Function Approximations in Reinforcement Learning

Lecture 6

Subir Varma

So Far..

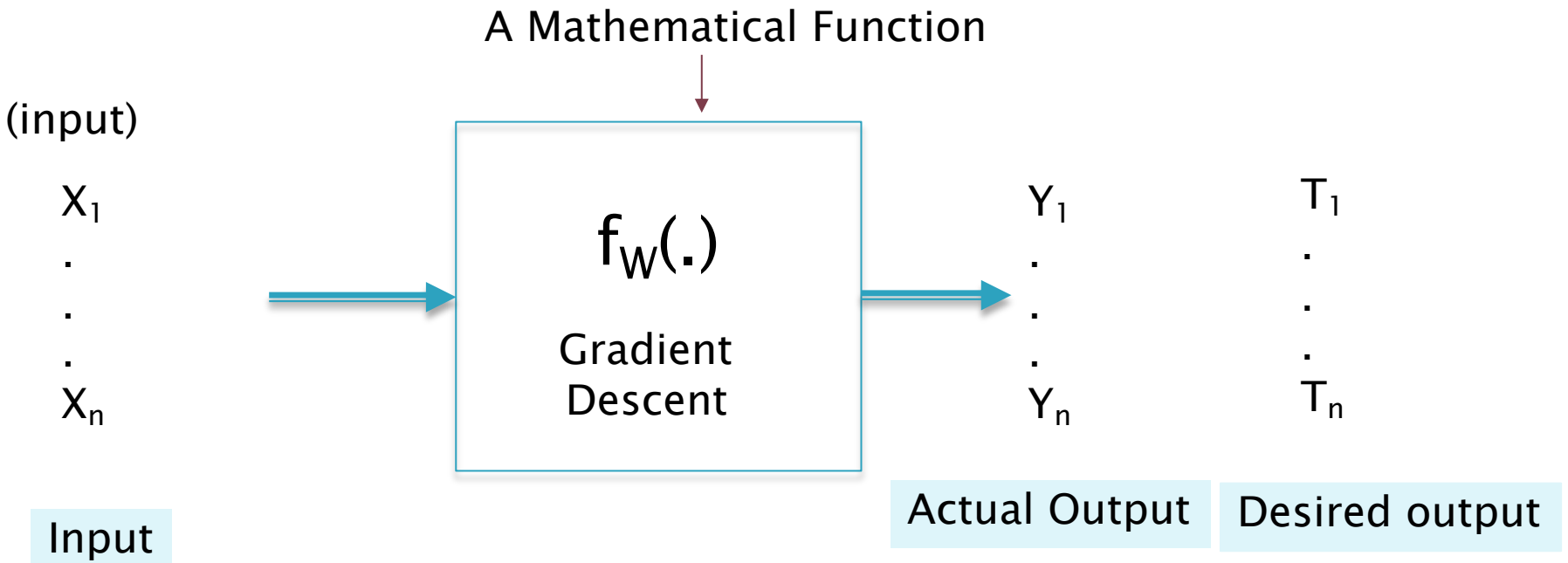
Tabular Reinforcement Learning

	A1	A2	A3	A4
S1	Q(S1,A1)	Q(S1,A2)	Q(S1,A3)	Q(S1,A4)
S2	Q(S2,A1)	Q(S2,A2)	Q(S2,A3)	Q(S2,A4)
S3	Q(S3,A1)	Q(S3,A2)	Q(S3,A3)	Q(S3,A4)
S4	Q(S4,A1)	Q(S4,A2)	Q(S4,A3)	Q(S4,A4)

This approach does not scale if the number of states is very large (in the multiple millions)

OR if S or A is continous

Machine Learning



Training Set: $(X_1,T_1),\ldots,(X_n,T_n)$

Objective Function: Distance(Y,T)

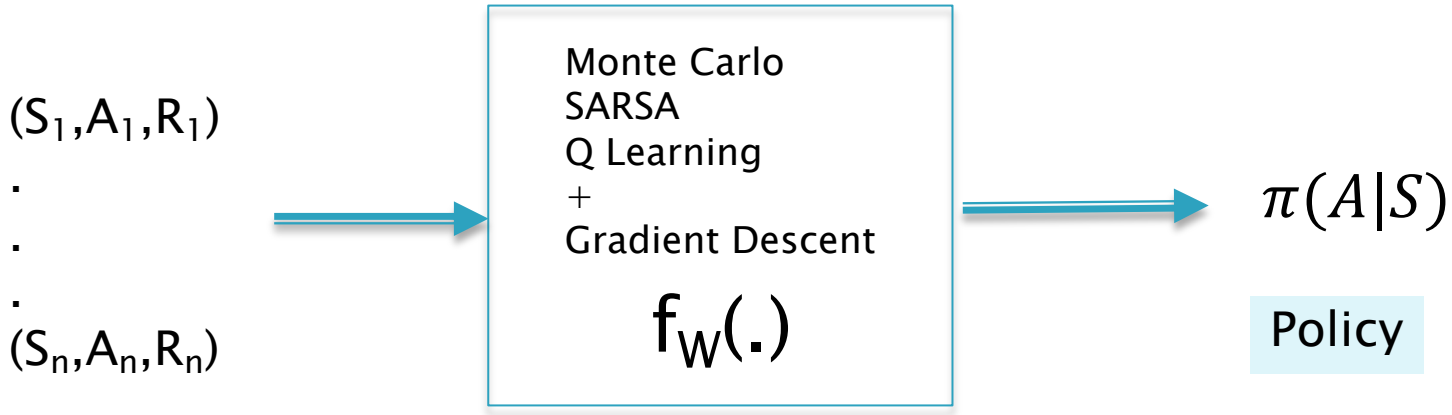
Tabular Reinforcement Learning

Rollouts ←→ Training Set

Objective Function: Total Reward

RL + ML = Deep RL (Functional RL)

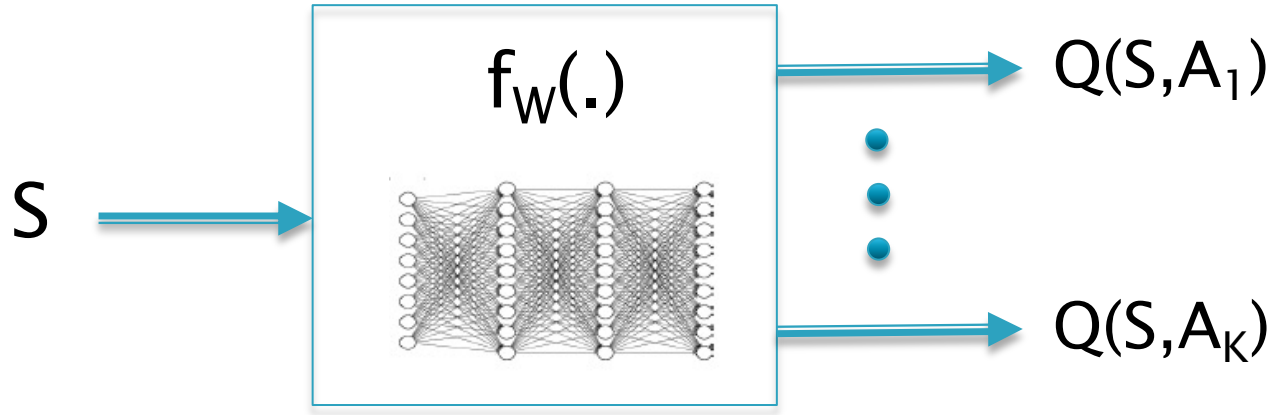
More Scalable



Rollouts ←→ Training Set

Objective Function: Total Reward

Deep RL Method 1: Approximating the Q Function



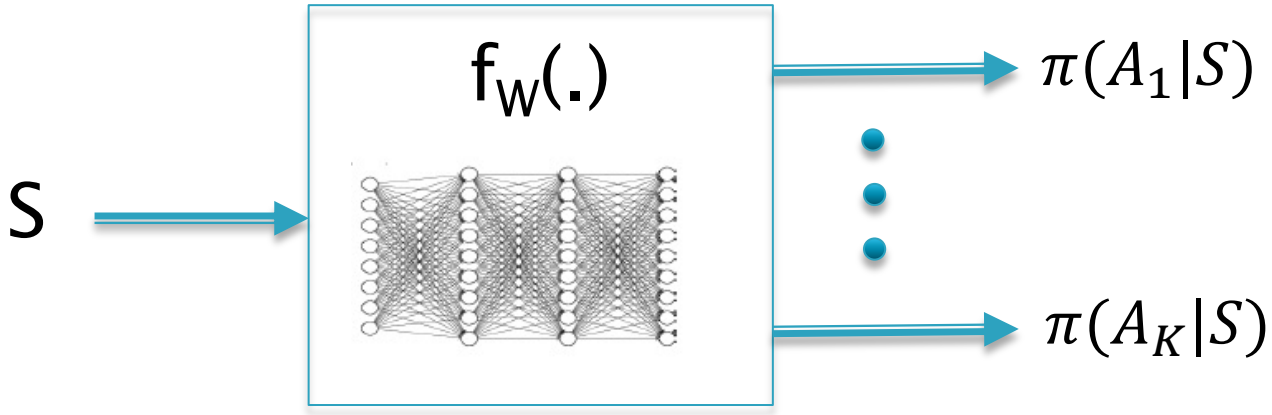
f_W is represented using a multilayer Neural Network

(Lecture 7)

Benefits:

- The number of Parameters W required to define the function f_W Is much less than the size of the state space for S
- The Parameters W can be learnt from the MDP data, using well known algorithms such as Backprop

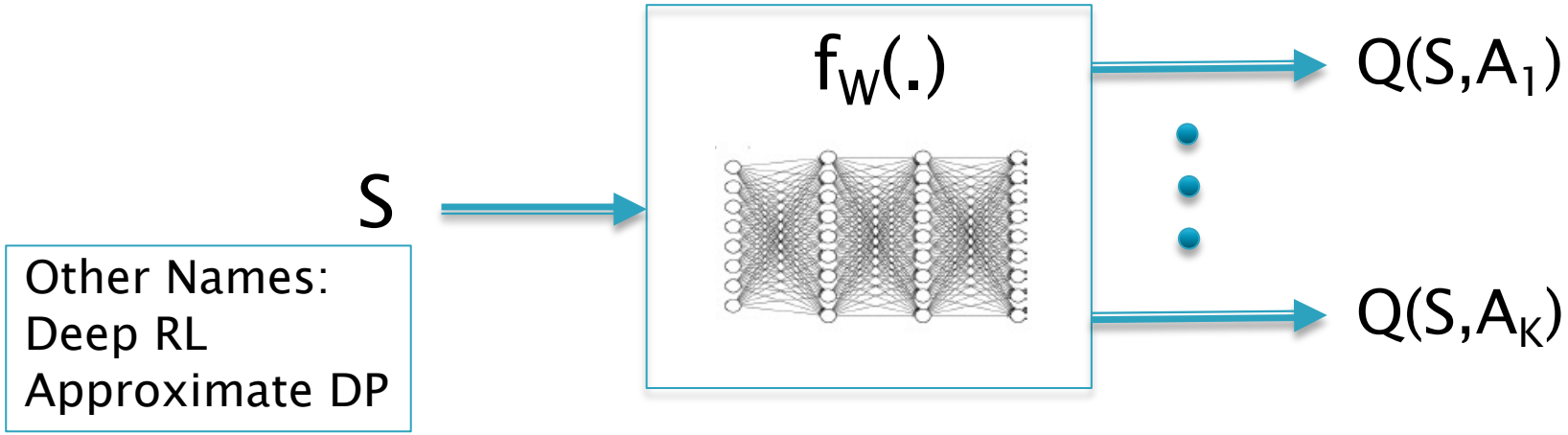
Deep RL Method 2: Approximating Policy Functions



(Lecture 8)

Policy Gradients Algorithms: Estimate Optimal Policy directly without first estimating the Value Function

A More Scalable Approach: Functional Reinforcement Learning



Other Names:
Deep RL
Approximate DP

Reinforcement Learning: How to make Optimal Decisions in an unknown environment

+

Deep Learning: How to solve complex problems in very large state spaces, especially with sensory data

=

Deep Reinforcement Learning: How to make Optimal Decisions for complex problems in large state spaces

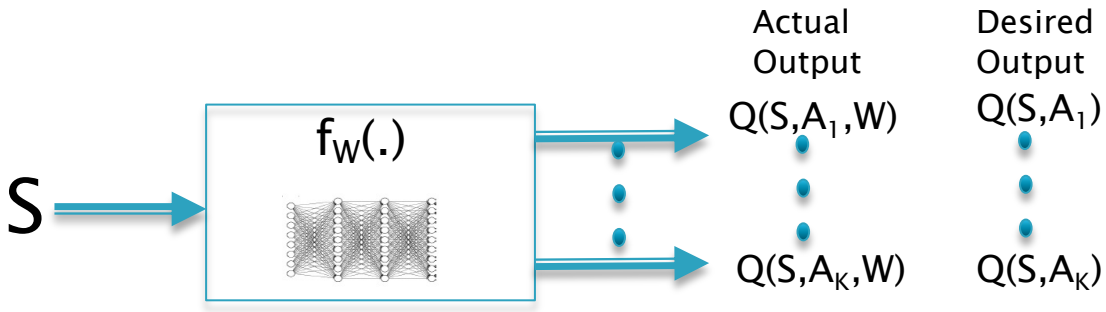
Deep RL with Q Function Approximation: High Level Approach

Run Sample Episodes from the System

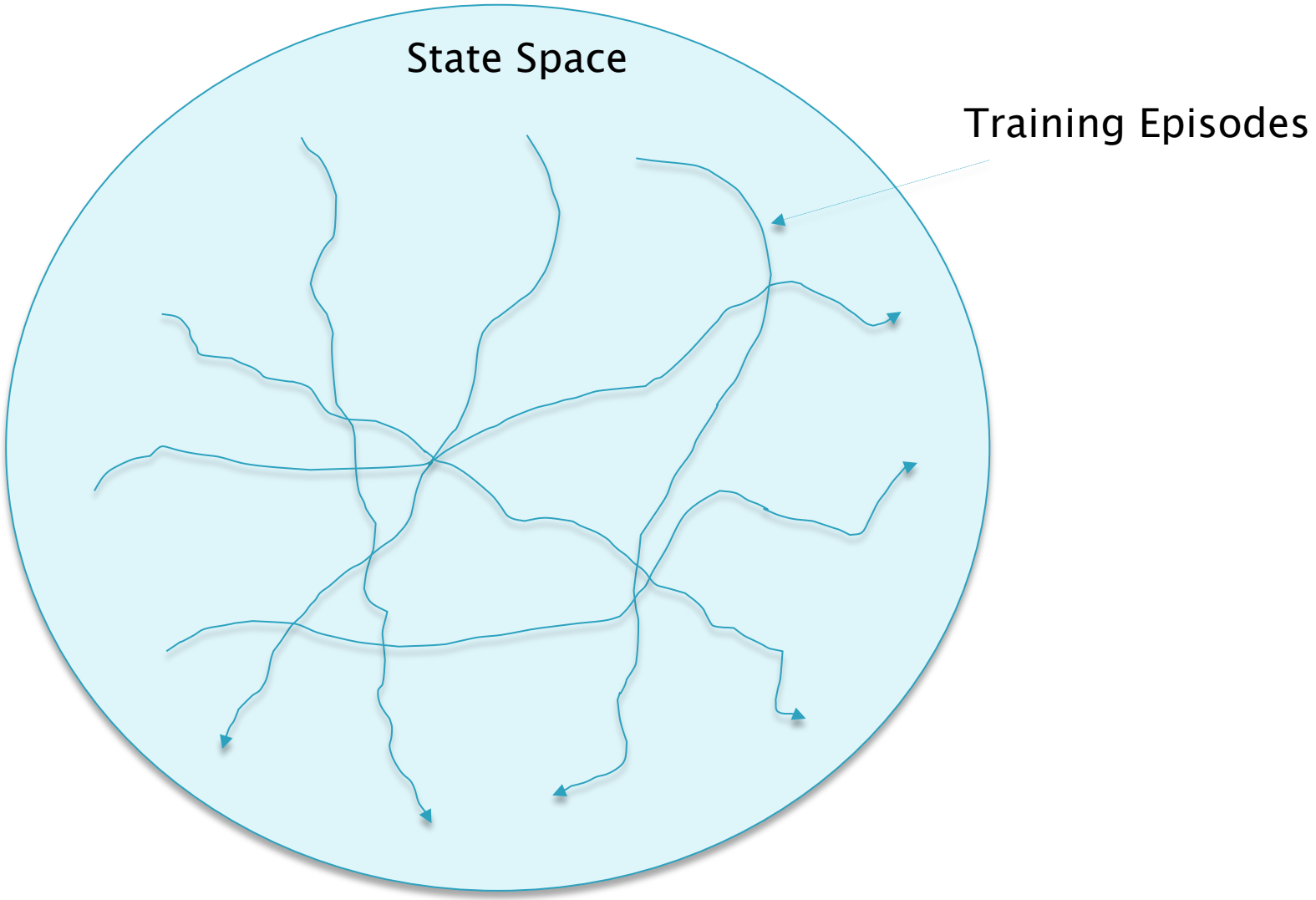
Use Monte Carlo, SARSA or Q-Learning to get samples of the mapping (S, A) → Q(S,A). This becomes the Training Data

Update the Neural Network weights
So that Q(S,A) and Q(S,A,W) move closer

We replace Table updates with NN Weight updates using Gradient Descent



Regression Problem

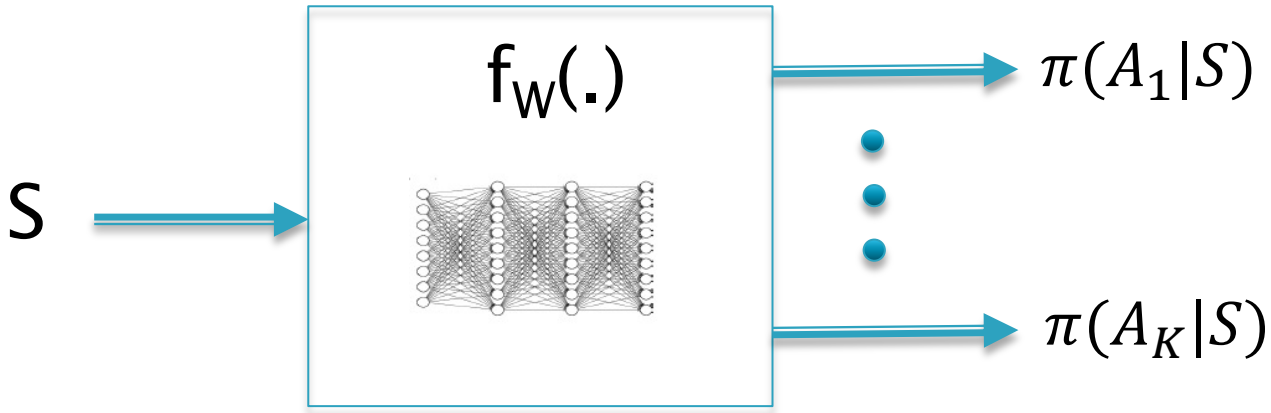


Neural Network approximates the Value Function for parts of the State space outside the sample episodes

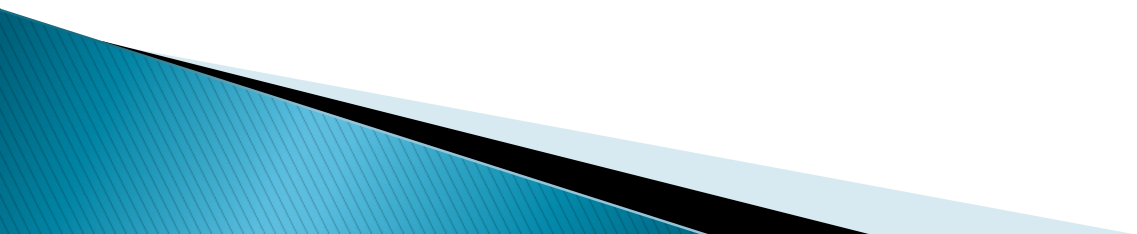
Deep RL with Policy Functions: High Level Approach

Run Sample Episodes from the System

After each episode: Modify the Neural Network to increase the probability of actions that lead to higher rewards, and decrease the probability of actions that lead to lower rewards.



Function Approximations Using Deep Learning Architectures

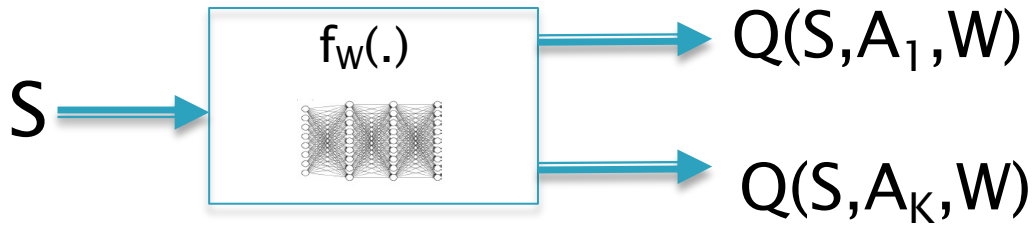


Choices for the function f_w

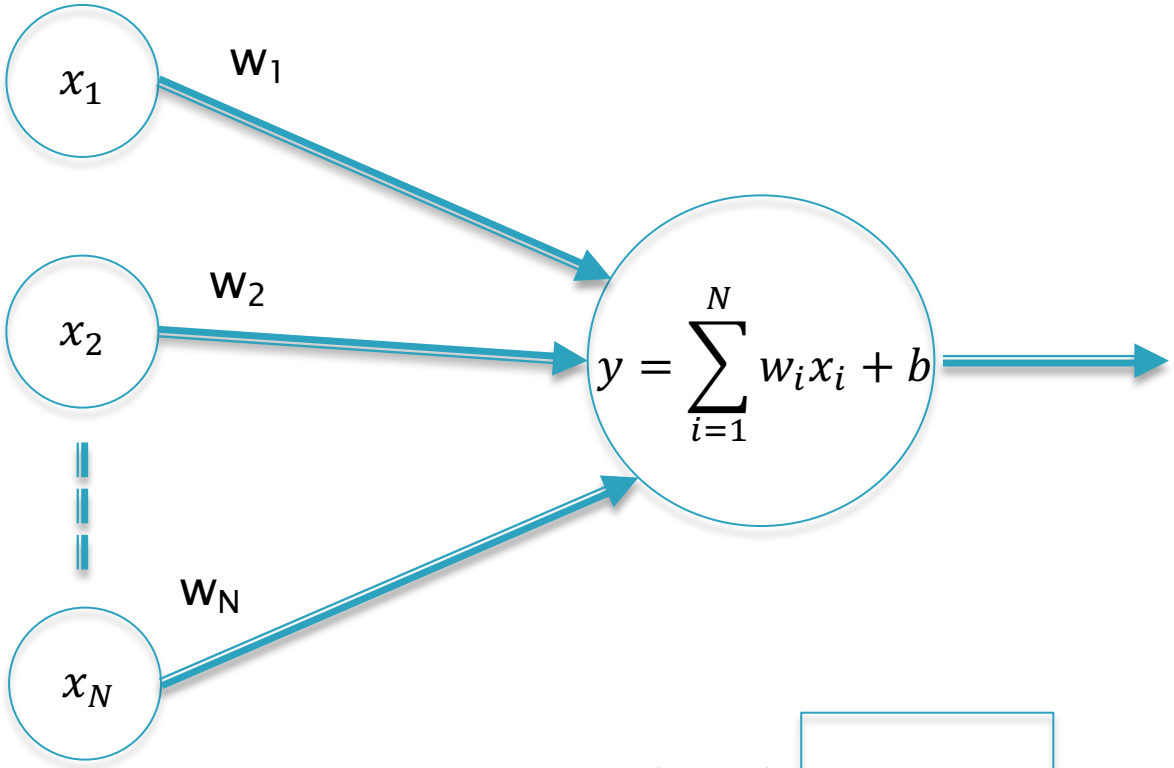
1. Linear Systems
2. Deep Feed Forward Systems
3. Convolutional Neural Networks
4. Recurrent Neural Networks
5. Transformers

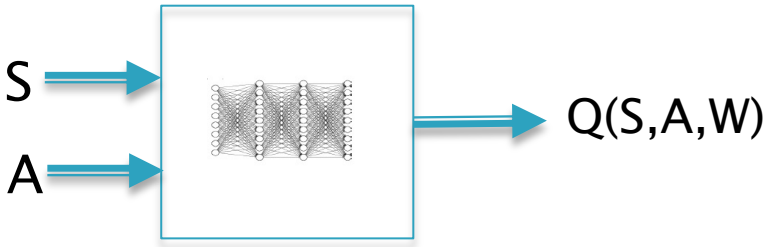
We will focus on these

Used by the Atari Game Playing RL System

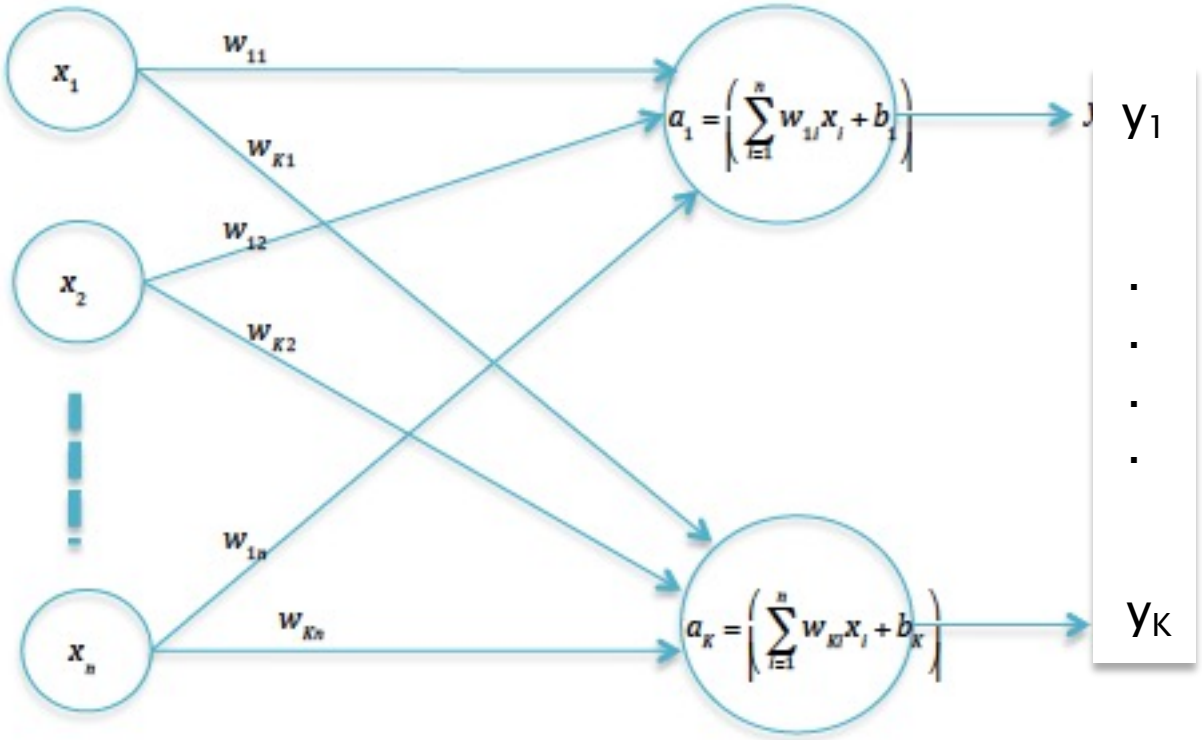


Linear Systems

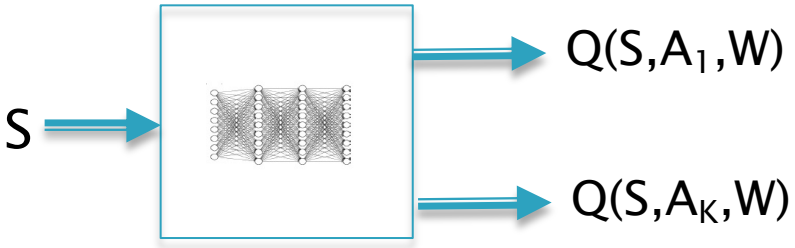




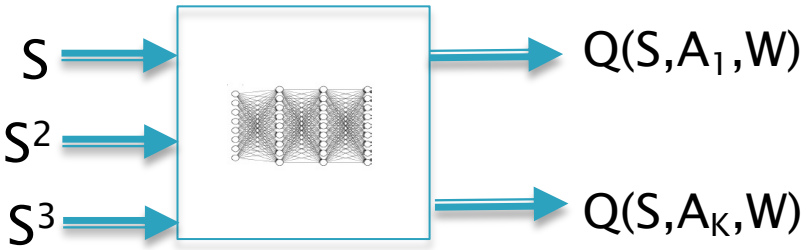
Linear System with K-ary Output



NK+K



What about Non-Linear Functions?

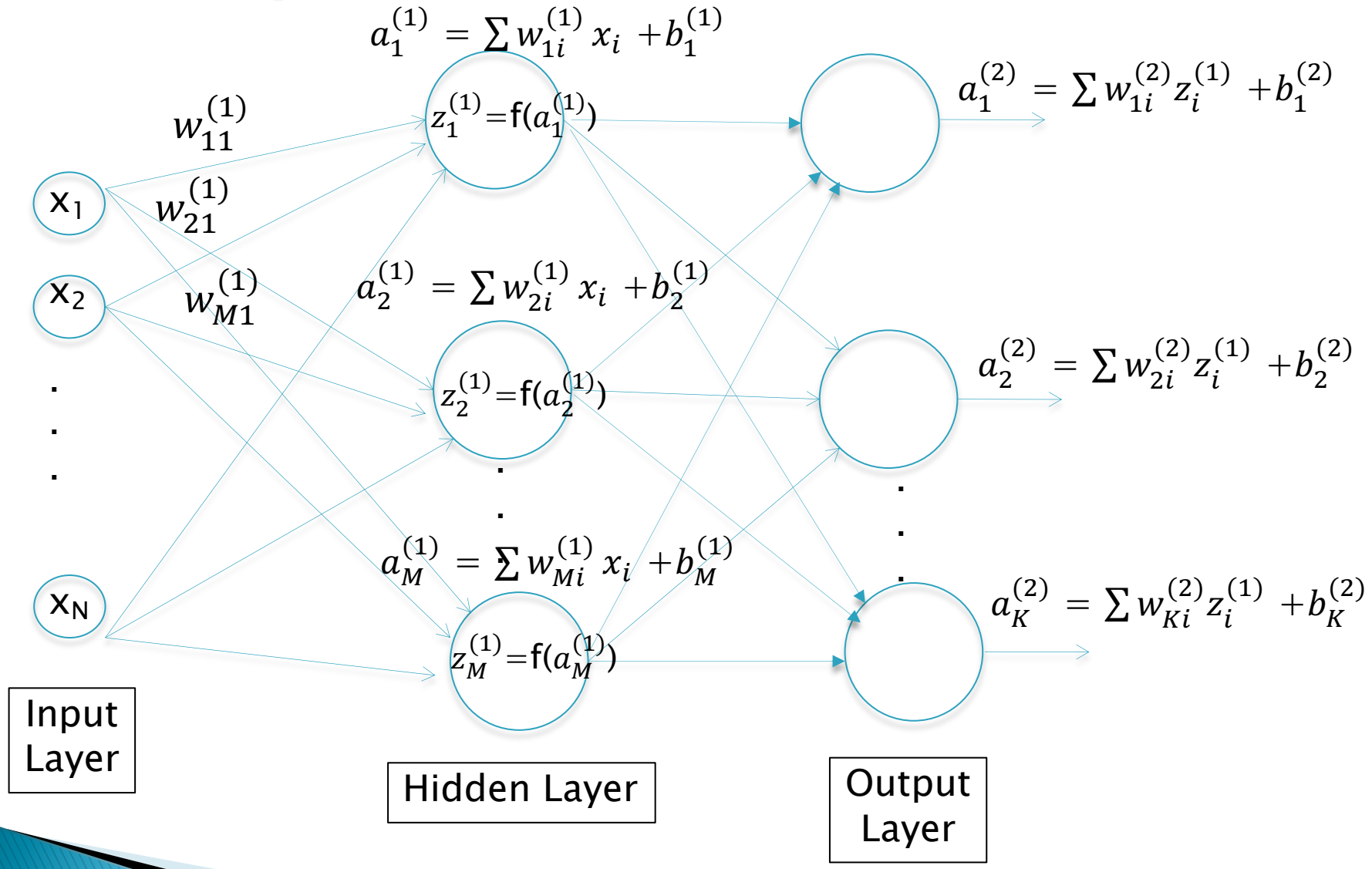


Two Solutions:

(1) Explicitly introduce non-linear inputs into a linear system
- Feature Selection

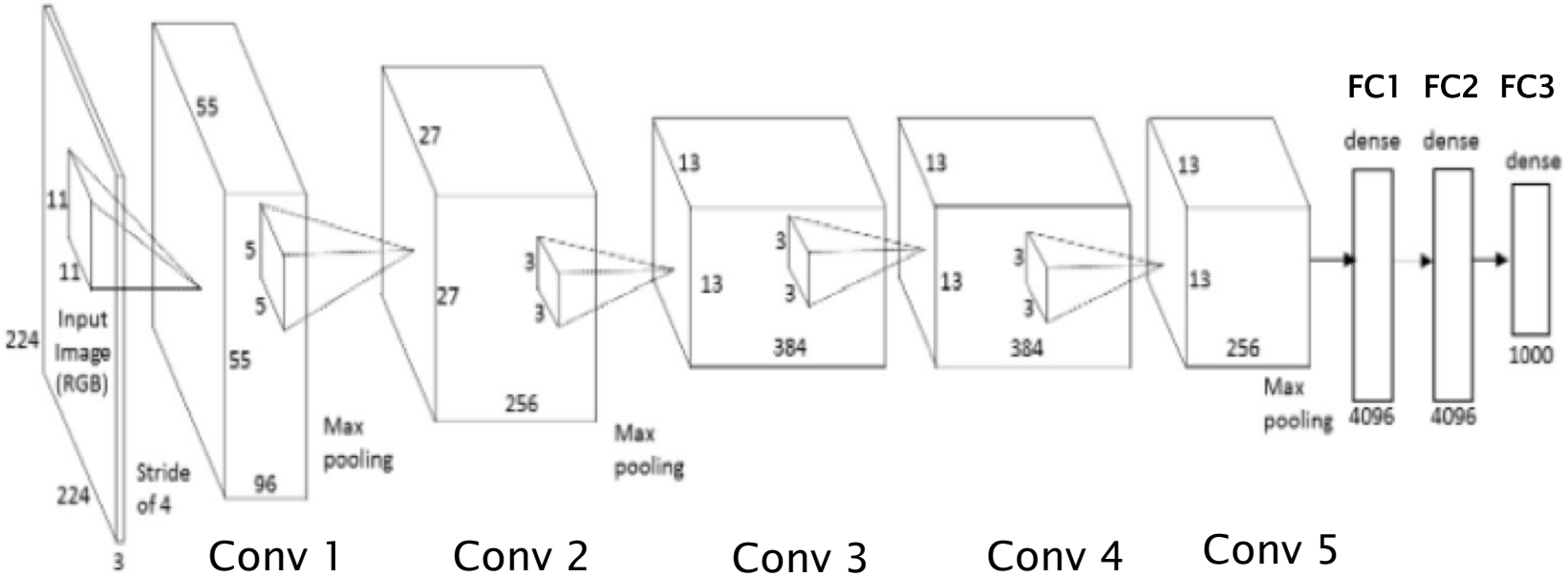
(2) Let the Training Process discover the non-linear function
- Deep Learning

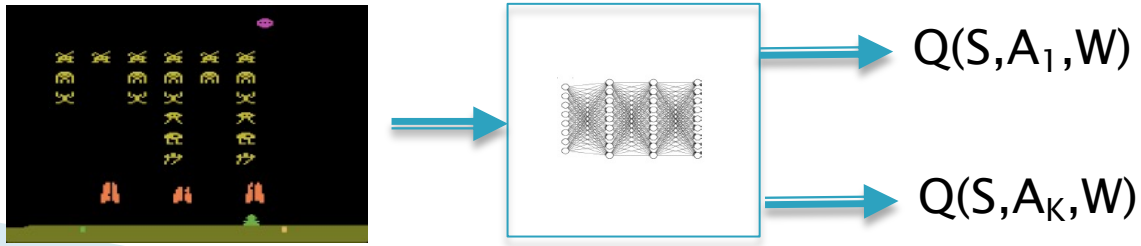
A Deep Feed Forward Network



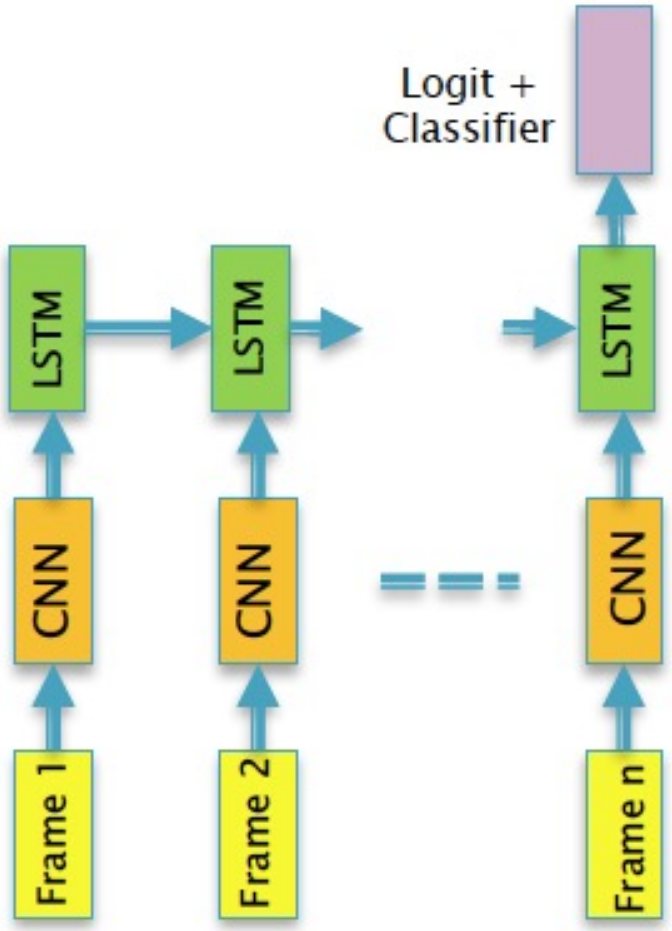
Discovers Non-Linear Functions during the Training Process

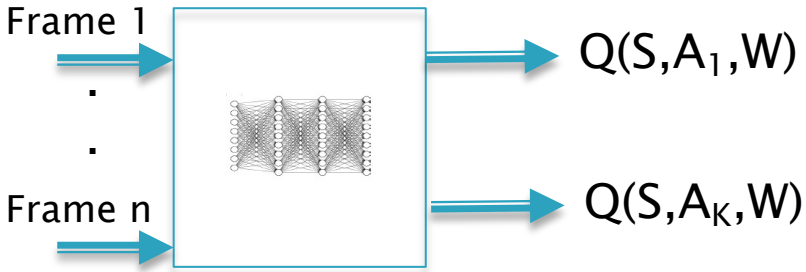
What if the State is an Image? Convolutional Neural Networks





What if the State has Memory (Video/Audio/Speech)? Recurrent Neural Networks/LSTMs/Transformers





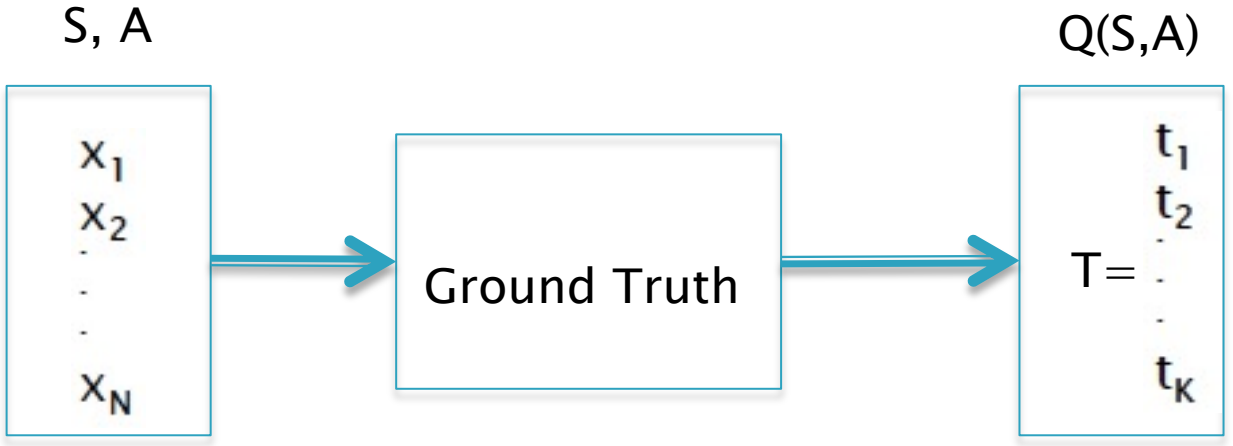
Training Neural Networks

There is a single algorithm that is used to train all types of Neural Networks!!

Stochastic Gradient Descent

Backprop: An efficient implementation of Stochastic Gradient Descent

Training Data – Supervised Learning

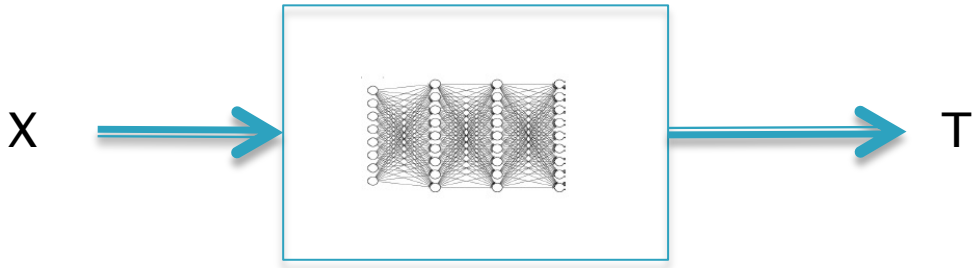


Input vector $X=(x_1,\ldots,x_N)$ is associated with
Output vector $T = (t_1, t_2, \ldots, t_K)$

$X(1) \rightarrow T(1)$
$X(2) \rightarrow T(2)$
.
.
$X(M) \rightarrow T(M)$

Training Dataset

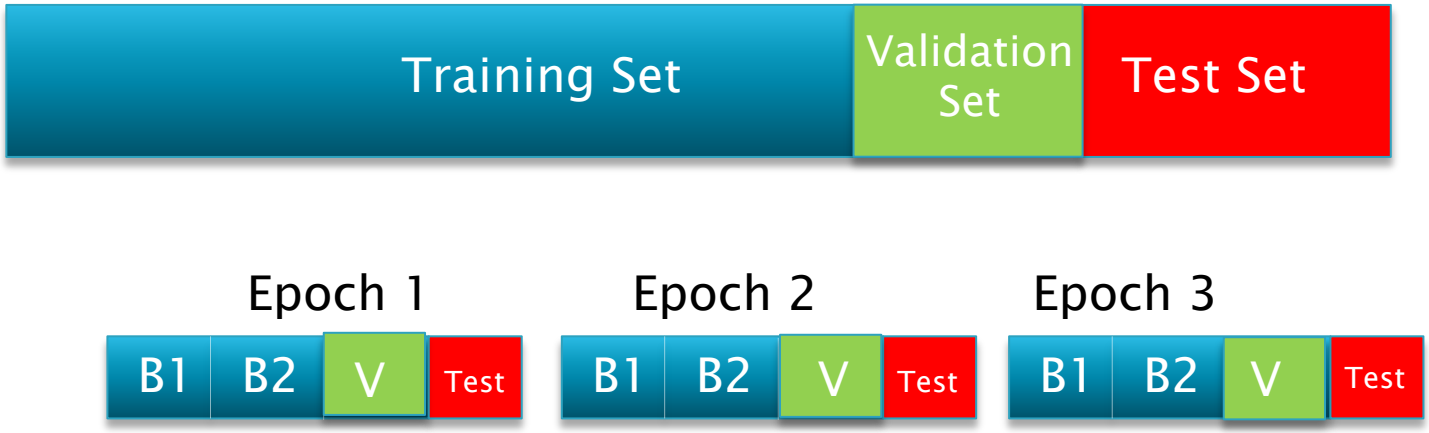
The Supervised Learning Problem



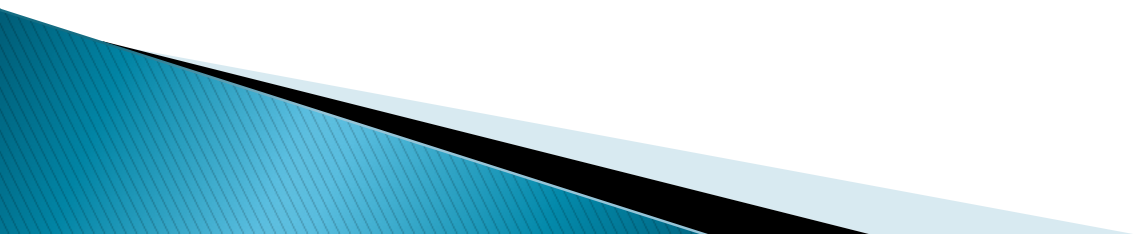
Problem: Find a model for the System, such that it is able to Predict “suitably good” values of T, for new values of X.

Test Data Set

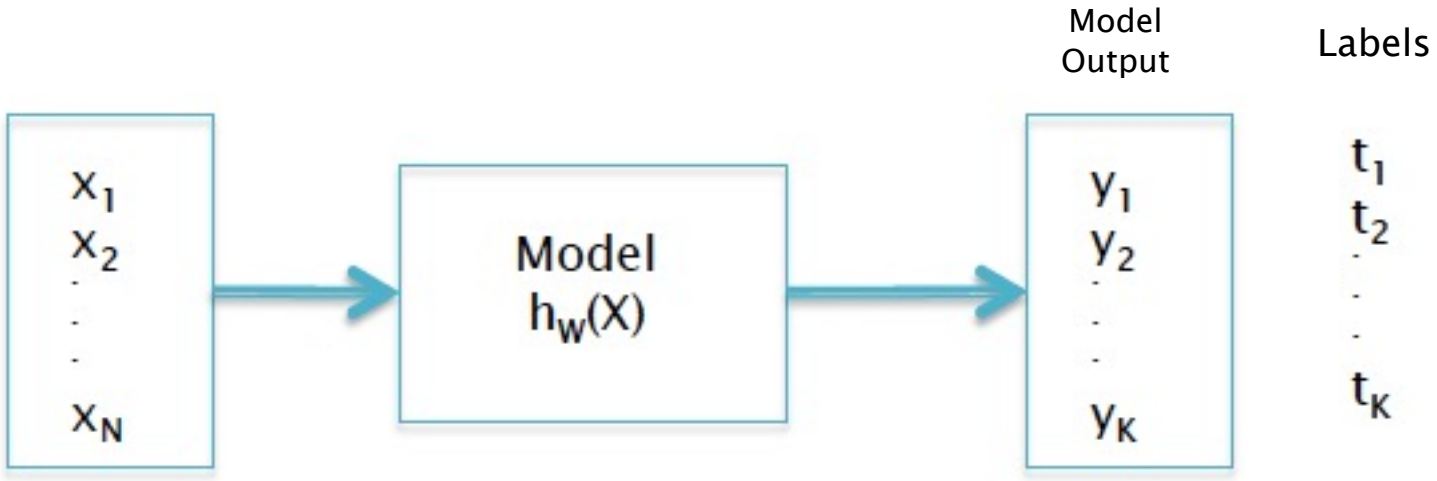
Training, Validation and Test Sets



Linear Regression



Linear Regression

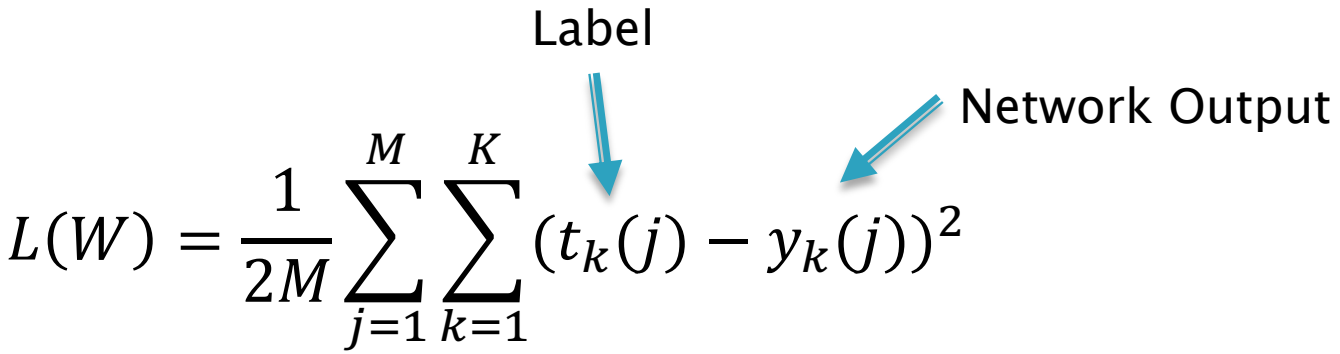


Application of the input vector $X = (x_1, x_2, \ldots, x_N)$ to the Model Results in the output vector $Y = (y_1, y_2, \ldots, y_K)$ while the desired outputs are $(t_1, t_2, \ldots, t_K)$

Training: Adjust the weights W, so that the "distance" between the Model Output y and the Label t is minimized

Testing: The model gives good results even for inputs that are not part of the Training Set

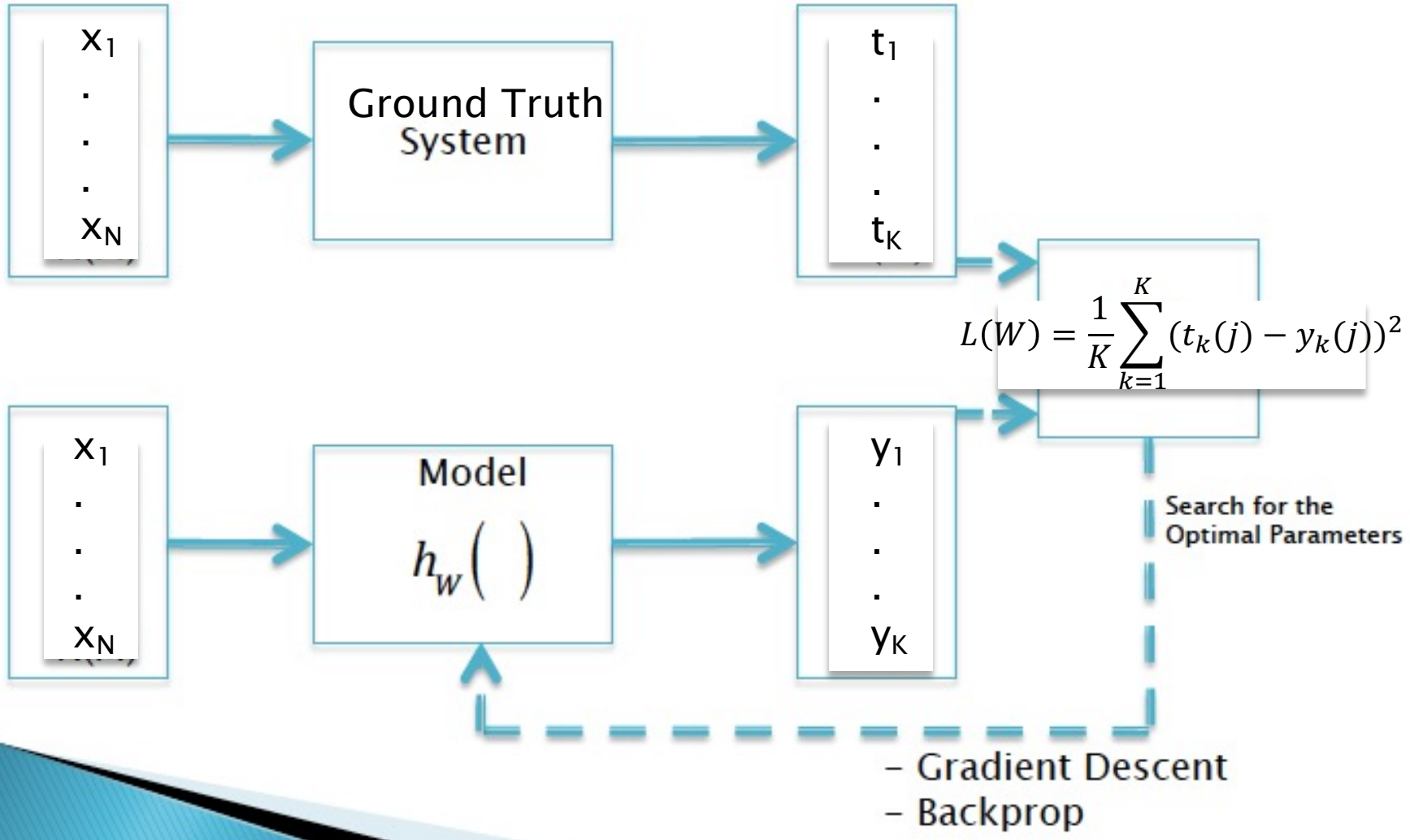
Distance Measure: Loss Function



Mean Square Error

Given the Training Data Set {X(j), T(j)}, j = 1,…,M,
The best parameters W are the ones that minimize the Mean Square Error

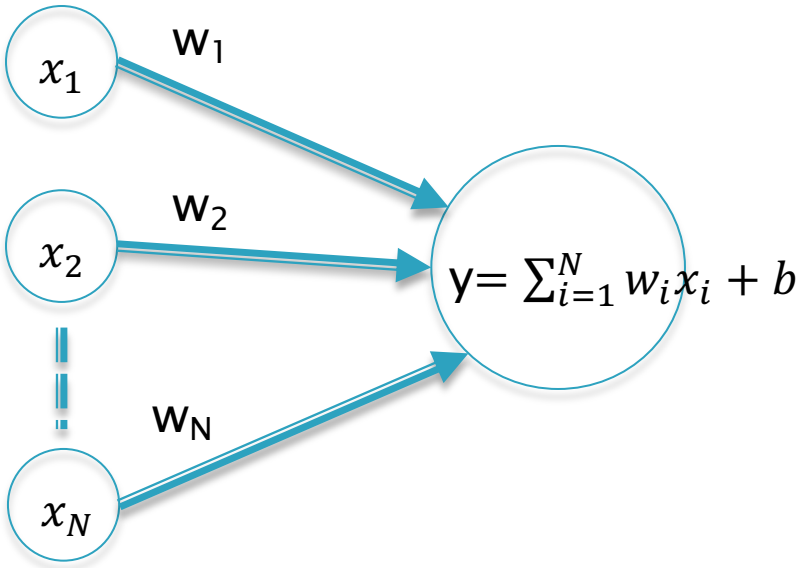
Solution to Regression Problem: Using Gradient Descent



Linear Models (Linear Regression)

Model Parameters have Linear Dependence

$$h_w(X^{(i)}) = W^T X^{(i)} + b = \sum_{i=1}^{n} w_i x_i + b$$



How to Find the Weights?

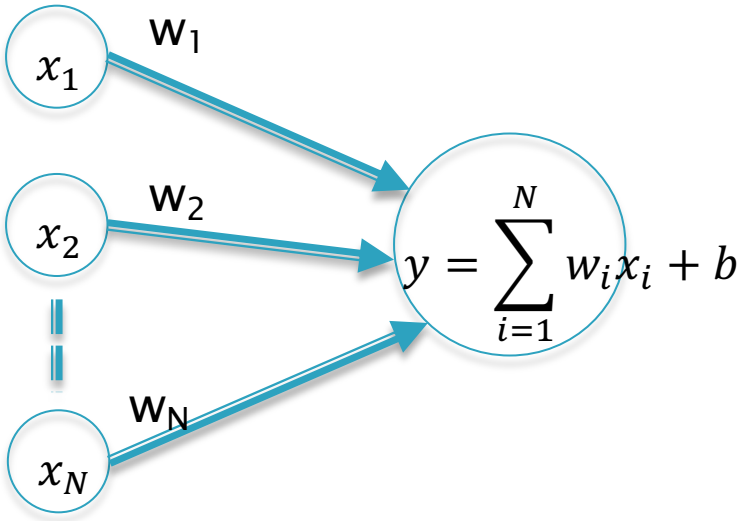
Given training samples
(X(j),T(j)), j=1,…,M

Find Weights that minimize the Loss Function

$$L(W) = \frac{1}{2M}\sum_{j=1}^{M}(y(j) - t(j))^2$$

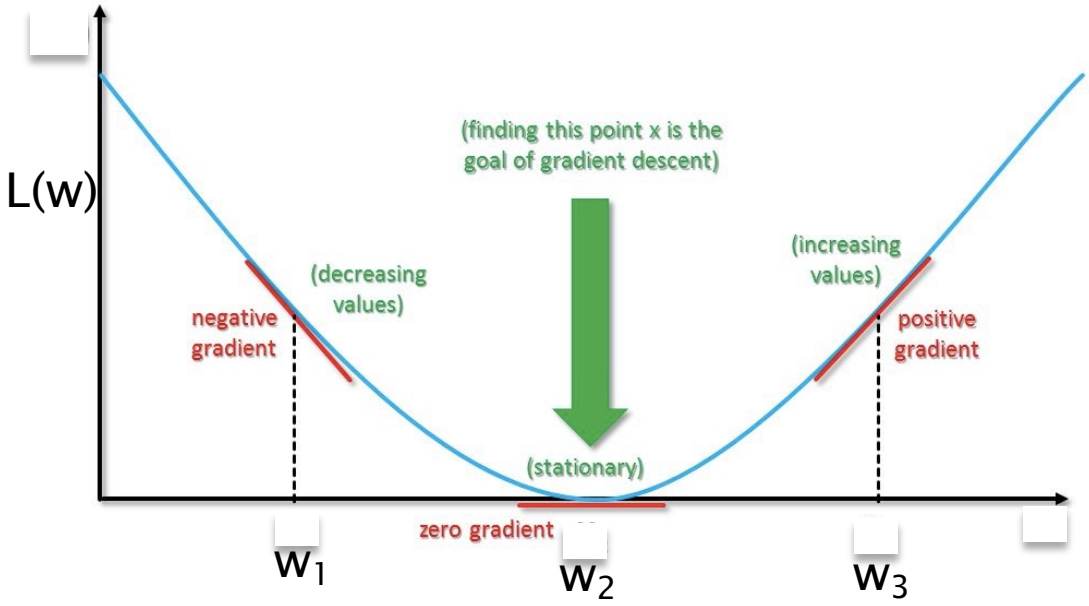
where

$$y(j) = \sum_{i=1}^{N} w_i x_i(j) + b$$



Gradient Descent: An Iterative Algorithm to find the Minimum

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$



Function Minimization by Iteration

Minimization Using Gradient Descent

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

where

Error for j[th] sample

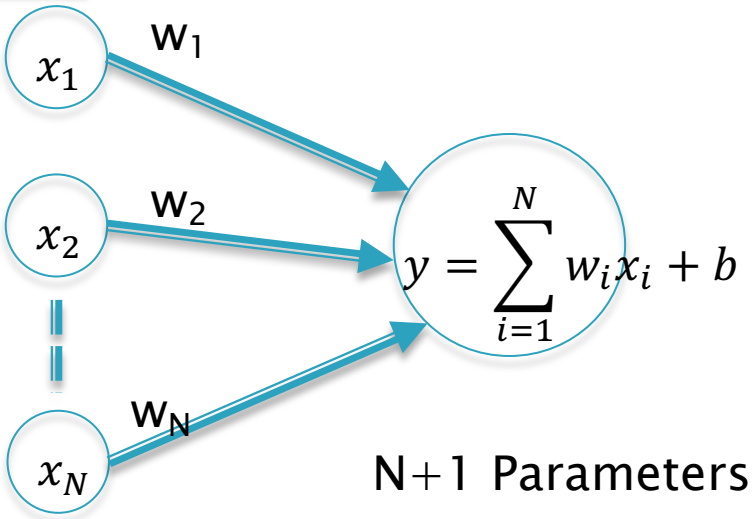
$$\frac{\partial L}{\partial w_i} = [y(j) - t(j)]x_i(j)$$

$$L(W) = (y(j) - t(j))^2$$

$$y(j) = \sum_{i=1}^{N} w_i x_i(j) + b$$

$$\frac{\partial L}{\partial b} = y(j) - t(j)$$

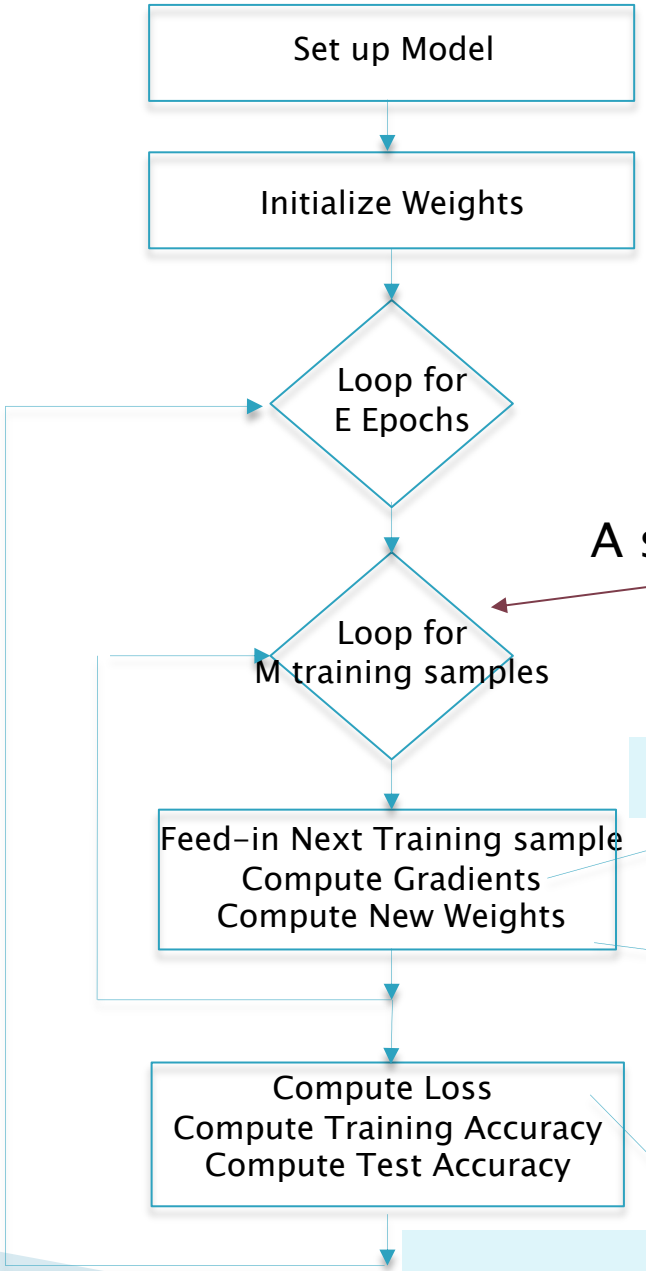
$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$



N+1 Parameters

Weight Updates Using Stochastic Gradient Descent

Linear Regression



$$\frac{\partial L}{\partial w_i} = x_i(j)[y(j) - t(j)]$$

$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$

$$L(W) = \frac{1}{2M}\sum_{j=1}^{M}(y\,(j) - t\,(j))^2$$

With K Outputs

$$L(W) = \frac{1}{K}\sum_{k=1}^{K}[y_k(j) - t_k(j)]^2$$

$$y_k(j) = \sum_{i=1}^{N} w_{ik}x_i(j) + b_k$$

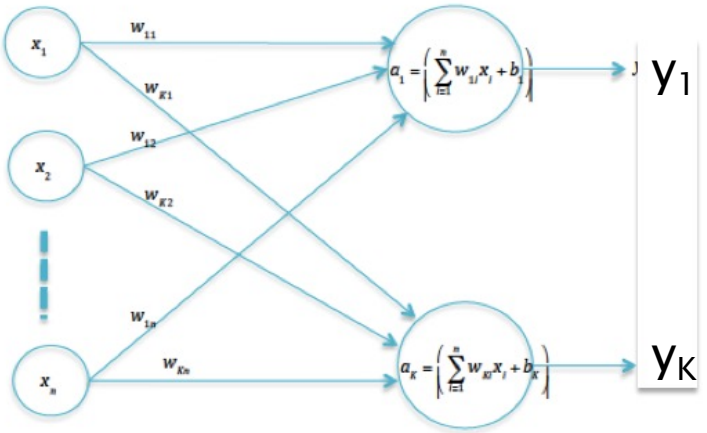
$$w_{ik} \leftarrow w_{ik} - \eta\frac{\partial L}{\partial w_{ik}}$$

where

$$\frac{\partial L}{\partial w_{ik}} = [y_k(j) - t_k(j)]x_i(j)$$

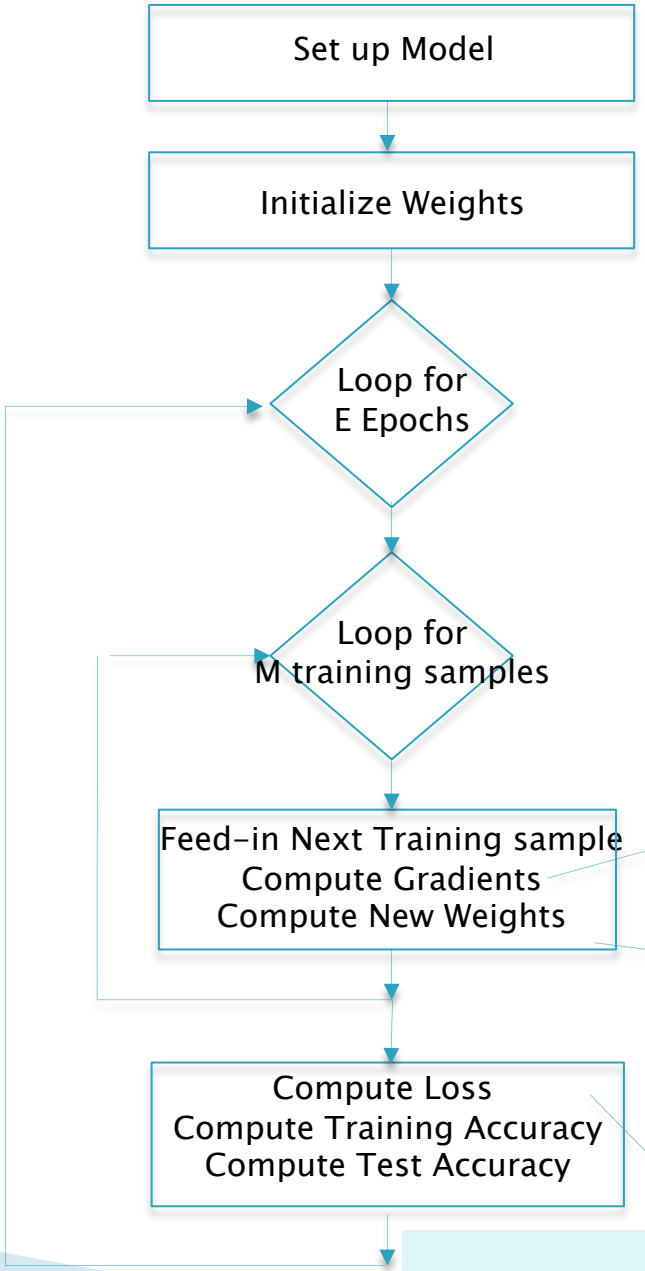
$$\frac{\partial L}{\partial b_k} = y_k(j) - t_k(j)$$

$$w_{ik} \leftarrow w_{ik} - \eta x_i(j)[y_k(j) - t_k(j)]$$



NK+K Parameters

Weight Updates Using Stochastic Gradient Descent



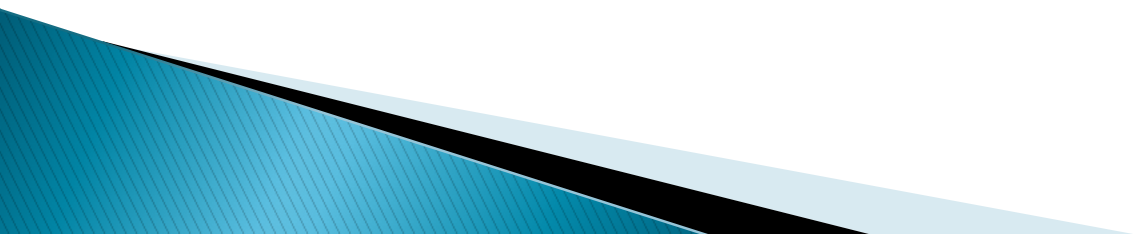
Linear Regression

$$\frac{\partial L}{\partial w_{ik}} = [y_k(j) - t_k(j)]x_i(j)$$

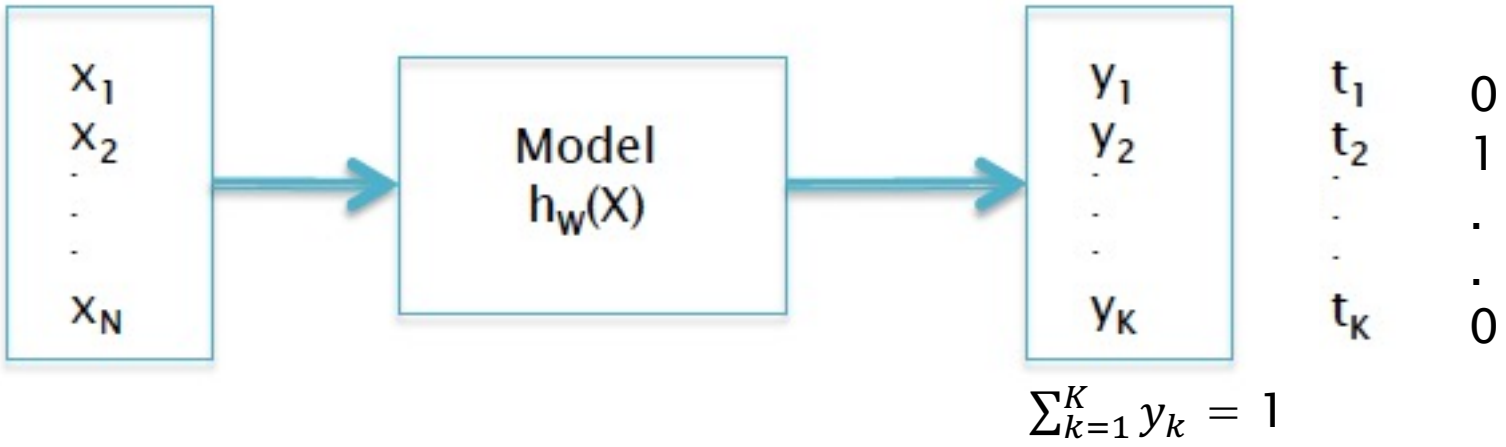
$$w_{ik} \leftarrow w_{ik} - \eta x_i(j)[y_k(j) - t_k(j)]$$

$$L(W) = \frac{1}{2M}\sum_{j=1}^{M}(y\,(j) - t\,(j))^2$$

Logistic Regression



Logistic Regression: Using the System for Estimating Probabilities



Application of the input vector $X = (x_1, x_2, \ldots, x_N)$ to the Model Results in the output vector $Y = (y_1, y_2, \ldots, y_K)$ while the desired outputs are $(t_1, t_2, \ldots, t_K)$

Training: Adjust the weights W, so that the "distance" between the Model Output y and the Label t is minimized

Testing: The model gives good results even for inputs that are not part of the Training Set

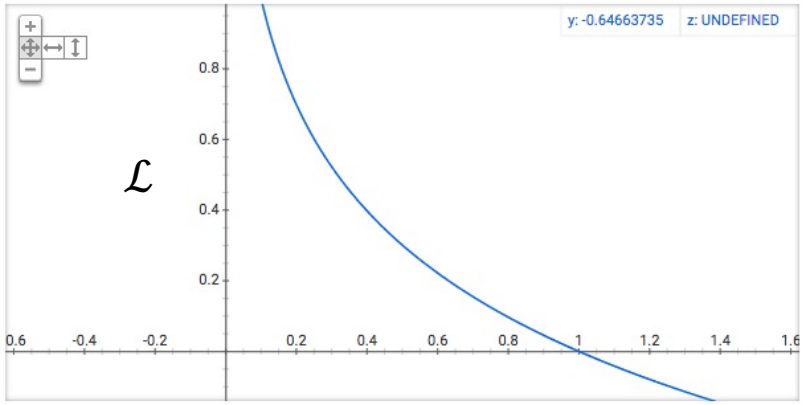
Loss Function for Probability Estimation

$$L(W) = -\frac{1}{M}\sum_{j=1}^{M}\sum_{k=1}^{K} t_k\,(j)\log y_k(\mathrm{j})$$

Cross Entropy Error

Given the Training Data Set {X(j), T(j)}, j = 1,…,M,
The best parameters W are the ones that minimize the Cross Entropy Error

The Cross Entropy Loss



$$t_q = 1$$

$$\mathcal{L} = -\log y_q,\ 0 \le y_q \le 1$$

Exact Match

$y_q = 1$

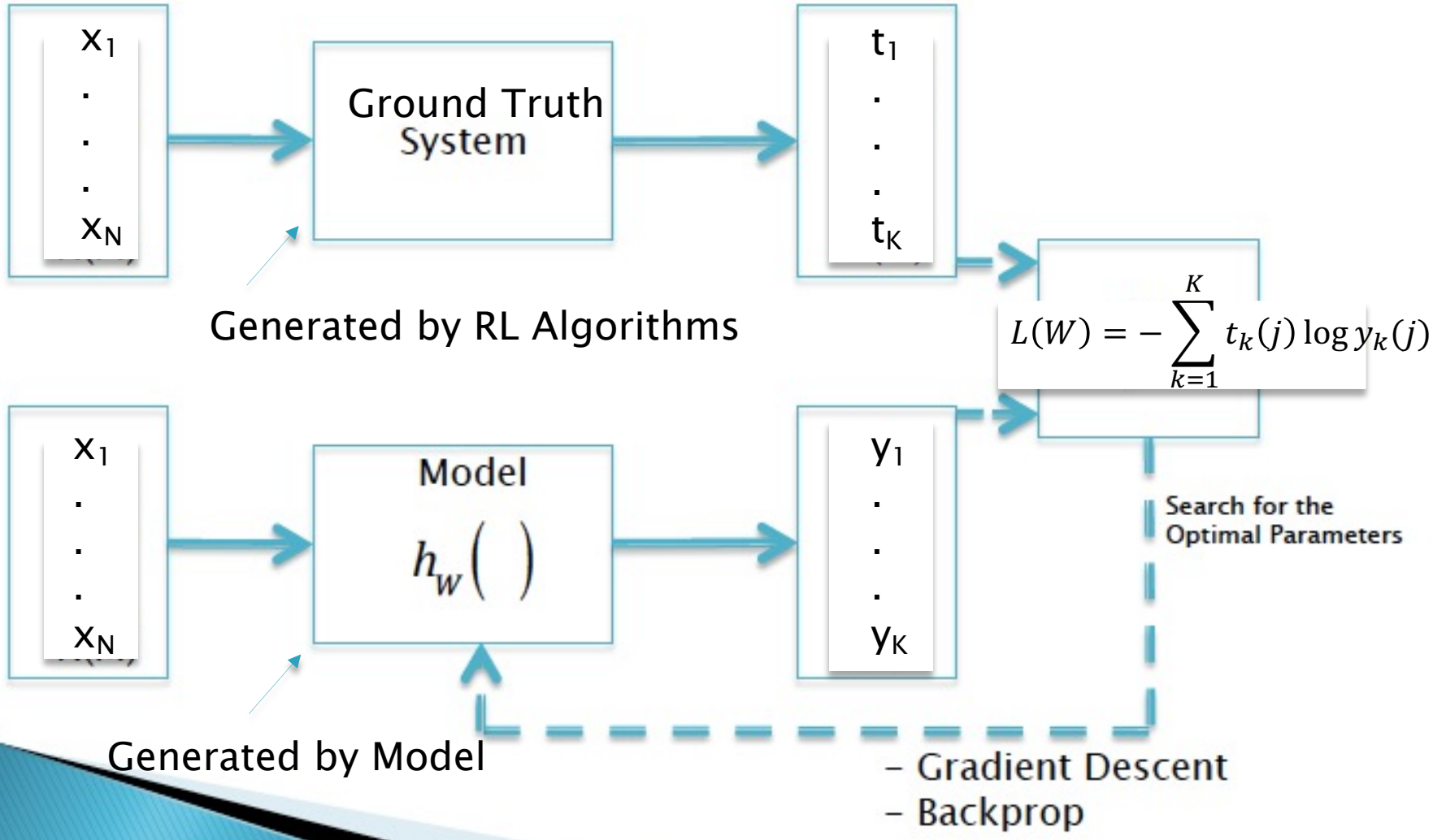
$\mathcal{L} = 0$

Complete Mismatch

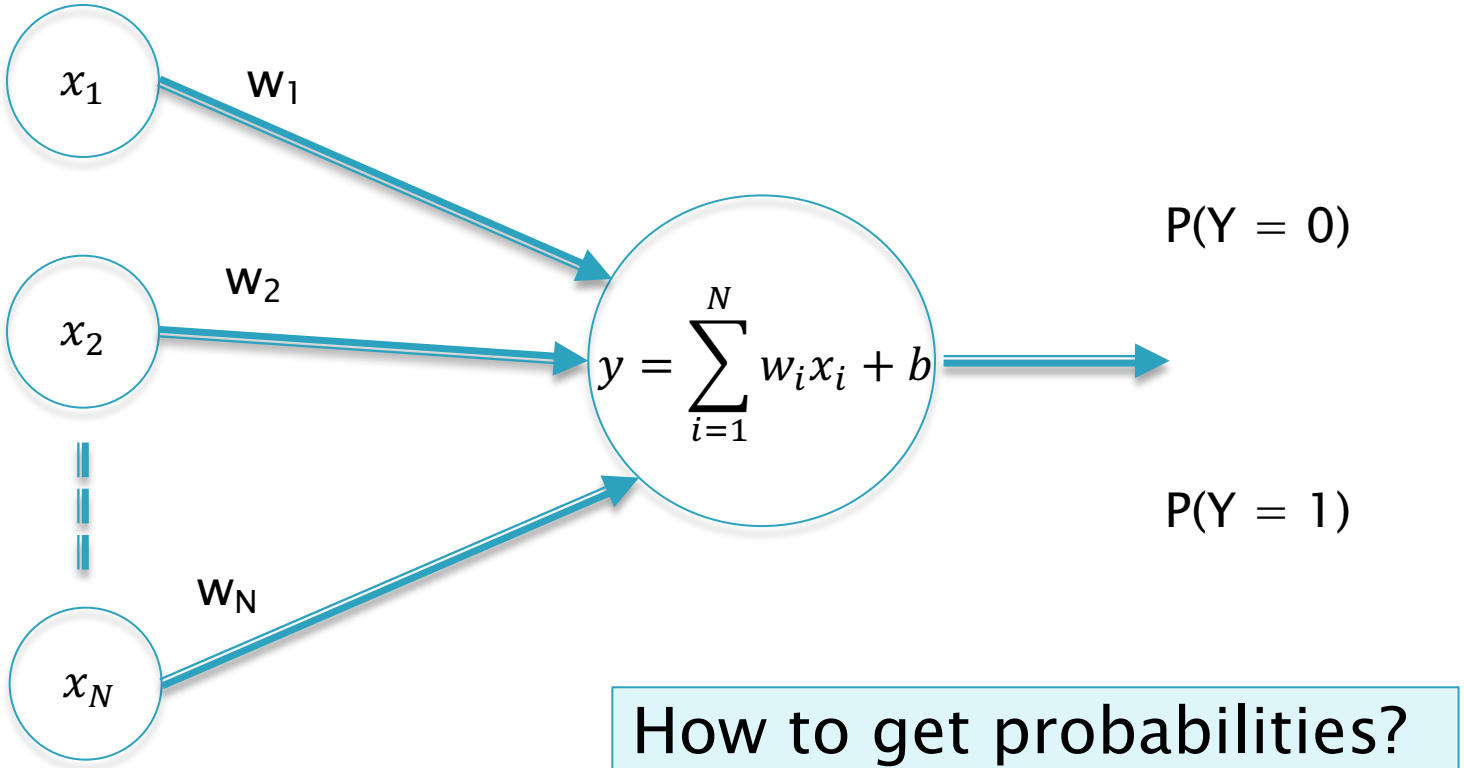
$y_q = 0$

$\mathcal{L} = \infty$

Solution to Regression Problem: Using Gradient Descent



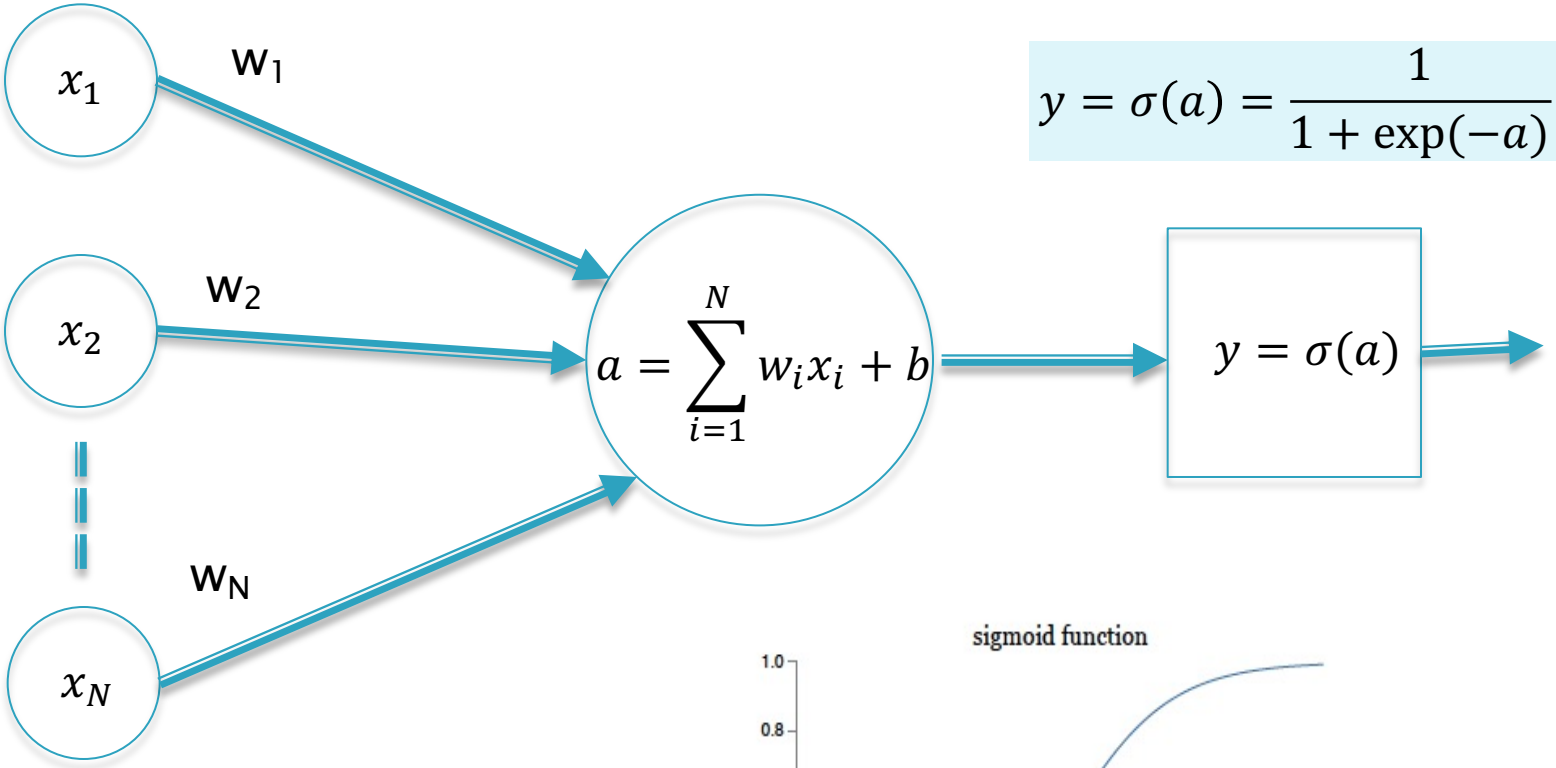
Linear Models for Probability Estimation: Logistic Regression

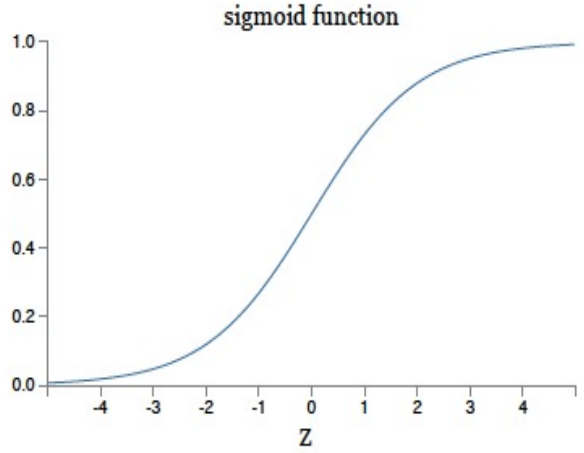


How to get probabilities?

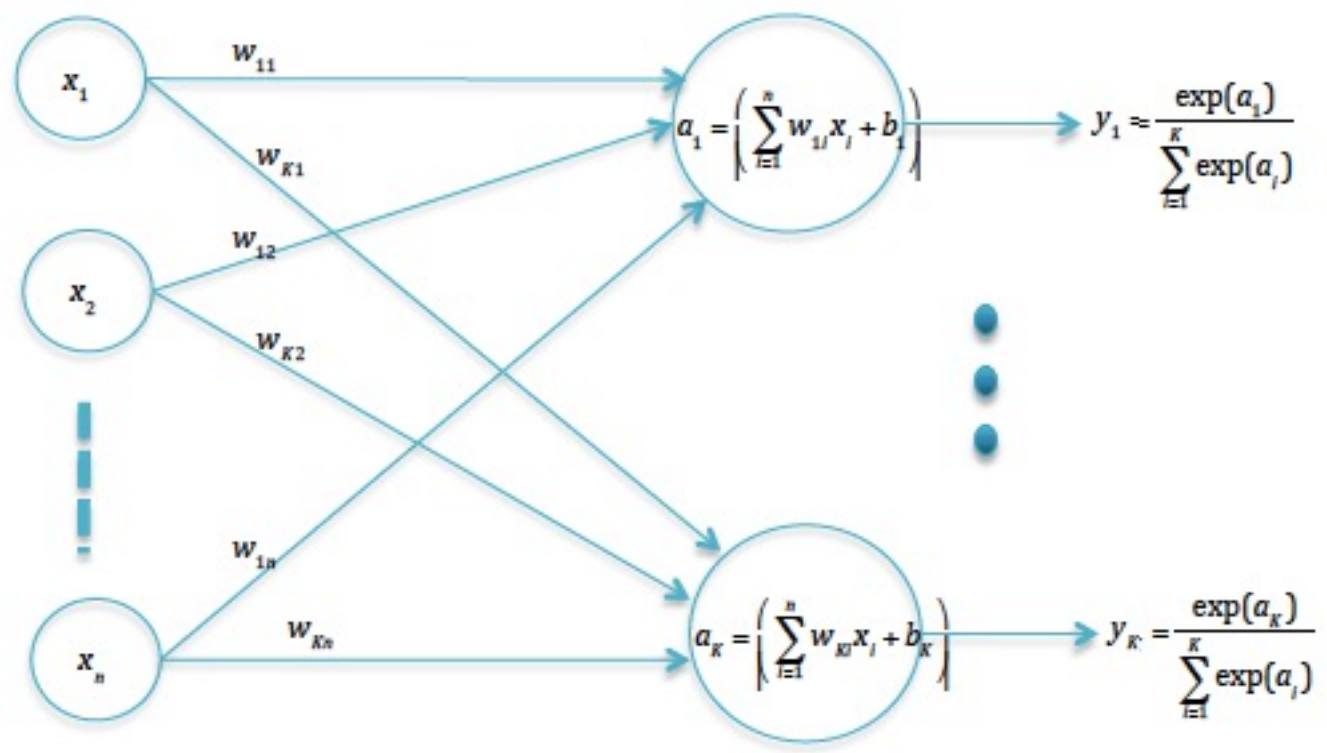
Convert Scores to Probabilities via the Sigmoid Function

$$y = \sigma(a) = \frac{1}{1 + \exp(-a)}$$





Logistic Regression with K Outputs: The Softmax Function



How to Find the Weights?

Given training samples
(X(j),T(j)), j=1,…,M

Find Weights that minimize the Loss Function

$$L(W) = -\frac{1}{M}\sum_{j=1}^{M}[t(j)\log y(j) + (1 - t(j)\log(1 - \mathrm{y(j)})]$$

$$y(j) = \frac{1}{1+\exp(-\sum_{i=1}^{n} w_i x_i(j) - b)}$$

No Closed Form solution
Will have to use Gradient Descent

$$w_i \leftarrow w_i - \eta\frac{\partial L}{\partial w_i}$$

Gradient Computation

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

where

$$y(j) = \frac{1}{1 + \exp(-\sum_{i=1}^{n} w_i x_i(j) - b)}$$

$$L(W) = -[t(j) \log y(j) + (1 - t(j)) \log(1 - y(j))]$$

$$\frac{\partial L}{\partial w_i} = [y(j) - t(j)] x_i(j)$$

$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$

Gradient Computation using Chain Rule of Differentiation

$$\mathcal{L} = -[t \log y + (1 - t) \log (1 - \mathrm{y})]$$

$$y = \frac{1}{1 + e^{-a}}, \qquad \mathrm{a} = \sum_{i=1}^{n} w_i x_i + b$$

Use Chain Rule:

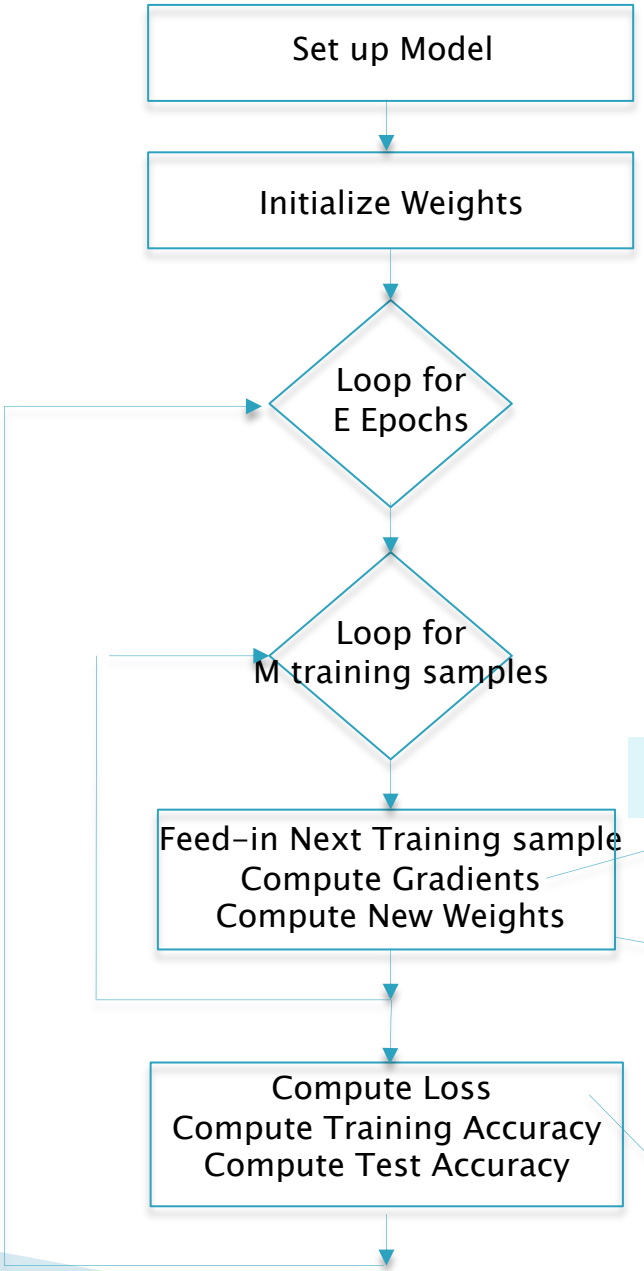
$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial a} \frac{\partial a}{\partial w_i}$$

$$\frac{y - t}{y(1 - y)} \qquad y(1 - y) \qquad x_i$$

$$\frac{\partial L}{\partial w_i} = [y(j) - t(j)] x_i(j)$$

Training Algorithm: Exactly the same as for Regression!

Training Using Stochastic Gradient Descent Logistic Regression



$$\frac{\partial L}{\partial w_i} = x_i(j)[y(j) - t(j)]$$

$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$

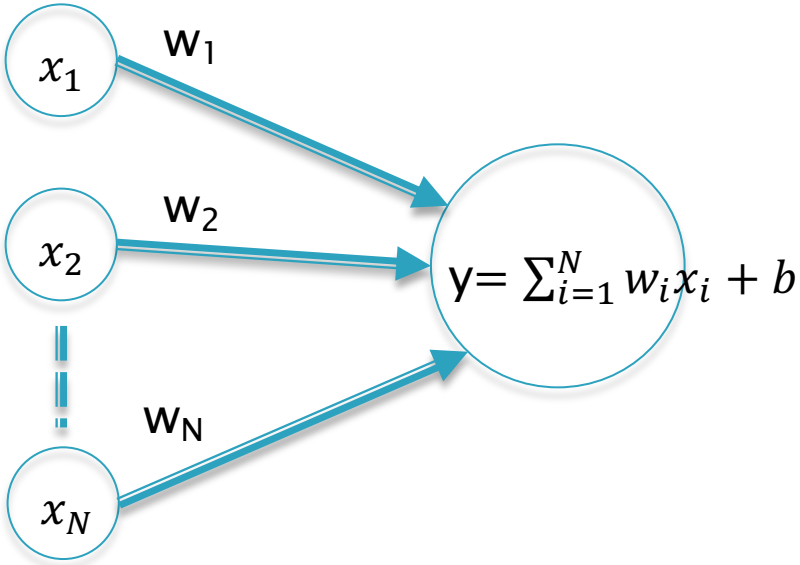
$$L(W) = -\frac{1}{M}\sum_{j=1}^{M}[t(j)\log y(j) + (1 - t(j)\log(1 - \mathrm{y(j)}]$$

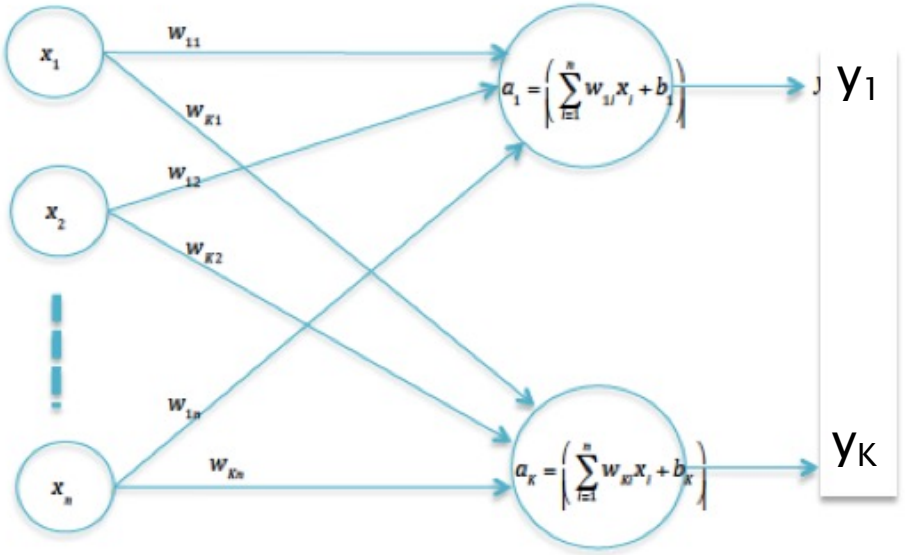
Issues in Running Gradient Descent Algorithms

- Weight Initialization
- Choosing the Learning Rate parameter η
- Deciding when to stop the algorithm
- Improving Generalization Error

These are called Hyper-Parameters

Summary – Regression

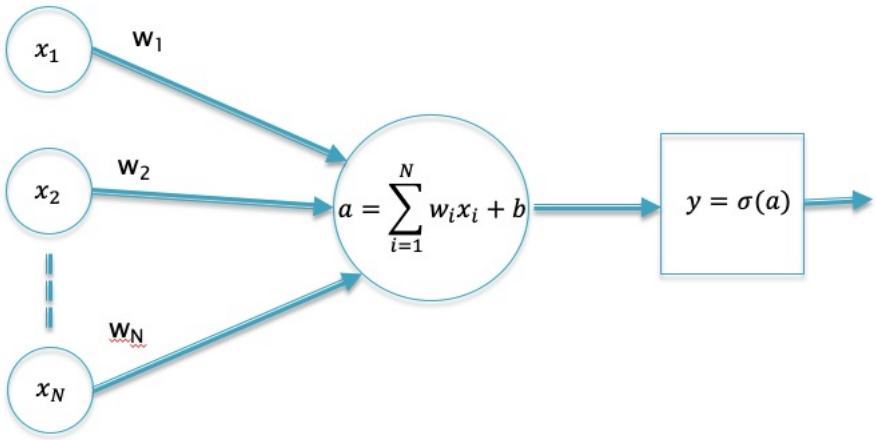


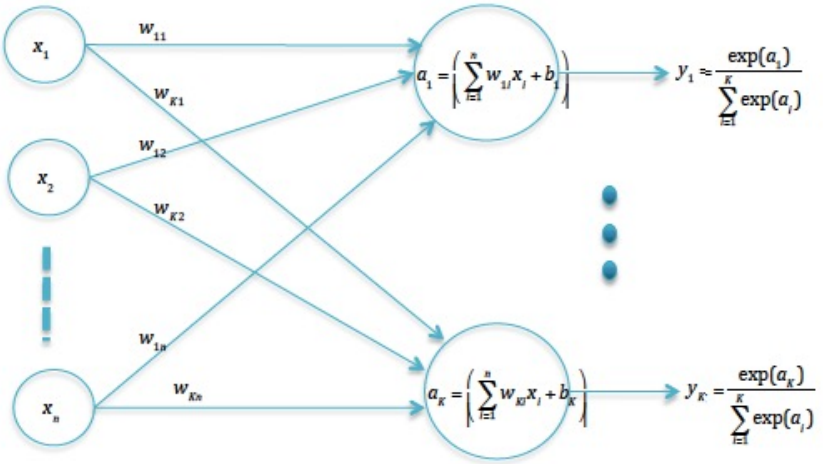


$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$

$$w_{ik} \leftarrow w_{ik} - \eta x_i(j)[y_k(j) - t_k(j)]$$

Summary – Logistic Regression





$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$

$$w_{ik} \leftarrow w_{ik} - \eta x_i(j)[y_k(j) - t_k(j)]$$

Further Reading

"Introduction to Deep Learning" by Varma and Das: https://srdas.github.io/DLBook2/

Chapter 1: Introduction
Chapter 2: Pattern Recognition
Chapter 5: Supervised Learning
Chapter 6: Linear Learning Models