Week 3 (Cont.)
Artificial Neural Networks: A very brief introduction

Nhung Nguyen
slides courtesy of Phong Le

Machine learning (supervised learning definition)

- Given an input x, find an output y
- Classification: y is one of N classes

Training: learn function y = f(x) given *n* examples $D_{train} = \{(x_1,y_1),\ldots,(x_n,y_n)\}$

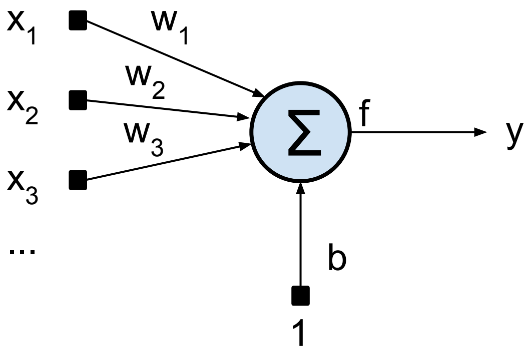
Modelling: f(x;θ) is often parameterized (e.g., a neural network)

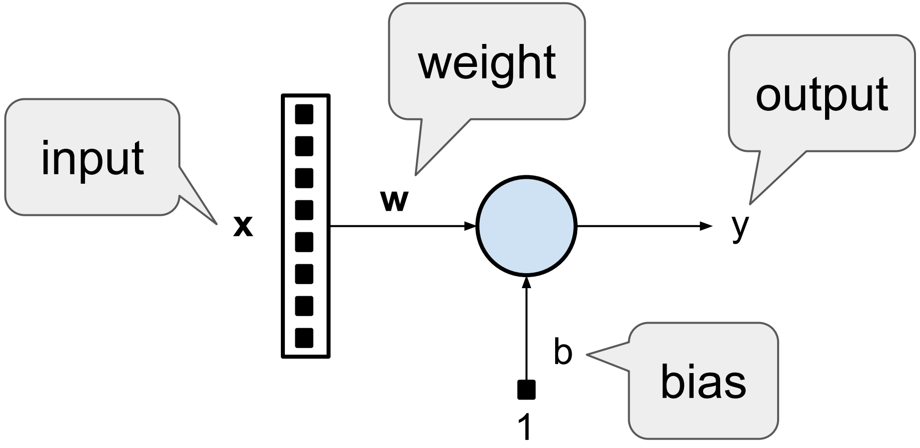
Testing: f("I'm a big fan of Tom Hanks";θ) = ?

x (input)	y (output)
I love the movie.	1 (positive)
The movie is horrible.	0 (negative)
The main actor is awesome.	1
Don't watch this movie.	0
...	...

Neurons

An artificial neuron is a computation unit which is loosely inspired by a biological neuron





$$z = \sum_{i=1}^{n} w_i x_i + b = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b$$

$$y = f(z)$$

$$z = \mathbf{w}^T \mathbf{x} + b \ ; \quad y = f(z)$$

Matrix-vector form (widely used in NN research)

Activation function *f*

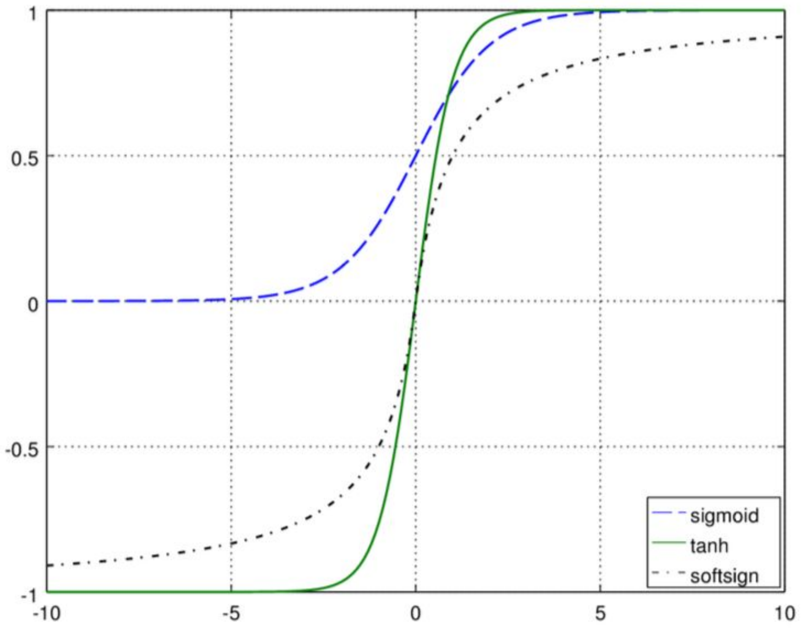
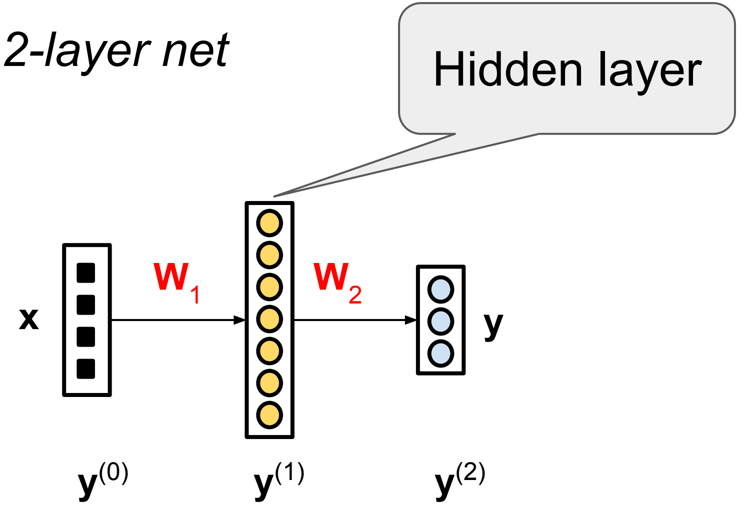


Figure 2.2: Activation functions: $\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$, $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$, $\mathrm{softsign}(x) = \frac{x}{1+|x|}$.

Multi-layer Neural Networks



A feed-forward neural network

Weight matrices (parameters to learn)

$$\mathbf{y}^{(0)} = \mathbf{x}$$

$$\mathbf{z}^{(1)} = \boxed{\mathbf{W}_1}\mathbf{y}^{(0)} + \mathbf{b}_1 \ ; \quad \mathbf{y}^{(1)} = f(\mathbf{z}^{(1)})$$

$$\mathbf{z}^{(2)} = \boxed{\mathbf{W}_2}\mathbf{y}^{(1)} + \mathbf{b}_2 \ ; \quad \mathbf{y}^{(2)} = f(\mathbf{z}^{(2)})$$

$$\mathbf{y} = \mathbf{y}^{(2)}$$

How many hidden layers do we need?

- At least one

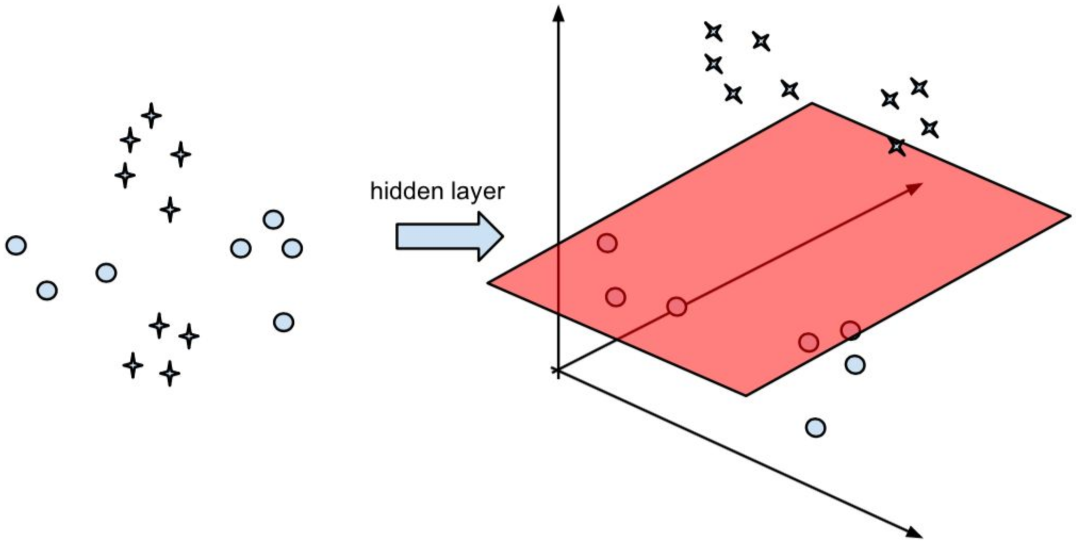


Figure 2.4: The role of the hidden layer in a two-layer feed-forward neural network is to project the data onto another vector space in which they are now linearly separable.

In theory, one hidden layer is enough: two-layer neural networks are *universal approximators* (they are capable of approximating any continuous function to any desired degree of accuracy).

How many hidden layers do we need? (cont.)

- In practice (deep learning), "deep" structures are prefered

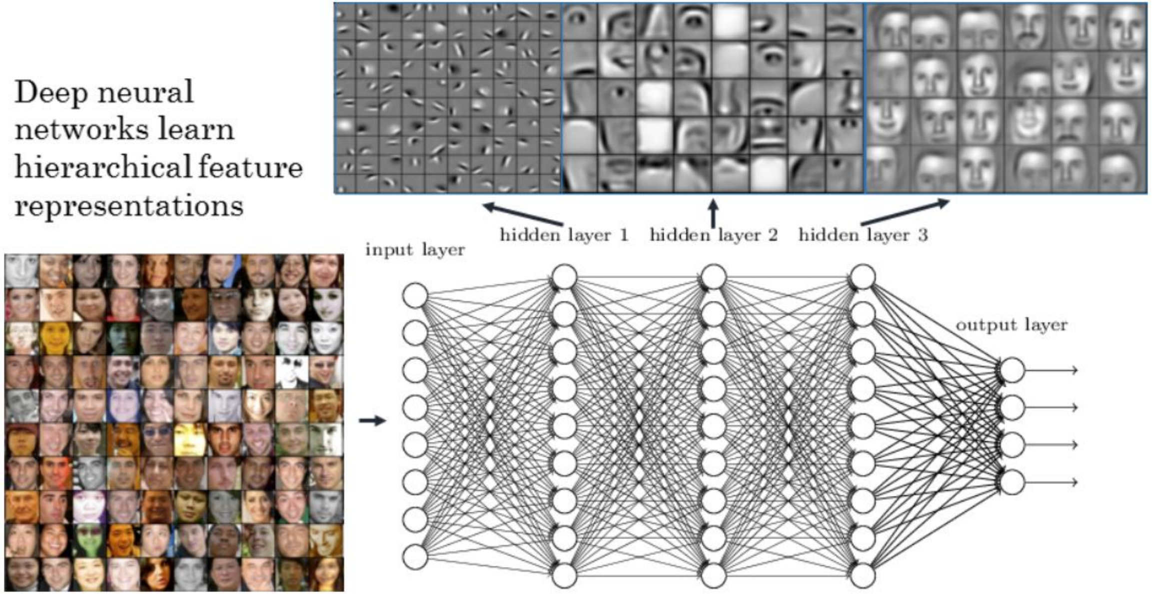
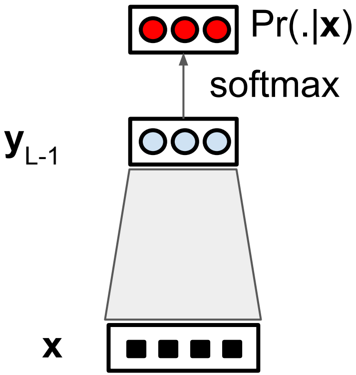


Figure 2.5: A four-layer neural network for face recognition. Higher layers (i.e. closer to the output layer) can extract more abstract features: the first hidden layer detects low level features such as edges and simple shapes; the second layer can identify more complex shapes like noses, eyes; and so on.

Classification task



- Assigning class c, one of N predefined classes $C=\{c_1,...,c_N\}$, to **x**
- Compute probability *Pr(c|**x**)* using *softmax*

$$Pr(c|\mathbf{x}) = \text{softmax}(c) = \frac{e^{u(c,\mathbf{y}_{L-1})}}{\sum_{c' \in C} e^{u(c',\mathbf{y}_{L-1})}}$$

$$[u(c_1, \mathbf{y}_{L-1}), ..., u(c_N, \mathbf{y}_{L-1})]^T = \mathbf{W}\mathbf{y}_{L-1} + \mathbf{b}$$

- If c_{true} is the correct class of **x**, we want $Pr(c_{true}|\mathbf{x})$ the highest.

Training a neural net (for classification)

- Training set: $D_{train} = \{(\mathbf{x}_1, c_1), ..., (\mathbf{x}_n, c_n)\}$, a set of pair $\mathbf{x}$ and its correct class c
- Our neural network has parameters θ (the set of all the weight matrices)
- Minimize the cross-entropy loss (i.e., negative log-likelihood)

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} -\log y_{i,c_i} + \frac{\lambda}{2}||\theta||_2^2 \equiv -\underbrace{\frac{1}{n}\sum_{i=1}^{n}\log Pr(c_i|\mathbf{x}_i)}_{\text{log likelihood}} + \underbrace{\frac{\lambda}{2}||\theta||_2^2}_{L_2 \text{ regularization}}$$

(Minibatch) Stochastic gradient descent

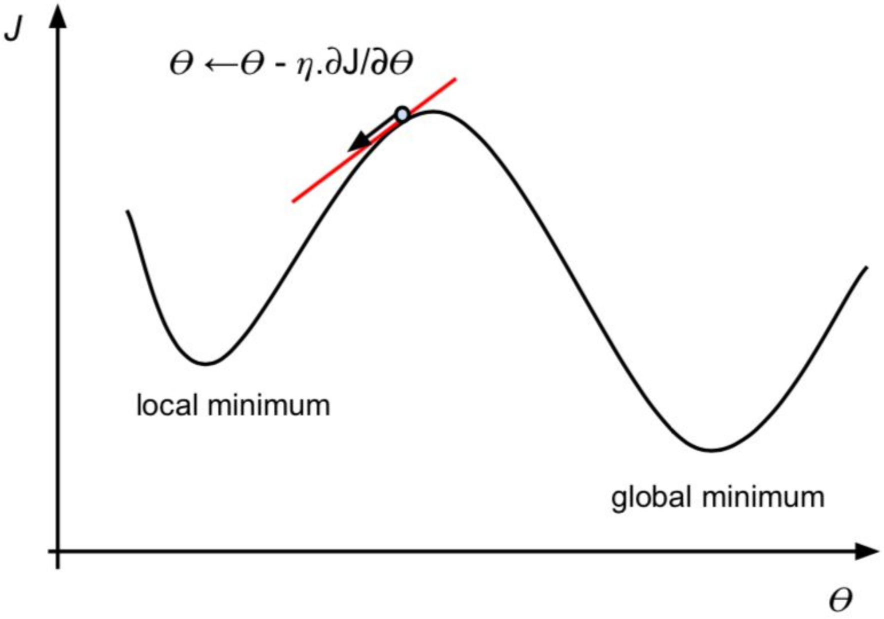


Figure 2.6: The gradient descent method. We iteratively update θ by adding to it an amount of $-\eta\frac{\partial J}{\partial \theta}$ until J converges. In this way, J gets smaller and smaller until it reaches a local minimum. This is similar to rolling a ball downhill.

1. sample a minibatch D′ of *m* examples from the training set D,

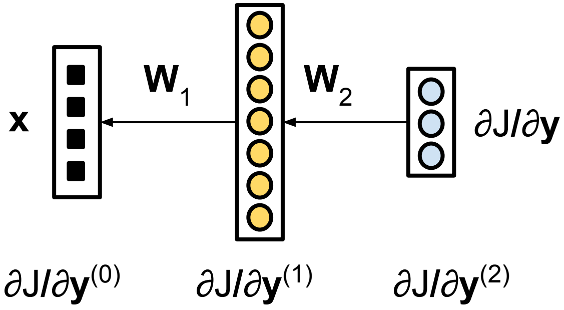
2. compute J(θ) on D′,

3. update the parameters:
$\theta \leftarrow \theta - \eta \partial J / \partial \theta$

4. if stopping criteria are not met, jump to step 1.

Computing gradients: error back-propagation

- Key: the chain rule $\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}}$



$$\frac{\partial J}{\partial \mathbf{y}^{(2)}} = \frac{\partial J}{\partial \mathbf{y}}$$

$$\frac{\partial J}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \frac{\partial J}{\partial \mathbf{y}^{(2)}} \; ; \quad \frac{\partial J}{\partial \mathbf{W}_2} = \frac{\partial J}{\partial \mathbf{z}^{(2)}} \mathbf{y}^{(1)T} \; ; \quad \frac{\partial J}{\partial \mathbf{y}^{(1)}} = \mathbf{W}_2^T \frac{\partial J}{\partial \mathbf{z}^{(2)}}$$

$$\frac{\partial J}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial J}{\partial \mathbf{y}^{(1)}} \; ; \quad \frac{\partial J}{\partial \mathbf{W}_1} = \frac{\partial J}{\partial \mathbf{z}^{(1)}} \mathbf{y}^{(0)T} \; ; \quad \frac{\partial J}{\partial \mathbf{y}^{(0)}} = \mathbf{W}_1^T \frac{\partial J}{\partial \mathbf{z}^{(1)}}$$

$$\frac{\partial J}{\partial \mathbf{x}} = \frac{\partial J}{\partial \mathbf{y}^{(0)}}$$

Summary

Important parts of a neural network:

- Activation function
- Number of layers
- Training:
  - Loss function
  - Stochastic gradient descent

Further reading

- Chapter 7: Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd ed. draft). https://web.stanford.edu/~jurafsky/slp3/