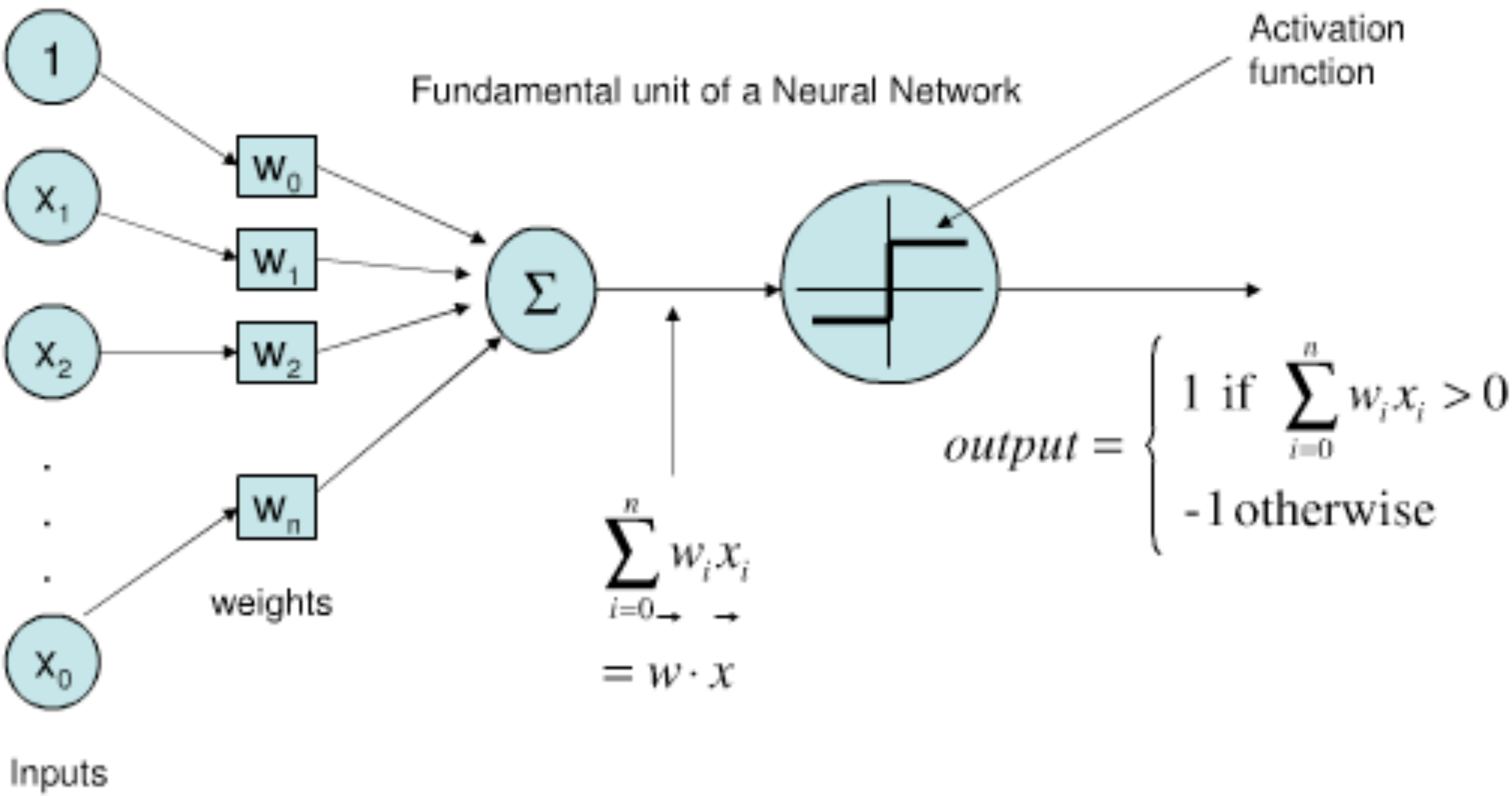
NEURAL NETWORKS

March 12, 2015 – General Assembly, Santa Monica

Mohsen Chitsaz, Ph.D.
Sr. Principal Data Scientist at Symantec

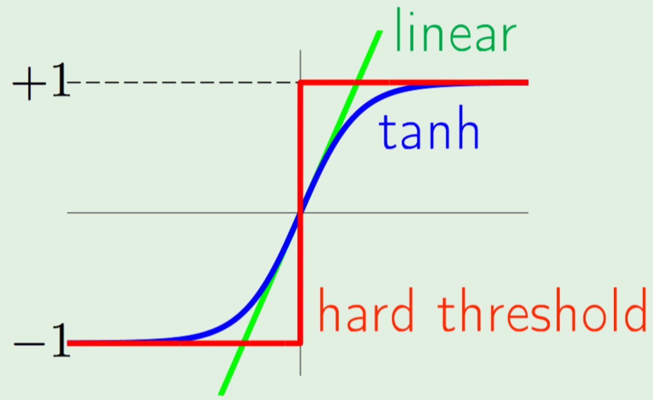
Lecture 10

Neuron Model



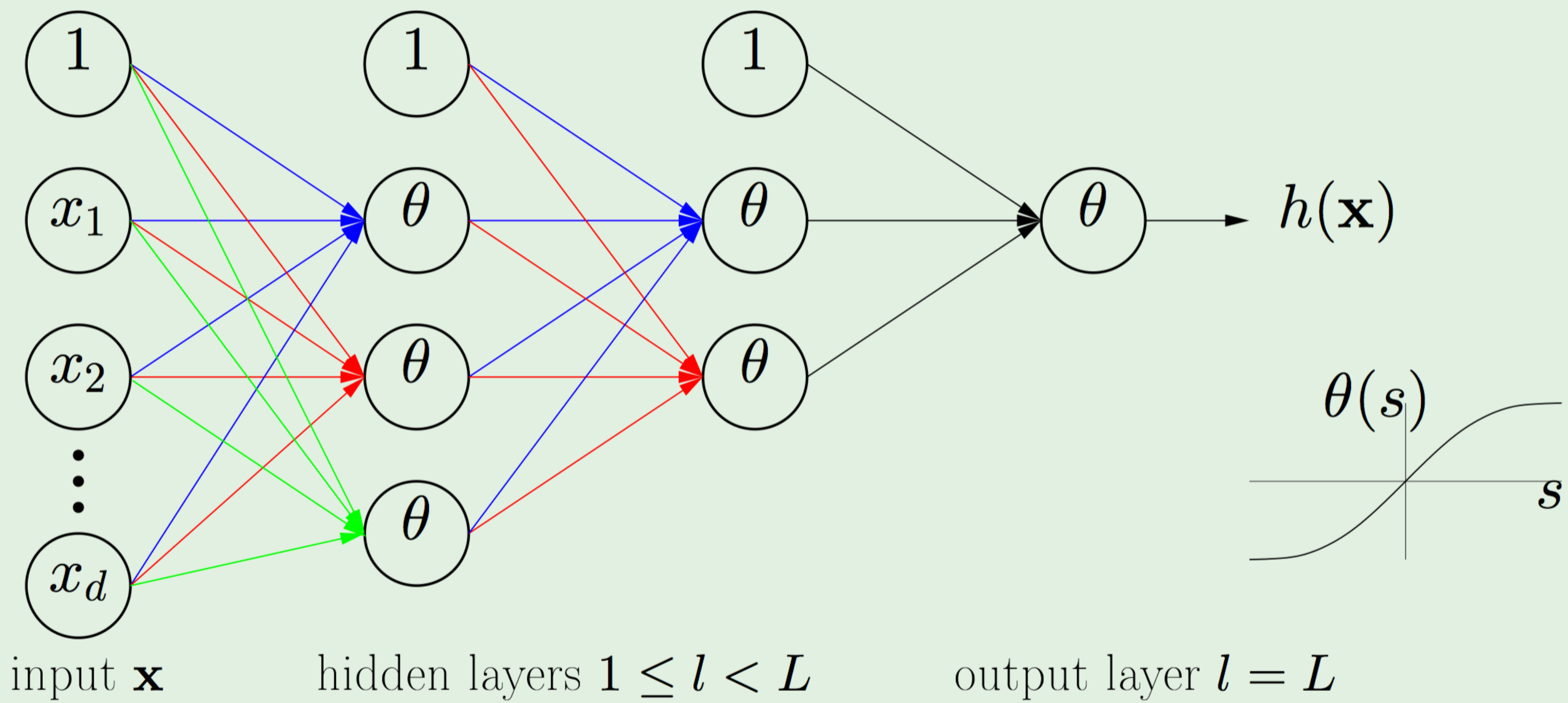
Transfer Functions

- Step Function
- Linear
- Sigmoid



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network Model



Classification Accuracy

		Model Prediction	
		True	**False**
Reality	**Positive**	TP	FP
	Negative	TN	FN

TP: Positive cases that were correctly predicted (Recall, Sensitivity)

FP: Positive cases that were not correctly predicted

TN: Negative cases that were predicted correctly (Specificity)

FN: Negative cases that were predicted incorrectly

Set things up

1. Importing all of the required libraries

```
import pybrain as pb
import matplotlib.pyplot as plt
from pybrain.supervised.trainers import BackpropTrainer
import pybrain.datasets.supervised as ds
from pybrain.structure.modules import LinearLayer, SigmoidLayer, TanhLayer
from pybrain.tools.shortcuts import buildNetwork
import pandas as pd
import numpy as np
```

```
%matplotlib inline
```

Importing data and cleaning

2. Importing the data and cleaning it up

```
data = pd.read_csv('breast_cancer.data',sep='\t')
data = data.replace(to_replace=' ',value=np.NaN)
data = data.dropna()
data = data.ix[1:,1:]
data = data.astype(float)
```

Defining input and target

3. Defining input and target data

```
input_data = data.ix[:,:-1]
target_data = np.int32(np.divide(np.vstack(data.ix[:,-1]),2)-1)
```

Create a dataset object

4. Create a dataset object to be used for training

```
data_net = ds.SupervisedDataSet(input_data, target_data)
```

Create a FF NN

5. Create a feed forward neural network

```
num_input_features = data_net.getDimension('input')
num_hidden_layer_nodes = 5
num_target_features = data_net.getDimension('target')
net = buildNetwork(num_input_features, \
                   num_hidden_layer_nodes, \
                   num_target_features, \
                   hiddenclass=TanhLayer, \
                   outclass=SigmoidLayer, \
                   bias=True)
```

Create a trainer obj

6. Create a "trainer" object to train our neural network

```
trainer = BackpropTrainer(net, data_net)
```

Training the NN

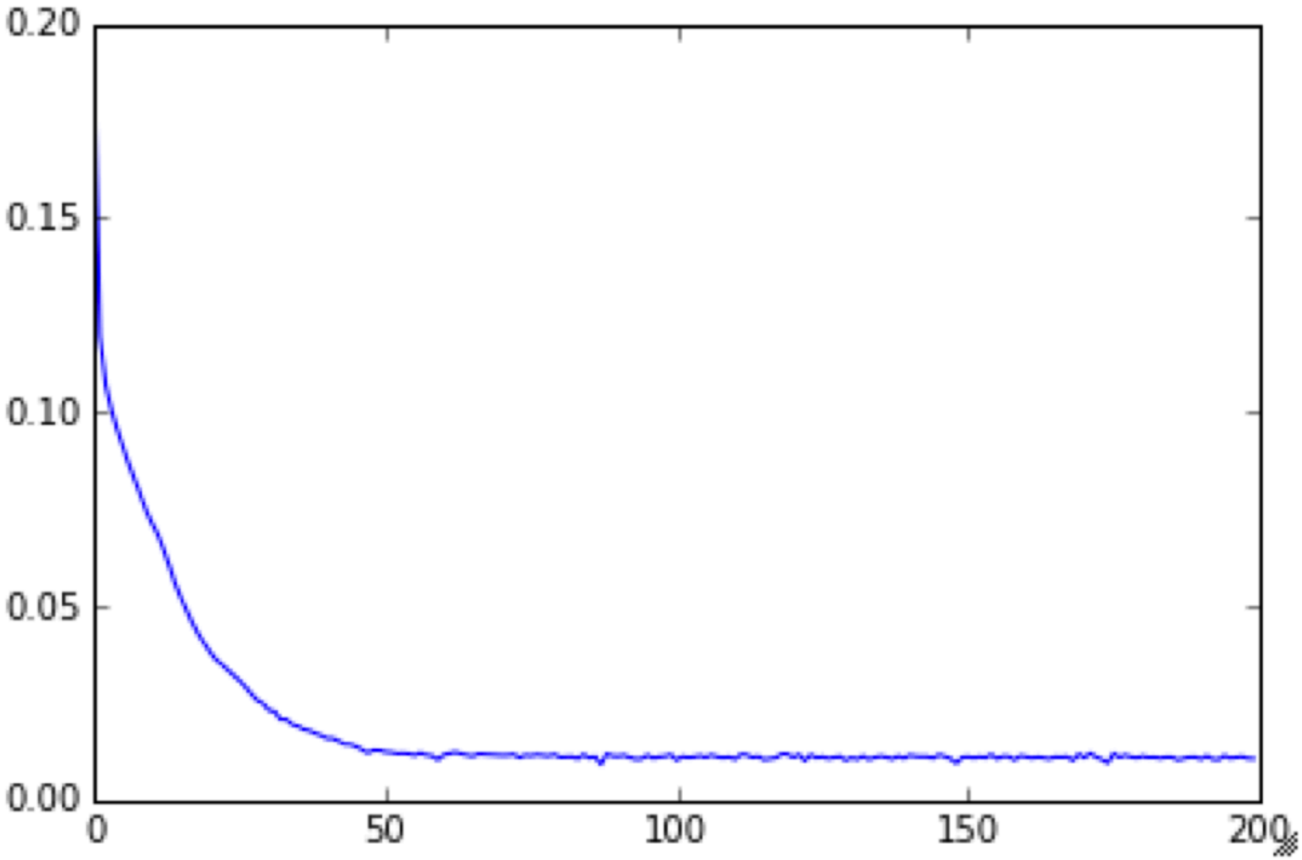
7. Training the network and computing the error at each epoch

```
epochs = 200
errors = np.zeros(epochs)
for i in xrange(epochs):
    errors[i] = trainer.train()
```

In sample error

```
plt.plot(errors)
```

```
[<matplotlib.lines.Line2D at 0x10c78c650>]
```



Predicting new data using the model

8. Using the model to predict the outputs

```
arr = net.activateOnDataset(data_net),target_data
```

```
threshold = 0.99; np.int32(np.divide(np.sign(arr[0]-threshold)+1,2))
```

Class Group Work (1)

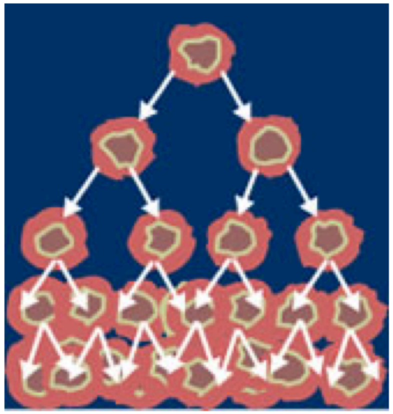
Breast Cancer Example

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer

Breast Cancer Data Set

Download: Data Folder, Data Set Description

Abstract: Breast Cancer Data (Restricted Access)



Data Set Characteristics:	Multivariate	**Number of Instances:**	286	**Area:**	Life
Attribute Characteristics:	Categorical	**Number of Attributes:**	9	**Date Donated**	1988-07-11
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	135598

Accuracy of the Model

- Divide your data with 20%, 80% ratio

- Train your neural net with a hidden layer that has 5 neurons and output layer of sigmoid [0,1]

- Plot in-sample and out-of-sample error for 200 epochs

- Find
  - TP, FP, TN, FN measures for the model (use 0.5 as threshold)

- How does the accuracy of in sample and out of sample change as we increase number of hidden layer nodes?

Performance Measurement

Let’s assume the following:

1. For every **positive** case that we predict **correctly,** we save the person 100,000$ in extra medical costs
2. For every **negative** case that we **wrongly** predict as **positive** we impose 10,000$ cost to the patient for extra checks

Find the optimal threshold for our classification that results in maximum performance based on our test data?

Class Group Work (2)

Class Group Work – Abalone Age

- Determining the age of Abalone is very laborious
- We want to find a formula that predicts the **age** of abalone based on some of its features

Class Group Work – Abalone Age

Feature	Description
sex	M, F, I, (Gender or Infant)
length	Longest shell measurement (mm)
diameter	Perpendicular to the length (mm)
height	With meat in shell (mm)
whole_weight (gr)	Whole weight (gr)
shucked_weight	Weight of meat (gr)
viscera_weight	Gut weight after bleeding (gr)
shell_weight	After being dried (gr)
rings	+1.5 gives the age in years

Abalone

- Build a neural network model that predicts the age of Abalone based on the features provided
  - Hint: the output layer has to be linear

Class Group Work (3)

Youtube Rating Prediction

1) Search for videos “*Madonna*”

2) *Parse the json result*

3) *Extract features of videos, e.g. number of likes, number of dislikes, number of views, number of days since its publish date*

4) *Come up with a formula that relates features to the rating*

https://gdata.youtube.com/feeds/api/videos?q=madonna&max-results=50&v=2&alt=json

Youtube videos

- Build a neural network model that predicts the age of Abalone based on the features provided
  - Hint: the output layer has to be linear

Example

- Try polynomial regression on youtube data and abalone examples

- How does the result of regression changes as we change the degree of polynomial from 5 to 2?

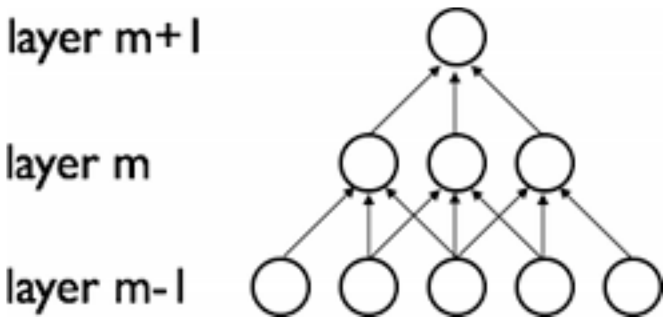
- Can you plot the in and out of sample R^2 score as a function of polynomial degree (k) for both problems? k=1..10

Group work

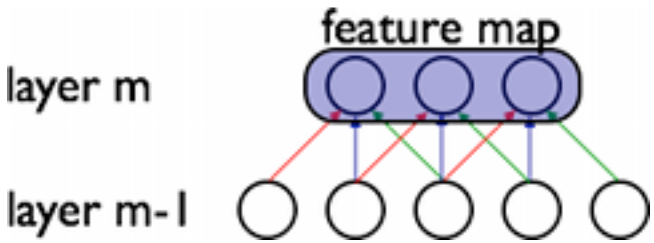
1) Extract the same set of financial indexes from 01/01/2000 to 01/01/2014
2) Pull the data for one the composite indexes (e.g. NASDAQ Composite .IXIC)
3) Try to come up with a linear or polynomial regression model that relates the indexes to the composite index
4) Assess the generality of your model by k-fold cross validation

Convolutional Neural Networks

- Sparse connectivity



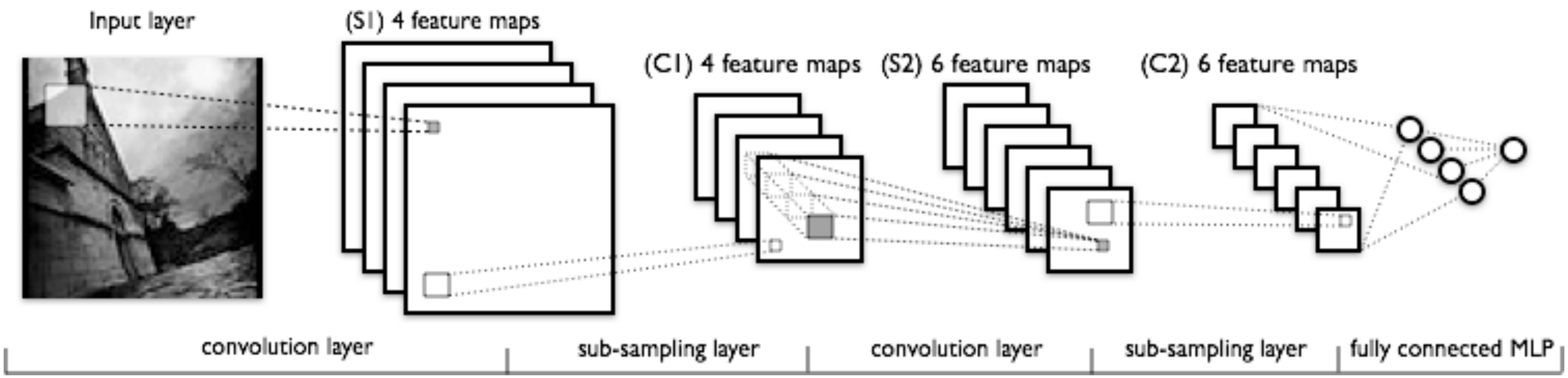
- Shared weights



Sub sampling

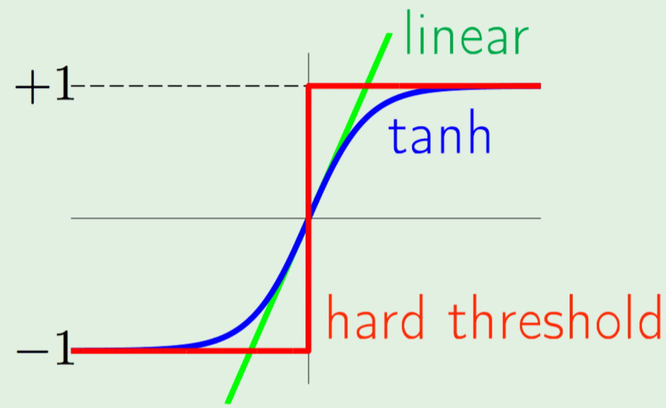
- Max Pooling
  - Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value.

Full Model (LeNet)



Questions?

$$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$



$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)}\, x_i^{(l-1)}\right)$$

$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Apply $\mathbf{x}$ to $x_1^{(0)} \cdots x_{d^{(0)}}^{(0)} \rightarrow \rightarrow x_1^{(L)} = h(\mathbf{x})$

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example $(\mathbf{x}_n, y_n)$ is

$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

To implement SGD, we need the gradient

$$\nabla\mathbf{e}(\mathbf{w}): \quad \frac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} \quad \text{for all } i, j, l$$

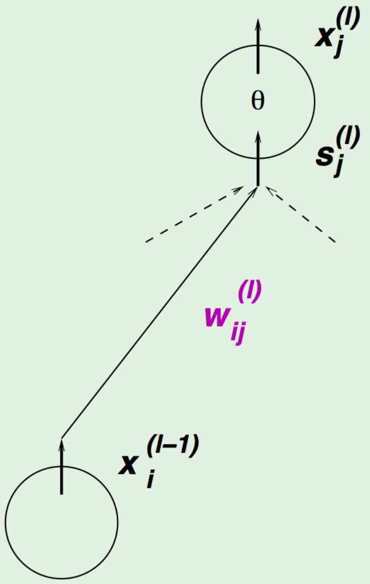
Computing $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$

We can evaluate $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$ one by one: analytically or numerically

A trick for efficient computation:

$$\frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}} \times \frac{\partial\ s_j^{(l)}}{\partial\ w_{ij}^{(l)}}$$

We have $\dfrac{\partial\ s_j^{(l)}}{\partial\ w_{ij}^{(l)}} = x_i^{(l-1)}$ We only need: $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}} = \delta_j^{(l)}$



δ for the final layer

$$\delta_j^{(l)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_1^{(L)}}$$

$$\mathrm{e}(\mathbf{w}) = (\, x_1^{(L)} - y_n)^2$$

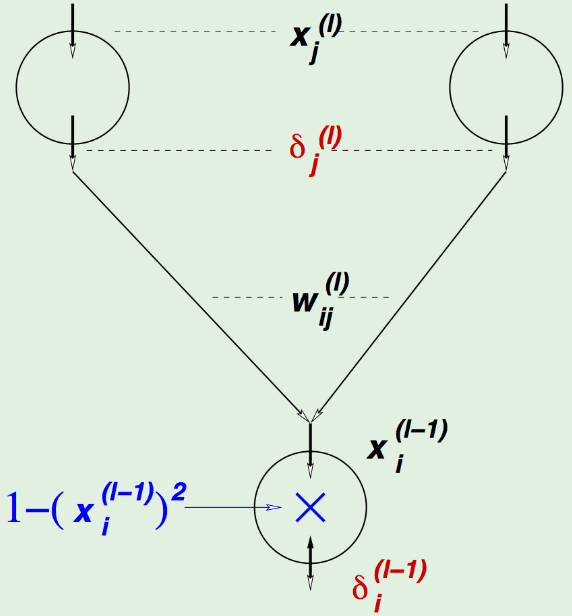
$$x_1^{(L)} = \theta(s_1^{(L)})$$

$$\theta'(s) = 1 - \theta^2(s)$$ for the tanh

Back propagation of δ

$$
\begin{aligned}
\delta_i^{(l-1)} &= \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}} \times \frac{\partial\ s_j^{(l)}}{\partial\ x_i^{(l-1)}} \times \frac{\partial\ x_i^{(l-1)}}{\partial\ s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \quad \delta_j^{(l)} \quad \times \quad w_{ij}^{(l)} \quad \times\ \theta'(s_i^{(l-1)})
\end{aligned}
$$

$$
\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)}\ \delta_j^{(l)}
$$



Backpropagation algorithm

Initialize all weights $w_{ij}^{(l)}$ at random

for $t = 0, 1, 2, \ldots$ **do**

  Pick $n \in \{1, 2, \cdots, N\}$

  Forward: Compute all $x_j^{(l)}$

  Backward: Compute all $\delta_j^{(l)}$

  Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \, x_i^{(l-1)} \delta_j^{(l)}$

  Iterate to the next step until it is time to stop

Return the final weights $w_{ij}^{(l)}$