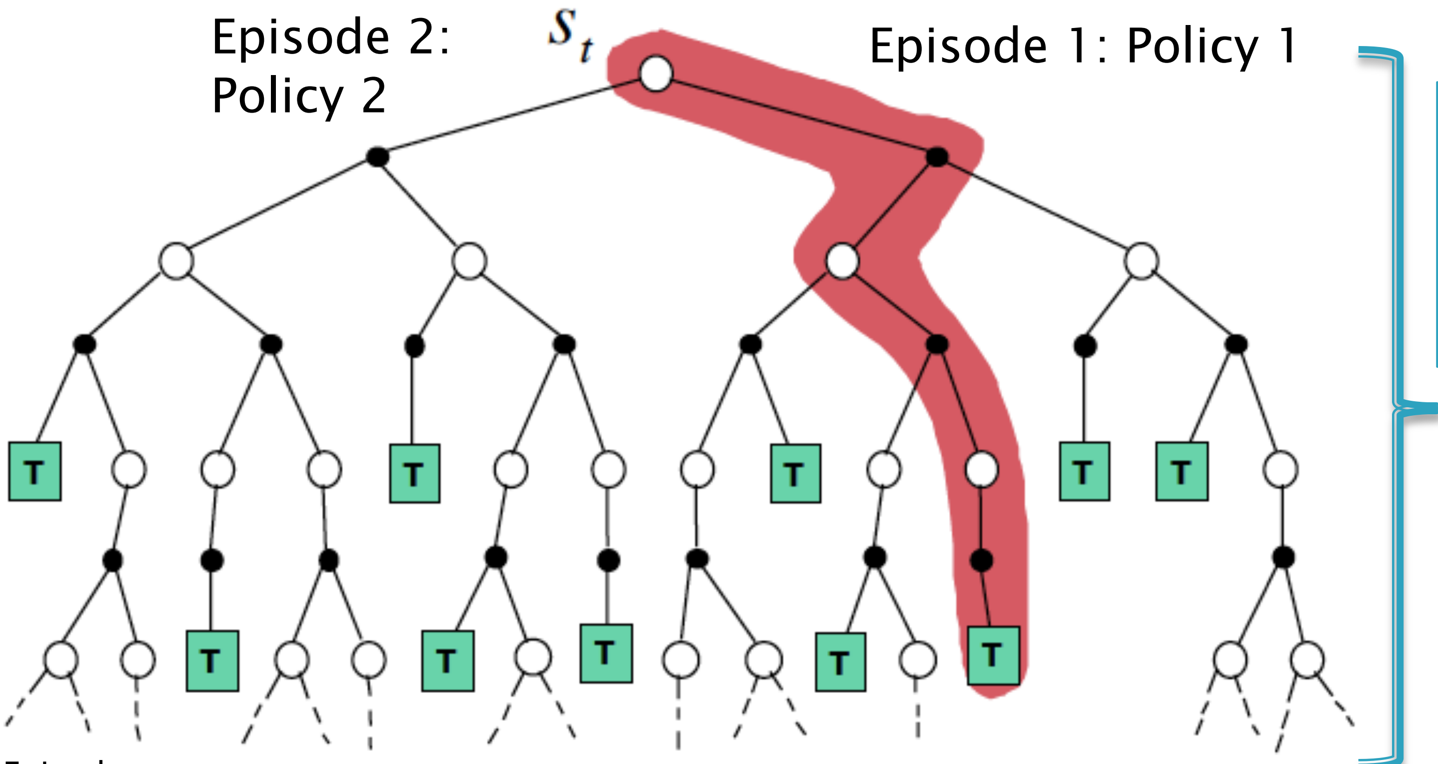
Function Approximations in Reinforcement Learning

Lecture 6

Subir Varma

Model Free Monte Carlo Control

$$Q(S, A) \leftarrow Q(S, A) + \alpha(G - Q(S, A))$$



All of the Actions are chosen according to Policy π

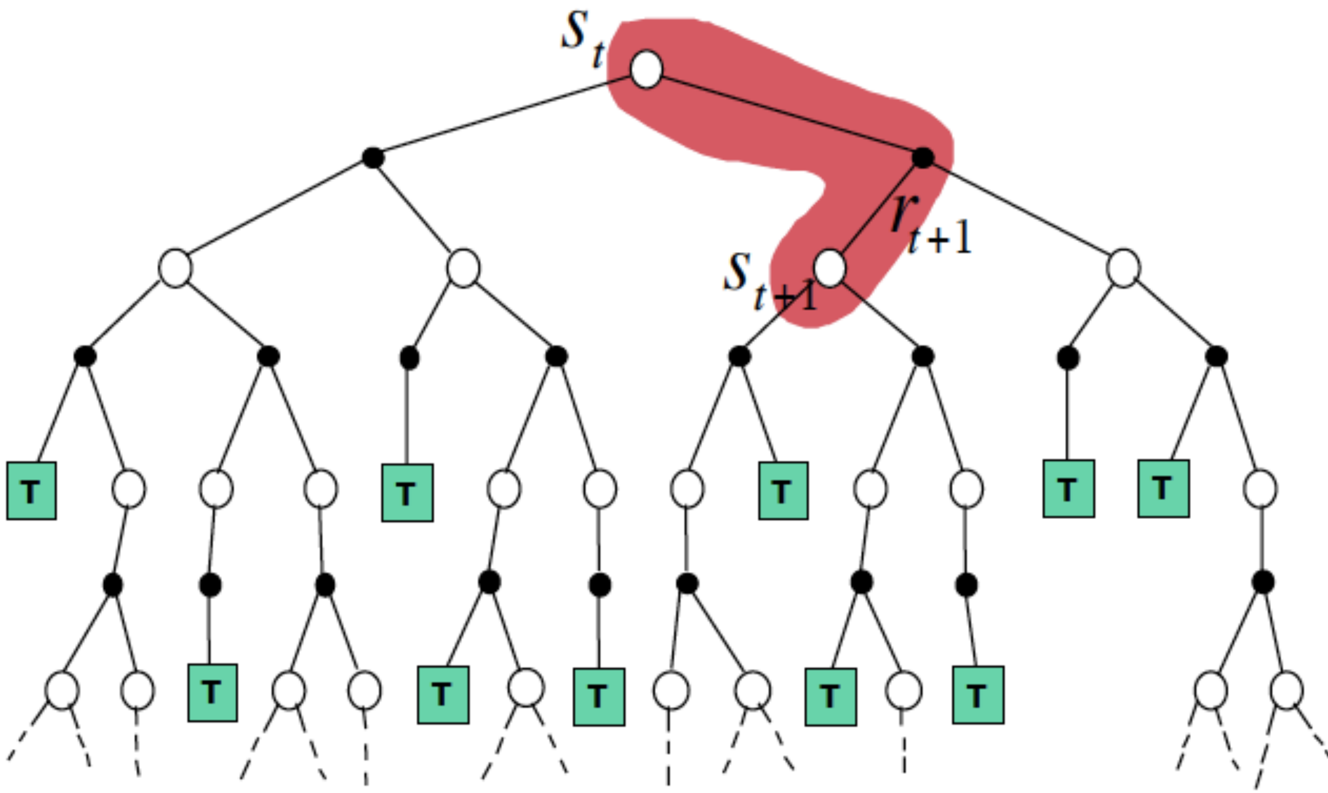
Episode

0,0 → 1,2 (–50) → 2,1 (32) → 0,0 (7) → 0,0 (7) → 0,0 (7) →
0,0 → 1,2 (–50) → 2,1 (32) → 0,0

Policy Improvement Update made at end of an Episode

Model Free On Policy Temporal-Difference – SARSA

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A) \right)$$



Actions are chosen according to Policy $\pi 1$

Actions are chosen according to Improved Policy π

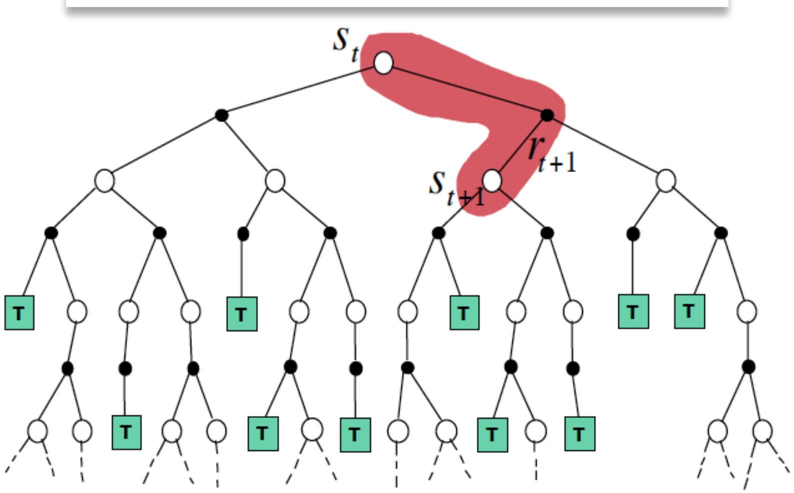
Policy being used to generate episode is the same as the policy being learnt

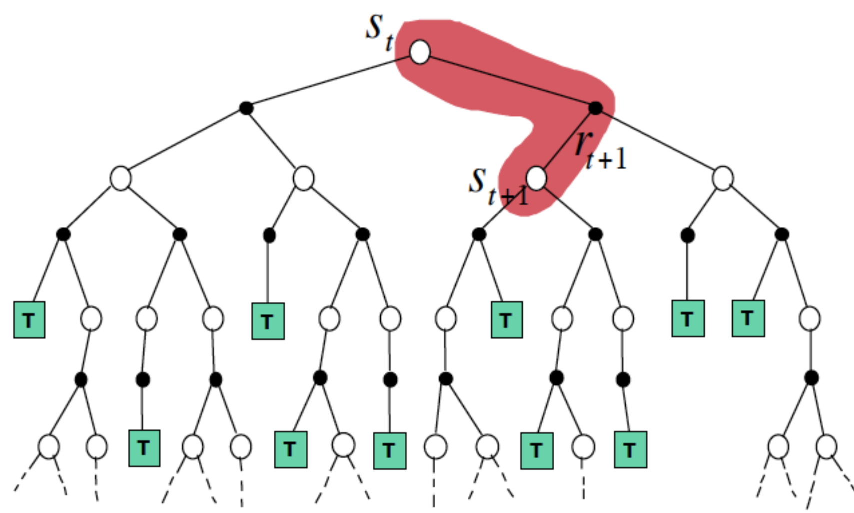
Q Learning

A special case of
Model Free
Off Policy Learning

Behavior Agent chooses actions using
The Q values that the Target Agent
computes

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$





Behavior Agent

Controls All Actions Actually Taken
Using epsilon-greedy algo

Target Agent

Follows Behavior Agent
AND In Parallel
Computes Best Possible Action

Two Policies

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 continuous

Tabular Reinforcement Learning

(S_1, A_1, R_1, S_1')
.
.
.
(S_n, A_n, R_n, S_n')

→

Monte Carlo
SARSA
Q Learning

(Table)

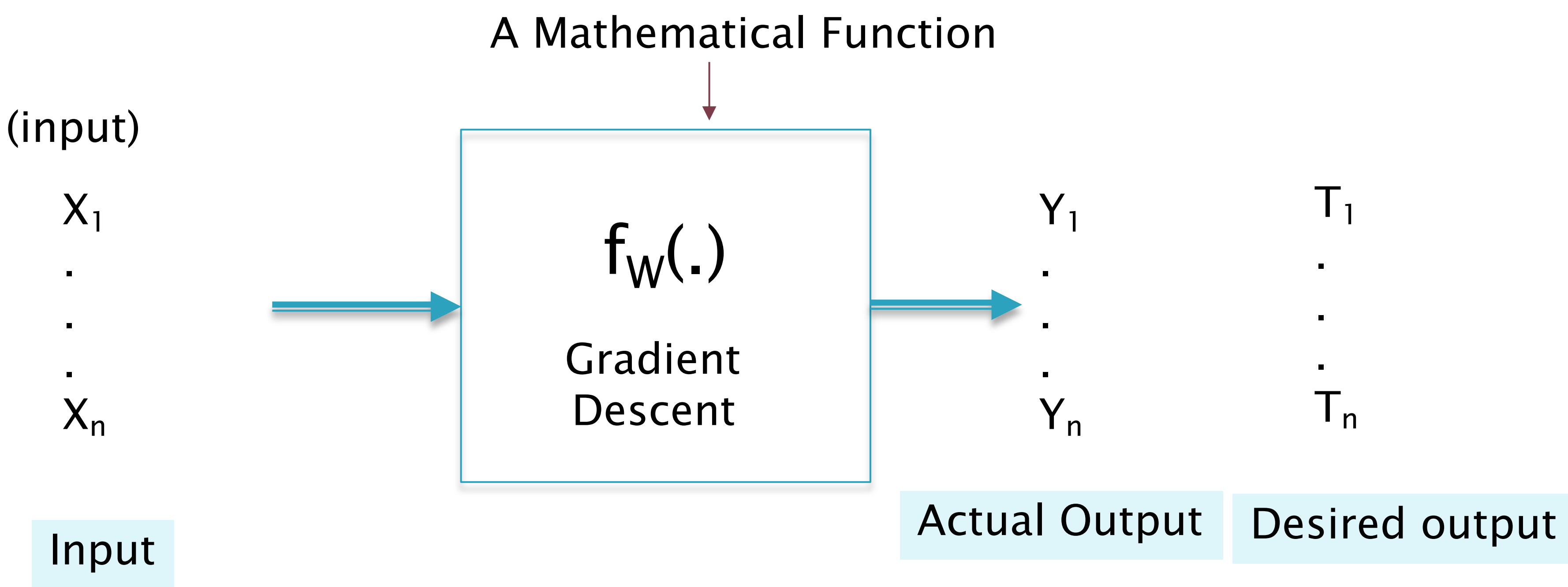
→ $\pi(A|S)$

Policy

Rollouts ←→ Training Set

Objective Function: Total Reward

Machine Learning



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

Objective Function: Distance(Y,T)

RL + ML = Deep RL (Functional RL)

More Scalable

(S_1,A_1,R_1,S_1')
.
.
.
(S_n,A_n,R_n,S_n')

Monte Carlo
SARSA
Q Learning
+
Gradient Descent

$f_W(.)$

$\pi(A|S)$

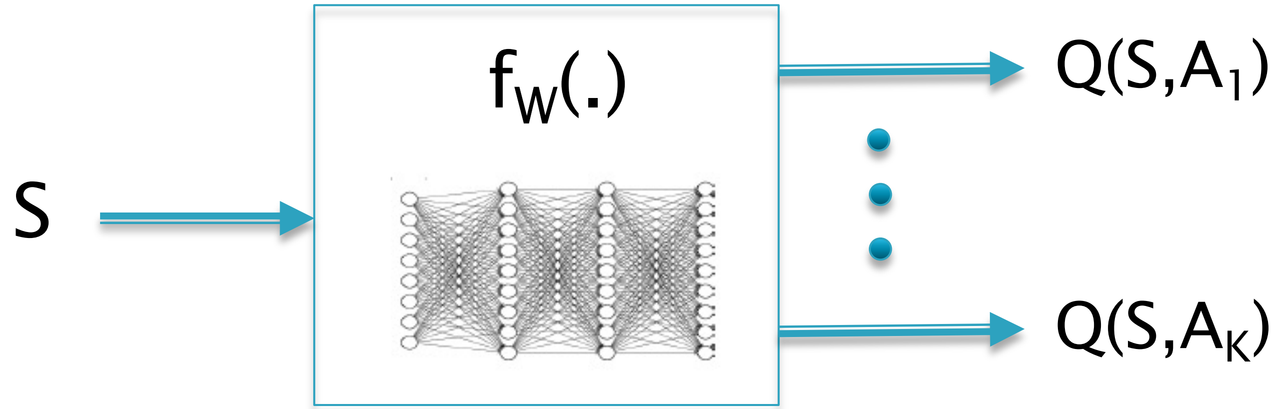
Policy

Rollouts ←→ Training Set

Objective Function: Total Reward

Instead of computing Table Entries, we are now computing Neural Network weights, but the number of weights is smaller, and the function generalizes

Deep RL Method 1: Approximating the Q Function



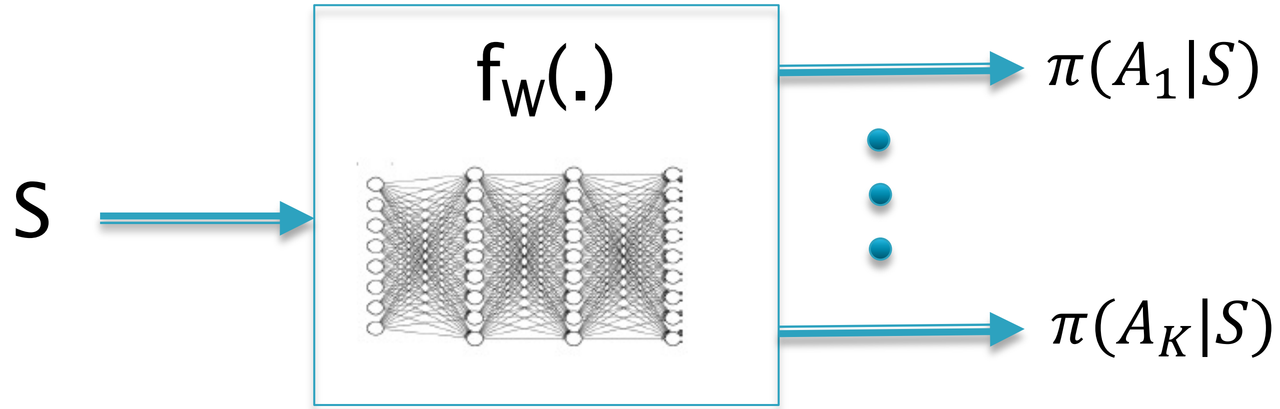
f_W is represented using a multilayer Neural Network

(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)

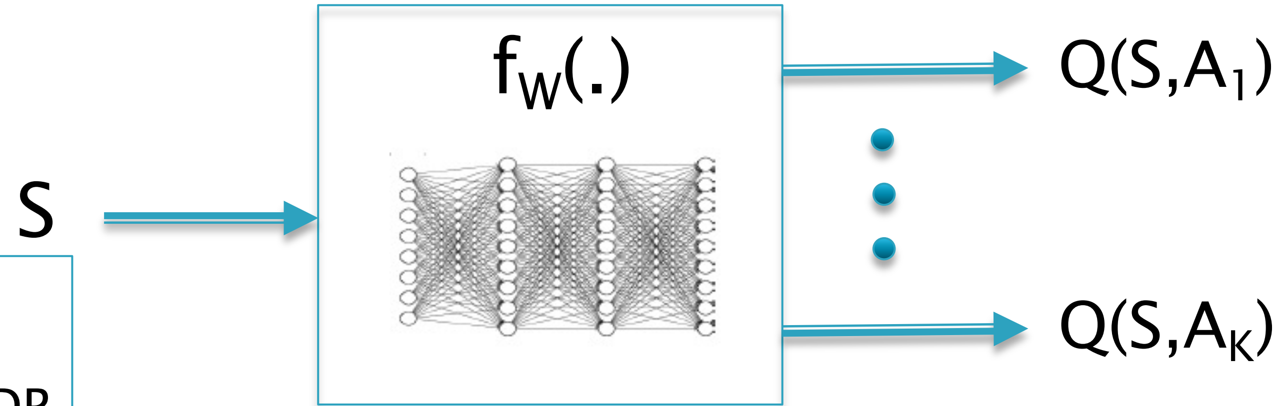
Policy Gradients Algorithms: Estimate Optimal Policy directly without first estimating the Value Function

Combining RL with DL

Two ways:

1. Use the Neural Network to approximate Q Functions
   - DQN: Deep Q Networks
   - A3C: Asynchronous Methods
2. Use the Neural Network to approximate the Policy
   - Policy Gradients Methods
   - Reinforce Algorithm
3. Use Neural Network to approximate both Q Function and Policy
   - Actor Critic Methods

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

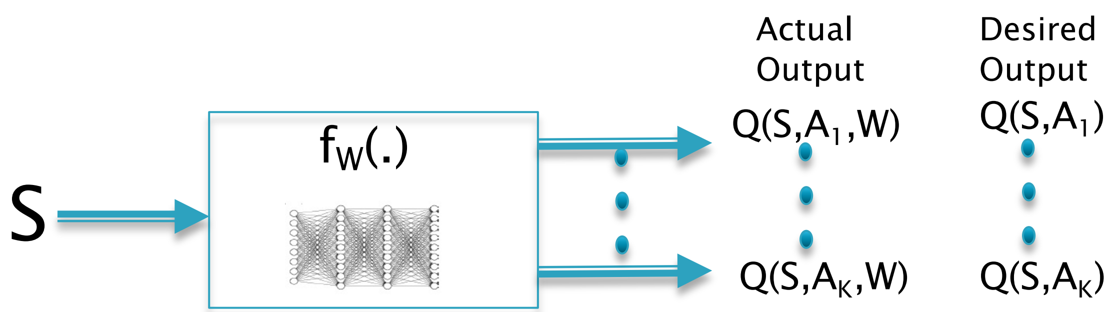
Deep RL with Q Function Approximation: High Level Approach

Run Sample Episodes from the System

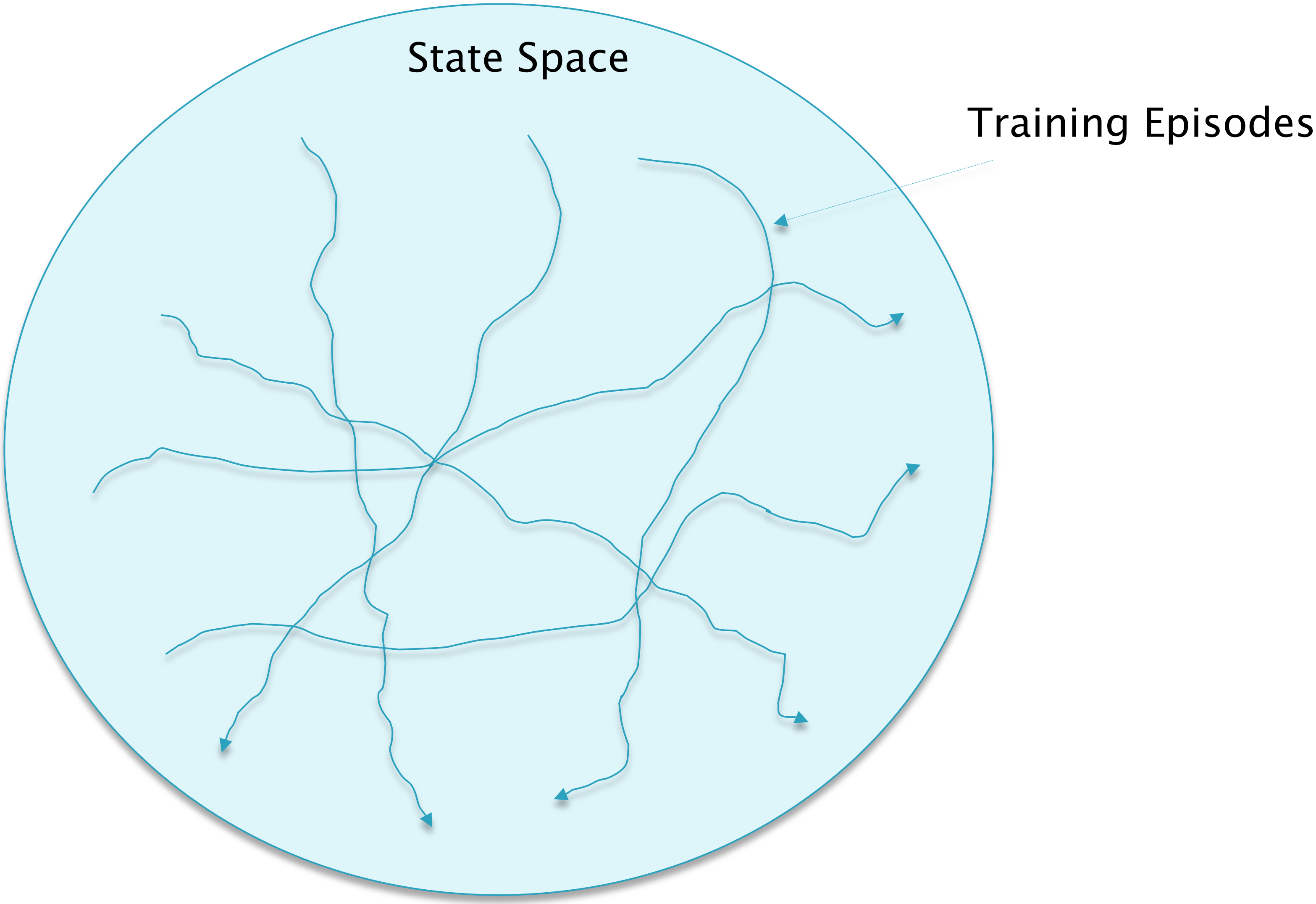
Use Monte Carlo, SARSA or Q-Learning to get samples of the mapping $(S,A,R,S') \rightarrow Q(S,A)$. This becomes the Training Data

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

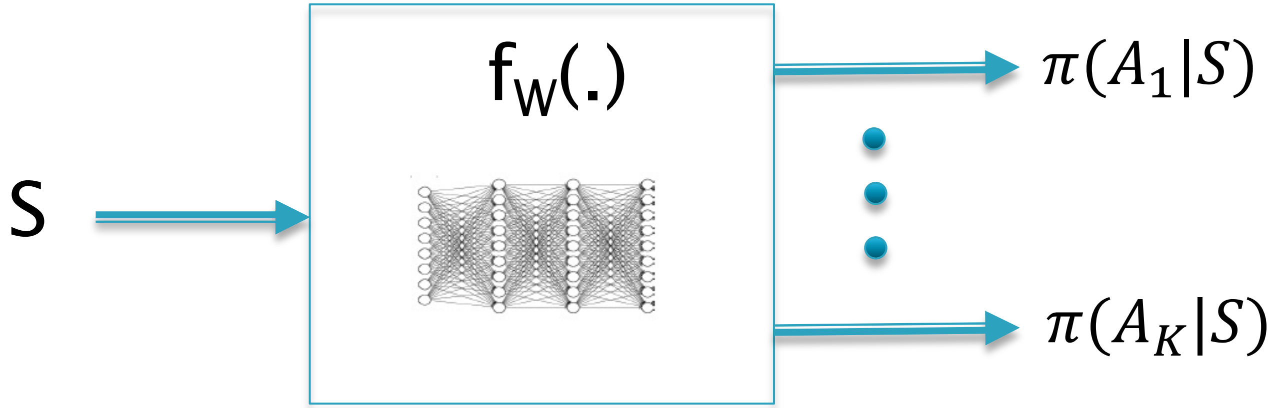


Neural Network approximates the Value Function for parts of the State space outside the sample episodes

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.



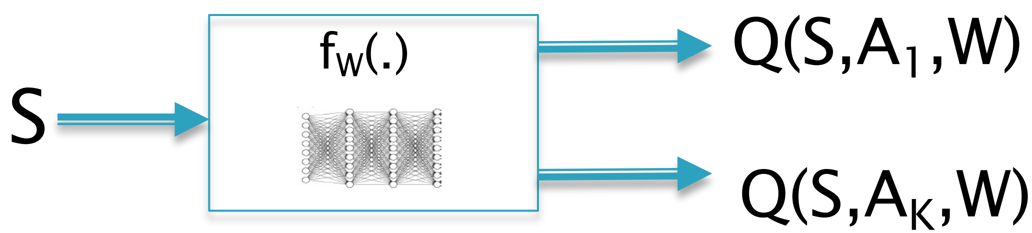
Function Approximations Using Deep Learning Architectures

Choices for the function f_W

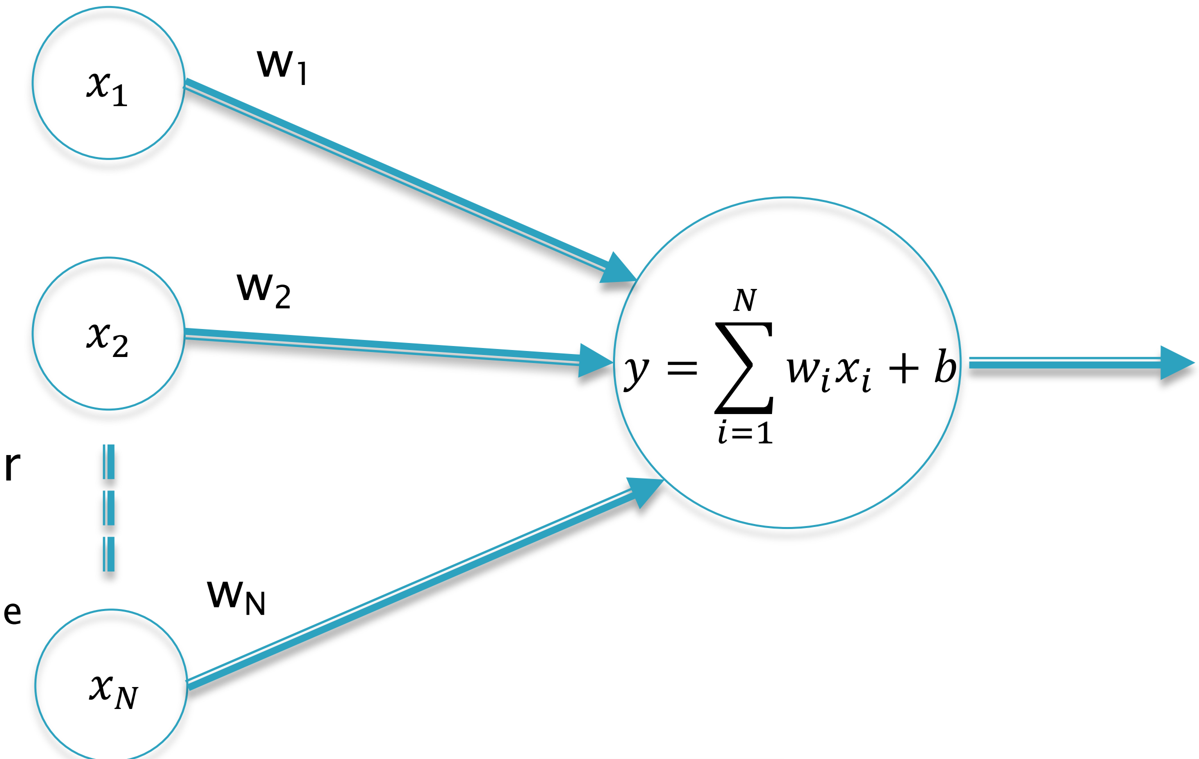
1. Linear Networks
2. Dense Feed Forward Networks
3. Convolutional Neural Networks
4. Recurrent Neural Networks
5. Transformers

We will focus on these

Used by the Atari Game Playing RL System

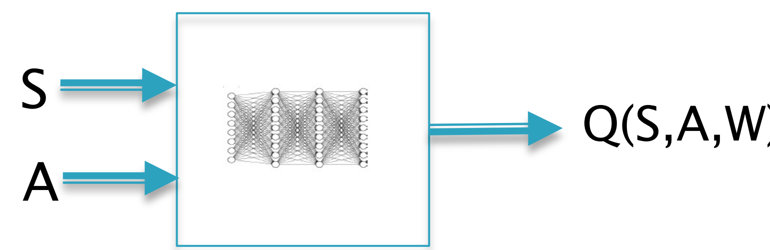


Linear Systems

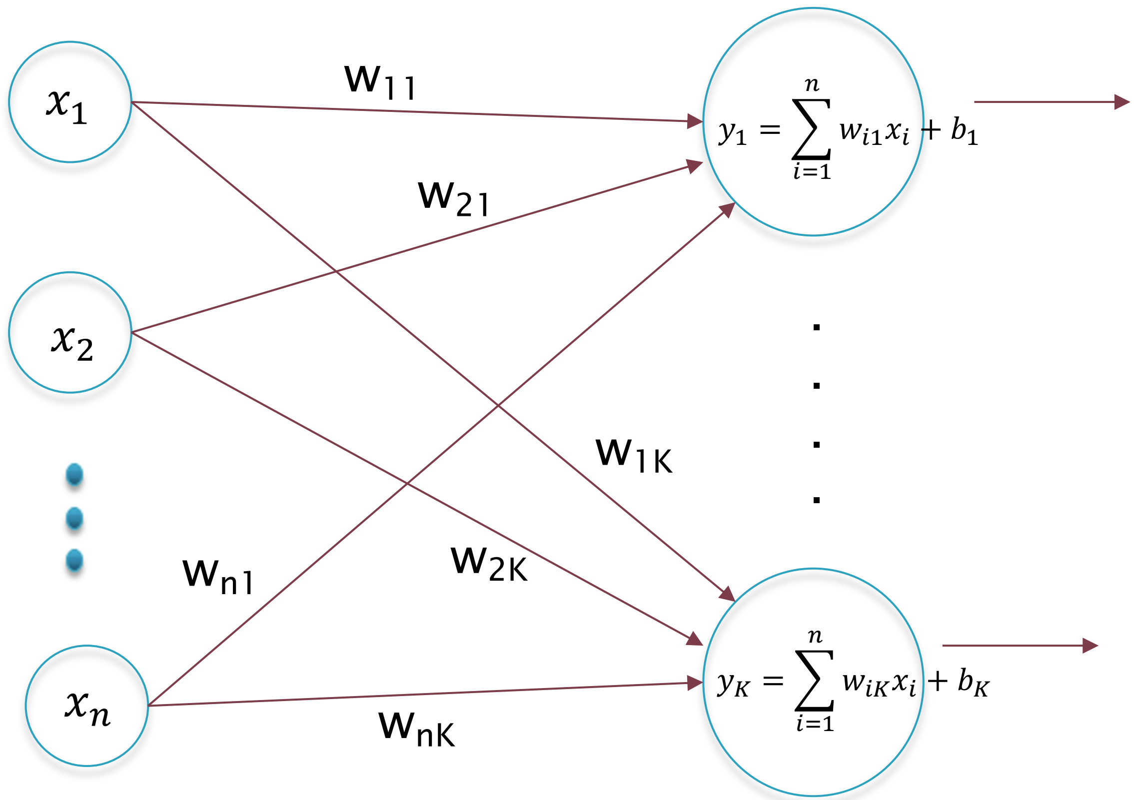


Notation: X is an input vector
$X = (x_1, x_2, \ldots, x_N)$

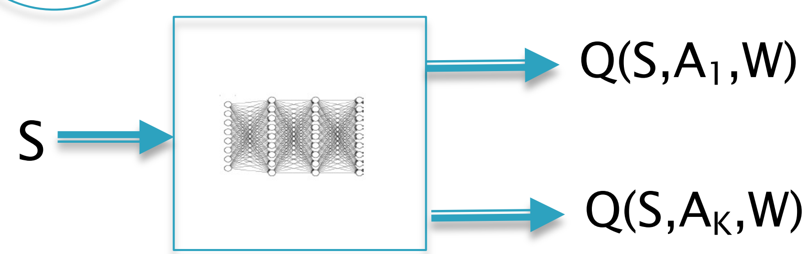
Components of the input vector



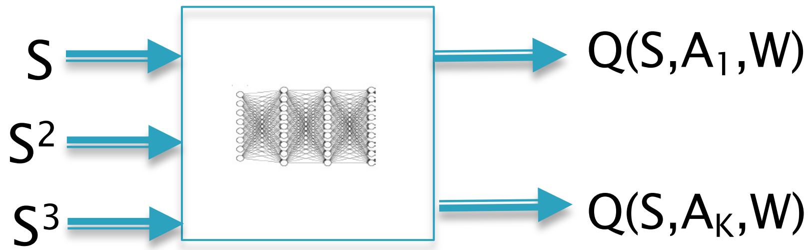
Linear System with K-ary Output



NK+K parameters



What about Non-Linear Functions?

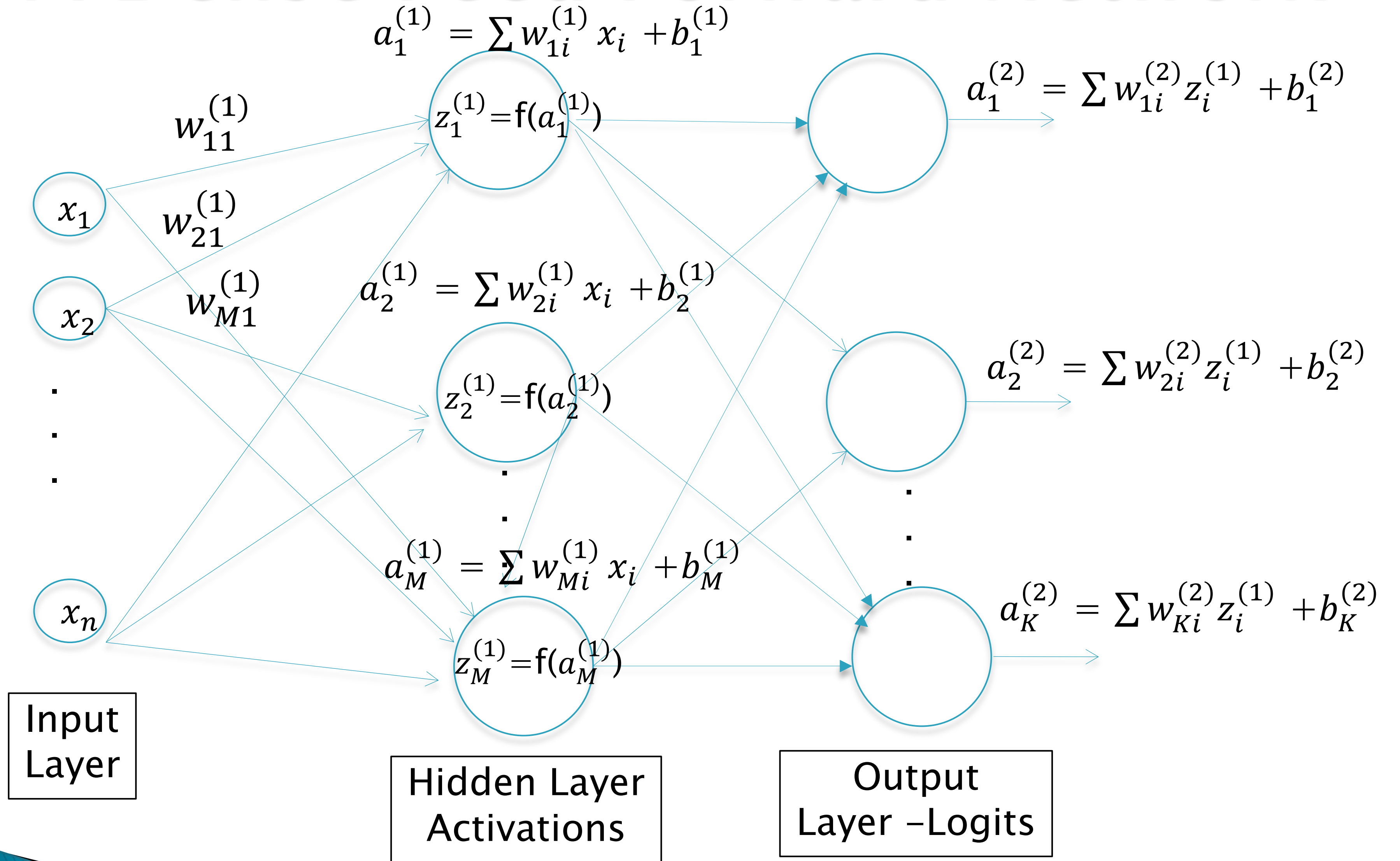


Two Solutions:

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

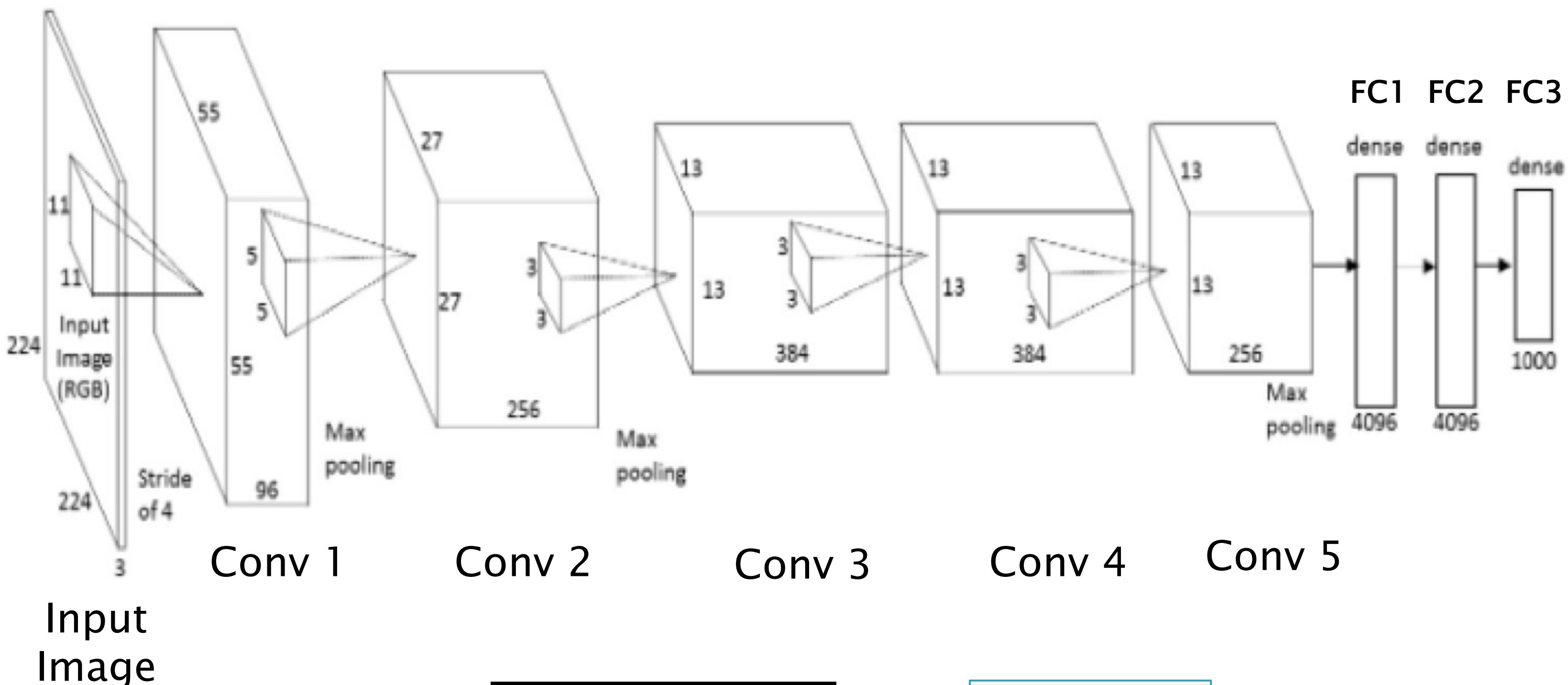
(2) Let the Training Process discover the non-linear function
 – Deep Learning

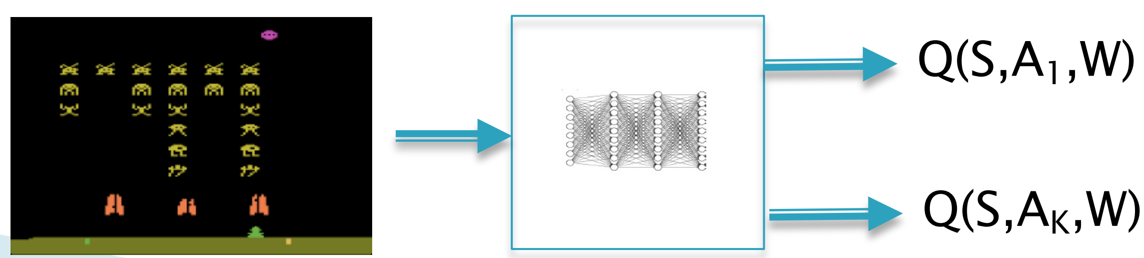
A Dense Feed Forward Network



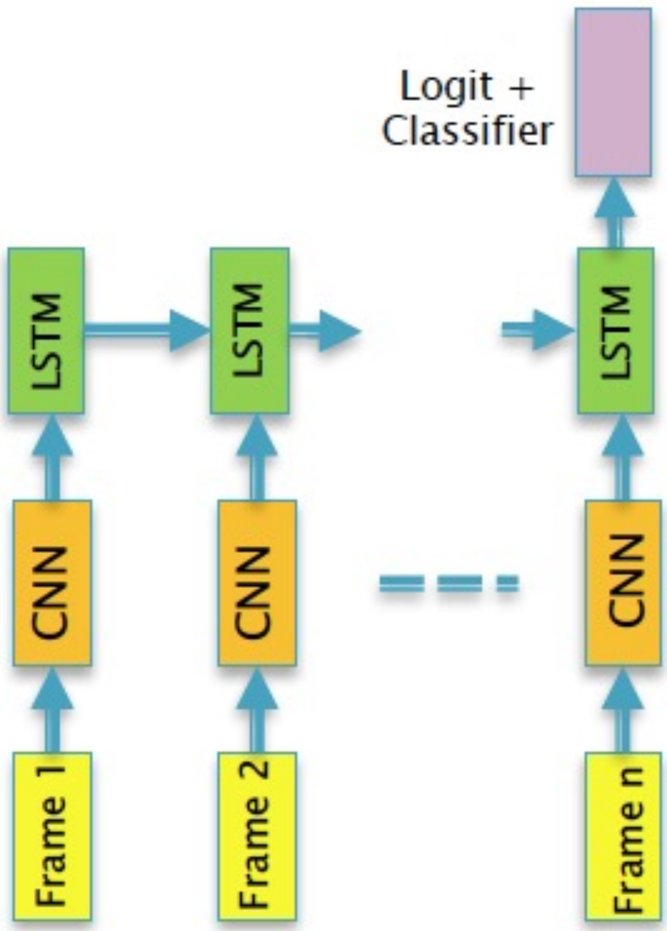
Discovers Non-Linear Functions during the Training Process

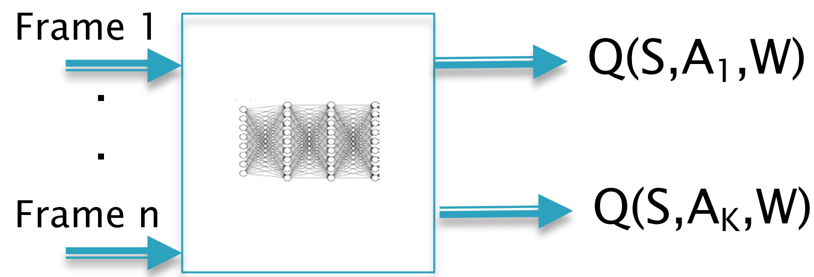
What if the Input State is an Image? Convolutional Neural Networks





What if the State is a Correlated Sequence (Video/Audio/Language)? Recurrent Neural Networks/LSTMs/Transformers





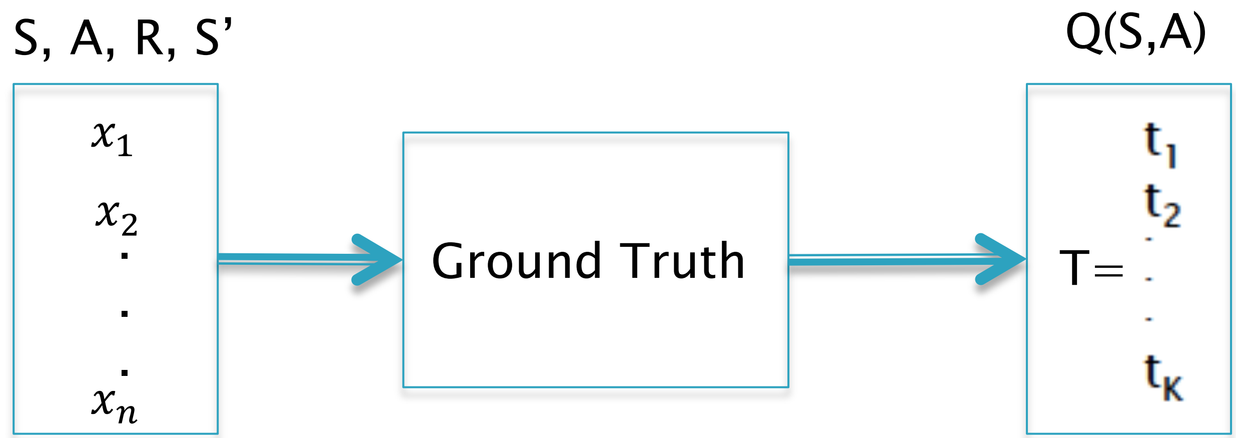
Training Neural Networks

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



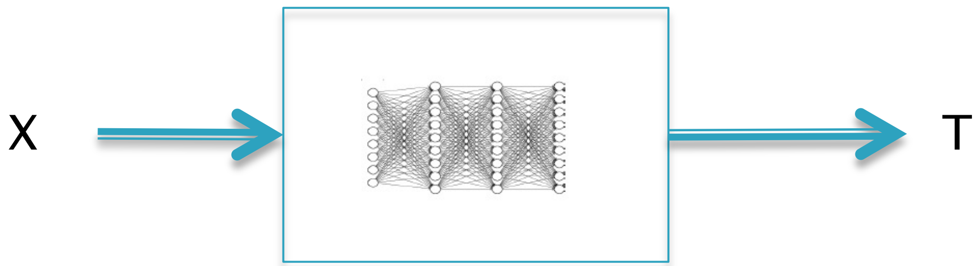
Input vector $X=(x_1,\ldots,x_N)$ is associated with Output 'desired' vector $T = (t_1, t_2, \ldots, t_K)$

Also called Label
Or 'Ground Truth'

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

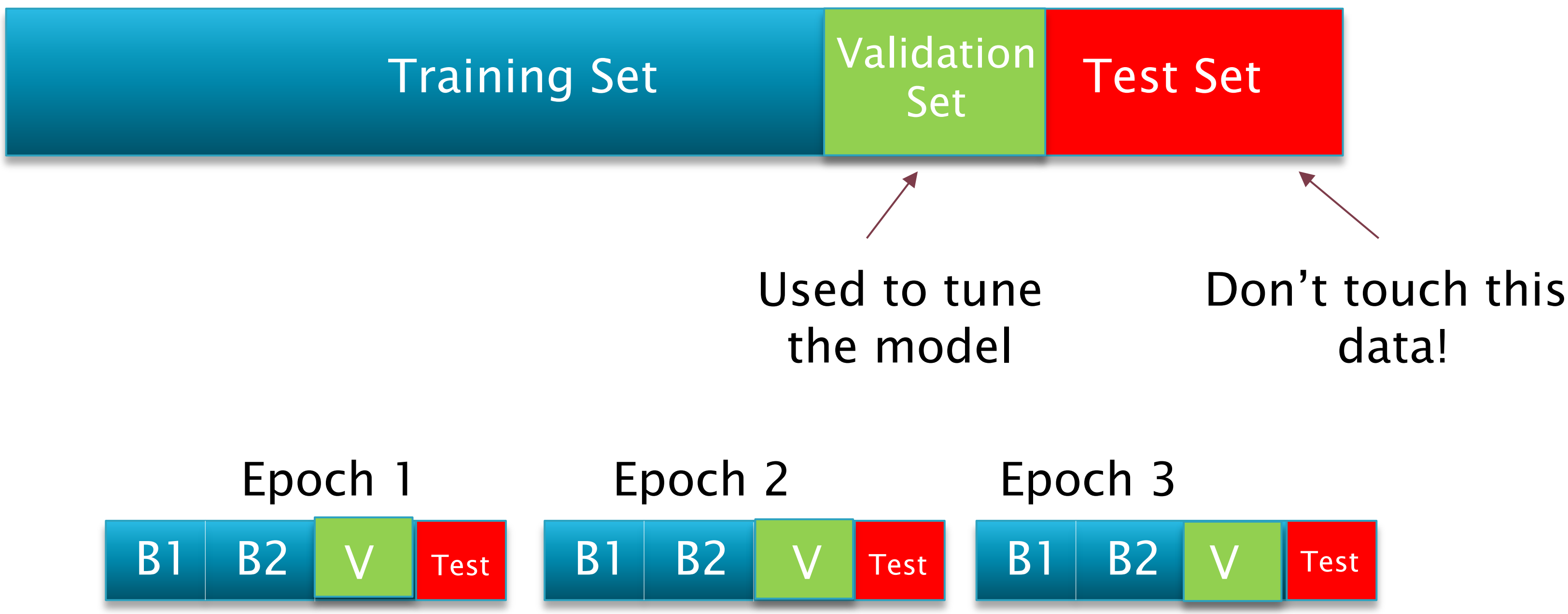
Training Dataset

The Supervised Learning Problem



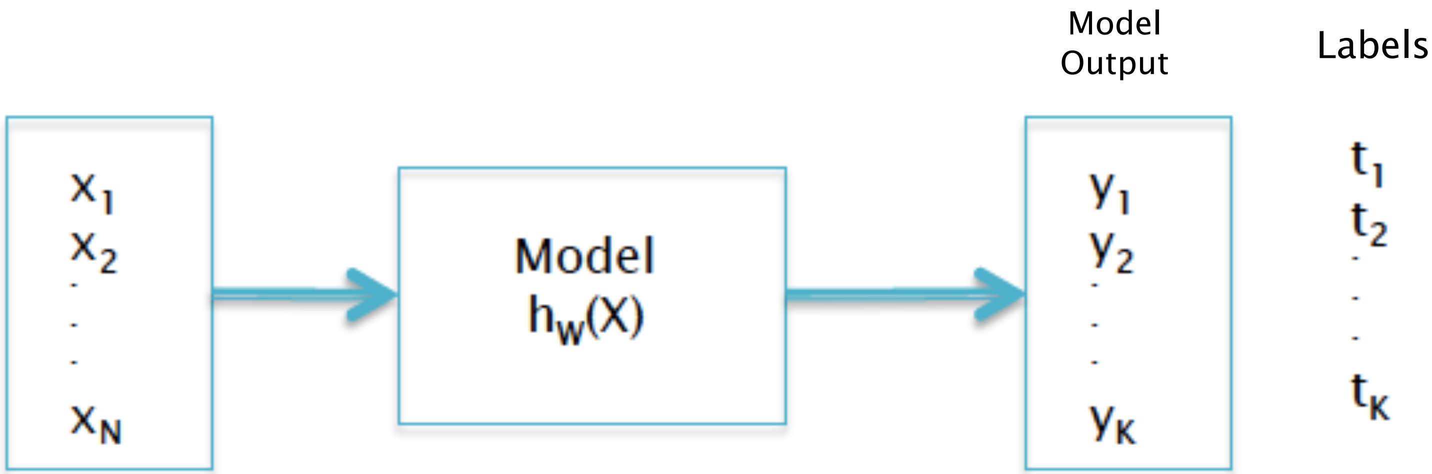
Problem: Find a model for the System, such that it is able to
Predict “suitably good” values of T, for new or un-seen values of X.

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

Label

Network Output

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

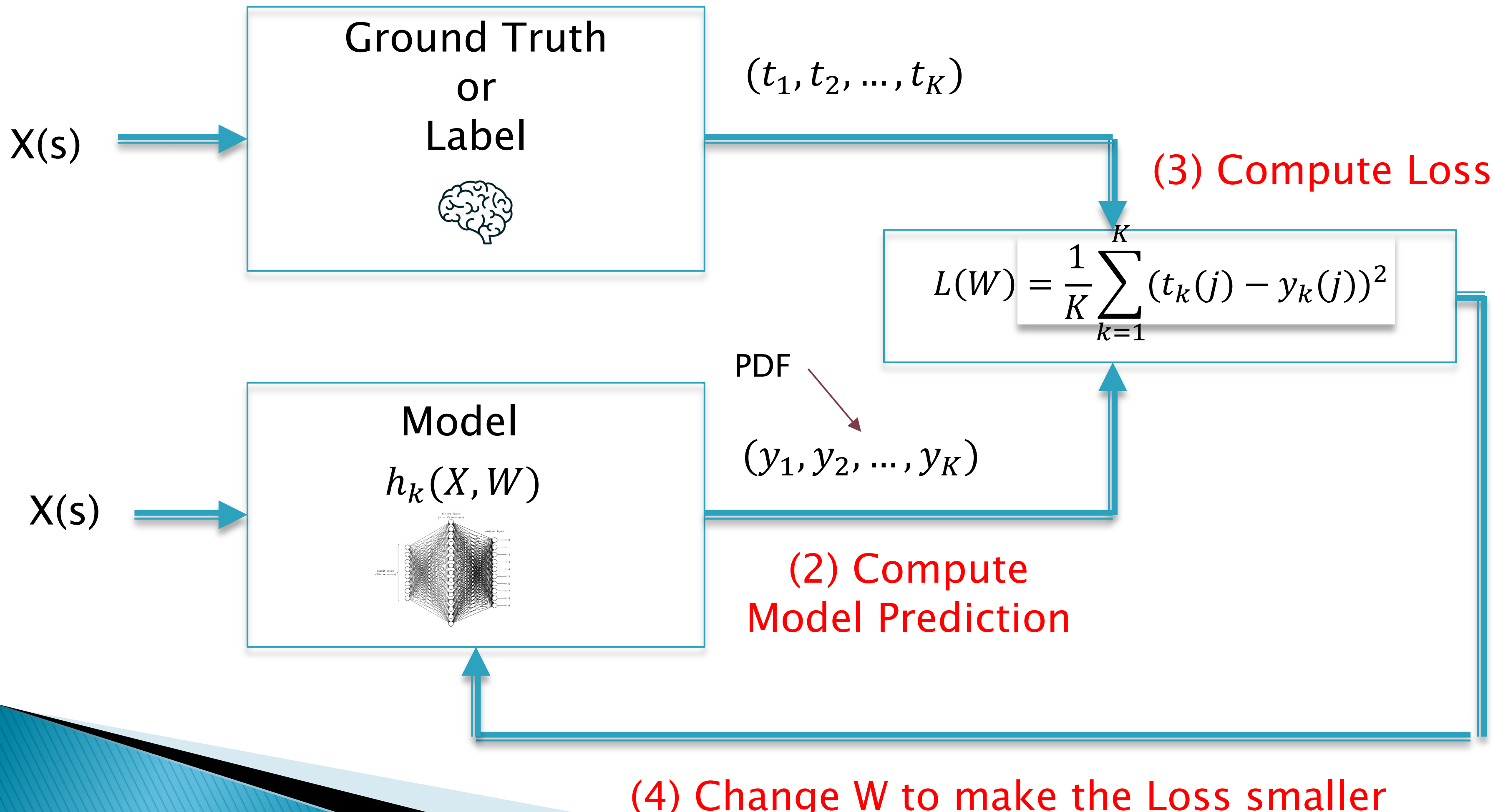
Mean Square Error

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

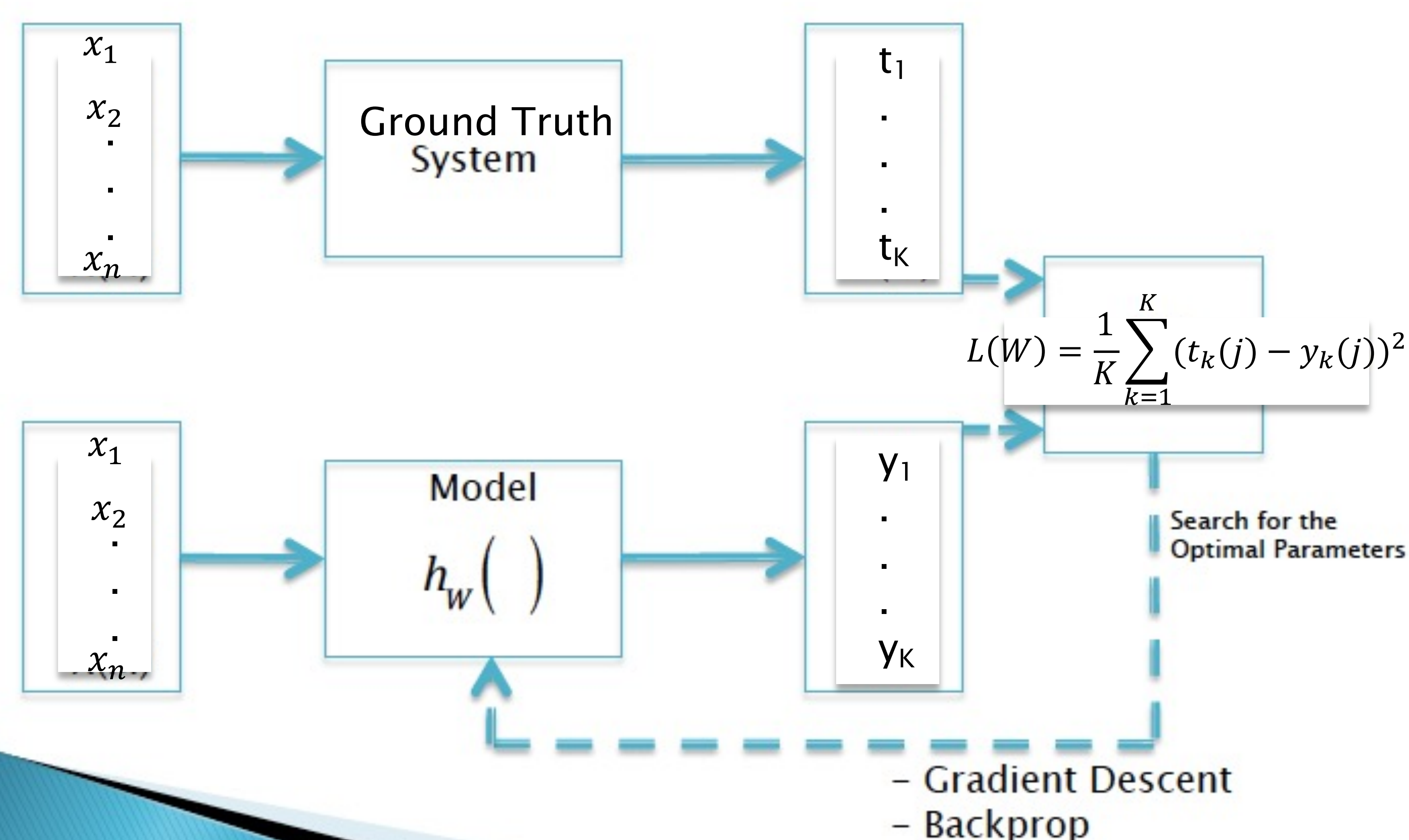
Solution to Classification Problem

(0) Collect Labeled Data (X(s),T(s))

(1) Choose Model h_k(X,W)



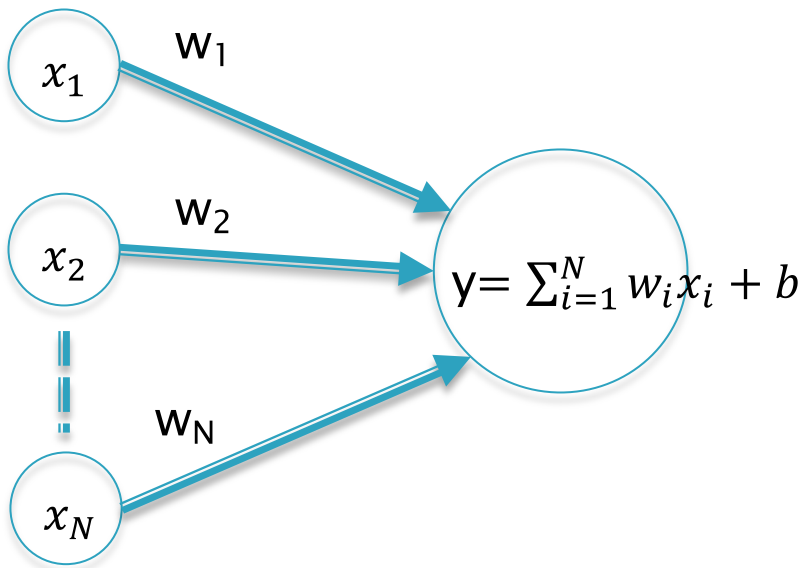
Solution to Regression Problem: Using Gradient Descent



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?

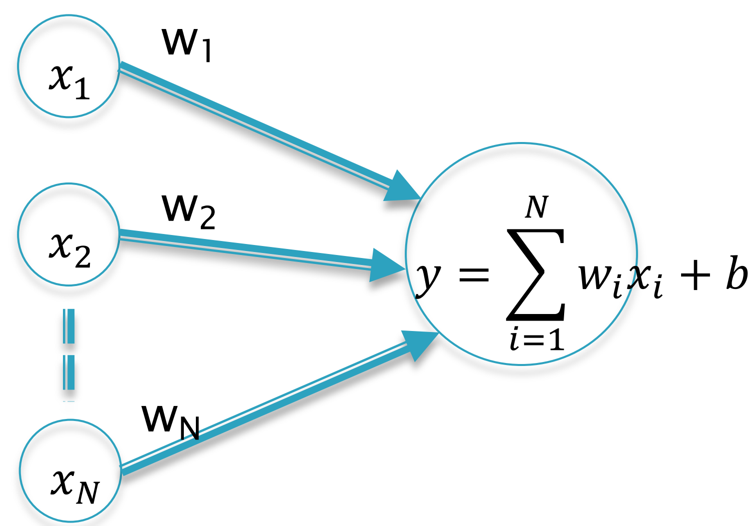
Given training samples
$(X(j),T(j))$, $j=1,\ldots,M$

Find Weights that minimize the Loss Function

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

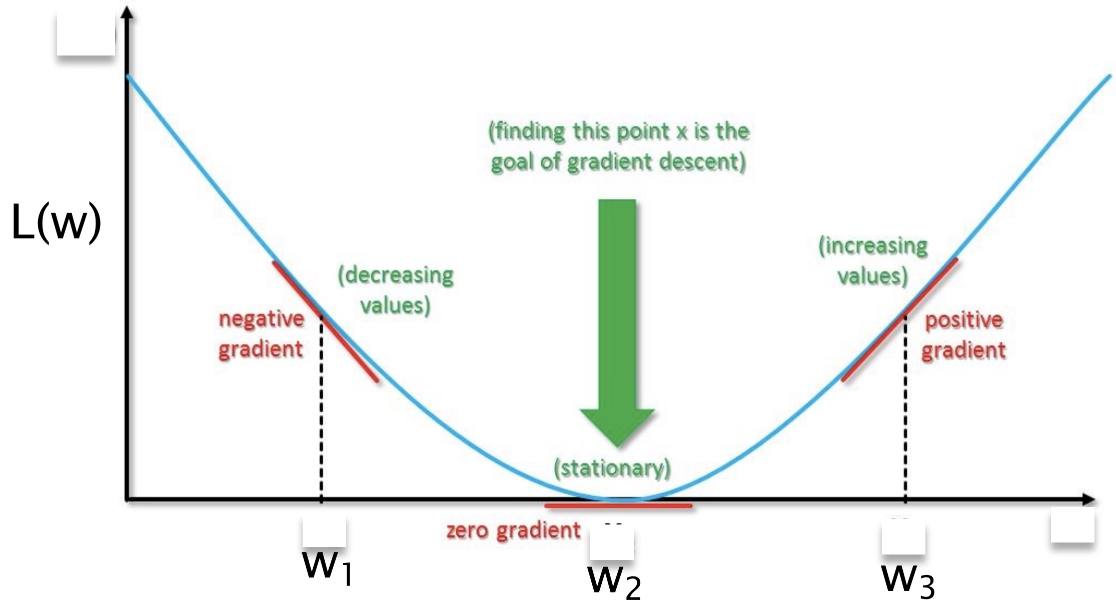
where

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



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 Stochastic Gradient Descent

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

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

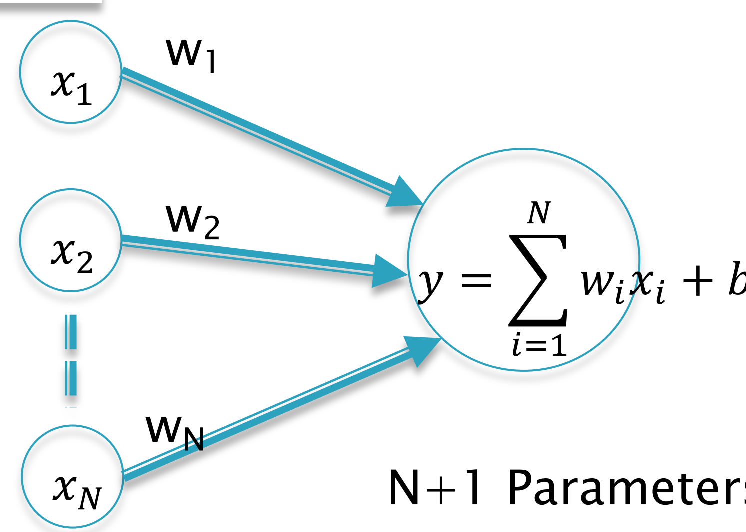
$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

where

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

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

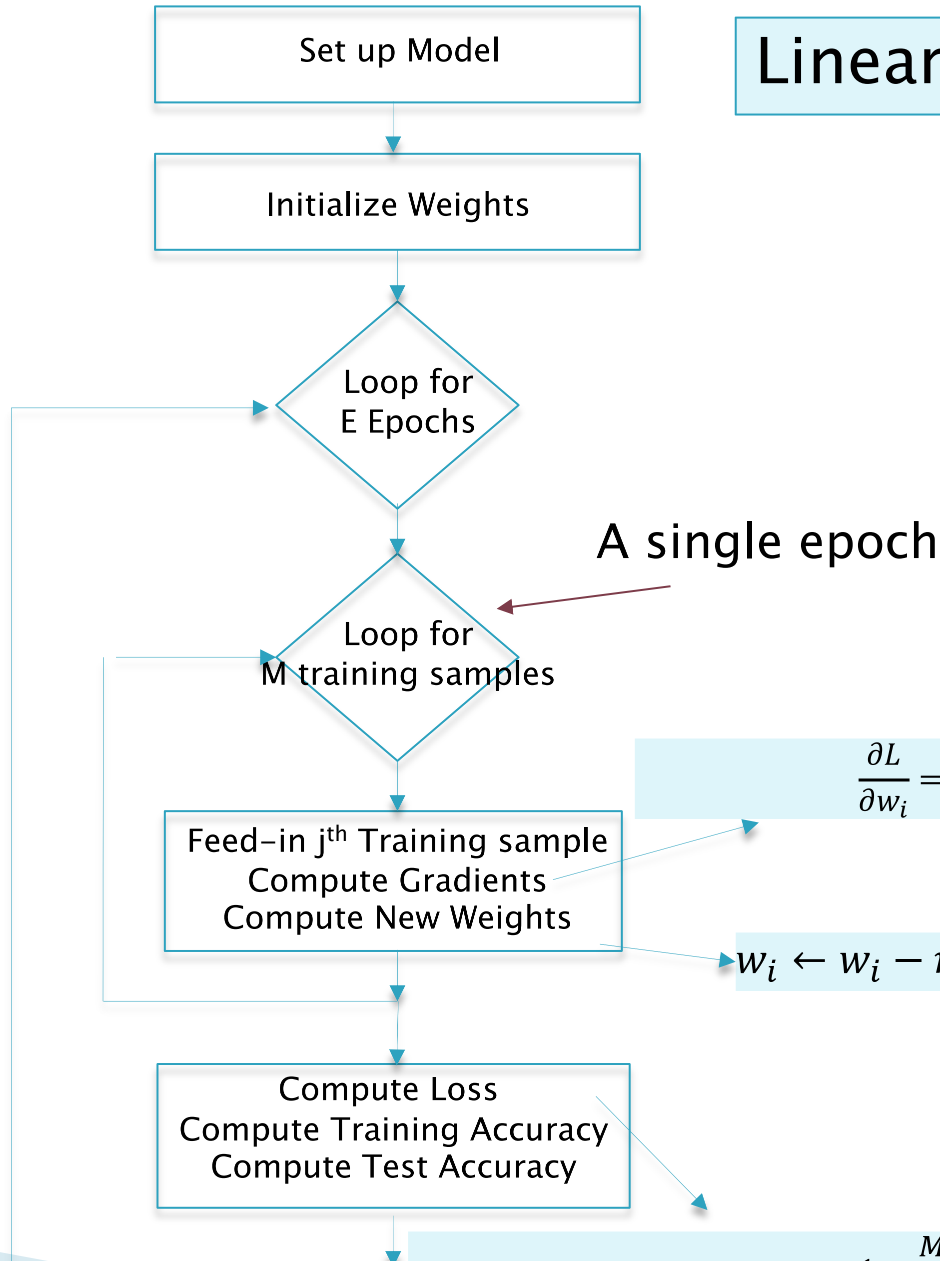
$$w_i \leftarrow w_i - \eta x_i [y - t]$$



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

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

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

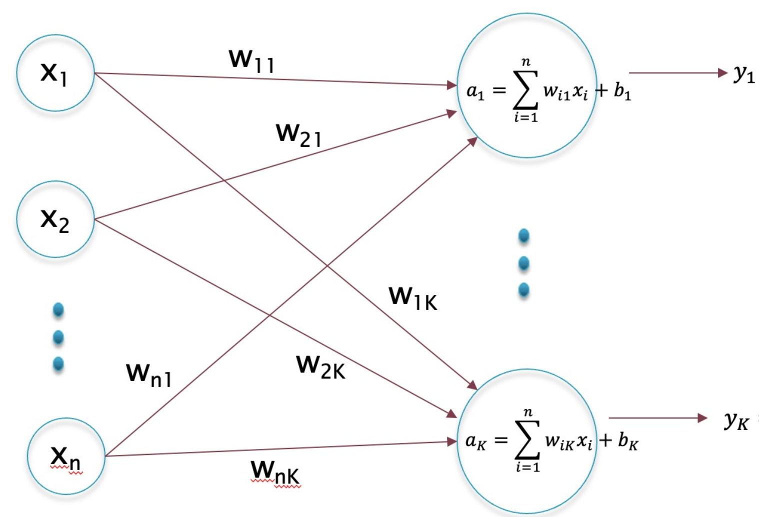
$$w_{ik} \leftarrow w_{ik} - \eta\frac{\partial L}{\partial w_{ik}}$$

where

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

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

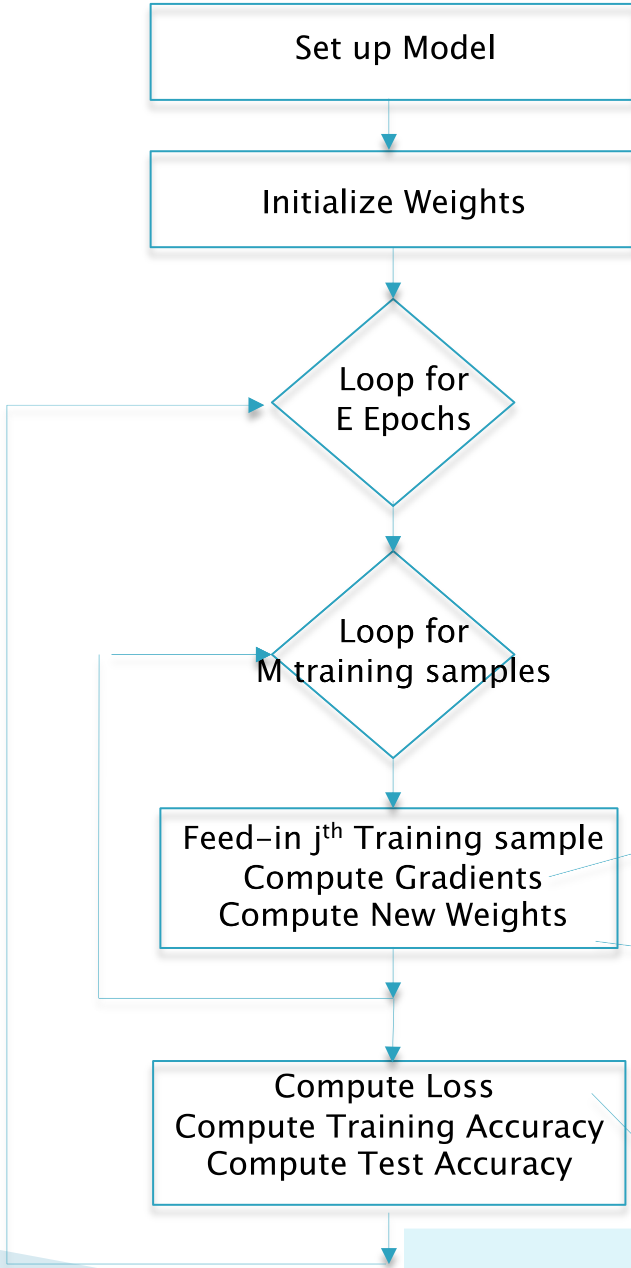
$$w_{ik} \leftarrow w_{ik} - \eta x_i[y_k - t_k]$$



NK+K Parameters

Weight Updates Using Stochastic Gradient Descent

Linear Regression



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

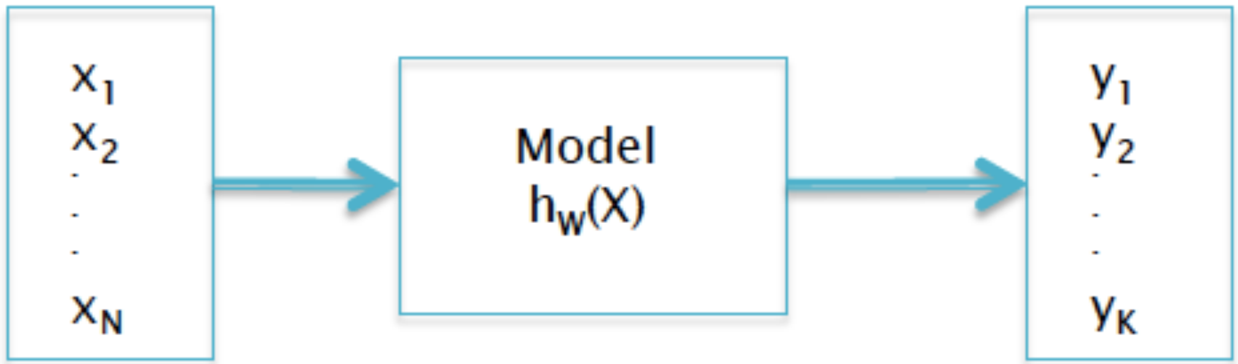
$$w_{ik} \leftarrow w_{ik}\eta x_i(j)[y_k(j) - t_k(j)]$$

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

Logistic Regression

Logistic Regression: Using the Neural Network for Estimating Probabilities

Classification Problem



Label

$$\begin{array}{ccc} t_1 & & 0 \\ t_2 & & 1 \\ \cdot & = & \cdot \\ \cdot & & \cdot \\ \cdot & & \\ t_K & & 0 \end{array}$$

$$\sum_{k=1}^{K} y_k = 1$$

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

Probabilistic Classification

$$\text{Label} = T \in \{1,2,\dots,K\}$$

X

$\mathrm{h}(X, W)$

$\mathrm{P}(\mathrm{T}=1|\mathrm{X}) = y_1 = h_1(X, W)$

$\mathrm{P}(\mathrm{T}=\mathrm{K}|\mathrm{X}) = y_K = h_K(X, W)$

Output is a Discrete Probability Density Function

$$y_k = h_k(X, W) = \mathrm{P}(\mathrm{Y} = \mathrm{k}|\mathrm{X})$$

$$\sum_{k=1}^{K} y_k = 1$$

In the context of Reinforcement Learning

$$y_k = \mathrm{P}(A_k = 1|X)$$

Gives the Distribution of Actions given an input State

Performance Measure for Probability Estimation: Reward Function

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

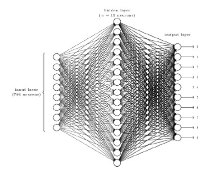
Cross Entropy

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

Example: K = 2 (Binary Cross Entropy)

X

$h(X, W)$

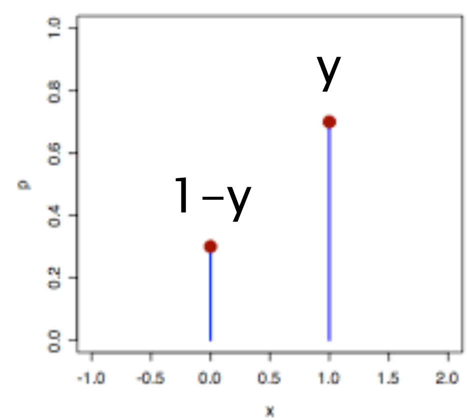


Single output

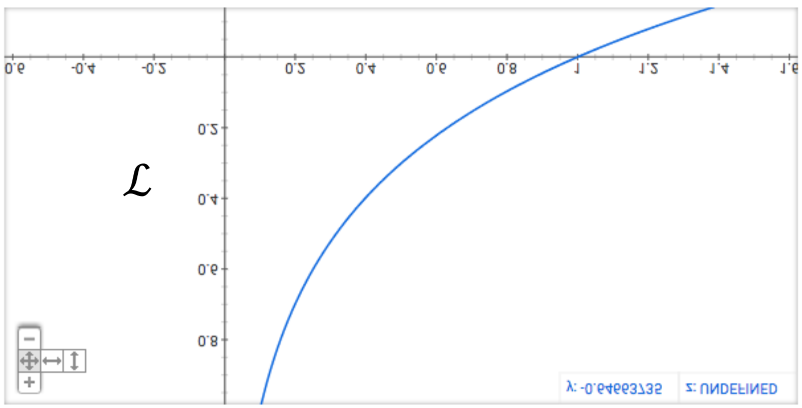
P(T=1|X) = y

Single Label: t
(1,0) and (0,1)

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



The Cross Entropy for (K =2)



$\mathcal{L}$

y_q

$t_q = 1$

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

Exact Match

$y_q = 1$

$\mathcal{L} = 0$

Complete Mismatch

$y_q = 0$

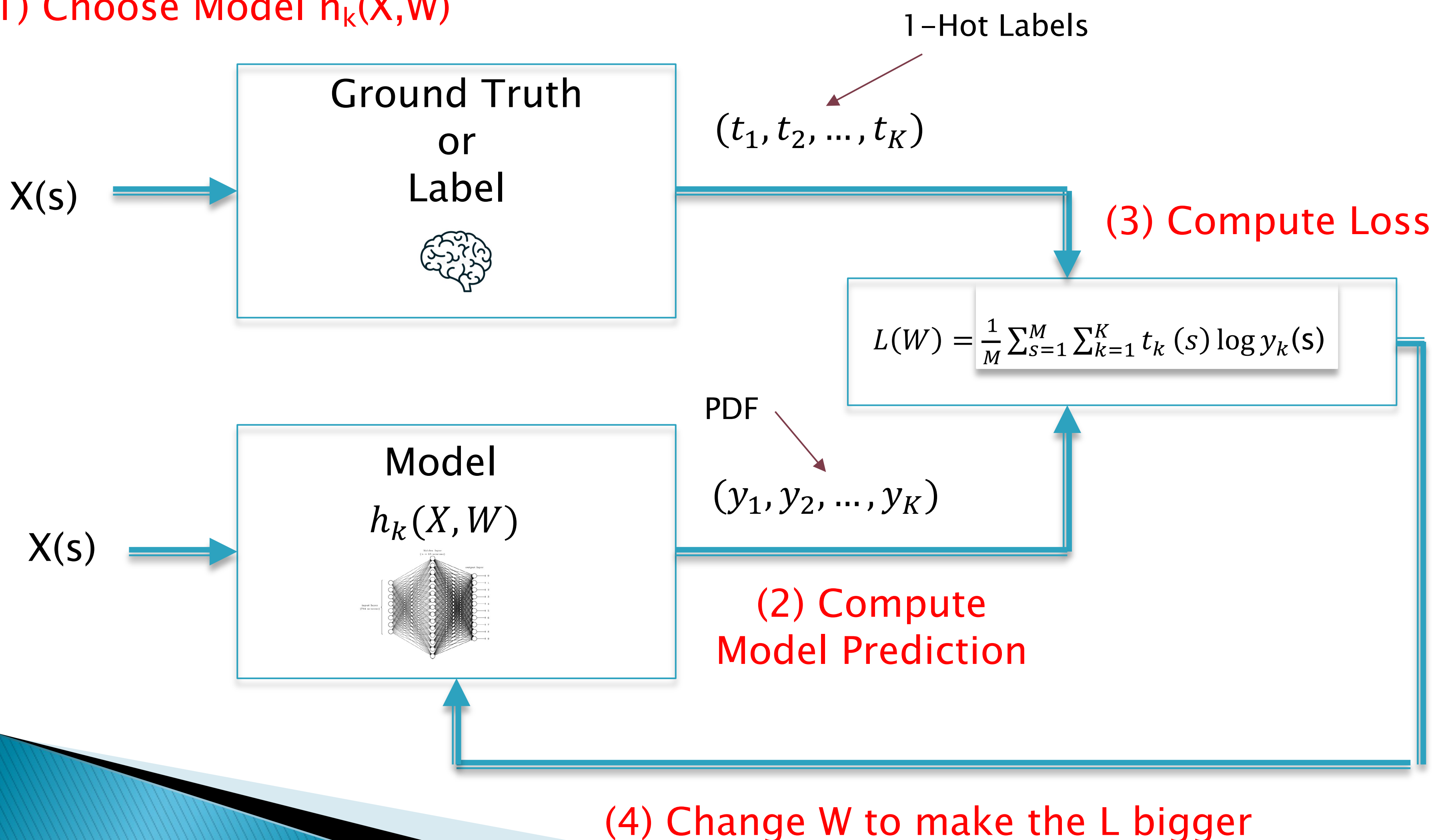
$\mathcal{L} = -\infty$

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

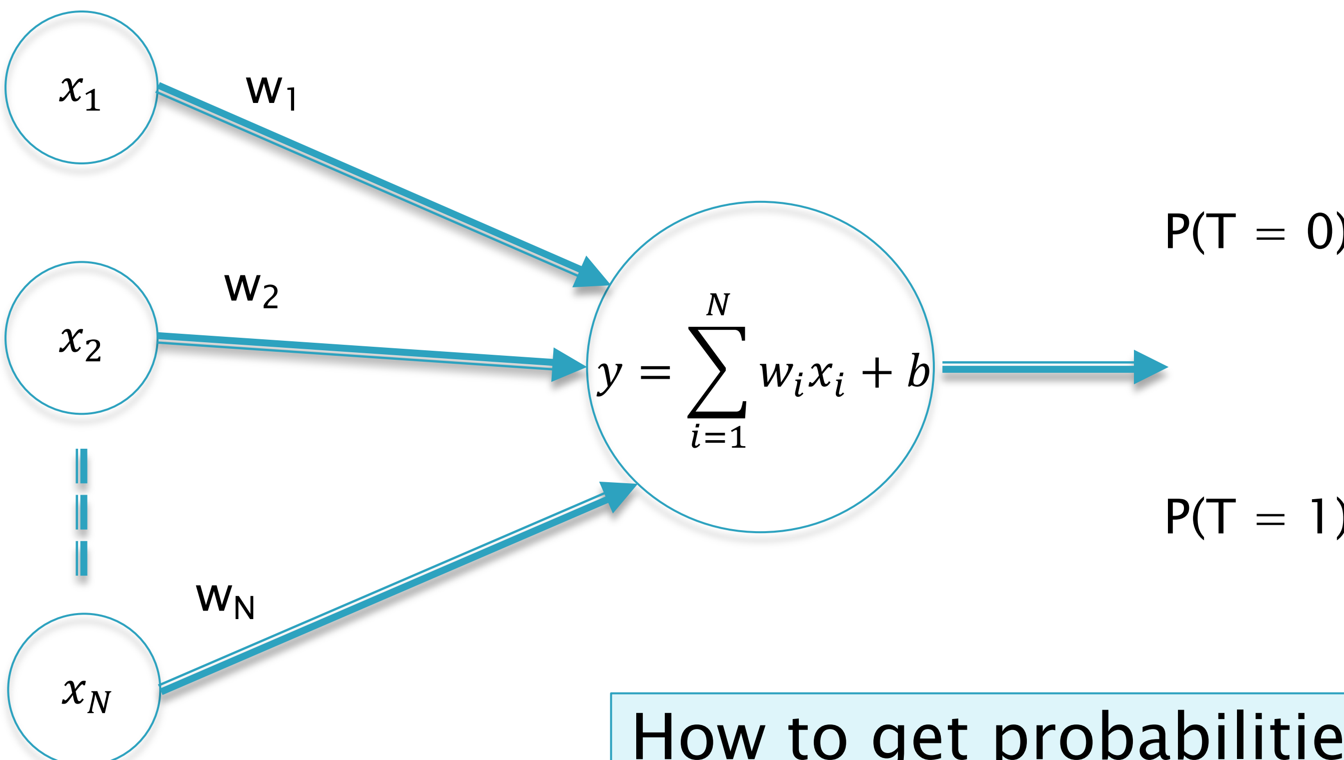
Solution to Classification Problem

(0) Collect Labeled Data (X(s),T(s))

(1) Choose Model $h_k(X,W)$



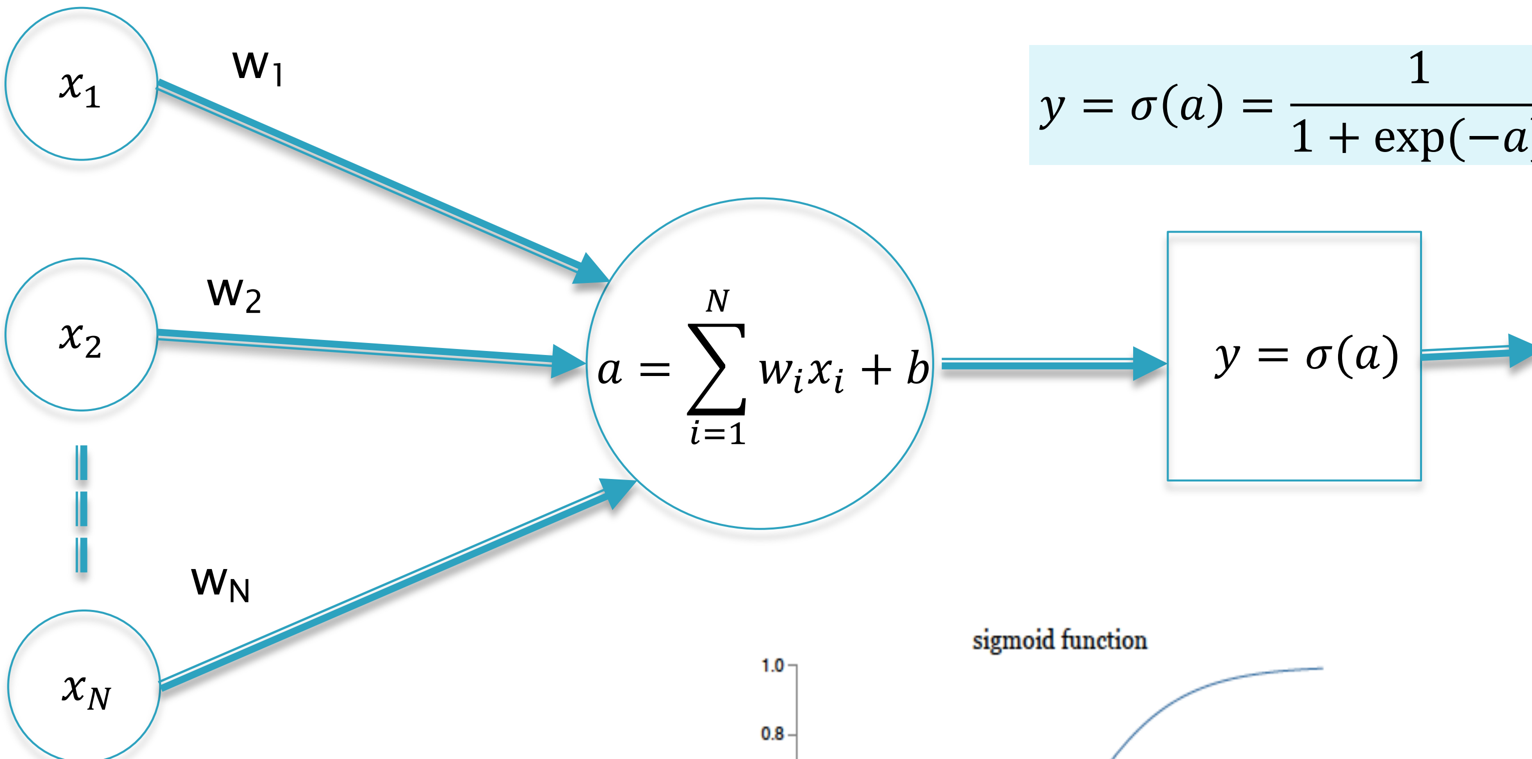
Linear Models for Probability Estimation: Logistic Regression

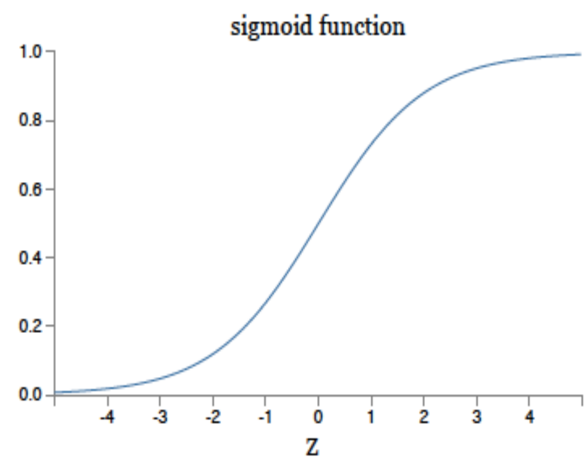


How to get probabilities?

Convert Scores to Probabilities via the Sigmoid Function

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





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

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

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

where

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

If

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

How did we get this?

then

$$w_i \leftarrow w_i + \eta x_i (t - y)$$

Gradient Computation using Chain Rule of Differentiation

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

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

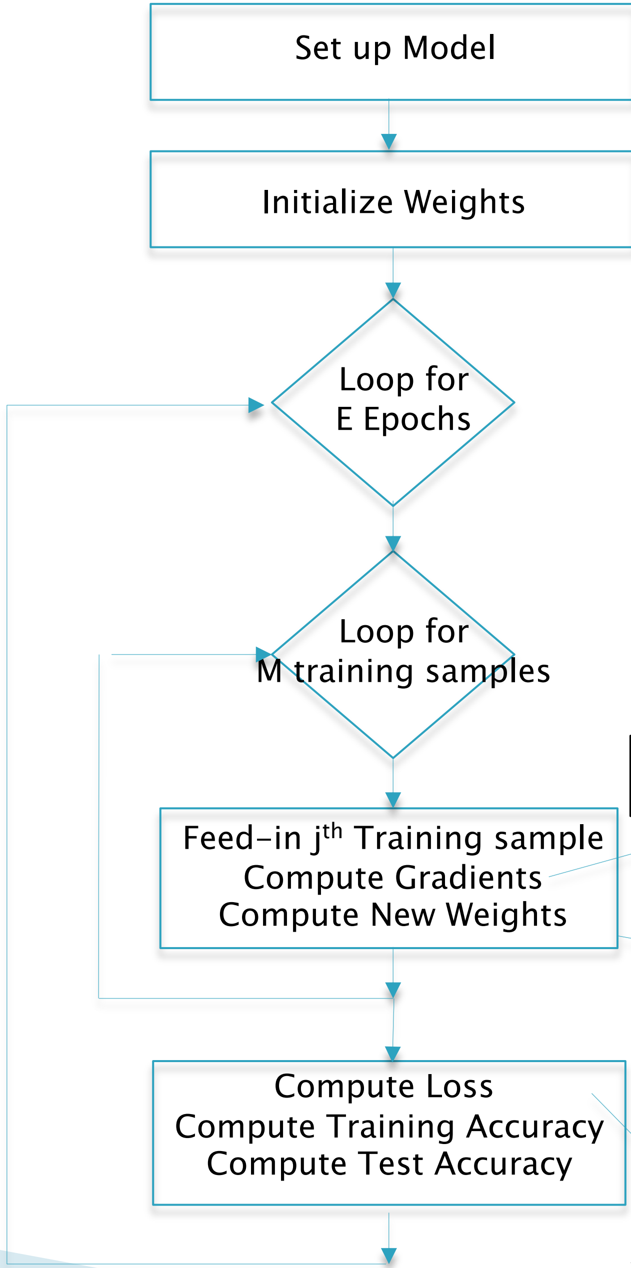
Use Chain Rule: $$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial y}\,\frac{\partial y}{\partial a}\,\frac{\partial a}{\partial w_i}$$

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

$$\boxed{\frac{\partial L}{\partial w_i} = (t-y)x_i}$$

Training Algorithm: Exactly the same as for Regression!

Training Using Stochastic Gradient Descent Logistic Regression

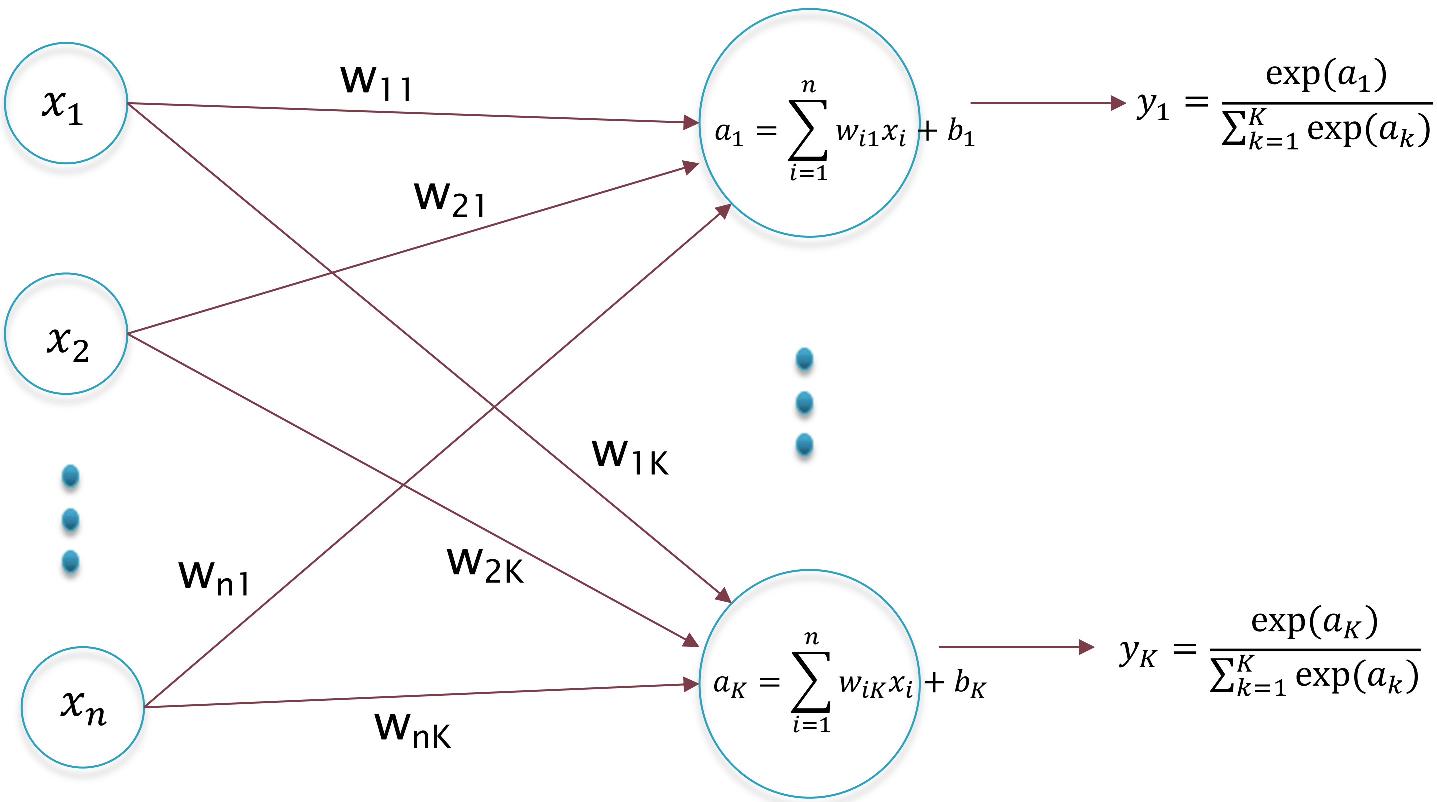


$$\frac{\partial L}{\partial w_i} = x_i(j)[t(j) - y(j)]$$

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

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

Logistic Regression with K Outputs: The Softmax Function



Training Equation

$$\boxed{w_{ik} \leftarrow w_{ik} + \eta x_i (t_k - y_k)}$$

What Does Training Do?

Assume that q^th output y_q in a training sample corresponds to the Ground Truth, i.e., the Label is given by

$T = (0,0,\ldots,1,\ldots0)$

q^{th} position

Then the Training Equation becomes

$$w_{iq} \leftarrow w_{iq} + \eta \frac{\partial L}{\partial w_{iq}} = w_{iq} + \eta x_i (1 - y_q) \quad \textit{for } k = q \textit{ and for all } i$$

$$w_{ik} \leftarrow w_{ik} - \eta x_i y_k, \quad for\ k \neq q \text{ and for all i}$$

This equation shows that if the q^{th} action is the correct one for input **X**, then its synapse weight is increased, while the synapse weights of the other actions are reduced

Training Equation

$$w_{ik} \leftarrow w_{ik} + \eta x_i (t_k - y_k)$$

Issues in Running Gradient Descent Algorithms

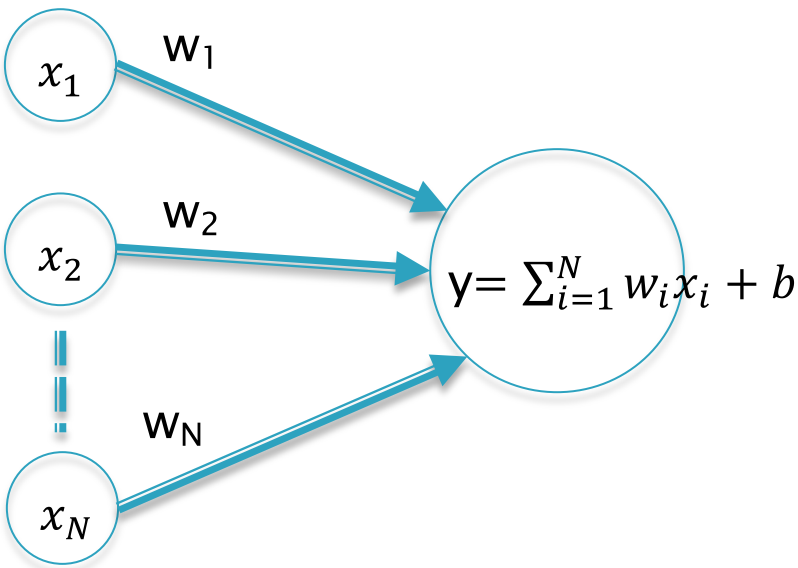
- Weight Initialization
- Choosing the Learning Rate parameter η
- Deciding when to stop the training
- Improving Generalization Error

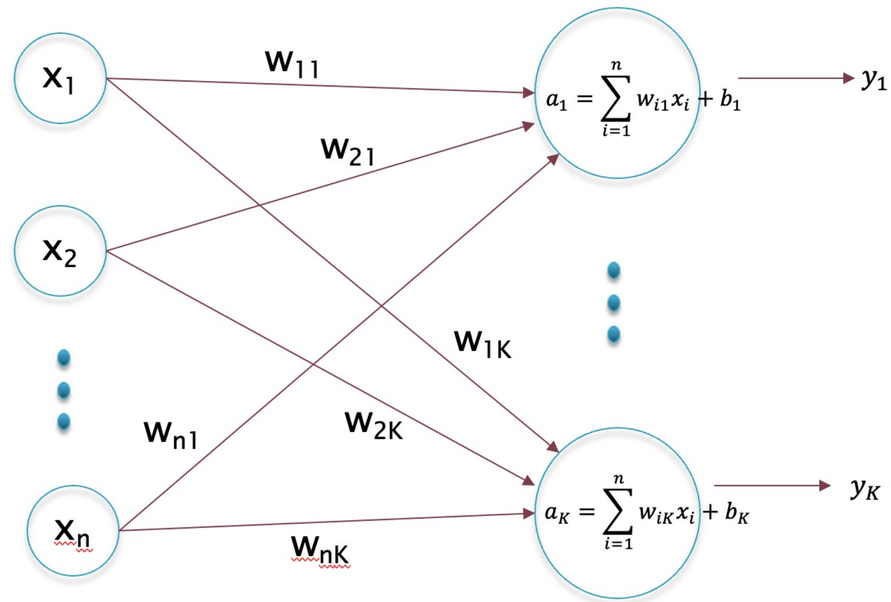
These are called Hyper-Parameters

Summary – Regression

Choose weights to Minimize Error

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





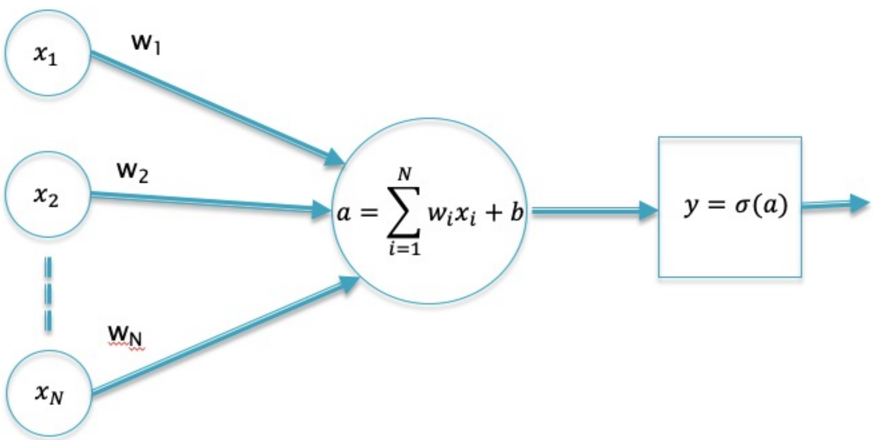
$$w_i \leftarrow w_i - \eta x_i (y - t)$$

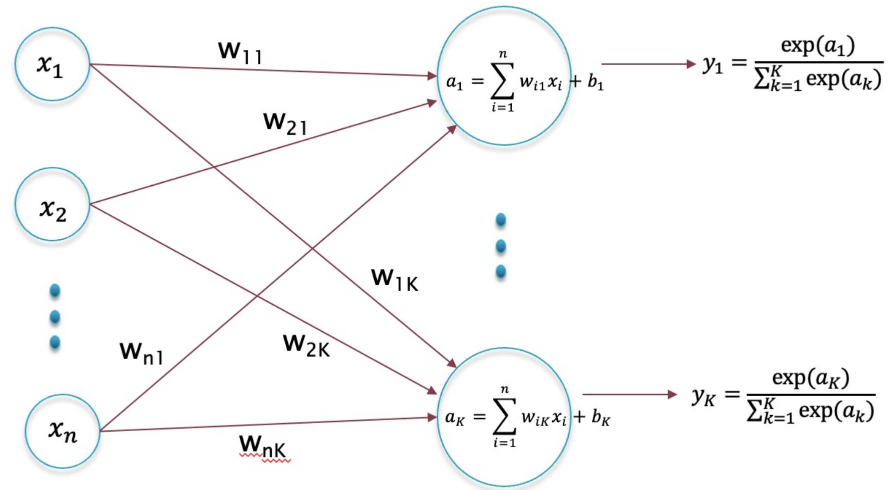
$$w_{ik} \leftarrow w_{ik} - \eta x_i (y_k - t_k)$$

Summary – Logistic Regression

Choose weights to
Maximize Reward

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





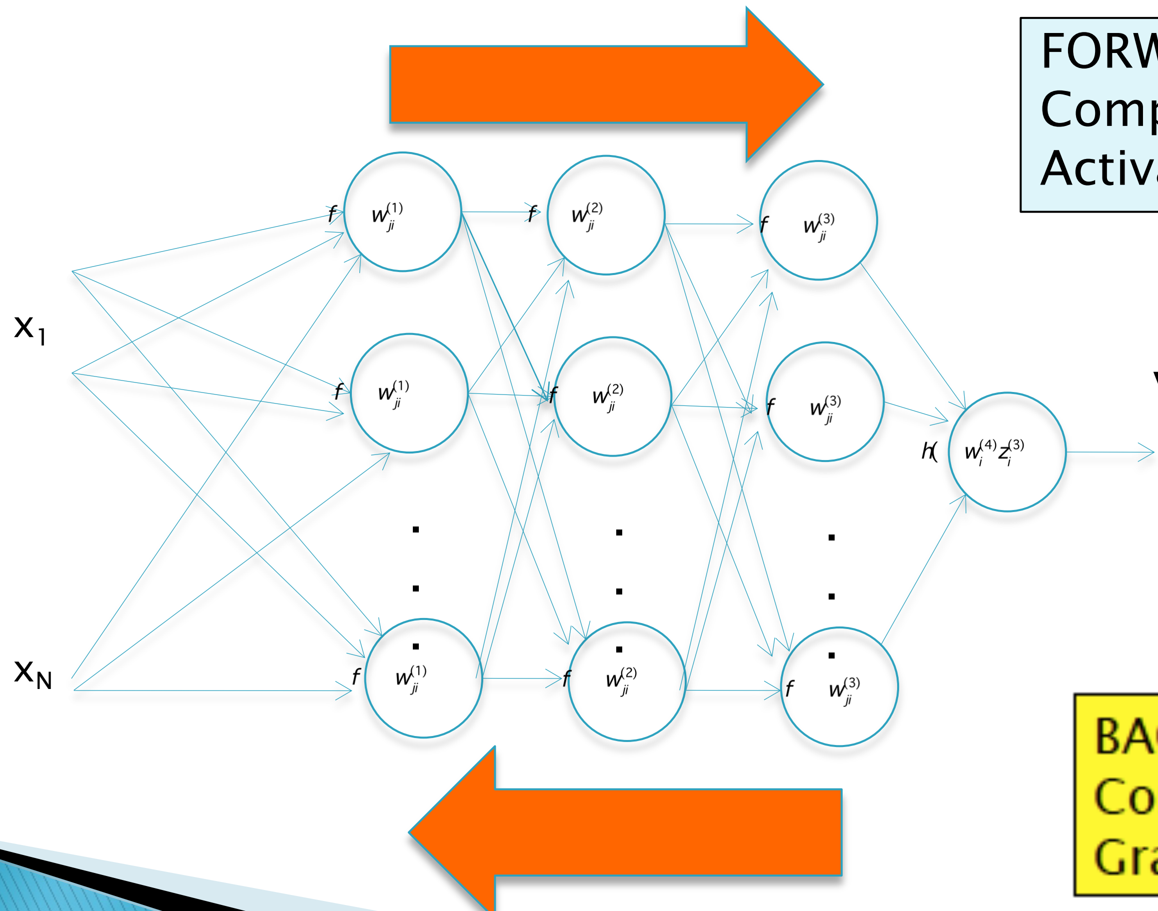
$$w_i \leftarrow w_i - \eta x_i(y - t)$$

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

A General Training Algorithm

Using Backprop

- Backprop requires only TWO passes to compute ALL the derivatives, irrespective of the size of the network!

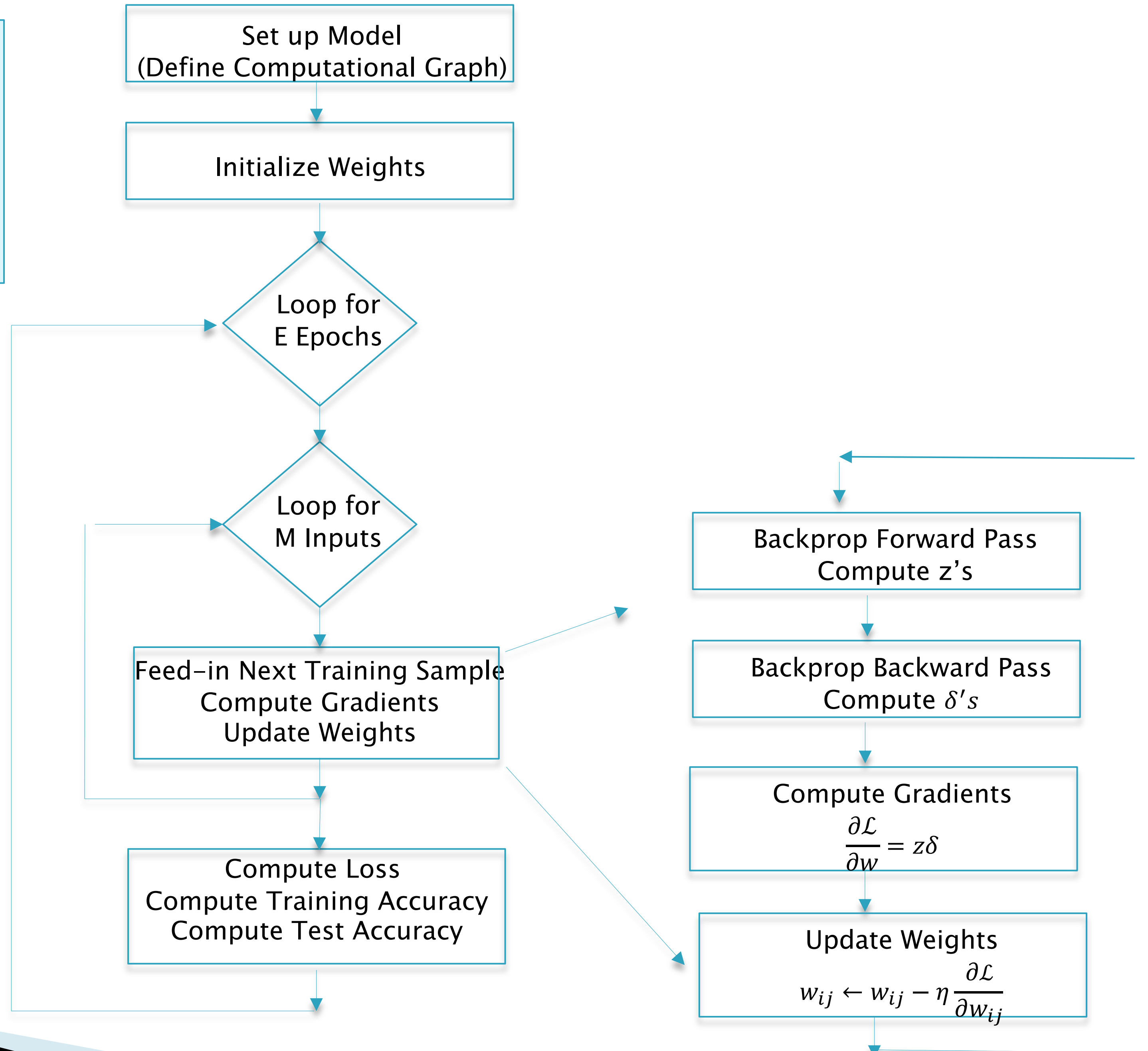


FORWARD PASS
Compute the Node Activations z

$$\frac{\partial \mathcal{L}}{\partial w} = z\delta$$

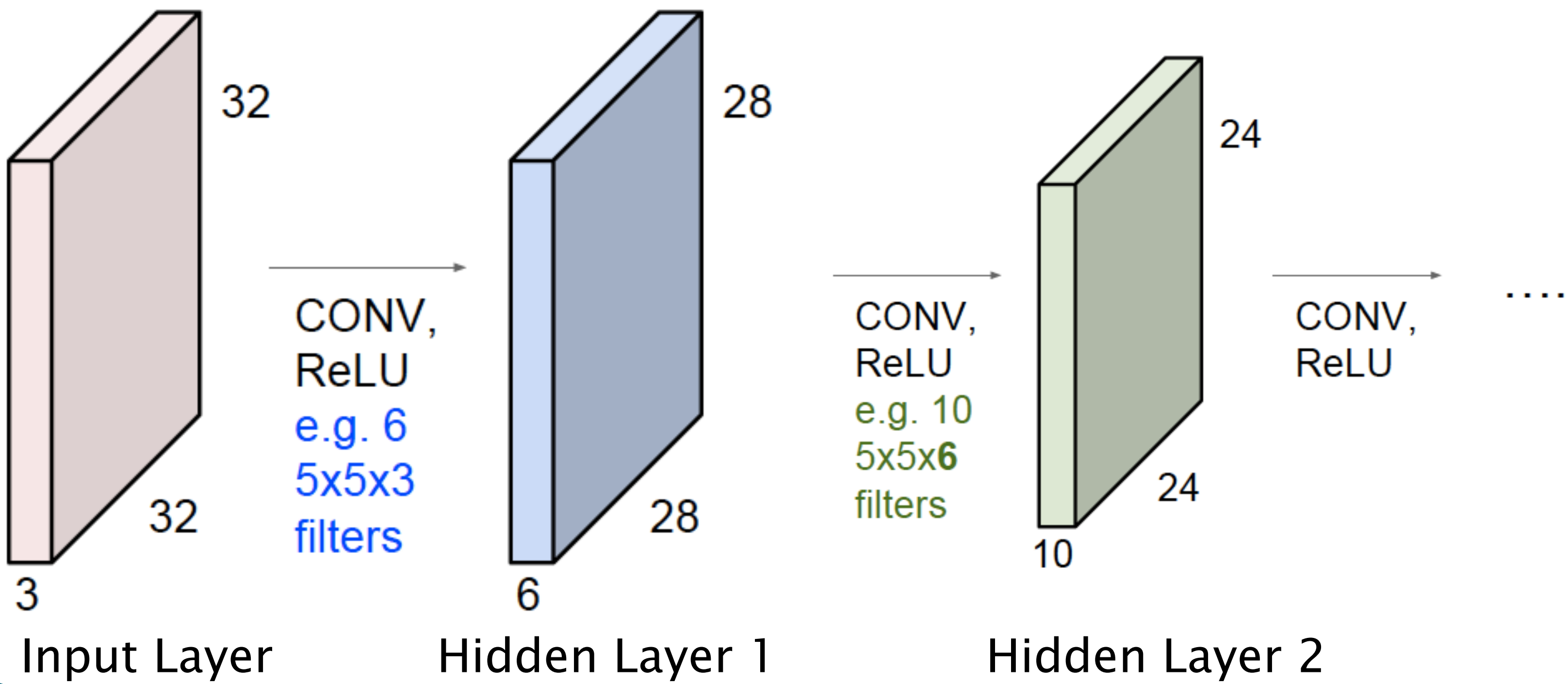
BACKWARD PASS
Compute the Node Gradients δ

A General Training Algorithm (Backprop)

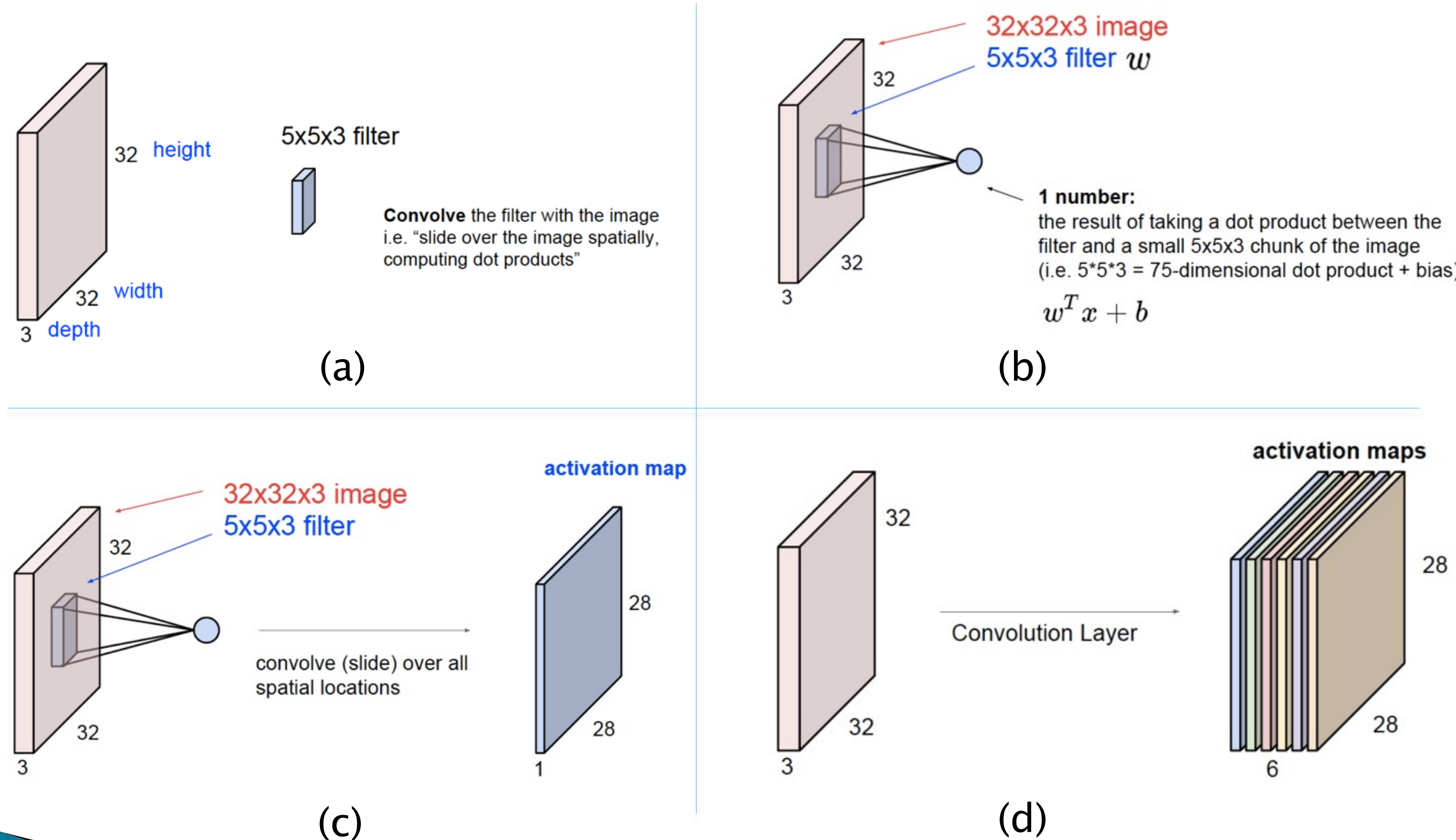


Convolutional Neural Networks

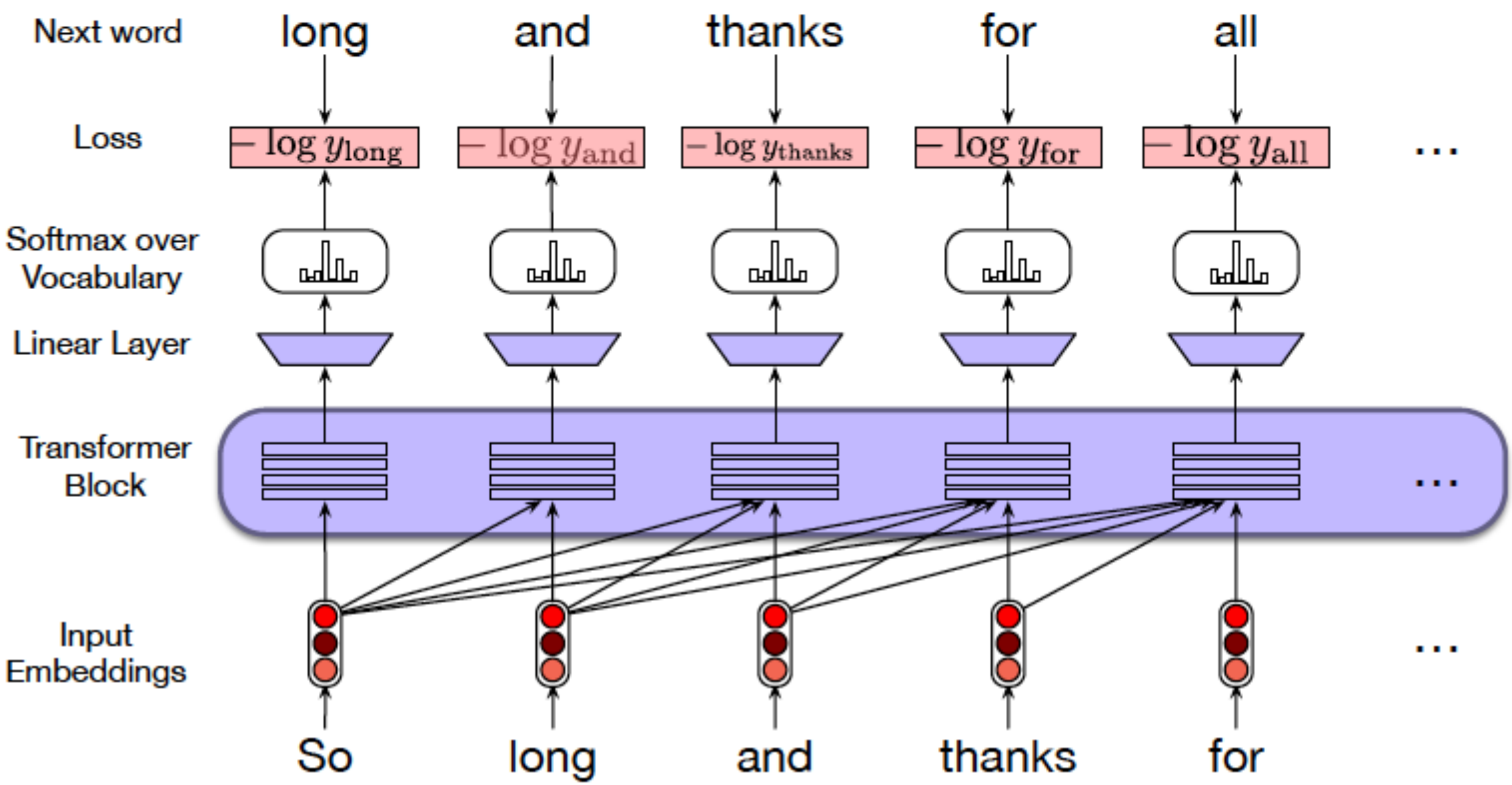
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



CNNs Summary



Transformers: LLMs



Further Reading

"Introduction to Deep Learning" by Varma and Das: https://subirvarma.github.io/GeneralCognitics/Books.html

Chapter 1: Introduction
Chapter 2: Pattern Recognition
Chapter 3: Supervised Learning
Chapter 4: Linear Neural Networks