Neural Networks - Part 2

Wenhu Chen

Lecture 7

Outline

Learning Goals

- Explain the steps of the gradient descent algorithm.
- Explain how we can modify gradient descent to speed up learning and ensure convergence.
- Describe the back-propagation algorithm including the forward and backward passes.
- Compute the gradient for a weight in a multi-layer feed-forward neural network.
- Describe situations in which it is appropriate to use a neural network or a decision tree.

Gradient Descent

Method to find local optima of differentiable a function f

- Intuition: gradient tells us direction of greatest increase, negative gradient gives us direction of greatest decrease
- Take steps in directions that reduce the function value
- Definition of derivative guarantees that if we take a small enough step in the direction of the negative gradient, the function will decrease in value
- How small is small enough?

Gradient Descent

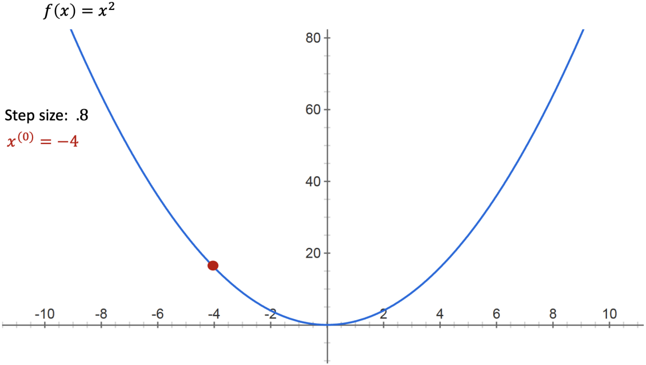
Gradient Descent Algorithm:

- Pick an initial point x_0
- Iterate until convergence

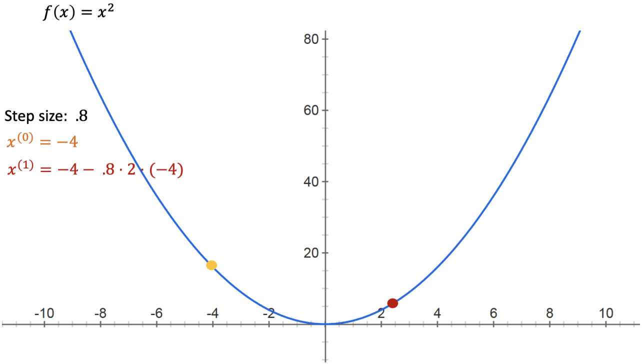
$$x_{t+1} = x_t - \gamma_t \Delta f(x_t) \tag{1}$$

where γ_t is the t^{th} step size.

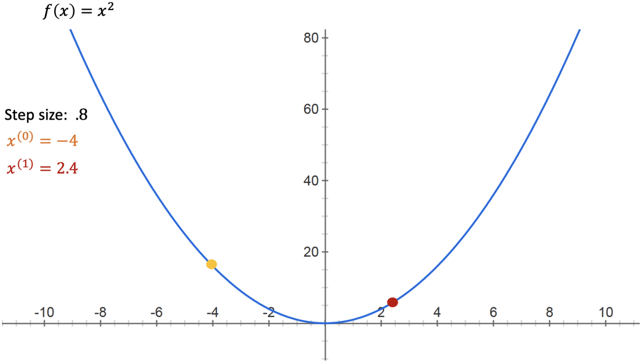
Gradient Descent Example



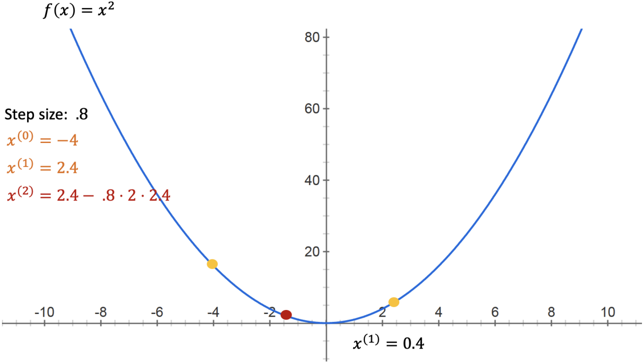
Gradient Descent Example



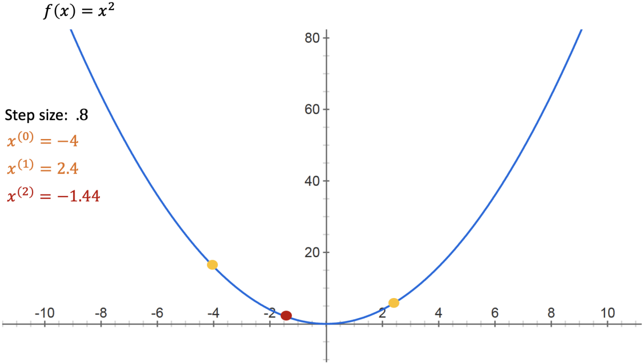
Gradient Descent Example



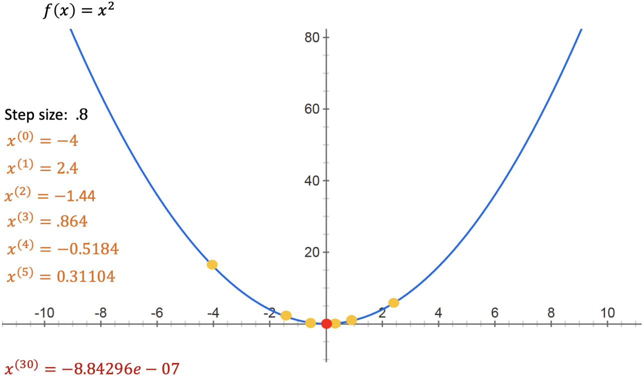
Gradient Descent Example



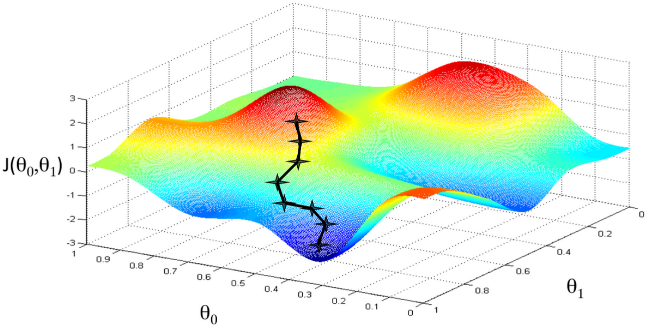
Gradient Descent Example



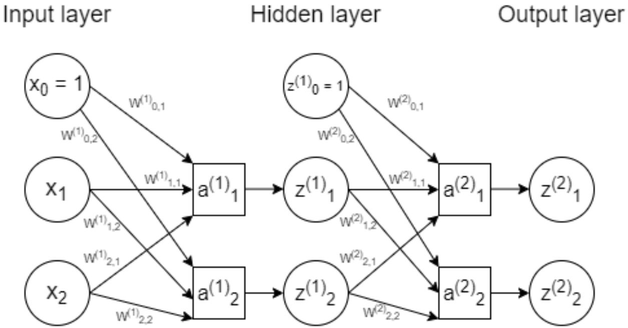
Gradient Descent Example



High-dimensional Gradient Descent Example



A 2-Layer Neural Network



A 2-Layer Neural Network

Assuming that we want the output of the 2-Layer neural network to be close to certain target value.

Let's assume we are doing spam classification:

The input x_1 and x_2 are two features: the email length x_1 and whether the email is coming from a trusted organization x_2.

We have paired training data, $x_1, x_2, y = \{0, 1\}$.

Therefore, we can feed x_1 and x_2 to the neural network to obtain its output $a_1^{(2)}$ and $a_2^{(2)}$.

Neural Network Approximation

Let's assume that $a_1^{(2)}$ denotes how likely the email is a spam and $a_2^{(2)}$ denotes how unlikely the email is a spam.

- If an input email is a spam, the desired output should be $[a_1^{(2)}, a_2^{(2)}] = [1, 0]$.
- If an input email is not a spam, the desired output should be $[a_1^{(2)}, a_2^{(2)}] = [0, 1]$.
- If an input email is indistinguishable, the desired output should be $[a_1^{(2)}, a_2^{(2)}] = [0.5, 0.5]$.

Measuring the Loss Function

Let's assume we want to measure the discrepancy between neural network output and the reference label. The discrepancy is also called loss function E. For example, we can have square difference loss as follows:

$$E = \sum_i (a_i^{(2)} - y_i)^2$$

We will be using E as the training signal to perform gradient descent.

Gradient Descent

"Walking downhill and always taking a step in the direction that goes down the most."

- A local search algorithm to find the minimum of a function.
- Steps of the algorithm:
  - Initialize weights randomly.
  - Change each weight in proportion to the negative of the partial derivative of the error with respect to the weight.

$$W := W - \eta \frac{\partial E}{\partial W}$$

  - η is the learning rate.
  - Terminate after some number of steps, when the error is small, or when the changes get small.

Why update the weight proportional to the negative of the partial derivative?

- Suppose that we want to find the minimum of $y = \mathbf{x}^T \cdot \mathbf{x}$.

  $\rightarrow$ Think of x as the weight and y as the error.
- Start with $\mathbf{x} = \mathbf{x}_0$.
- $\frac{\partial y}{\partial \mathbf{x}} = 2\mathbf{x}$
- In what direction should we change the value of $\mathbf{x}$?

Why update the weight proportional to the negative of the partial derivative?

- Suppose that we want to find the minimum of $y = \mathbf{x}^T \cdot \mathbf{x}$.

  $\rightarrow$ Think of x as the weight and y as the error.

- Start with $\mathbf{x} = \mathbf{x}_0$.
- $\frac{\partial y}{\partial \mathbf{x}} = 2\mathbf{x}$
- In what direction should we change the value of $\mathbf{x}$?

  $\rightarrow$ If the gradient is positive, we want to decrease x_0. If the gradient is negative, we want to increase x_0.

  We want to move in the direction of the negative of the gradient.

Why update the weight proportional to the negative of the partial derivative?

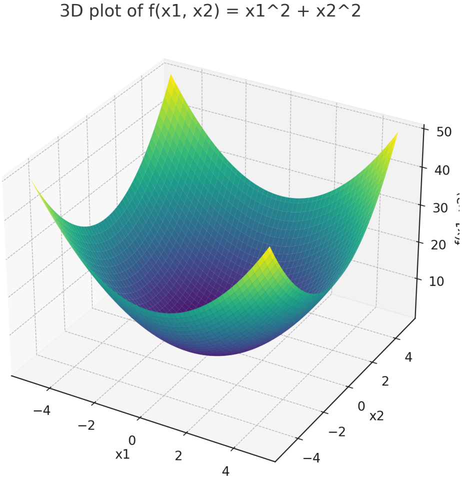
- By what amount should we change the value of **x**?
  What is the step size?

  → If the gradient is large, the curve is steep and we are likely far from the minimum. We can afford to take a larger step. If the gradient is small, the curve is flat and we are likely close to the minimum. We want to take a smaller step.

  Take a step proportional to the gradient.

Visualization

when $[x_1, x_2]$ are larger, there gradient $\frac{\partial y}{\partial \mathbf{x}}$ also gets larger.



How do we update the weights based on the data points?

- Gradient descent updates the weights after sweeping through all the examples.
- To speed up learning, update weights after each example.
  - **Incremental gradient descent** $\rightarrow$ update weights after each example.
  - **Stochastic gradient descent** $\rightarrow$ same as incremental version except each example is chosen randomly.

    $\rightarrow$ With cheaper steps, weights become more accurate more quickly, but not guaranteed to converge as individual examples can move the weights away from the minimum.

How do we update the weights based on the data points?

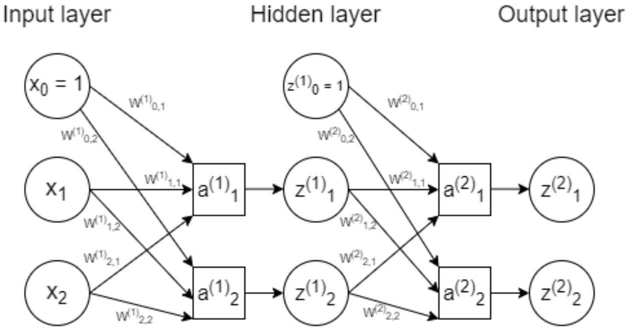
- Trade off learning speed and convergence.
  - **Batched gradient descent**

    $\rightarrow$ update weights after a batch of examples.

    batch $=$ all the examples $\longrightarrow$ gradient descent.

    batch $=$ one example $\longrightarrow$ incremental gradient descent.

A 2-Layer Neural Network



Let $\hat{y}$ be the output of a network (i.e. prediction).
For this network, $\hat{y} = z^{(2)}$

The Backpropagation Algorithm

- An efficient method of calculating the gradients in a multi-layer neural network.
- Given training examples $(\vec{x}_n, \vec{y}_n)$ and an error/loss function $E(\hat{y}, y)$. Perform 2 passes.
  - **Forward pass:** compute the error E given the inputs and the weights.
  - **Backward pass:** compute the gradients $\dfrac{\partial E}{\partial W^{(2)}_{j,k}}$ and $\dfrac{\partial E}{\partial W^{(1)}_{i,j}}$.
- Update each weight by the sum of the partial derivatives for all the training examples.

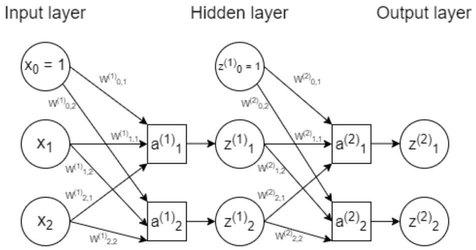
Forward Pass for a 2-layer Network

Calculate the values of $z_j^{(1)}$ and $z_k^{(2)}$ and E.

$$a_j^{(1)} = \sum_i x_i W_{i,j}^{(1)} \qquad z_j^{(1)} = g(a_j^{(1)}) \tag{2}$$

$$a_k^{(2)} = \sum_j z_j^{(1)} W_{j,k}^{(2)} \qquad z_k^{(2)} = g(a_k^{(2)}) \tag{3}$$

$$E(z^{(2)}, y) \tag{4}$$

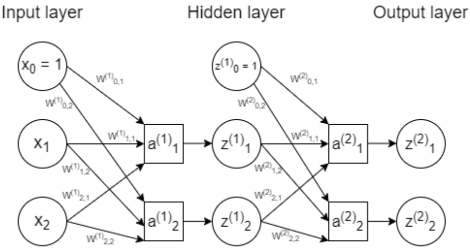


Backward Pass for a 2-layer Network

Calculate the gradients for $W_{i,j}^{(1)}$ and $W_{j,k}^{(2)}$.

$$\frac{\partial E}{\partial W_{j,k}^{(2)}} = \frac{\partial E}{\partial a_k^{(2)}} z_j^{(1)} = \delta_k^{(2)} z_j^{(1)}, \quad \delta_k^{(2)} = \frac{\partial E}{\partial z_k^{(2)}} g'(a_k^{(2)}) \tag{5}$$

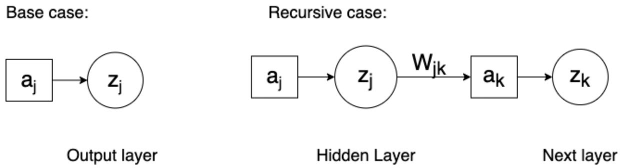
$$\frac{\partial E}{\partial W_{i,j}^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}} x_i = \delta_j^{(1)} x_i, \qquad \delta_j^{(1)} = \left(\sum_k \delta_k^{(2)} W_{j,k}^{(2)} \right) g'(a_j^{(1)}) \tag{6}$$



The recursive relationship

For unit j of layer ℓ, $\delta_j^{(\ell)} = \dfrac{\partial E}{\partial a_j^{(\ell)}}$.

$$\delta_j^{(\ell)} = \begin{cases} \dfrac{\partial E}{\partial z_j^{(\ell)}} \times g'(a_j^{(\ell)}), & \text{base case, } j \text{ is an output unit} \\ \left(\displaystyle\sum_k \delta_k^{(\ell+1)} W_{j,k}^{(\ell+1)} \right) \times g'(a_j^{(\ell)}), & \text{recursive case, } j \text{ is a hidden unit} \end{cases} \tag{7}$$



A concrete example of forward and backward pass

Calculate $W^{(2)}_{j,k}$ and $W^{(1)}_{i,j}$ given the information below.

- The error function is the sum of squares error.

$$E = \sum_k (\hat{y}_k - y_k)^2$$

- The activation function is the sigmoid function.

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

The derivative of $g(x)$

Sigmoid Function Derivative:

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

It means that during forward propagation, we can save the intermediate values of $g(x)$ to directly compute $\frac{\partial g(x)}{\partial x}$.

The recursive relationship

For the i-th layer output $x^{(i)}$:

$$\frac{\partial g(x^{(i)})}{\partial x^{(i)}} =$$

$$\begin{pmatrix} g(x_1^{(i)})(1-g(x_1^{(i)})) & 0 & \cdots & 0 \\ 0 & g(x_2^{(i)})(1-g(x_2^{(i)})) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g(x_d^{(i)})(1-g(x_d^{(i)})) \end{pmatrix}$$

where j indexes the j-th element in the i-th vector $g(x_j^{(i)})$.

The recursive relationship

At i-th layer, assuming there are d neurons:

Through backward propagation, the derivative w.r.t to $g(x_i)$ is denoted as $\delta_i = \frac{\partial E}{\partial g(x^{(i)})} \in \mathbb{R}^d$.

$$\delta_{i-1} = \frac{\partial E}{\partial g(x^{i-1})} = \frac{\partial E}{\partial g(x^{(i)})} \odot \frac{\partial g(x^{(i)})}{\partial x^{(i)}} \cdot \frac{\partial x^{(i)}}{\partial g(x^{(i-1)})}$$

According to definition: $\frac{\partial x^{(i)}}{\partial g(x^{(i-1)})} = W_i \in \mathbb{R}^{d \times d'}$, where d' is the number of neurons in $i-1$-th layer.

Therefore, we can conclude:

$$\delta_{i-1} = \delta_i \odot \frac{\partial g(x^{(i)})}{\partial x^{(i)}} \cdot W_i$$

where $\delta_{i-1} \in \mathbb{R}^{d'}$

The recursive relationship

Backward Propagation Algorithm:

- Initialize W_i for all the layers.
- Feedforward x into neural network and save intermediate values $g(x^{(1)}), g(x^{(2)}), \cdots$.
- Compute $\delta_n = \frac{\partial E}{\partial z}$.
- For i = n $\rightarrow$ 1; do
  - $\delta_{i-1} = \delta_i \odot \frac{\partial g(x^{(i)})}{\partial x^{(i)}} \cdot W_i$
  - Compute $\frac{\partial E}{\partial W_i} = \delta_i \odot \frac{\partial g(x^{(i)})}{\partial x^{(i)}} \otimes g\big(x^{(i-1)}\big)$
- Obtain all $\frac{\partial E}{\partial W_i}$ for gradient descent.

Revisiting Learning Goals

- Explain the steps of the gradient descent algorithm.
- Explain how we can modify gradient descent to speed up learning and ensure convergence.
- Describe the back-propagation algorithm including the forward and backward passes.
- Compute the gradient for a weight in a multi-layer feed-forward neural network.
- Describe situations in which it is appropriate to use a neural network or a decision tree.