Machine Learning

CSE 142

Xin (Eric) Wang

Monday, November 29, 2021

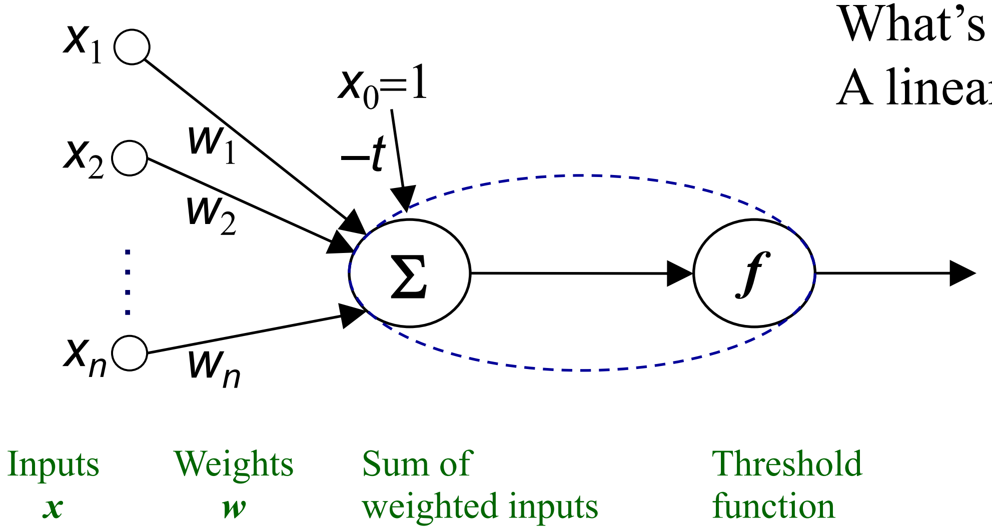
Today

- Neural networks and deep learning (cont.)

Notes

- HW#4 due Wednesday
  - Ask questions on Piazza / come to our office hours for help
- Final exam (Thursday, December 9, 4-6pm, here)
  - Similar to the midterm in style – important to understand the concepts we've covered and how to apply them
  - Covers all material of the course (lectures, reading, DS, midterm, HWs)
    - More weight on the material since the midterm (about 1/3 vs. 2/3)
  - Good practice: midterm, homework
    - Practice exam will be posted later this week
  - Open book
    - No internet search; no earphones; no talking; no keyboard typing.
    - I'll also provide some information, formulas, etc.

A simple neural network



What's this neural network called?
A linear perceptron!

$$o = f\left(\sum_{i=0}^{n} w_i x_i\right) = \begin{cases} 1 & \text{if} > 0 \\ -1 & \text{else} \end{cases}$$

$-t$: threshold value or bias

$$\underbrace{\left(\sum_{i=0}^{n} w_i x_i\right)}_{\text{Homogeneous}} = \underbrace{\left(\sum_{i=1}^{n} w_i x_i\right) - t = w^T x - t}_{\text{Non-homogeneous}}$$

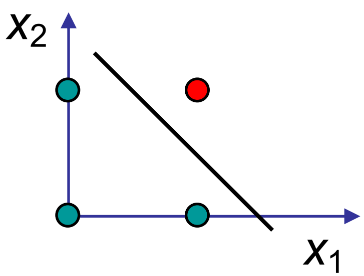
f : activation function – may be a thresholding unit (binary output):

$$f(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{otherwise} \end{cases} \quad \text{or} \quad f(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

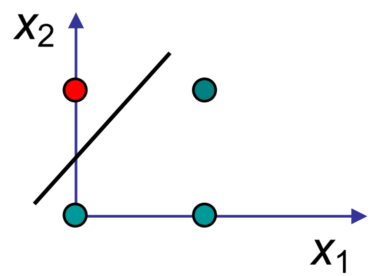
What can be decided by a perceptron?

- The decision surface is a hyperplane given by $\sum_{i=0}^{n} w_i x_i = 0$
  - 2D case: the decision surface is a line
  - 3D case: the decision surface is a plane
  - N-D case: the decision surface is an (N-1)D hyperplane
- Represents many useful functions: for example:

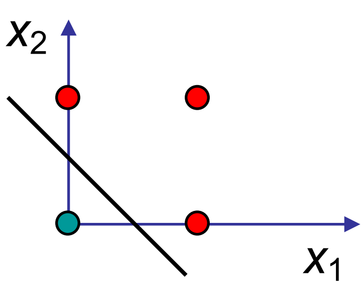
$x_1 \wedge x_2$



$\neg x_1 \wedge x_2$

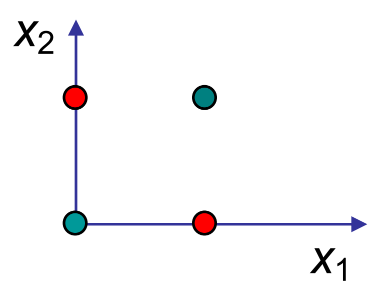


$x_1 \vee x_2$



x_1 XOR x_2

NO!

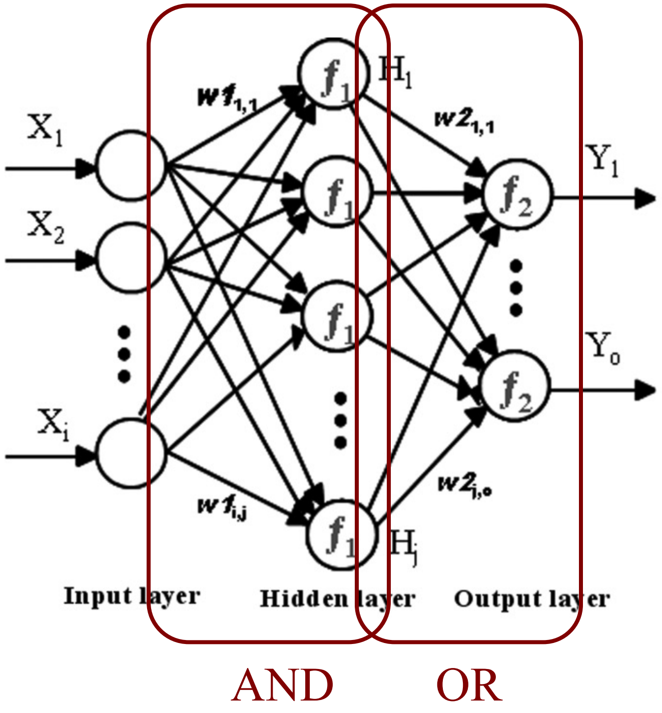


Implementing general Boolean functions

- Solution:
  - A network of perceptrons
  - Any Boolean function representable as disjunctive normal form (DNF)
    - 2 layers
    - Disjunction (layer 2) of conjunctions (layer 1)

- Example of XOR in DNF

$$x_1 \text{ XOR } x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$



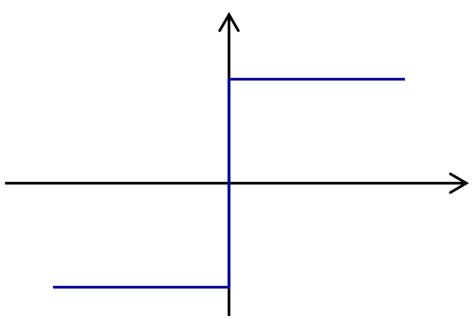
Network output

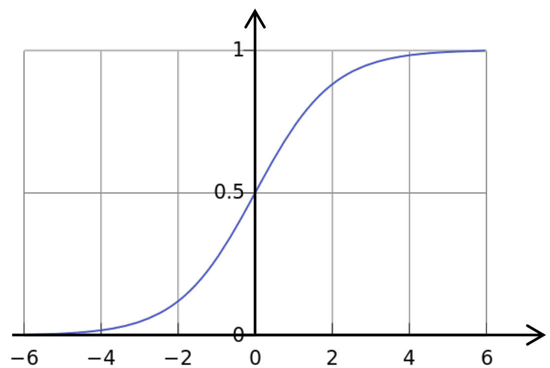
- Unlike the perceptron, most neural networks output one or more weights (rather than a binary classification)
- So we replace the thresholding unit in the perceptron with the sigmoid (or logistic) function $\sigma(x)$ or the $\tanh(x)$ function

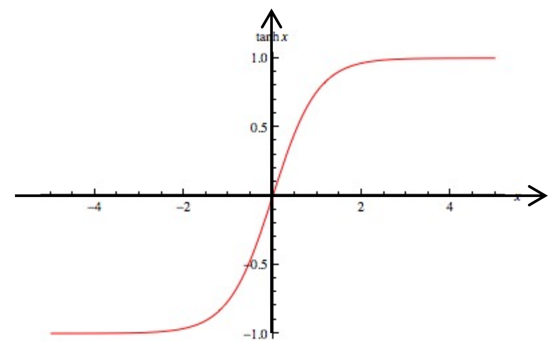
$$f(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma(x) = \tanh(x)$$







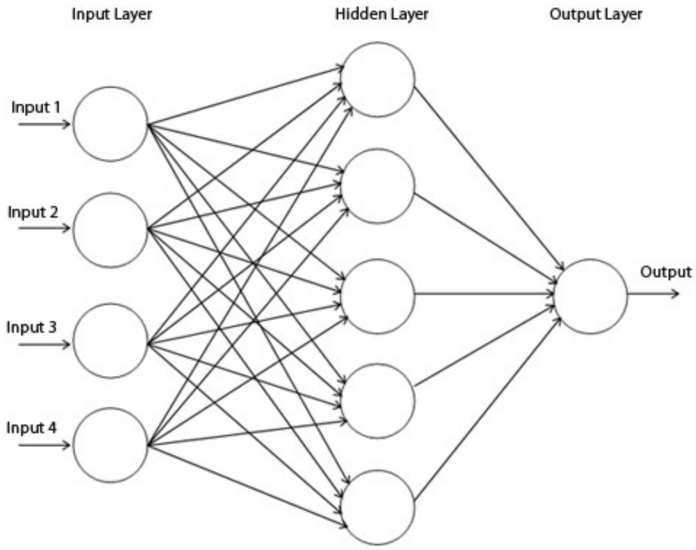
$f(x) \in \{-1, 1\}$

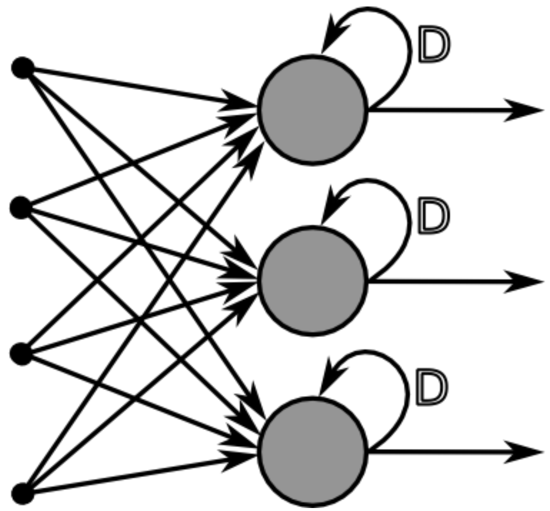
$f(x) \in (0, 1)$

$f(x) \in (-1, 1)$

- Nonlinear
- Differentiable

Neural networks





Feedforward network

- Information only moves forward, from input to output
- No cycles in the graph
- A.k.a. multi-layer perceptron

Recurrent network

- Directed cycles exist in the network/graph

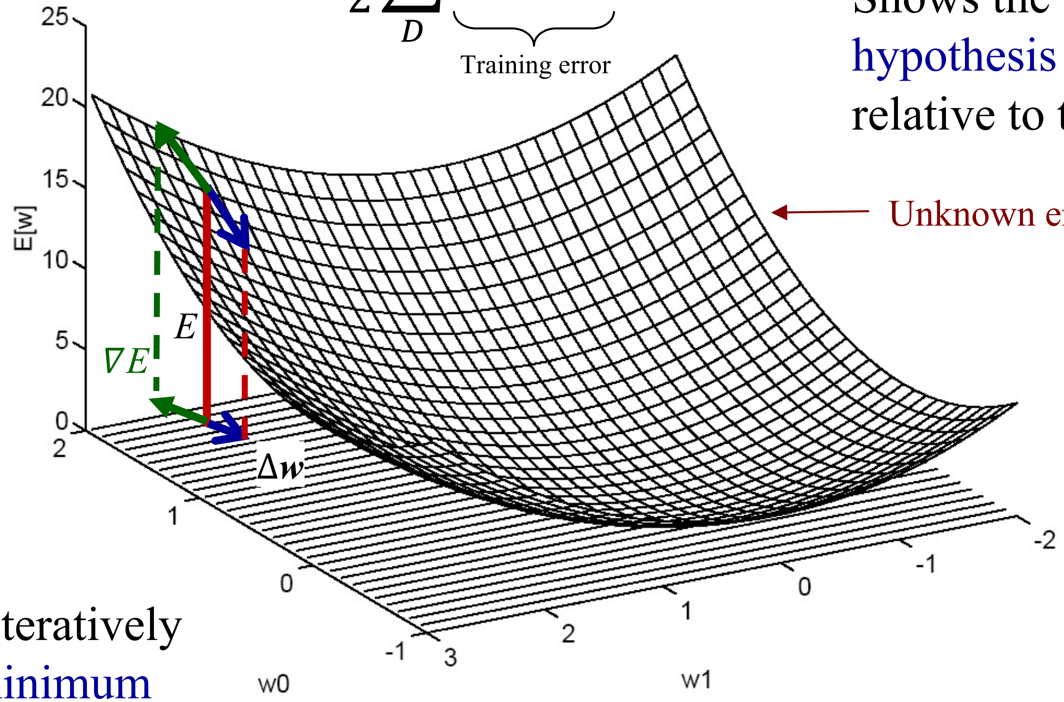
Typical neural network learning

- The output (target) can be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes
- Training data: attribute-value pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$
  - E.g., for ALVINN, $\boldsymbol{x}_i$ is the input (30x32) image, y_i is the steering direction
- The training data may contain errors (i.e., noisy)
- Long training time, fast execution (evaluation) time
  - E.g., real-time steering response for ALVINN
- In training, use gradient descent to search the hypothesis space of possible weight vectors to find the $\boldsymbol{w}$ that best fits the training examples
  - So a neural network hypothesis is a set of weights $\boldsymbol{w}$

The hypothesis space and gradient descent

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{D}\underbrace{(y_i - o_i)^2}_{\text{Training error}}$$

Training error $E(\boldsymbol{w})$



Shows the error E of the hypothesis $\boldsymbol{w} = (w_0, w_1)$ relative to the training data

Unknown error function $E(\boldsymbol{w})$

Hypothesis/weight space $\boldsymbol{w}$

Gradient descent iteratively searches for the minimum error over the complete training data by moving in the direction $(\delta w_0, \delta w_1)$, at each step, that most reduces the error.

Think globally, act locally!

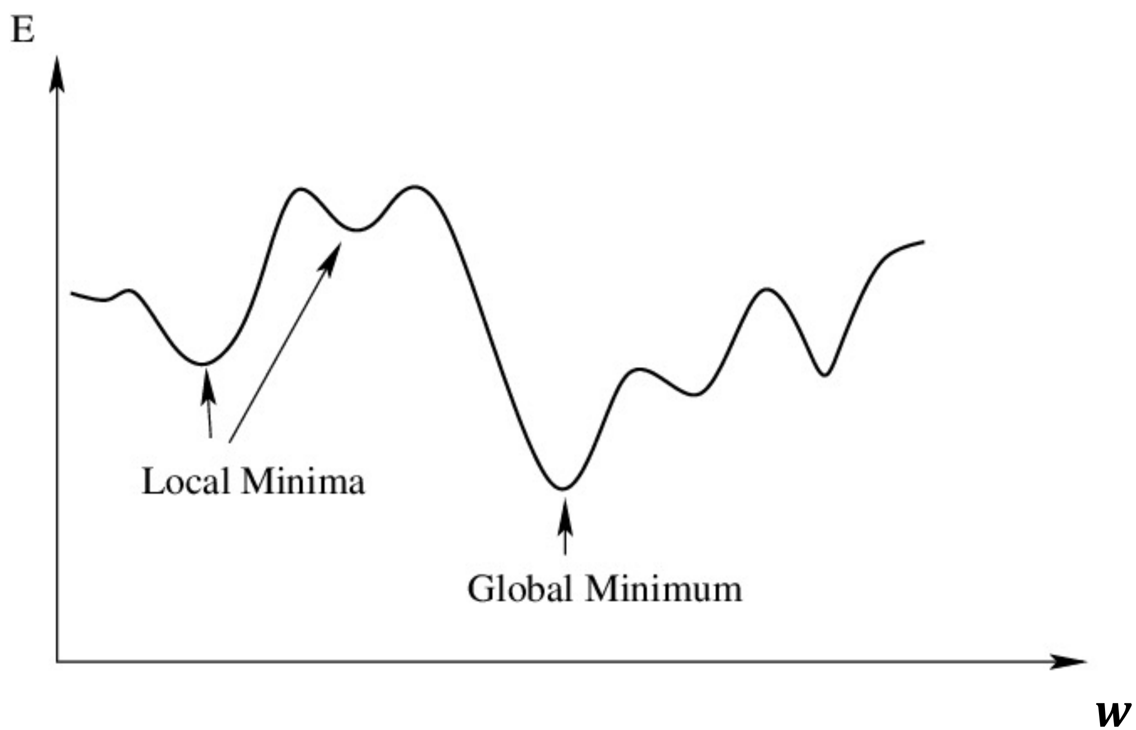
So $\boldsymbol{w} = \boldsymbol{w} + \Delta\boldsymbol{w}$

$$\Delta\boldsymbol{w} = -\eta\nabla E(\boldsymbol{w})$$

where $\nabla E(\boldsymbol{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right)$ Gradient of E with respect to $\boldsymbol{w}$

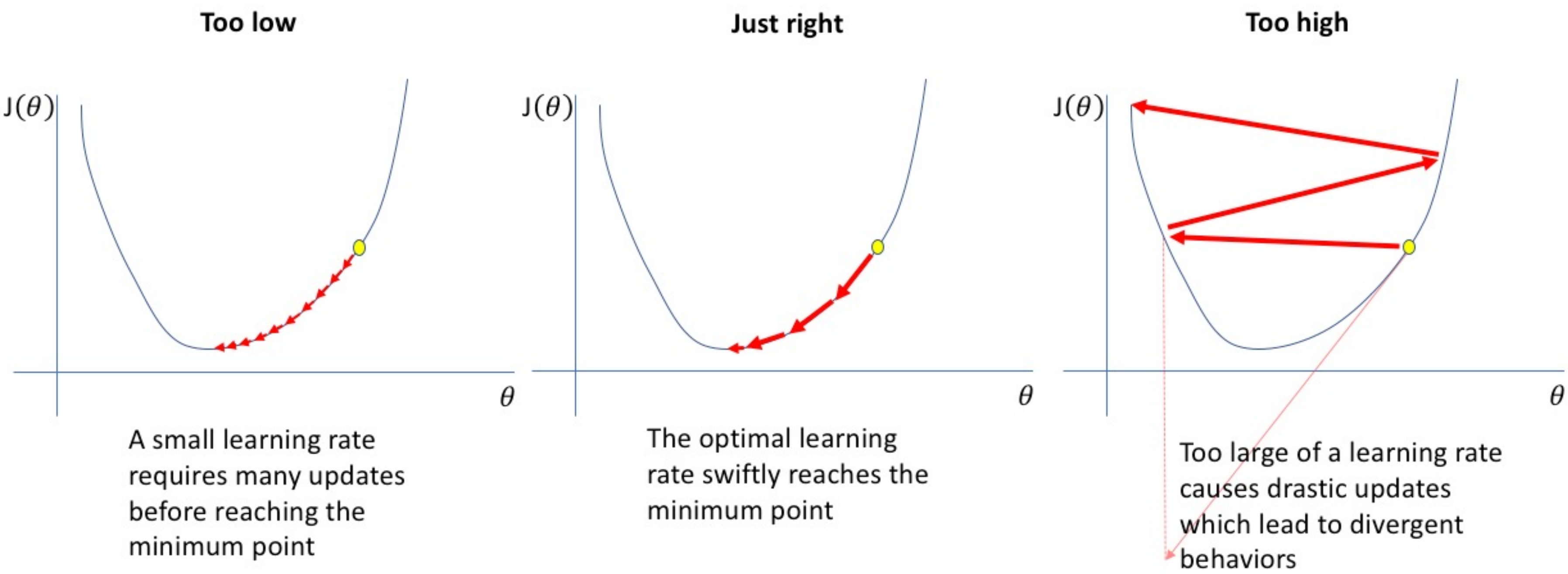
Global minimum vs. local minimum

- Gradient descent is not guaranteed to reach global minimum
- Initialization is important
  - E.g., initializing NN demo: https://www.deeplearning.ai/ai-notes/initialization/



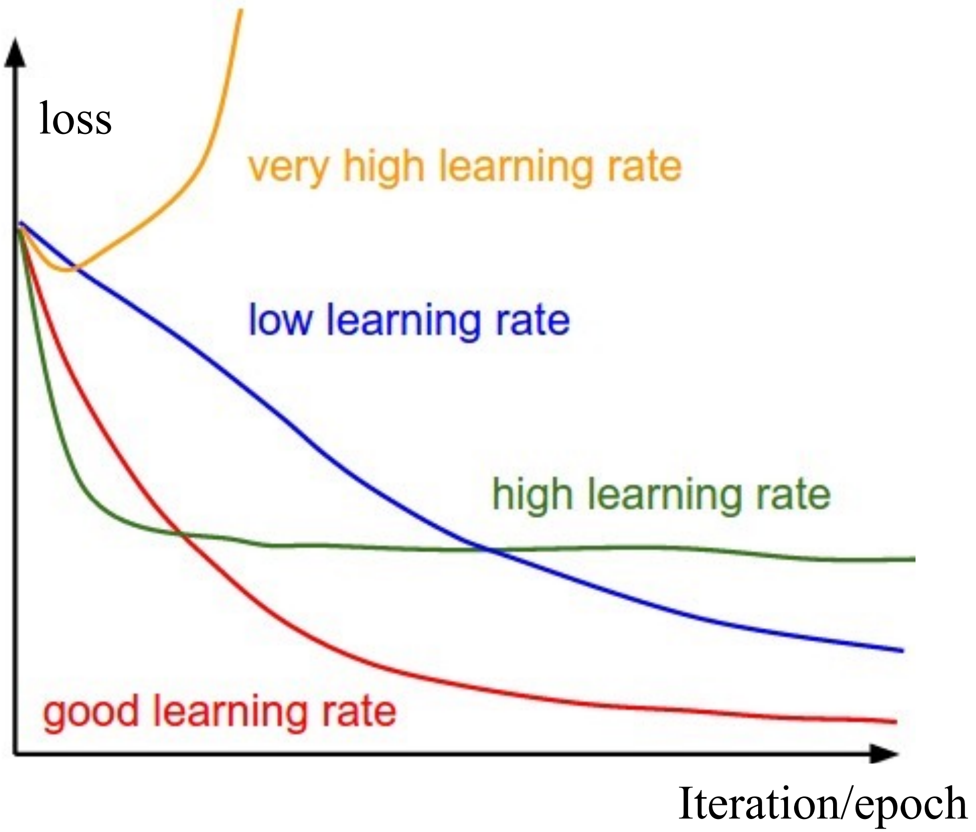
Importance of learning rate

- Learning rate determines how big the gradient descent steps are towards the minimum



Importance of learning rate

- Learning rate determines how big the gradient descent steps are towards the minimum
- Good practice: plot the loss function as the optimization runs



Gradient descent

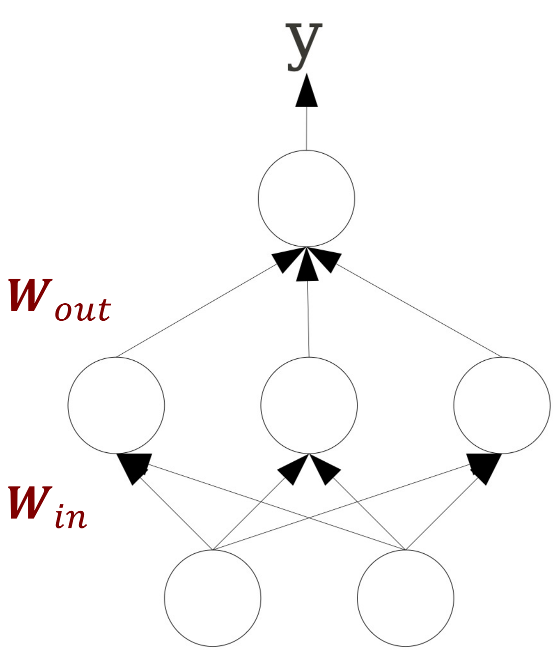
- Gradient descent is an important general paradigm for learning
- Can be applied whenever
  - The hypothesis space contains continuously parameterized hypotheses
    - E.g., the weights in a linear unit
  - The error on training data can be computed with respect to these hypotheses
- This will converge to a solution even with noisy, non-separable training data
- Practical difficulties in applying gradient descent:
  - Convergence can be slow (e.g., can require thousands of steps)
  - Converges to a local minimum – no global guarantee
- A gradient descent algorithm might update the weights after each training sample or after each complete epoch

Quiz: gradient descent

- See Canvas.

Backpropagation

- The backpropagation algorithm learns weights for a multilayer network
- Use gradient descent and chain rule to minimize the training loss



No activation functions:

$$\boldsymbol{y} = \boldsymbol{W}_{out}\boldsymbol{W}_{in}\boldsymbol{x}$$

sigmoid on the hidden layer and the output layer:

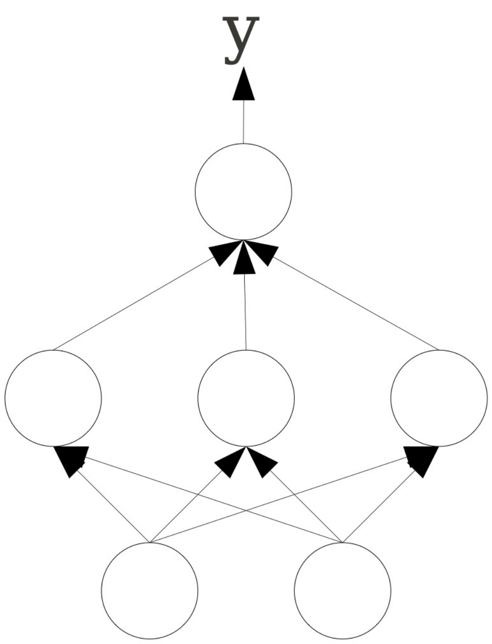
$$\boldsymbol{y} = S(\boldsymbol{W}_{out}\, S(\boldsymbol{W}_{in}\boldsymbol{x}))$$

Backprop trains the network by iteratively propagating errors backwards from output units

How to compute gradients?

- Use gradient descent to minimize the squared loss between the target values and the network output values:

$$E = \frac{1}{2}\sum_{p}(d^{p} - y^{p})^{2}$$



$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$