- Sigmoid outputs are not zero-centered
- It is undesirable to have all the values squashed near the tails, where the gradient is ≈ 0

SIGMOID 3/3


f(x) = 1/(1+e^-5x)
f(x) = 1/(1+e^-10x)
y
x
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
−1.0 −0.9 −0.8 −0.7 −0.6 −0.5 −0.4 −0.3 −0.2 −0.1
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

TANH

This is a zero-centered non-linear function.



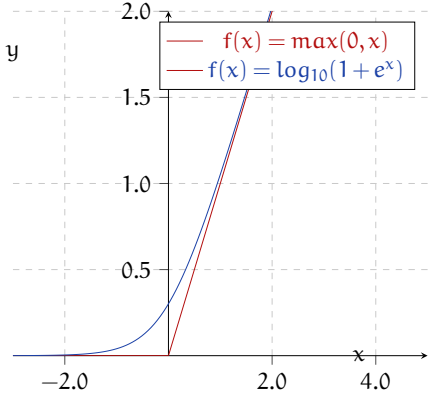
RELU - RECTIFIED LINEAR UNIT

- There is a continuous gradient for the neurons to be in active state
- Produces a non-zero gradient for values closer to zero
- Leaky ReLu

$$f(x) = \begin{cases} x, & \text{if} x > 0 \\ 0.01x, & \text{otherwise} \end{cases}$$

- Produces efficient propagation of the gradient
- Computationally efficient
- Scale invariant
- Unbounded and not zero centered

Learning rate (usually, very small) has to be fine tuned to minimize the death of neurons



MULTICLASS DECISION FUNCTION

- Basic linear classifiers make binary decisions
- In NLP problems, we often need more than two classes
  - Document classification
  - Sentiment analysis - positive, negative, neutral, non-sentiment
- We need a decision function that scores all K classes at once
- Stacking binary (case-by-case) decision functions is hard to manage

SOFTMAX

- Need a function that takes a vector of K real-valued scores and normalizes it into a probability distribution over K classes
- Need a function that keeps the classes well separated (ideal condition)
- Need a function that assigns each class a probability mass

$$p(c_j|x) = \frac{p(x|c_j)p(c_j)}{\sum_k p(x|c_k)p(c_k)}$$
$$= \frac{\exp(a_j)}{\sum_k \exp(a_k)} \Rightarrow \mathrm{Softmax}$$

ACTIVATION FUNCTION - PYTHON CODE

```
import numpy as np
def sigmoid(X,W,b):
    return 1.0/(1.0+ np.exp(-(np.dot(W.T,X)+b)))
def tanh(X,W,b):
    z = np.dot(W.T,X)+b
    return (np.exp(z) - np.exp(-z))/(np.exp(z) + np.exp(-z))
def relu(X,W,b):
    return np.maximum(np.dot(W.T,X) + b,0)
def leaky_relu(X,W,b):
    return np.maximum(0.01*(np.dot(W.T,X)+b),np.dot(W.T,X)+b)
def softmax(X,W,b):
    z_exp = np.exp(np.dot(W,X)+b)
    return z_exp/np.sum(z_exp)
```

```
if __name__ == '__main__':
^^IW = np.array([[1,2,3],[2,3,8],[1,5,7]])
^^IX=np.array([0.2, 0.1, 0.3])
^^Ib=1.5

^^Iprint(sigmoid(X,W,b)) ^^I# [0.90024951 0.97587298 0.99330715]
^^Iprint(tanh(X,W,b)) ^^I^^I# [0.97574313 0.99877117 0.9999092 ]
^^Iprint(relu(X,W,b))^^I^^I# [2.2 3.7 5. ]
    print(leaky_relu(X,W,b))# [2.2 3.7 5. ]
^^Iprint(softmax(X,W,b))   # [0.08672022 0.52462674 0.38865305]
```

REFERENCES I