Acoustic Modeling and Neural Network

Instructor: Tom Ko

The emission probabilities

- Recall that there is a $b_j(x_t)$ in the Viterbi algorithm and forward-backward algorithm
- $b_j(x_t) = \mathrm{p}(x_t \mid s_j)$
- This represents the relationship between the audio signal and the phonetic units and is modeled by the acoustic model.

Start from the most simplest case

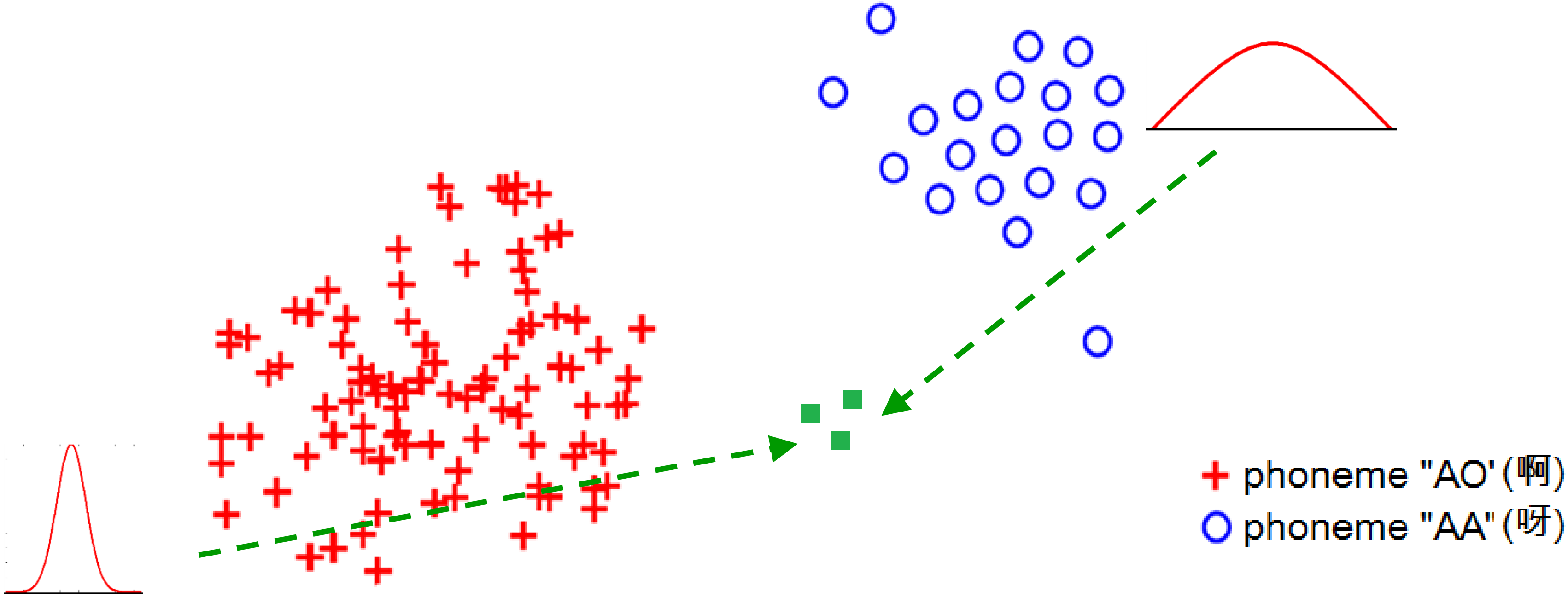
- Assume that there are only two sound units to classify: AO(啊) and AA(呀)
- Let x be one of the testing samples, we want to compute P(AO|x) and P(AA|x)

$$\boxed{P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)},}$$

- Thus we need P(x|AO)P(AO) and P(x|AA)P(AA)
- For the likelihood terms, we use Gaussian models:

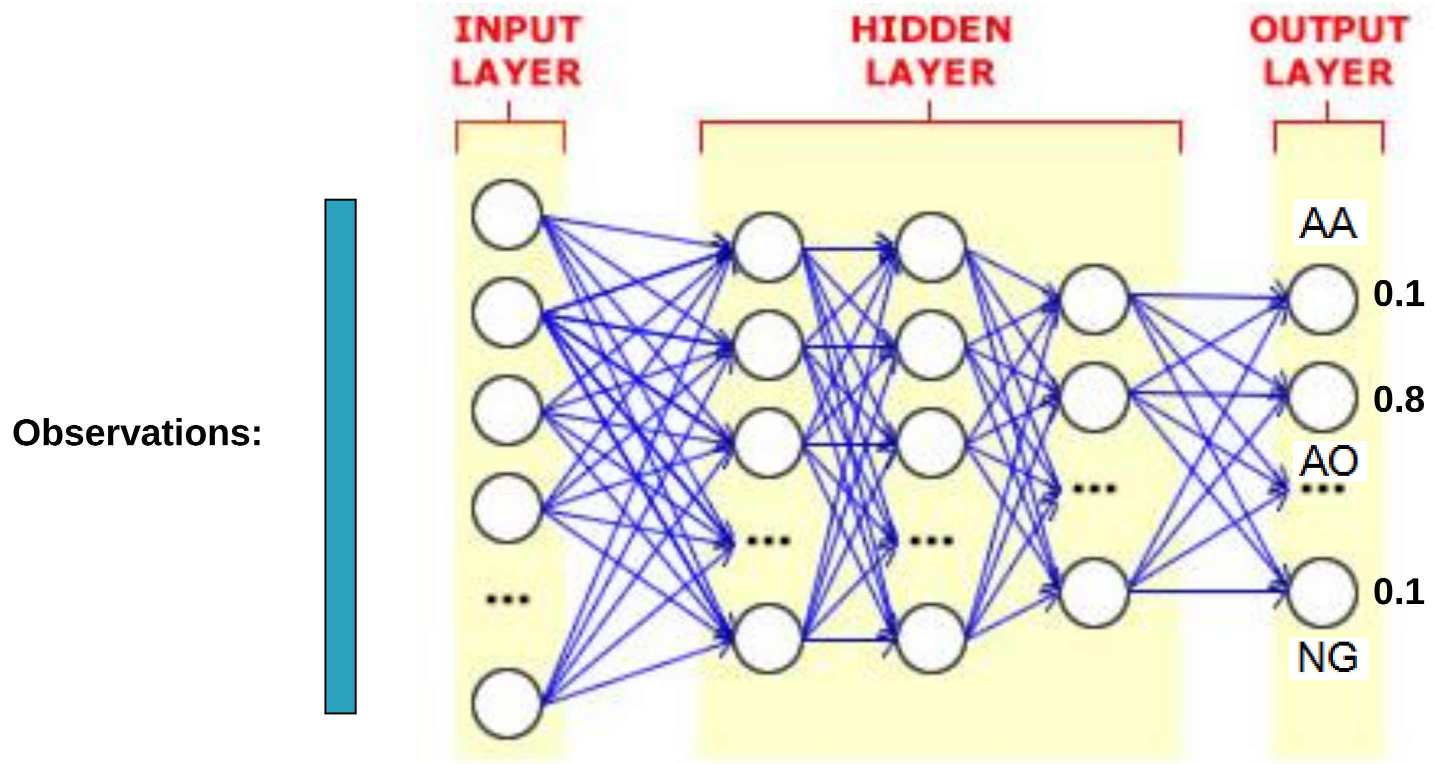
$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}\sigma_1}\exp\left[-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2\right]$$

Classification with GMMs



- Obtain P(AO) and P(AA) by simple counting
- Build a Gaussian model for each class: $N(\mu_1,\sigma_1^2)$ and $N(\mu_2,\sigma_2^2)$
- Compute P(x|AO) and P(x|AA)
- Make decision depending on P(AO|x) and P(AA|x)

Classification with a neural network



- The output is directly the posterior $\mathrm{p}(s_j \mid x_t)$
- .

Generative vs. discriminative

- Let's say you have input data x and you want to classify the data into labels y.
- A generative model learns the **joint** probability distribution $p(x,y)$
- A discriminative model learns the **conditional** probability distribution $p(y|x)$ – which you should read as *"the probability of y given x"*.
- Suppose you have the following data in the form (x,y):
- (1,0), (1,0), (2,0), (2, 1), (1,2), (1,2)
- $p(x,y)$ is

	y=0	y=1	y=2
x=1	2/6	0	2/6
x=2	1/6	1/6	0

- $p(y|x)$ is

	y=0	y=1	y=2
x=1	1/2	0	1/2
x=2	1/2	1/2	0

Generative vs. discriminative

- GMM is a generative model
  - Each GMM is built for each class separately
  - It expresses how to generate the features if we knew it was of a particular class.
- NN is a discriminative model
  - It will learn to assign high weight to features that directly improve its ability to discriminate between possible classes
- The overall observation is that discriminative models generally outperform generative models in classification tasks.

Generative vs. discriminative

- In Bayesian learning, the posterior probability is transformed into a likelihood and a prior. They are learnt separately.
  - Thus, it is a generative classifier.
- How about a discriminative classifier which learn and compute the posterior probability directly?
  - Logistic regression is a discriminative classifier
  - A neural network can be viewed as a series of logistic regression classifiers stacked on top of each other

Logistic regression

- Consider a single input observation x, which is a vector of features [$\mathbf{x}_1$, ..., $\mathbf{x}_n$]
- Logistic regression solves this task by learning, from a training set, a vector of weights and a bias term.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

- The resulting single number z expresses the weighted sum of the evidence for the class.

$$z = w \cdot x + b$$

The sigmoid classifier

- Let us first assume a two-class classification problem.
- We'll pass z through the sigmoid function **σ(z)**
- The sigmoid function (named because it looks like an s) is also called the logistic function

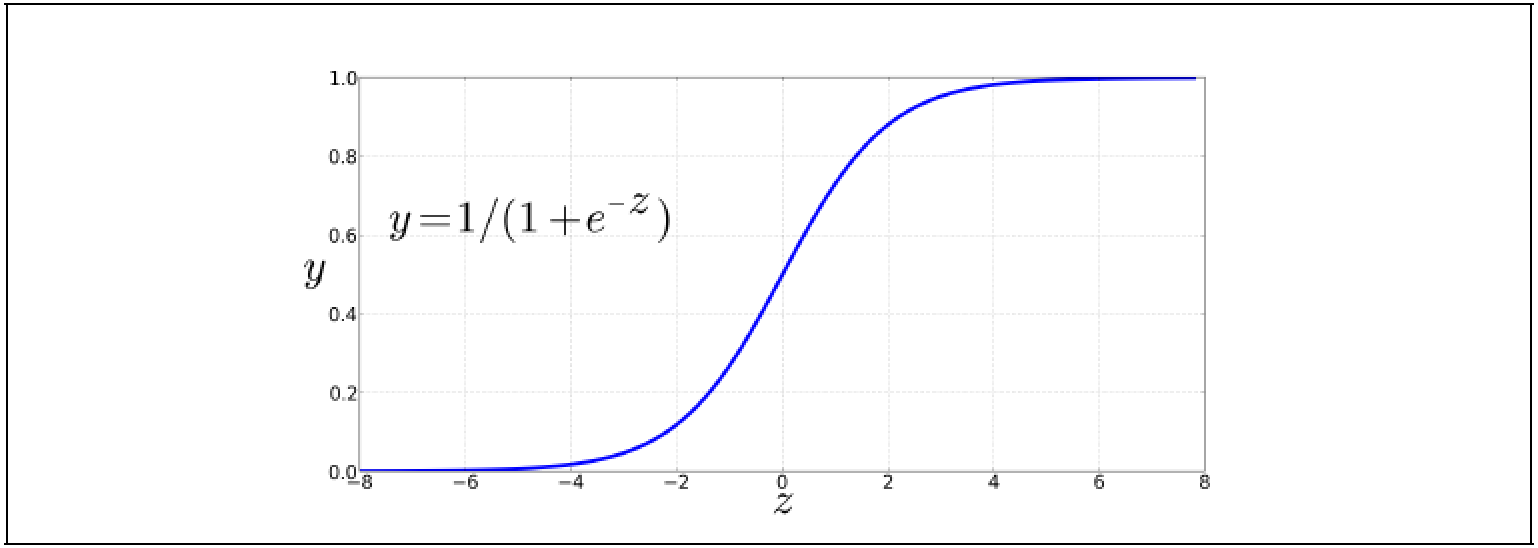
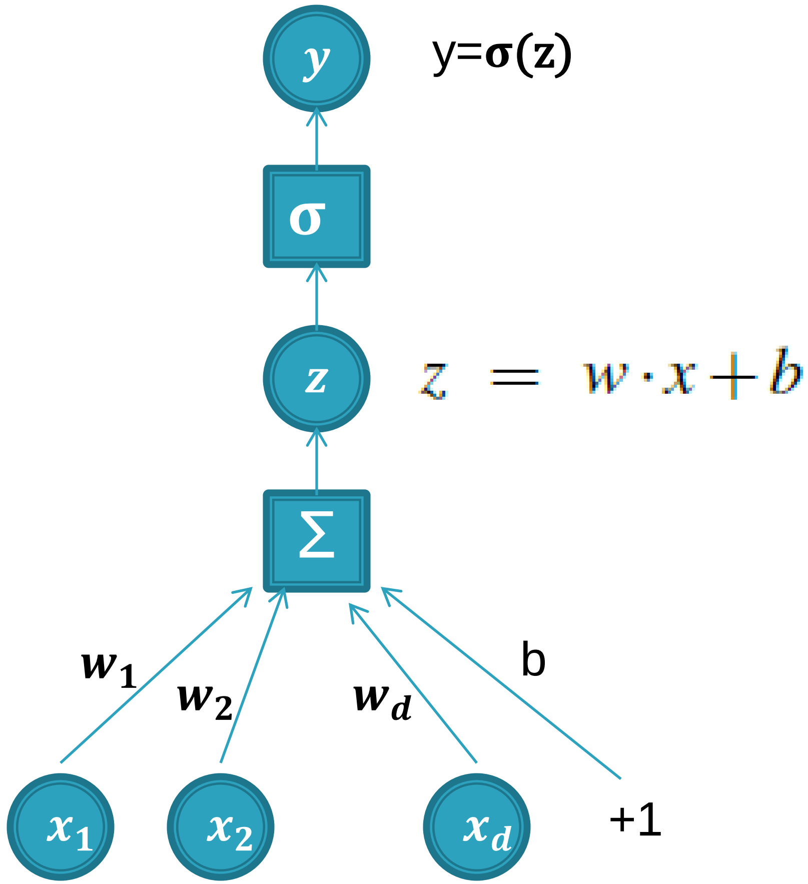


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. Because it is nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1.

The sigmoid classifier



The sigmoid classifier

- The sigmoid has a number of advantages:
  - It take a real-valued number and maps it into the range [0;1], which is just what we want for a probability.
  - Because it is nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1.
  - It's differentiable
- If $\boldsymbol{\sigma}(\mathbf{z})$ represents the probability P(w_1|x), then 1−$\boldsymbol{\sigma}(\mathbf{z})$ represents the probability P(w_2|x)
- For a test instance x, we say w_1 if the probability $\boldsymbol{\sigma}(\mathbf{z})$ is more than 0.5, and no otherwise.
- We call 0.5 the decision boundary

Loss function

- We need a loss function that expresses, for an observation x, how close the classifier output ($\hat{y} = \boldsymbol{\sigma}(\mathbf{z})$) is to the correct output (y, which is 0 or 1). We'll call this:

$$L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$$

Loss function

- The probability of class w_1 (y=1, the positive class) computed by the model given an observation x is $\hat{y}$
- Similarly, the probability of class w_2 (y=0, the negative class) computed by the model is $(1-\hat{y})$
- Thus, we can express p(y|x) as

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

$$\begin{aligned}\log p(y|x) &= \log\left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

- This is the term that we want to maximize.
- By flipping the sign, this is just happened to be the cross-entropy loss.

Cross-entropy loss function

- The cross-entropy loss is defined to be

$$L_{CE}(\hat{y}, y) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

- By substituting $\hat{y} = \sigma(w \cdot x + b)$

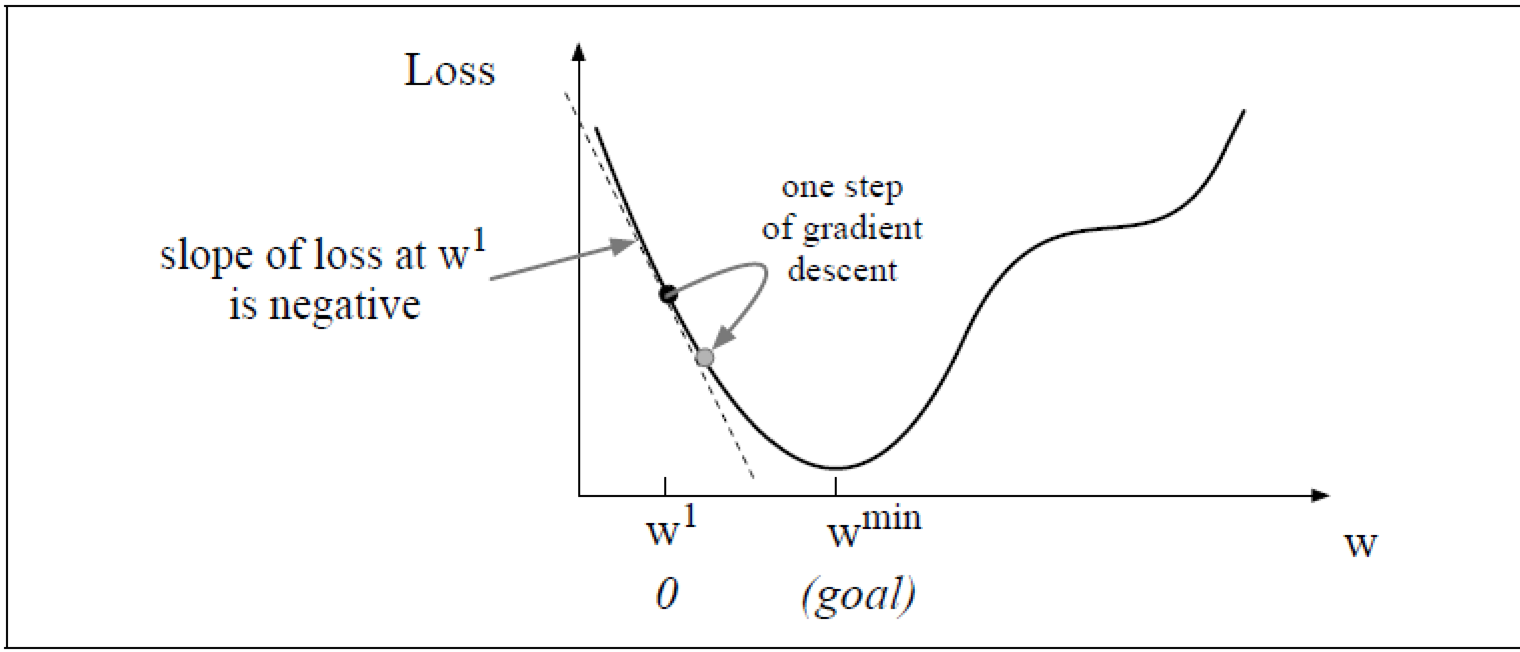
$$L_{CE}(w,b) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

- The total loss of the whole training set:

$$\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

Gradient descent

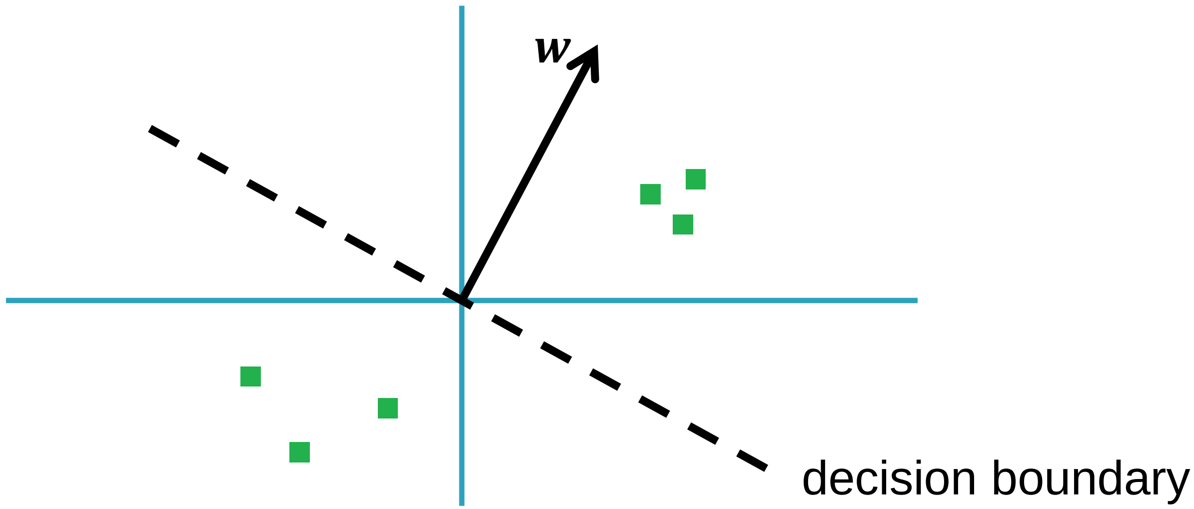
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$



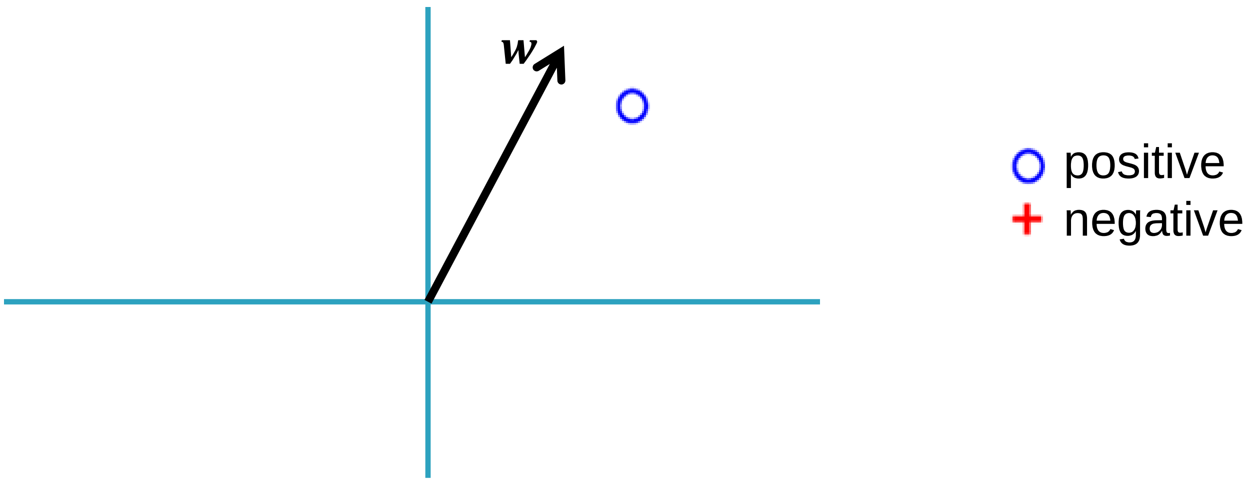
$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

How a sigmoid classifier classifies

- $\boldsymbol{w} \cdot \boldsymbol{x} = |\boldsymbol{w}|\ |\boldsymbol{x}|\ \cos\theta$
- We assume the bias term is 0 in this case, it will shift the decision boundary for non zero values.

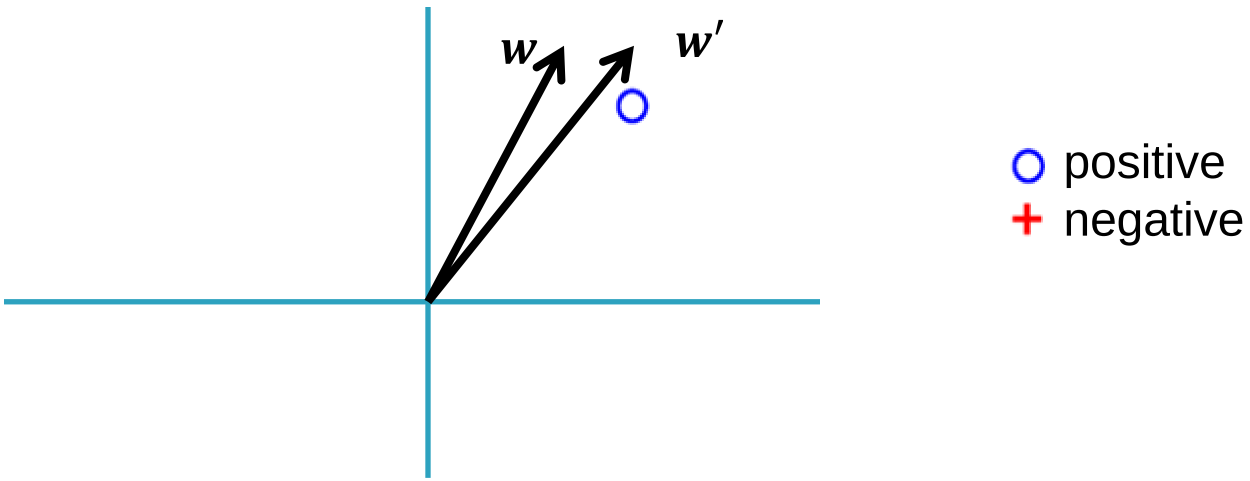


How it changes in training

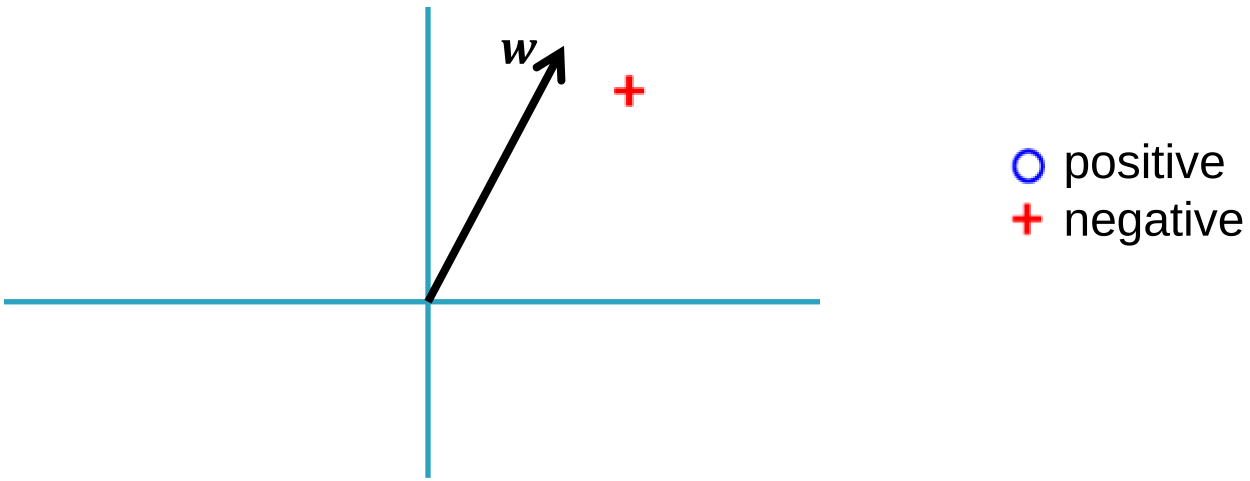


How it changes in training

- $\boldsymbol{w}$ will draw close to positive samples

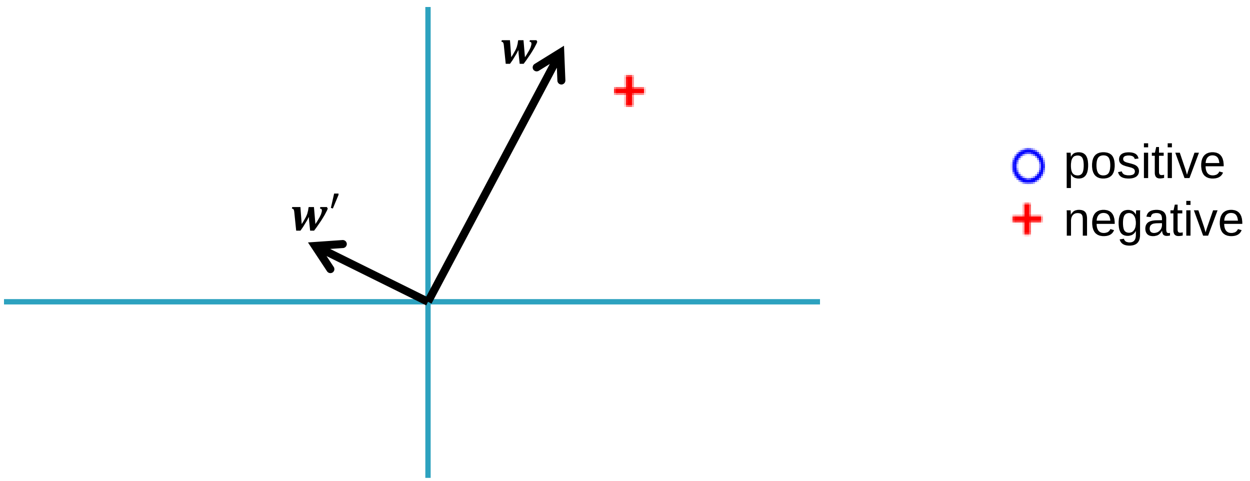


How it changes in training



How it changes in training

- $\boldsymbol{w}$ will draw away from negative samples



Softmax Classifier

- Softmax classifier is a generalization of sigmoid classifier when the number of classes > 2.
- It is also called multinominal logistic regression
- For a vector z of dimensionality k, the softmax is defined as:

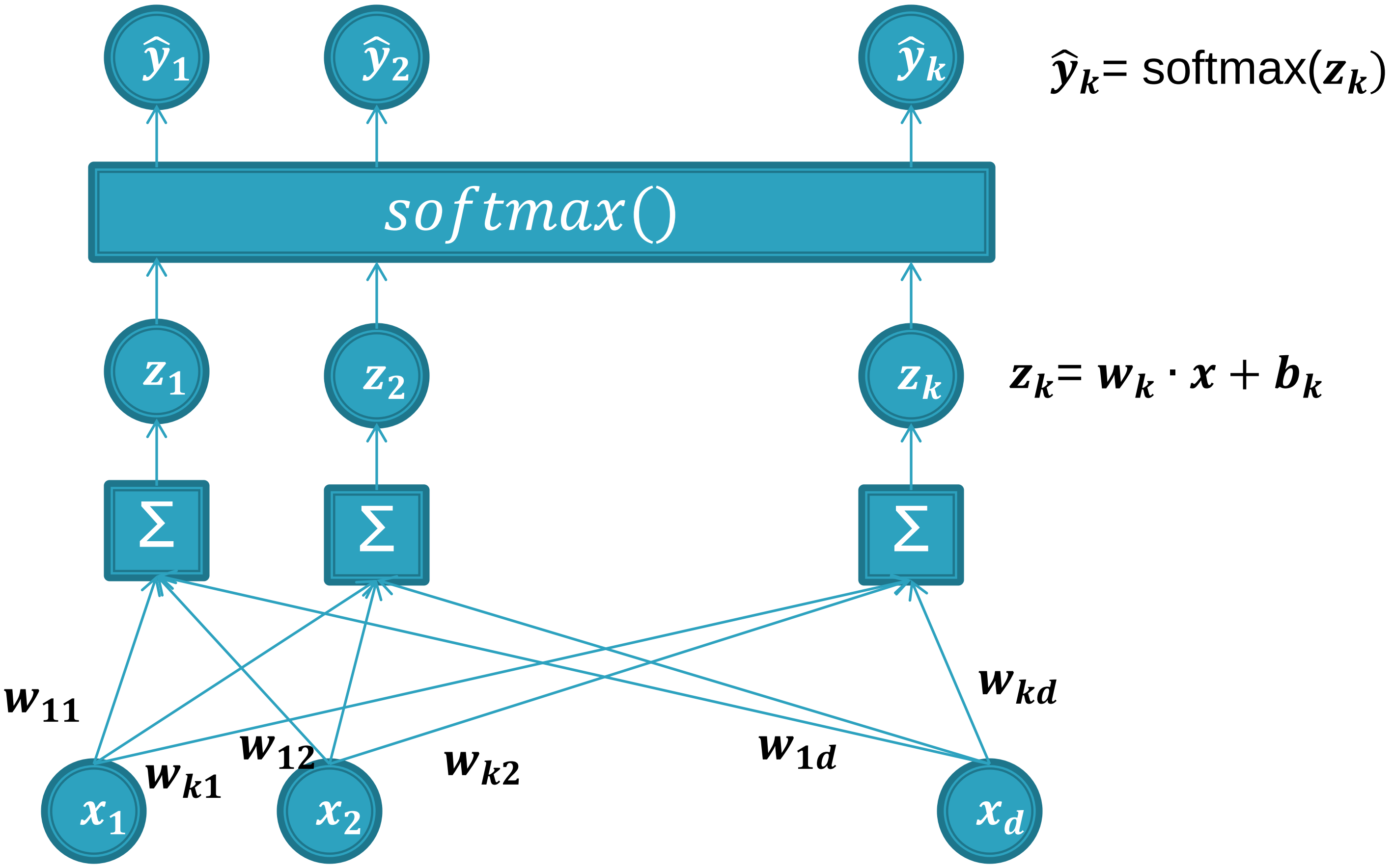
$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq k$$

- The probability of class c among K classes P(w_c|x) =

$$\frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

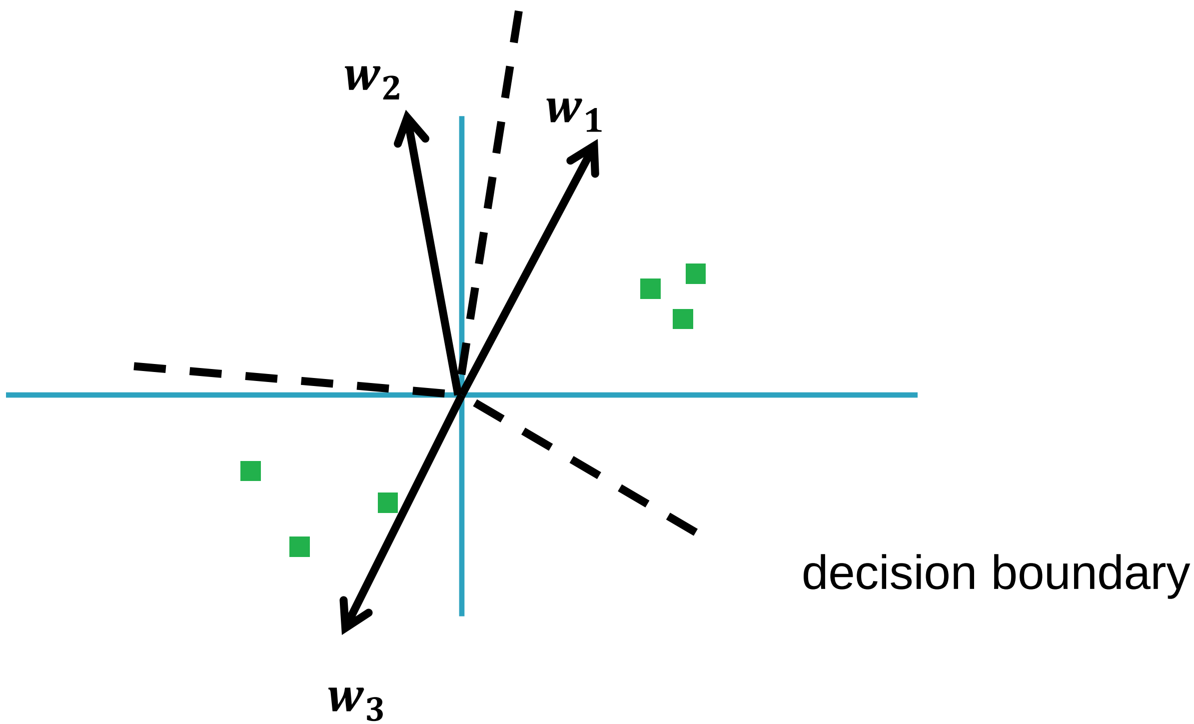
- The outputs of a softmax sum up to 1

The softmax classifier



How a softmax classifier classifies

- $\boldsymbol{w} \cdot \boldsymbol{x} = |\boldsymbol{w}|\ |\boldsymbol{x}|\ \cos\theta$



Derivative of softmax

- $$\frac{\partial \hat{y}_k}{\partial z_i} = \frac{\partial \frac{e^{z_k}}{\sum e^{z_j}}}{\partial z_i} = \frac{\sum e^{z_j} \frac{\partial e^{z_k}}{\partial z_i} - e^{z_k} \frac{\partial \sum e^{z_j}}{\partial z_i}}{[\sum e^{z_j}]^2}$$
- If i == k, the above term becomes
- $$\frac{\sum e^{z_j} e^{z_k} - e^{z_k} e^{z_k}}{[\sum e^{z_j}]^2} = \frac{e^{z_k} (\sum e^{z_j} - e^{z_k})}{[\sum e^{z_j}]^2}$$
- $$= \frac{e^{z_k}}{\sum e^{z_j}} \frac{\sum e^{z_j} - e^{z_k}}{\sum e^{z_j}} = \frac{e^{z_k}}{\sum e^{z_j}} (1 - \frac{e^{z_k}}{\sum e^{z_j}}) = \hat{y}_k (1 - \hat{y}_k)$$
- If i != k, the term becomes
- $$\frac{- e^{z_k} e^{z_i}}{[\sum e^{z_j}]^2} = - \frac{e^{z_k}}{\sum e^{z_j}} \frac{e^{z_i}}{\sum e^{z_j}} = - \hat{y}_k \hat{y}_i$$

Cross-entropy loss for multi class

- The cross-entropy loss is defined to be

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

  - where

$$\hat{y}_k = \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^{K} e^{w_j \cdot x + b_j}}$$

- Thus,

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^{K} e^{w_j \cdot x + b_j}}$$

$$y_k = \begin{cases} 1 & \text{if } x \text{ is sample of class } k \\ 0 & \text{otherwise} \end{cases}$$

Learning in softmax classifier

- $\frac{\partial y_k}{\partial z_i} = \frac{\partial \frac{e^{z_k}}{\sum e^{z_j}}}{\partial z_i} = \frac{\sum e^{z_j} \frac{\partial e^{z_k}}{\partial z_i} - e^{z_k} \frac{\partial \sum e^{z_j}}{\partial z_i}}{[\sum e^{z_j}]^2}$
- Consider a class k sample, the CE loss becomes $-log\hat{y}_k$
- $\frac{\partial L_{CE}}{\partial w_k} = -\frac{\partial log\hat{y}_k}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_k} \frac{\partial z_k}{\partial w_k} = -\frac{1}{\hat{y}_k} \hat{y}_k (1 - \hat{y}_k) x = (\hat{y}_k - 1) x$
- Consider a class i (i !=k) sample, the CE loss becomes $-log\hat{y}_i$
- $\frac{\partial L_{CE}}{\partial w_k} = -\frac{\partial log\hat{y}_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_k} \frac{\partial z_k}{\partial w_k} = -\frac{1}{\hat{y}_i} (- \hat{y}_k \hat{y}_i) x = \hat{y}_k \ x$

- For a set of training data, the gradients for w_k are accumulated

Update the parameters

- The parameters will be updated according to

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

- Where η is the learning rate, we assume it to be 1 for now
- Here $\boldsymbol{w^t}$ is the column vector of class k at training step t
- Consider a class k sample, the change in the parameters will be
- $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - (\boldsymbol{\hat{y}_k} - \mathbf{1})\ \boldsymbol{x} = \boldsymbol{w^t} + (\mathbf{1} - \boldsymbol{\hat{y}_k})\ \boldsymbol{x}$
- As $(\mathbf{1} - \boldsymbol{\hat{y}_k})$ is always > 0, the update of $\boldsymbol{w^t}$ is like adding a scale of x to it. It is like moving $\boldsymbol{w^t}$ towards x.
- When $\boldsymbol{w^t}$ is already pointing in the similar direction as x, this move will increase the magnitude of $\boldsymbol{w^t}$. Thus, it is general to observe that, when there are more training samples of class k, the magnitude of column vector of class k is larger.

Update the parameters

- The parameters will be updated according to

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

- Where η is the learning rate, we assume it to be 1 for now
- Here $\boldsymbol{w^t}$ is the column vector of class k at training step t
- Consider a class i (i !=k) sample, the change in the parameters will be
- $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \boldsymbol{\hat{y}_k}\ \boldsymbol{x}$
- As $\boldsymbol{\hat{y}_k}$ is always > 0, the update of $\boldsymbol{w^t}$ is like subtracting a scale of x to it. It is like moving $\boldsymbol{w^t}$ away from x.

Update the parameters

- The parameters will be updated according to

$$w^{t+1} = w^{t} - \eta \frac{d}{dw} f(x; w)$$

- Where η is the learning rate, we assume it to be 1 for now
- Here $\boldsymbol{w^t}$ is the column vector of class k at training step t
- Consider a class i (i !=k) sample, the change in the parameters will be
- $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \boldsymbol{\hat{y}_k}\ \boldsymbol{x}$
- As $\boldsymbol{\hat{y}_k}$ is always > 0, the update of $\boldsymbol{w^t}$ is like subtracting a scale of x to it. It is like moving $\boldsymbol{w^t}$ away from x.

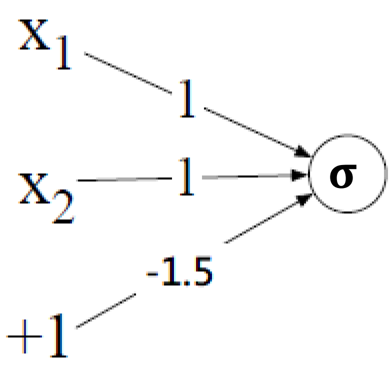
The XOR problem

- Consider the very simple task of computing simple logical functions of two inputs,

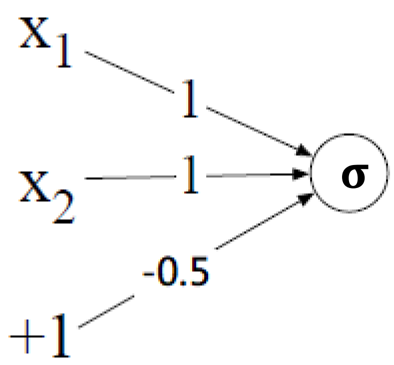
AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

The XOR problem

- Can you deduce the values of w and b for the XOR case?



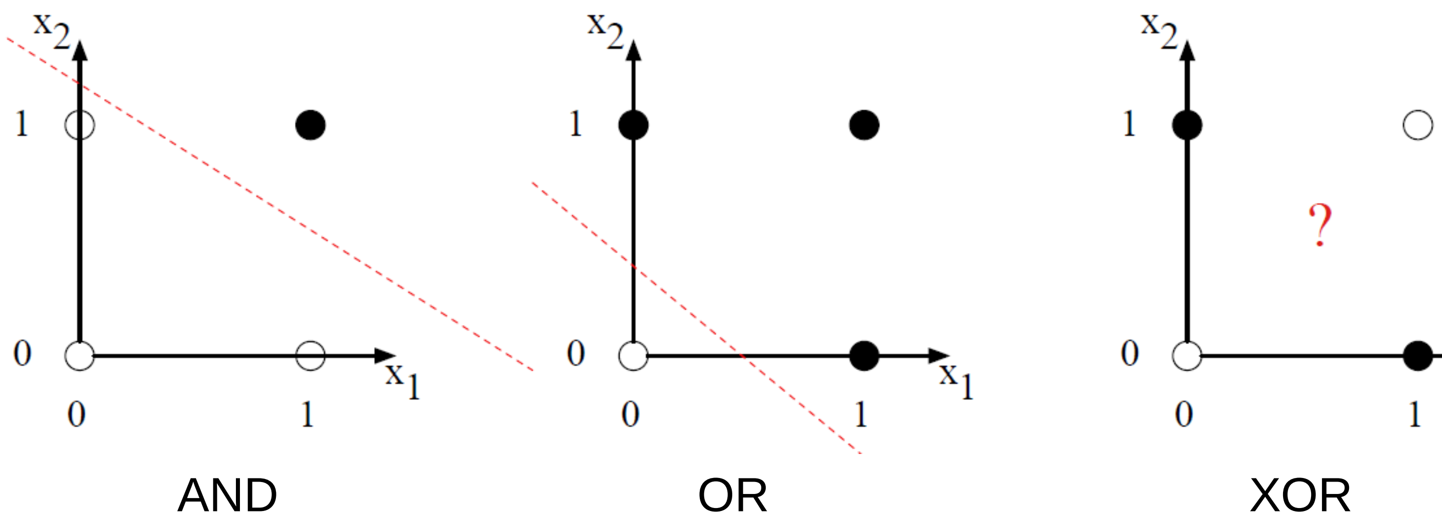
AND function



OR function

The XOR problem

- Consider the very simple task of computing simple logical functions of two inputs,

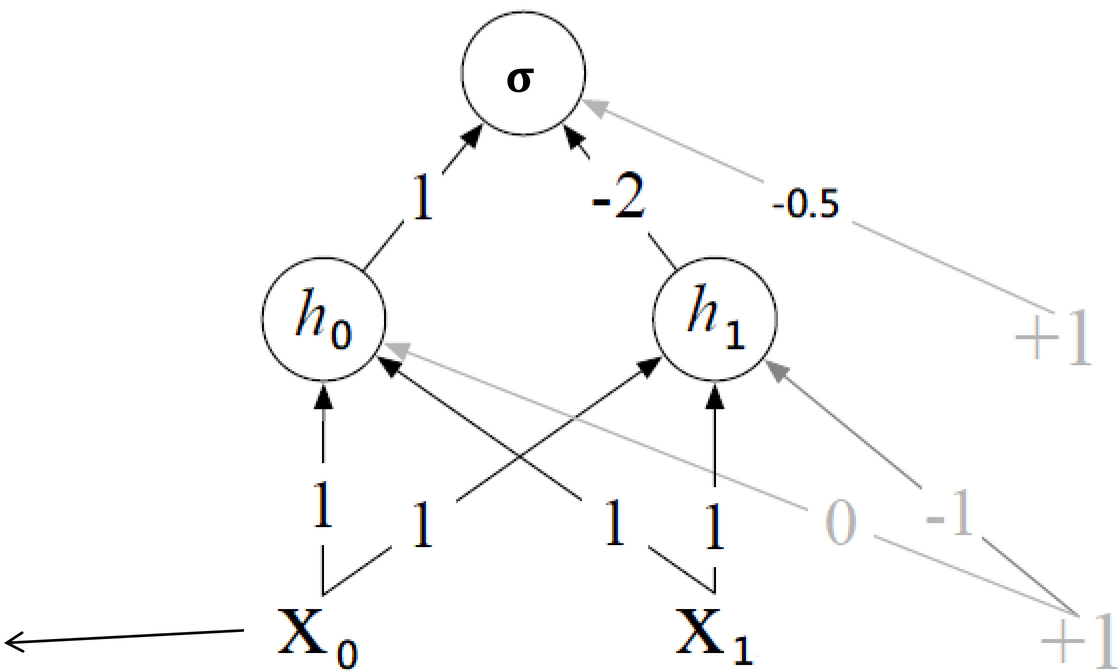


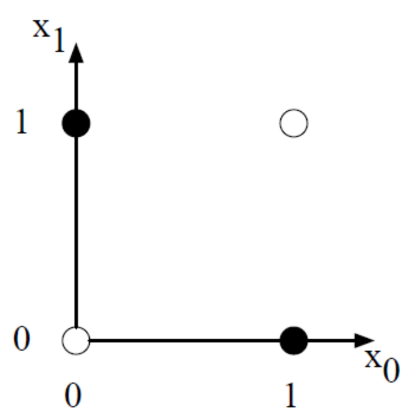
The XOR problem

- XOR is not a linearly separable function, it cannot be solved by logistic regression.
- Solution: neural network

Solution for the XOR problem

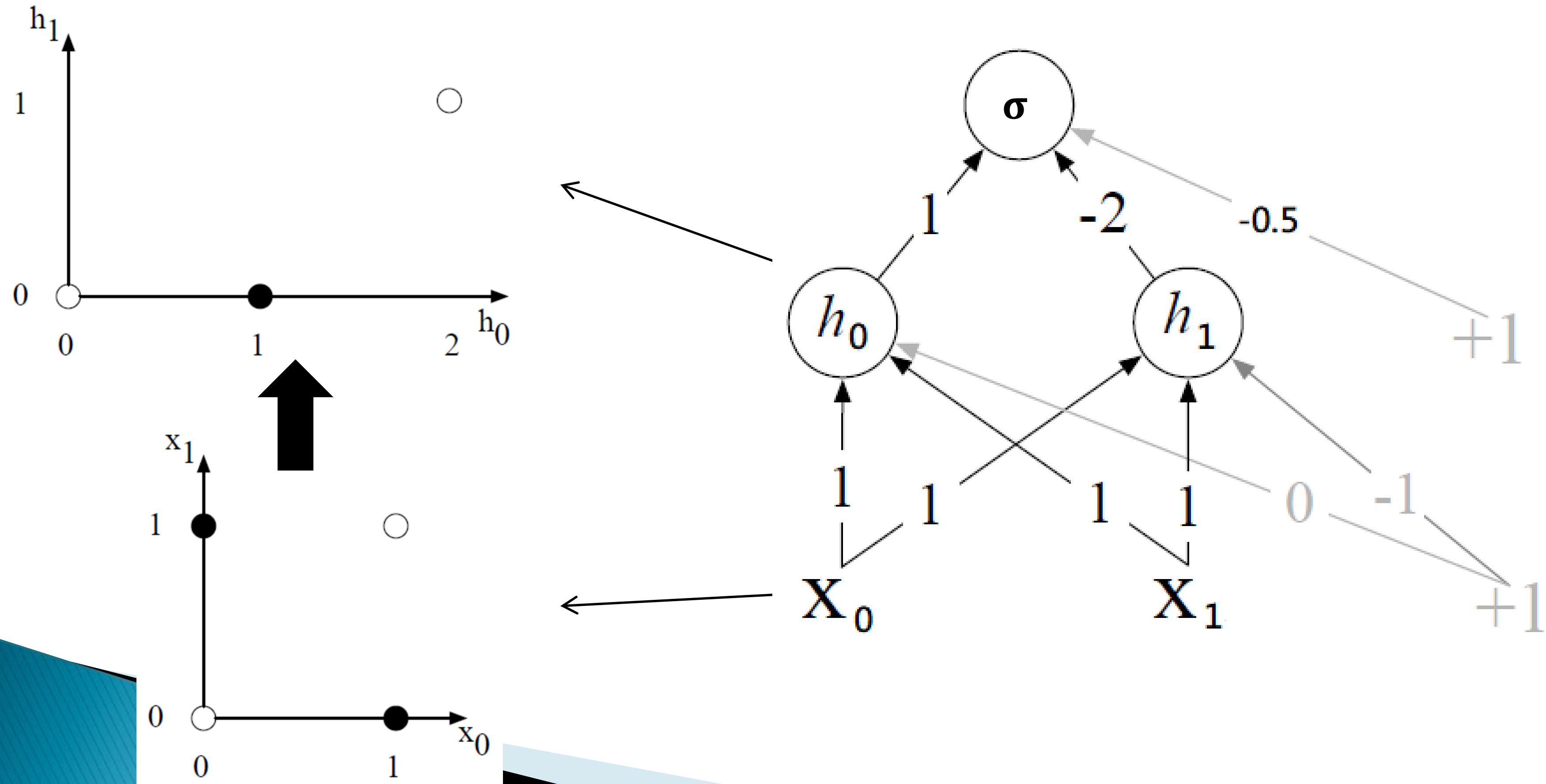
- Using a 2-layer neural network with ReLU-based units



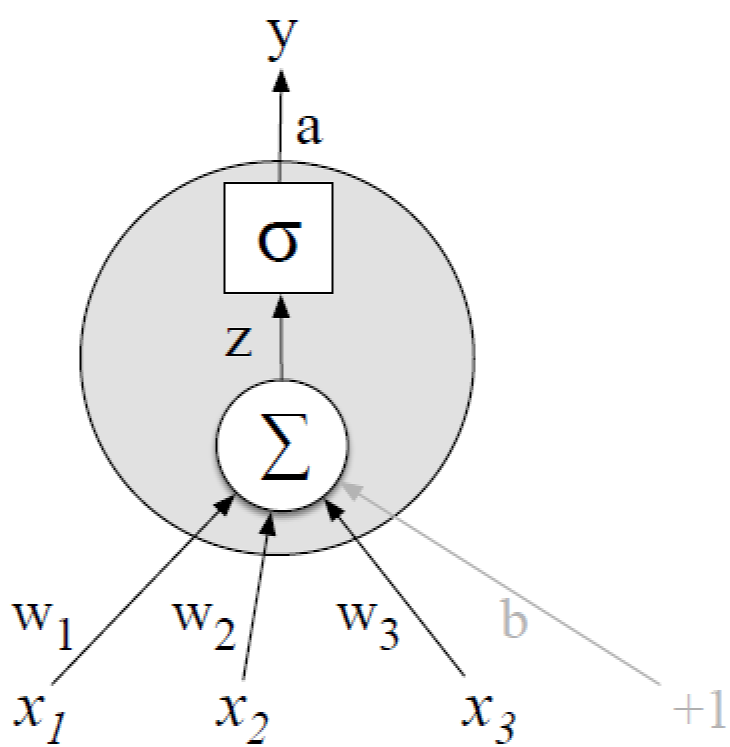


Solution for the XOR problem

- The hidden layer will learn to form useful representation

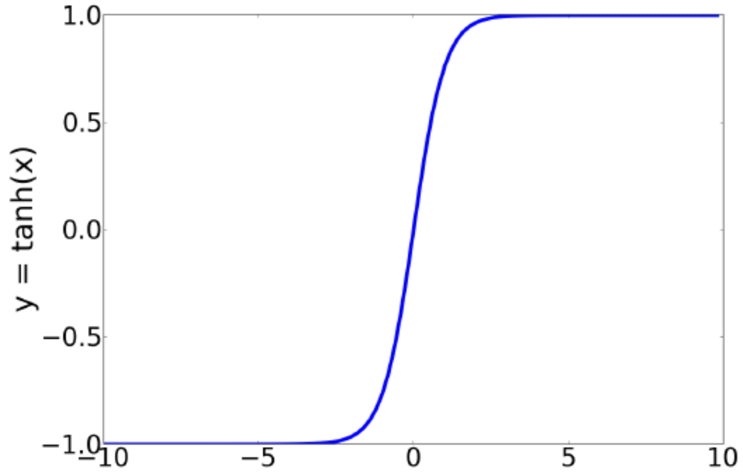


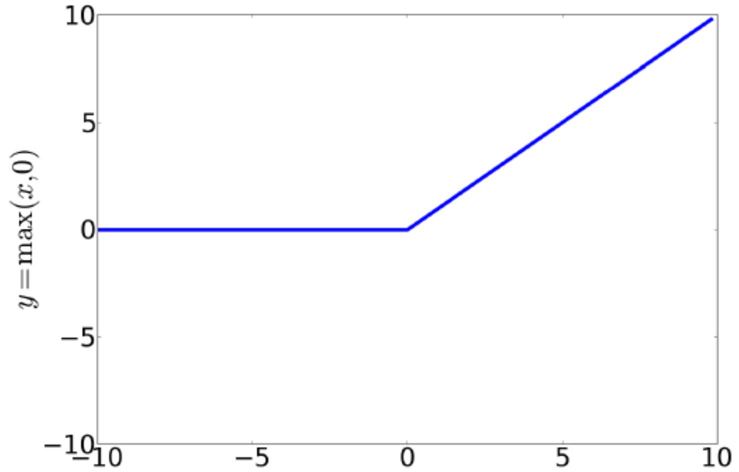
An example of neural unit



Activation functions

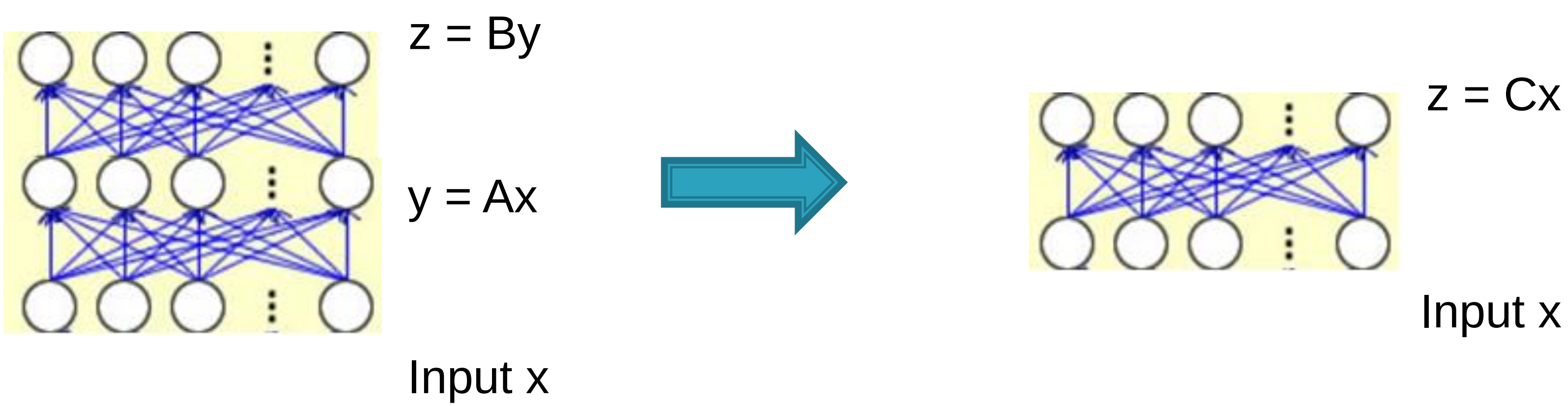
- Nowadays, the sigmoid is not commonly used as an activation function.
- Instead, people use
  - Tanh function
  - Rectified linear unit (ReLU)





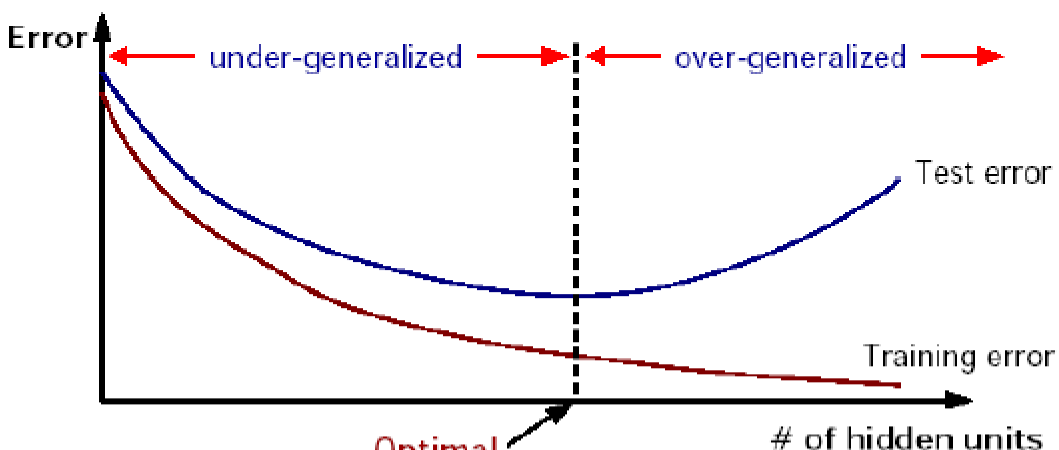
Without the non-linear layer

- A network formed by many layers of purely linear units can always be reduced to a single layer of linear units with appropriate weights.

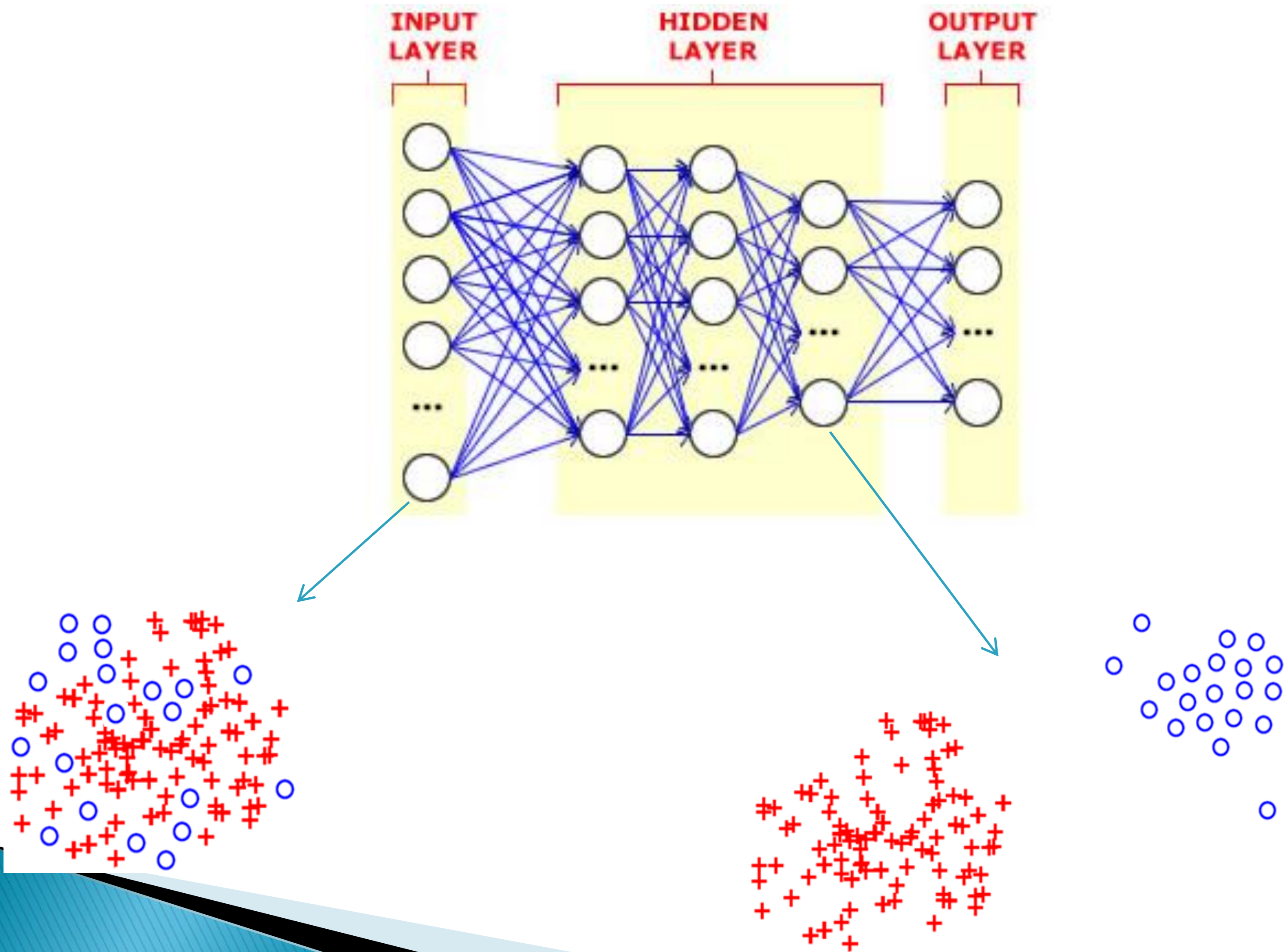


What is a Deep neural network?

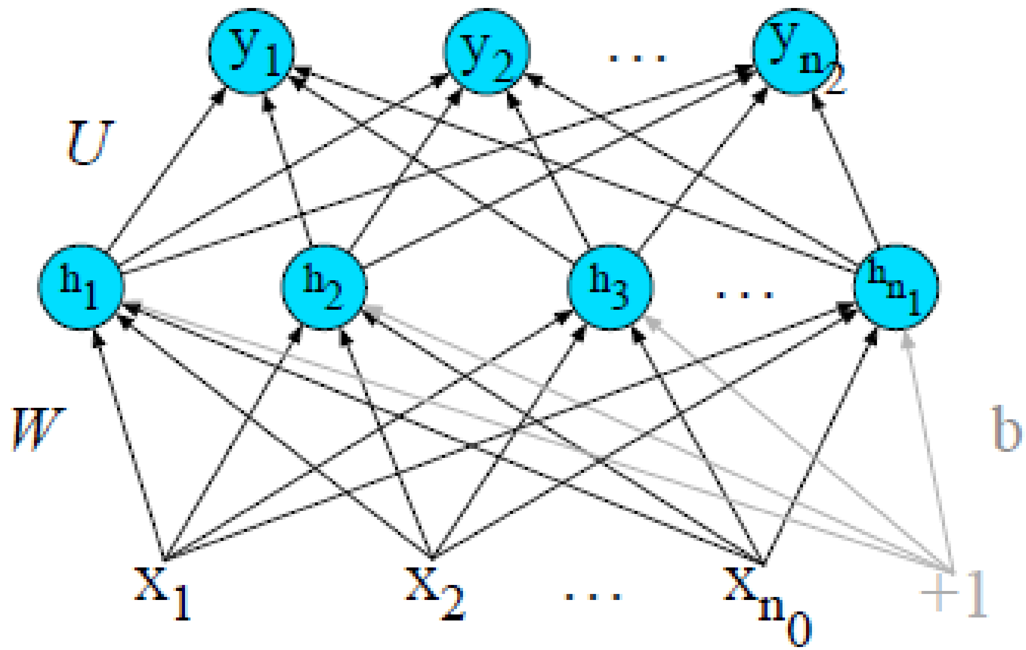
- Why need networks with > 2 hidden layers?
  - by using extra layers we might find a network with fewer weights in total while still achieving the same level of accuracy
- How may hidden units?
  - not too many nor too few



Classification with a neural network

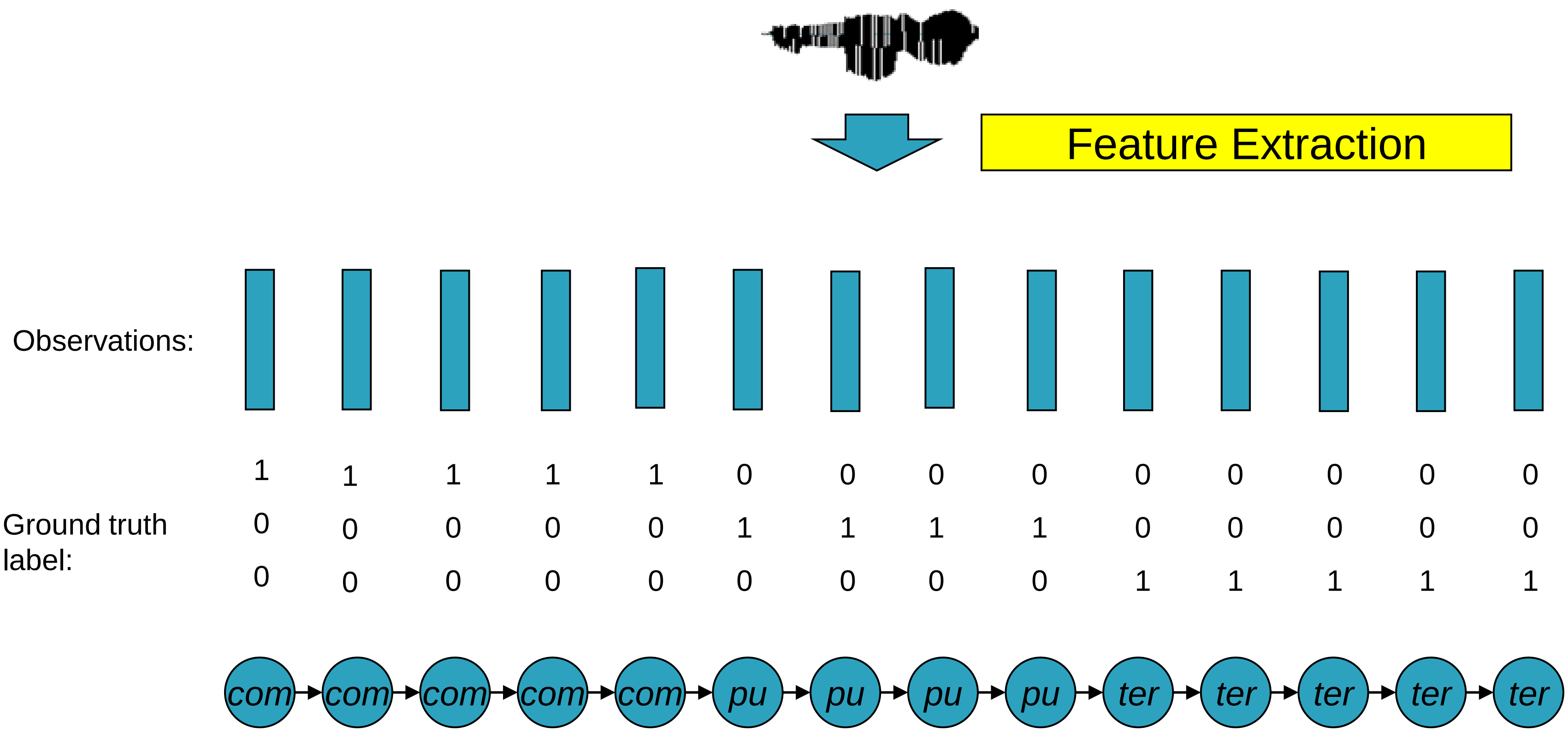


Backpropagation

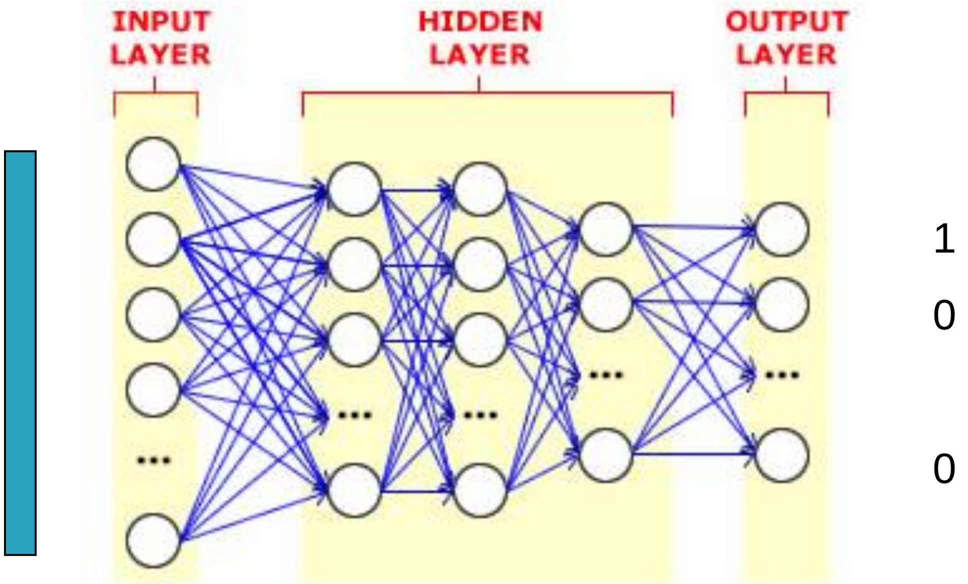


State alignment for training

- Given a 1-sec audio, resulting in 100 MFCCs, with a transcription of word "computer"



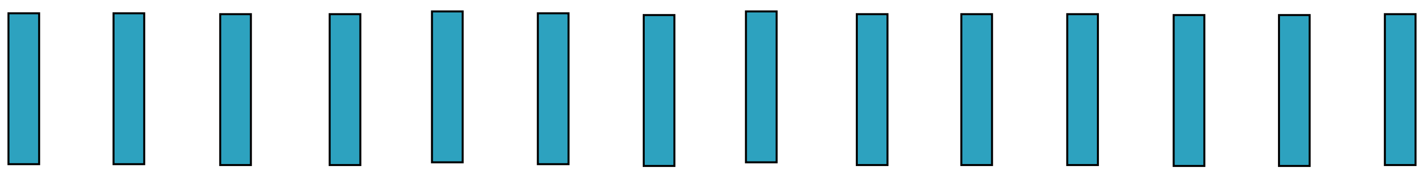
Feed-forward and backprogapate



Soft alignment vs. hard alignment

- Given a 1-sec audio, resulting in 100 MFCCs, with a transcription of word “computer”

Observations:



Ground truth label:	1	1	1	0.9	0.7	0.5	0.2	0.05	0	0	0	0	0	0
	0	0	0	0.1	0.3	0.5	0.8	0.9	0.8	0.4	0.2	0	0	0
	0	0	0	0	0	0	0	0.05	0.2	0.6	0.8	1	1	1

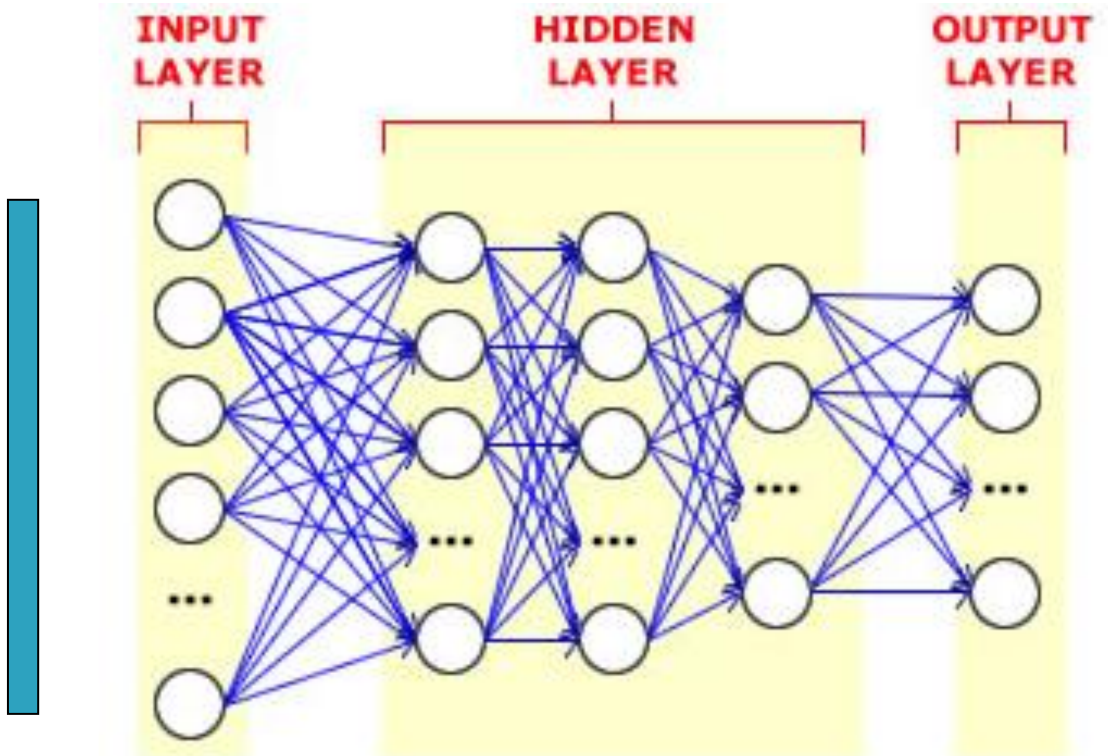
com → *com* → *com* → *com* → *com* → *pu* → *pu* → *pu* → *pu* → *ter* → *ter* → *ter* → *ter* → *ter*

Architectures of neural network

- Fully connected neural network
- Convolutional neural network (CNN)
- Recurrent neural network (RNN)
  - Long short-term memory (LSTM)

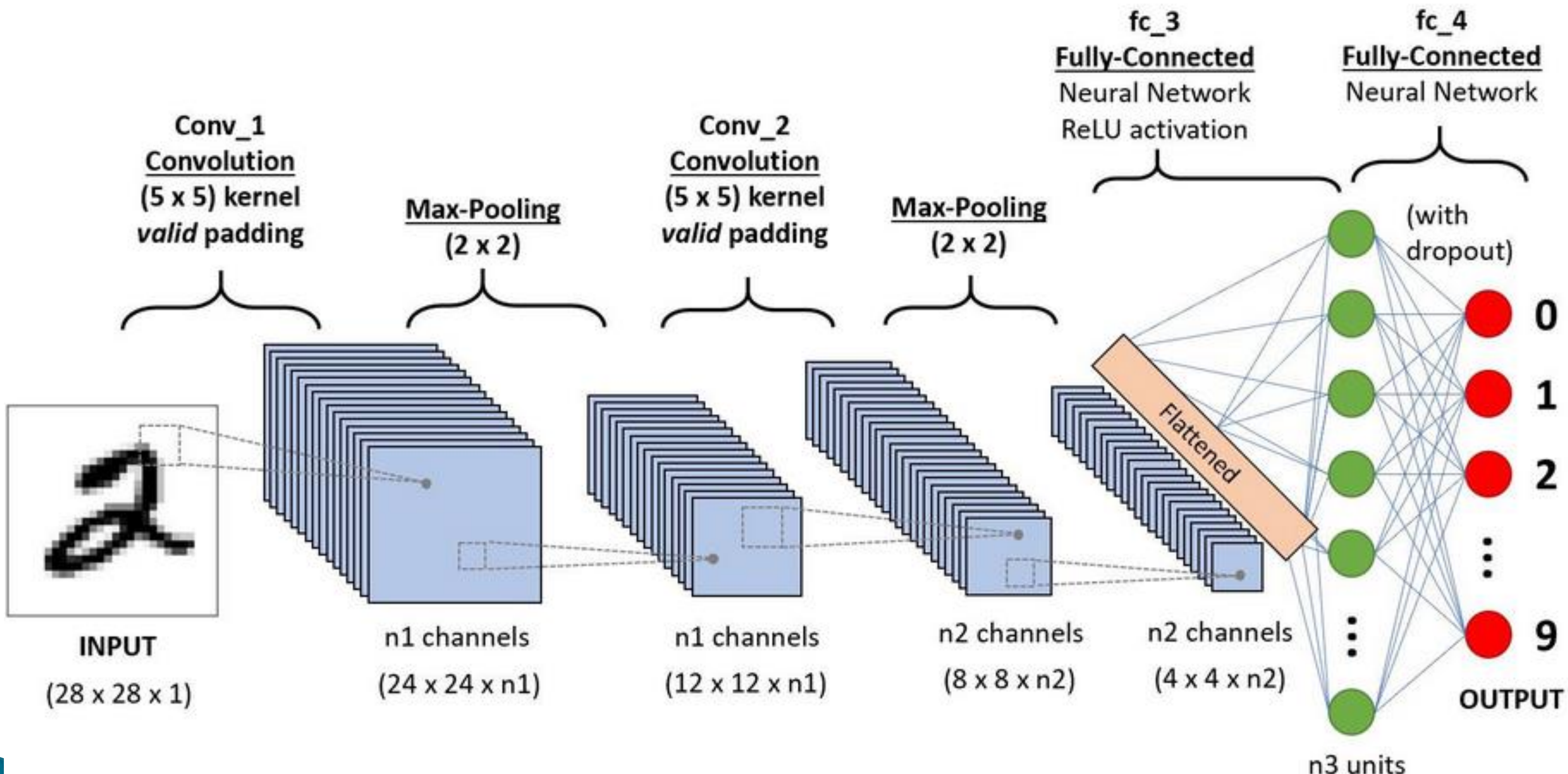
Fully connected neural network

Observations:

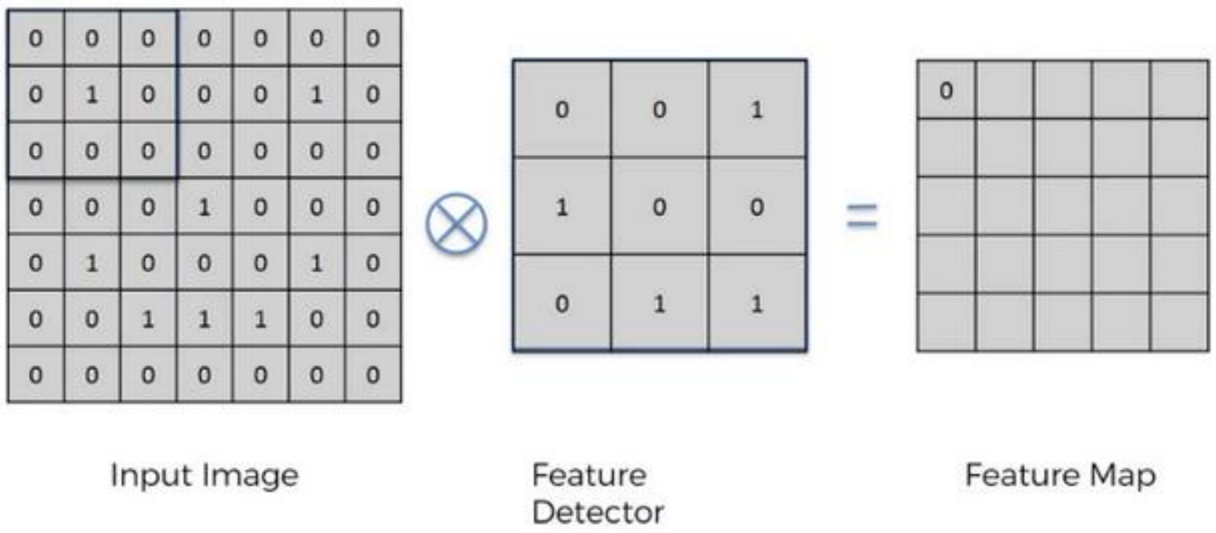


- How to handle when the input is 2D (e.g. an image)?

Convolutional neural network (CNN)

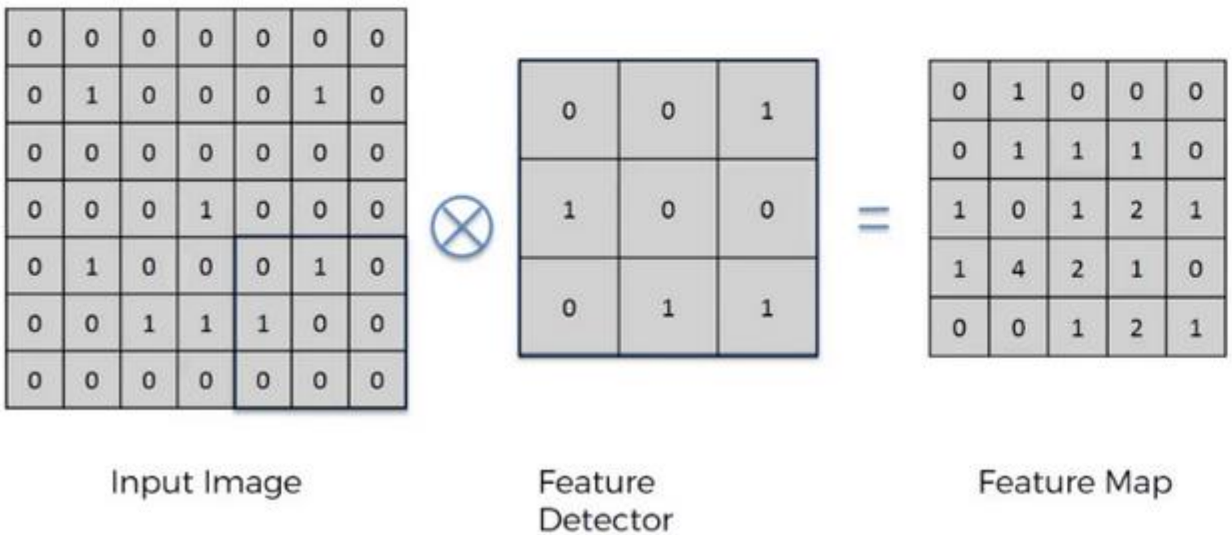


Convolutional layer

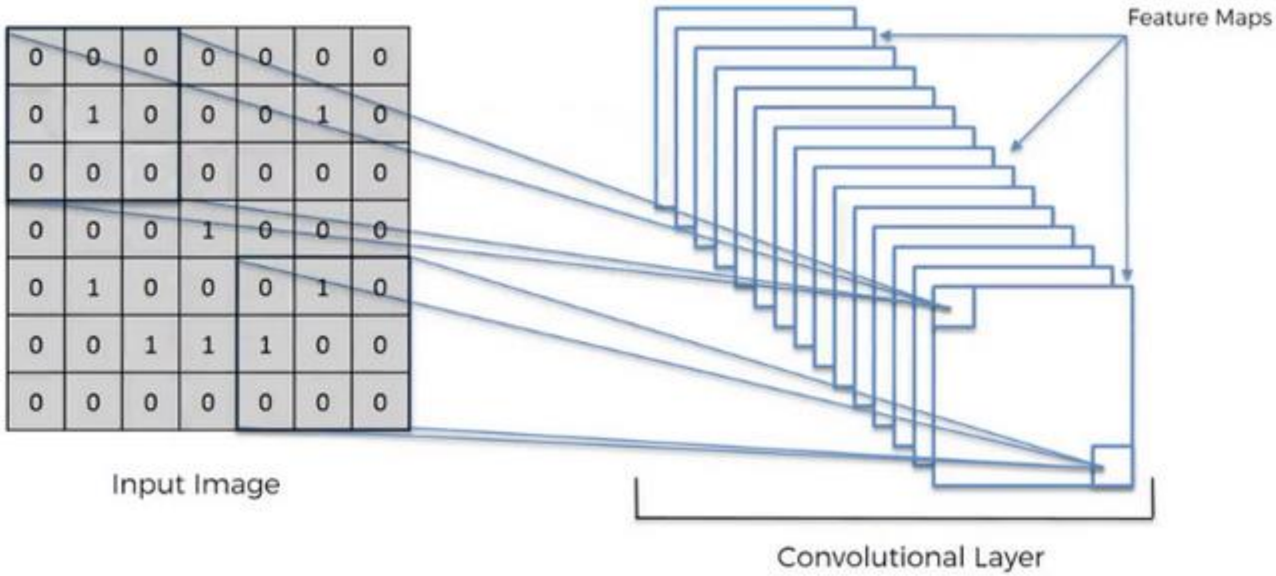


- Figures from https://medium.com/jameslearningnote

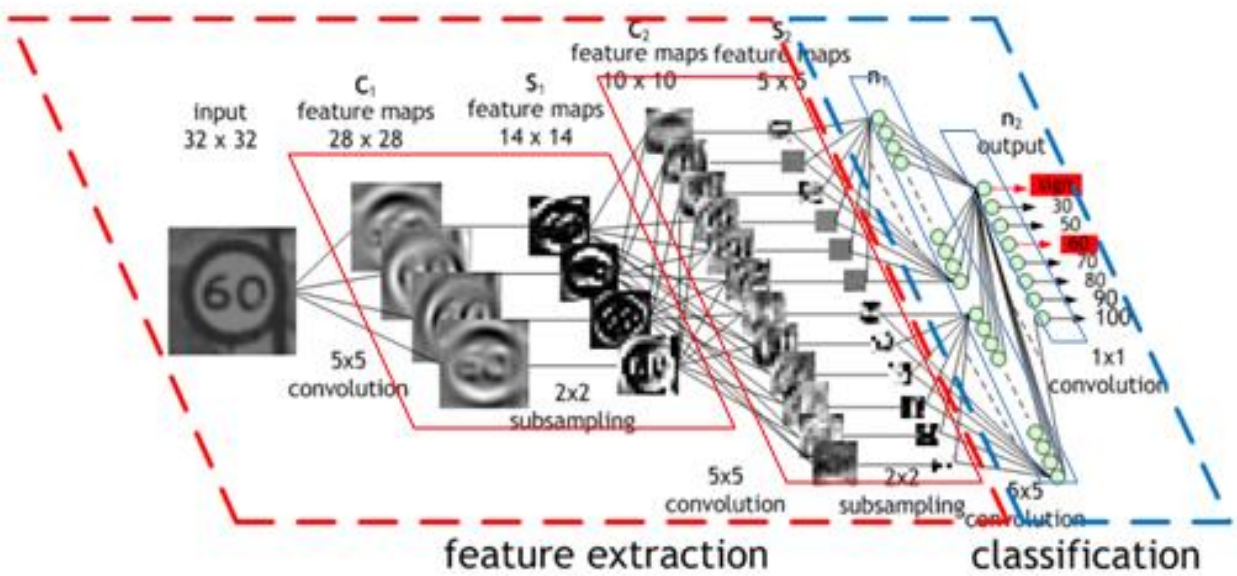
Convolutional layer



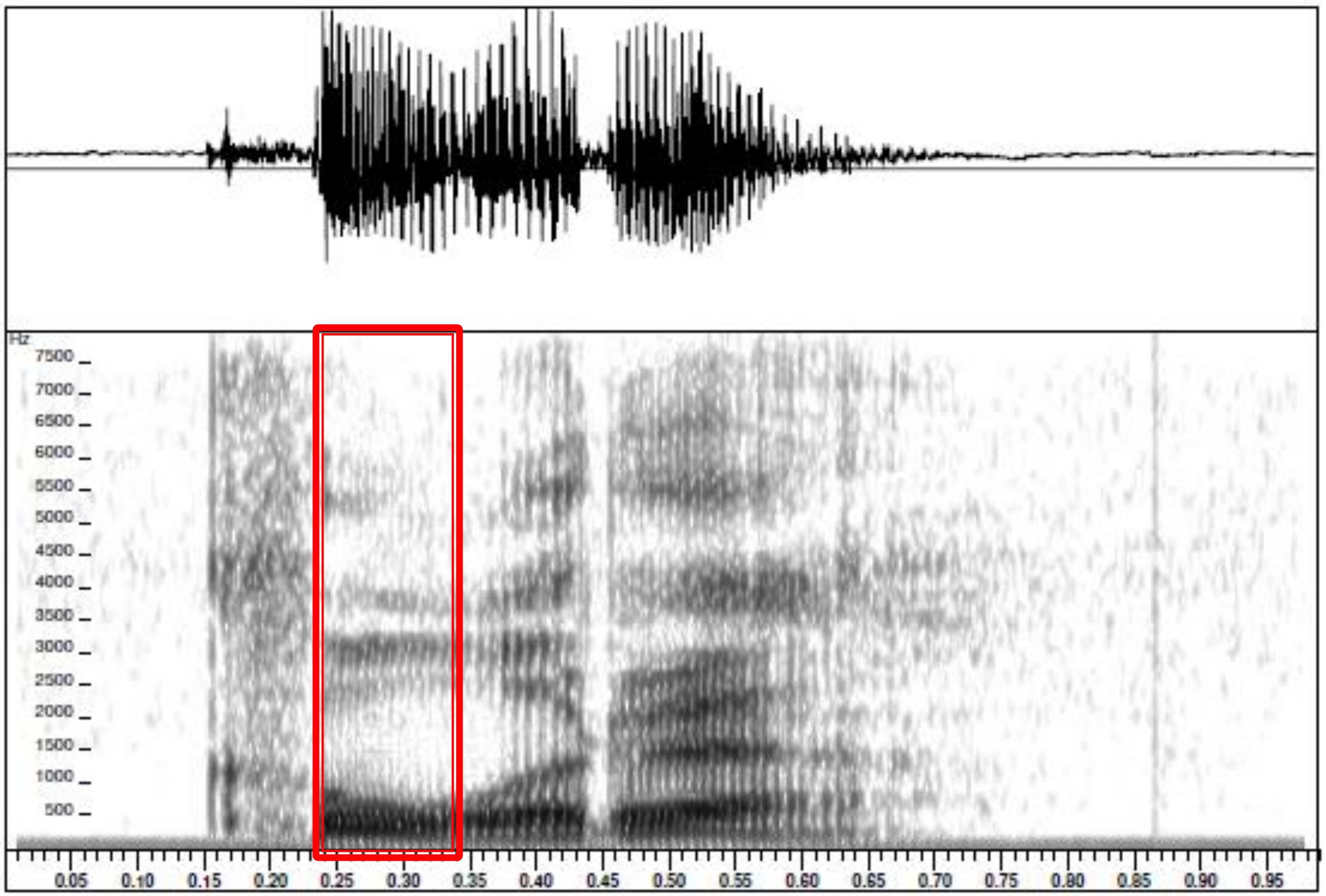
Feature maps



CNN as feature extraction

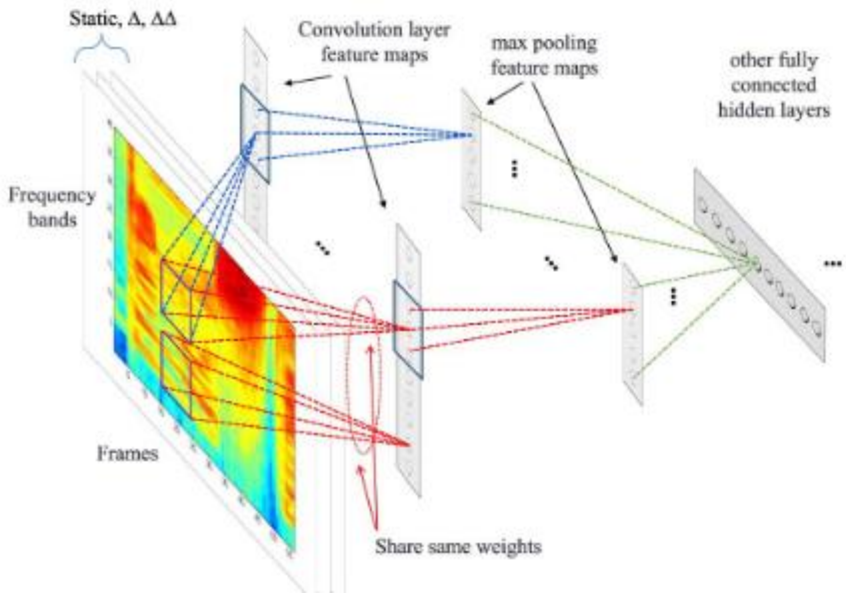


The spectrogram



CNN in speech recognition

- Treat the spectrogram as an image
- Convolving along the time axis and the frequency axis



Reading list

- Speech and Language Processing, version 3
  - Chapter 5 and 7