Machine Learning 2007: Lecture 8

Instructor: Tim van Erven (Tim.van.Erven@cwi.nl)
Website: `www.cwi.nl/~erven/teaching/0708/ml/`

October 31, 2007

Overview

- **Organisational Matters**
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

Course Organisation

Final Exam:

- You have to enroll for the final exam on tisvu (when possible.)
- The final exam will be more difficult than the intermediate exam.

Mitchell:

- Read: Chapter 4, sections 4.1 – 4.4.

Course Organisation

Final Exam:

- You have to enroll for the final exam on tisvu (when possible.)
- The final exam will be more difficult than the intermediate exam.

Mitchell:

- Read: Chapter 4, sections 4.1 – 4.4.

This Lecture:

- Explanation of linear functions as inner products is needed to understand Mitchell.
- Neural networks are in Mitchell. I have some extra pictures.
- Convex functions are not discussed in Mitchell.
- I will give more background on gradient descent.

Overview

- Organisational Matters
- **Linear Functions as Inner Products**
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

Linear Functions as Inner Products

Linear Function:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

- $\mathbf{x} = (x_1, \ldots, x_d)^\top$ is a d-dimensional feature vector.
- $\mathbf{w} = (w_0, w_1, \ldots, w_d)^\top$ is a $d+1$-dimensional weight vector.

Linear Functions as Inner Products

Linear Function:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

- $\mathbf{x} = (x_1, \ldots, x_d)^\top$ is a d-dimensional feature vector.
- $\mathbf{w} = (w_0, w_1, \ldots, w_d)^\top$ is a $d+1$-dimensional weight vector.

As an Inner Product (a standard trick):

We may change $\mathbf{x}$ into a $d+1$-dimensional vector $\mathbf{x}'$ by adding an imaginary extra feature x_0, which always has value 1:

$$\mathbf{x} = (x_1, \ldots, x_d)^\top \quad \Rightarrow \quad \mathbf{x}' = (1, x_1, \ldots, x_d)^\top$$

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{d} w_i x'_i = \langle \mathbf{w}, \mathbf{x}' \rangle$$

Linear Functions as Inner Products

Linear Function:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

- $\mathbf{x} = (x_1, \ldots, x_d)^\top$ is a d-dimensional feature vector.
- $\mathbf{w} = (w_0, w_1, \ldots, w_d)^\top$ is a $d+1$-dimensional weight vector.

As an Inner Product (a standard trick):

We may change $\mathbf{x}$ into a $d+1$-dimensional vector $\mathbf{x}'$ by adding an imaginary extra feature x_0, which always has value 1:

$$\mathbf{x} = (x_1, \ldots, x_d)^\top \quad \Rightarrow \quad \mathbf{x}' = (1, x_1, \ldots, x_d)^\top$$

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{d} w_i x'_i = \langle \mathbf{w}, \mathbf{x}' \rangle$$

- Mitchell writes $\mathbf{w} \cdot \mathbf{x}'$ for $\langle \mathbf{w}, \mathbf{x}' \rangle$.

Overview

- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - **The Perceptron**
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

Artificial Neurons

An Artificial Neuron:

An (artificial) **neuron** is some function h that gets a feature vector $\mathbf{x}$ as input and outputs a (single) label y.

The Perceptron:

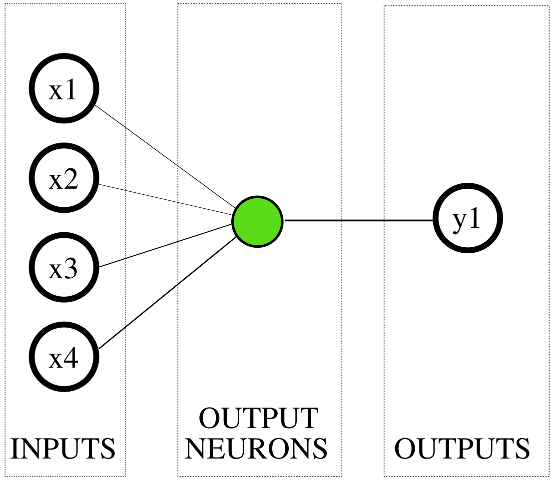
The most famous type of (artificial) neuron is the perceptron:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots w_d x_d > 0, \\ -1 & \text{otherwise.} \end{cases}$$

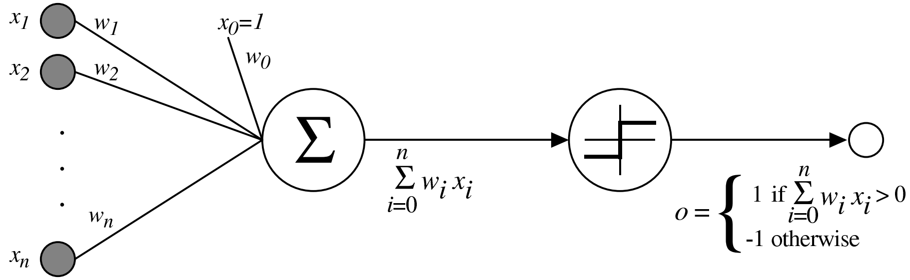
- Applies a threshold to a linear function of $\mathbf{x}$.
- Has parameters $\mathbf{w}$.

Different Views of The Perceptron

Simple Neural Network:



Mitchell's Drawing:

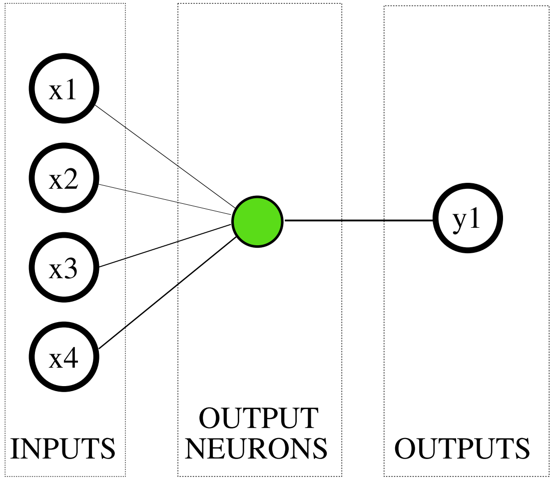


Equation:

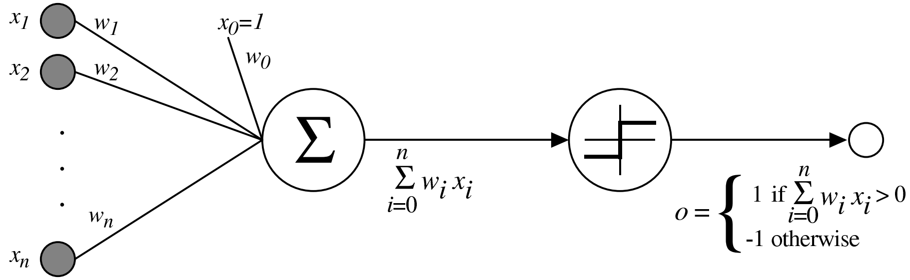
$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots w_d x_d > 0, \\ -1 & \text{otherwise.} \end{cases}$$

Different Views of The Perceptron

Simple Neural Network:



Mitchell's Drawing:



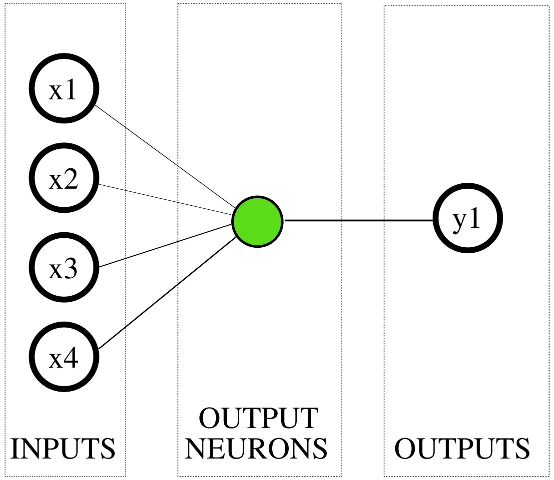
Equation:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots w_d x_d > 0, \\ -1 & \text{otherwise.} \end{cases}$$

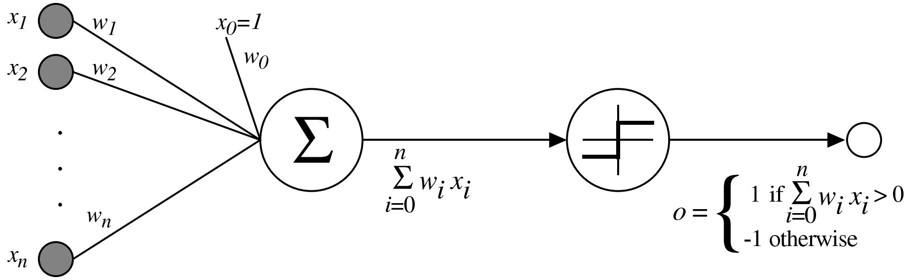
- One of the most simple neural networks consists of just one perceptron neuron.
- A perceptron does **classification**.

Different Views of The Perceptron

Simple Neural Network:



Mitchell's Drawing:



Equation:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots w_d x_d > 0, \\ -1 & \text{otherwise.} \end{cases}$$

- One of the most simple neural networks consists of just one perceptron neuron.
- A perceptron does **classification**.
- The network has no hidden units, and just one output.
- It may have any number of inputs.

Decision Boundary of the Perceptron

Decision boundary: $w_0 + w_1 x_1 + \ldots + w_d x_d = 0$

- This is where the perceptron changes its output y from -1 (-) to $+1$ (+) if we change $\mathbf{x}$ a little bit.
- For $d = 2$ this decision boundary is always a line.

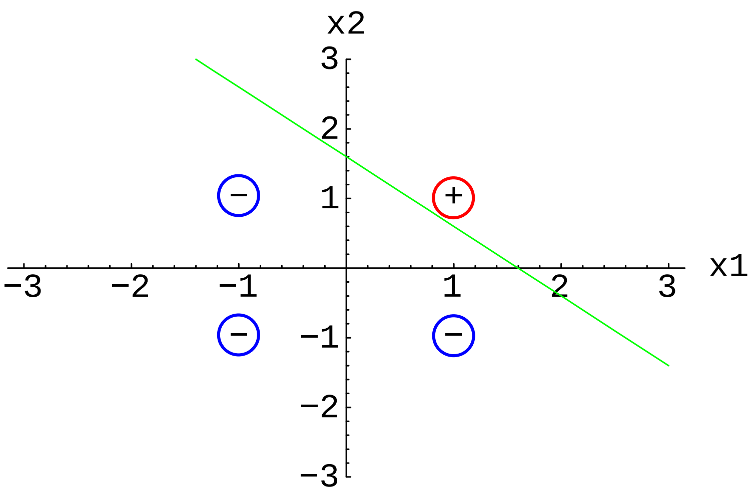
Decision Boundary of the Perceptron

Decision boundary: $w_0 + w_1 x_1 + \ldots + w_d x_d = 0$

- This is where the perceptron changes its output y from -1 (-) to $+1$ (+) if we change $\mathbf{x}$ a little bit.
- For $d = 2$ this decision boundary is always a line.

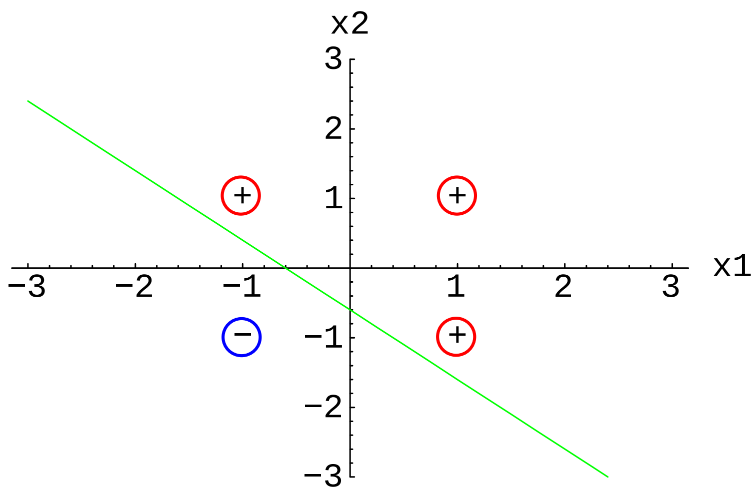
Representing Boolean Functions **(-1 = false, 1 = true):**

AND



$w_0 = -0.8, w_1 = 0.5, w_2 = 0.5$

OR

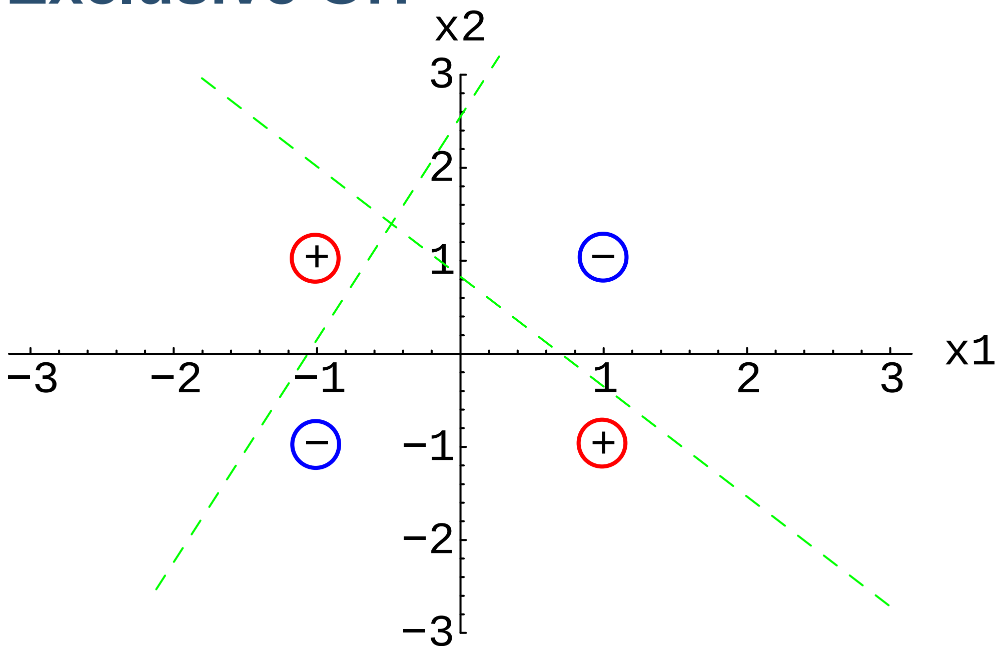


$w_0 = 0.3, w_1 = 0.5, w_2 = 0.5$

Wrong in Mitchell!

Perceptron Cannot Represent Exclusive Or

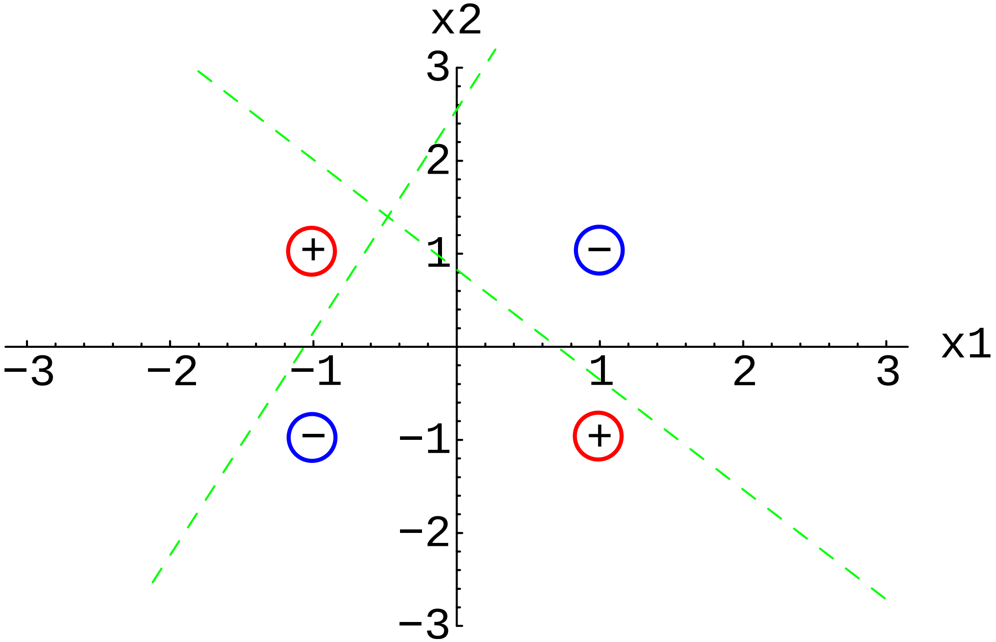
Exclusive Or:



- There exists no line that separates the inputs with label ‘-’ from the inputs with label ‘+’. They are not **linearly separable**.

Perceptron Cannot Represent Exclusive Or

Exclusive Or:

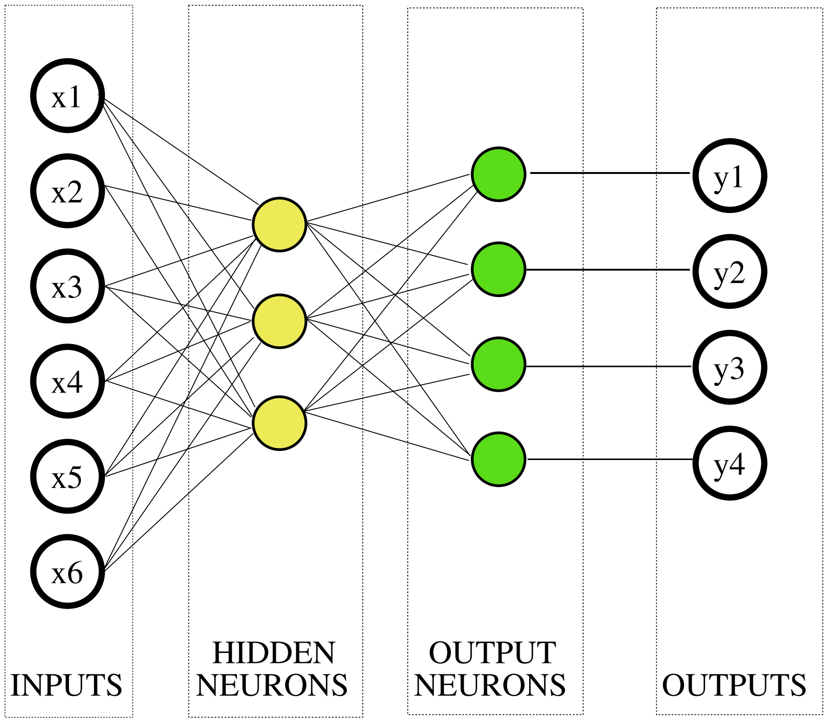


- There exists no line that separates the inputs with label '-' from the inputs with label '+'. They are not **linearly separable**.
- The decision boundary for the perceptron is always a line.
- Hence a perceptron can **never** implement the 'exclusive or' function, whichever weights we choose!

Overview

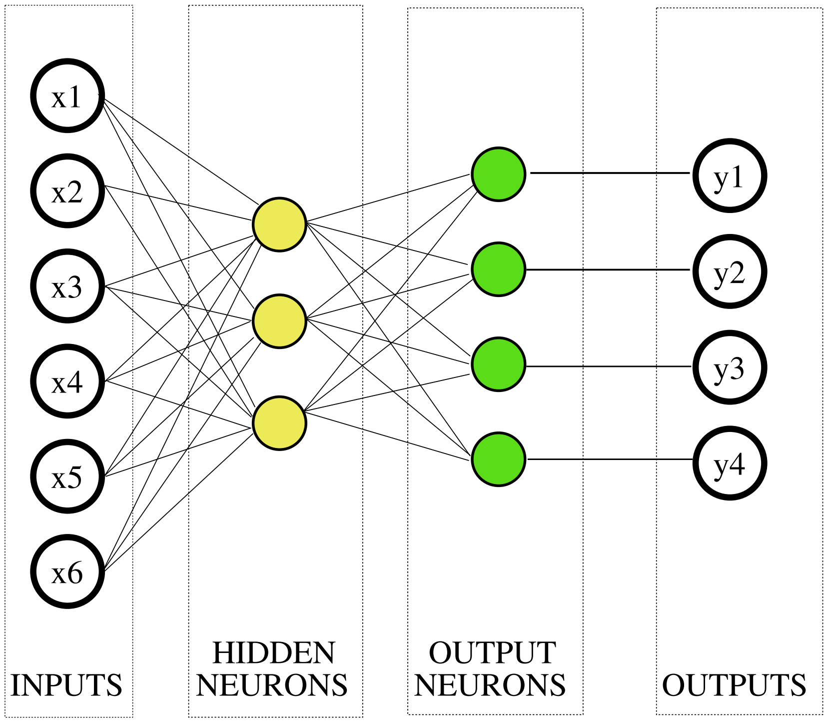
- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - **General Neural Networks**
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

Artificial Neural Networks



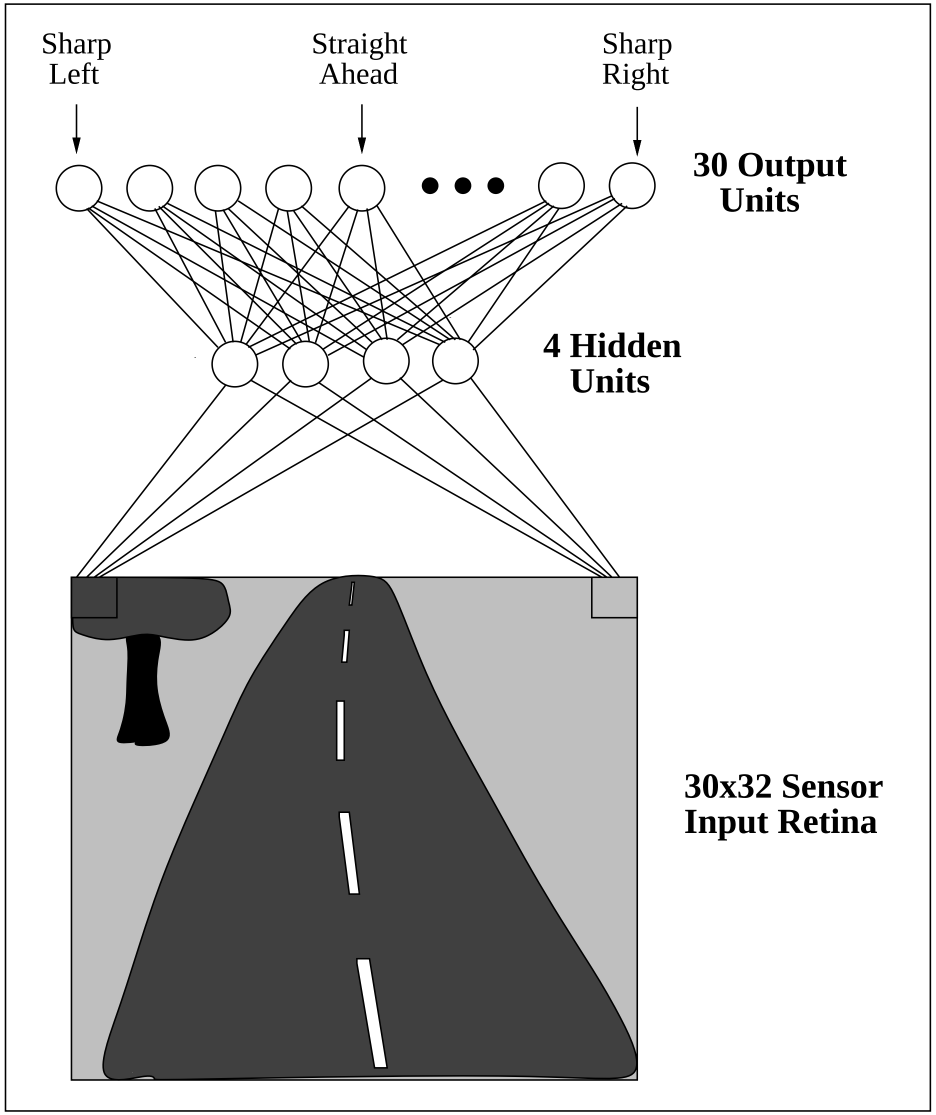
- We can create an (artificial) **neural network** (NN) by connecting neurons together.
- We hook up our feature vector $\mathbf{x}$ to the input neurons in the network. We get a label vector $\mathbf{y}$ from the output neurons.

Artificial Neural Networks



- We can create an (artificial) **neural network** (NN) by connecting neurons together.
- We hook up our feature vector $\mathbf{x}$ to the input neurons in the network. We get a label vector $\mathbf{y}$ from the output neurons.
- The parameters of the neural network $\mathbf{w}$ consist of all the parameters of the neurons in the network taken together in one big vector.

NN Example: ALVINN

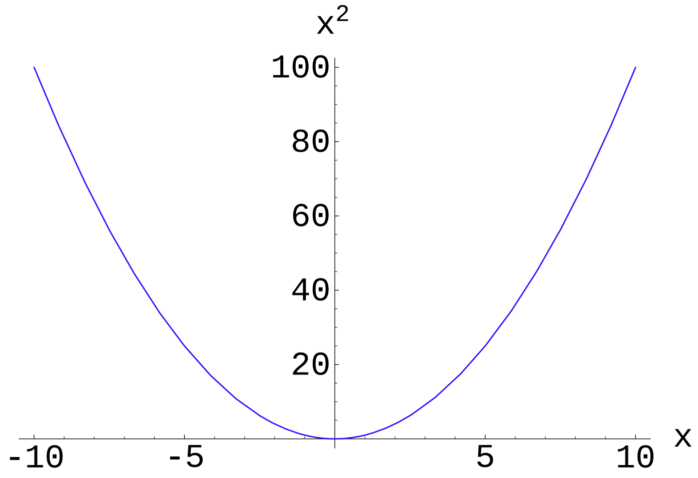


Overview

- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - **Convex Functions**
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

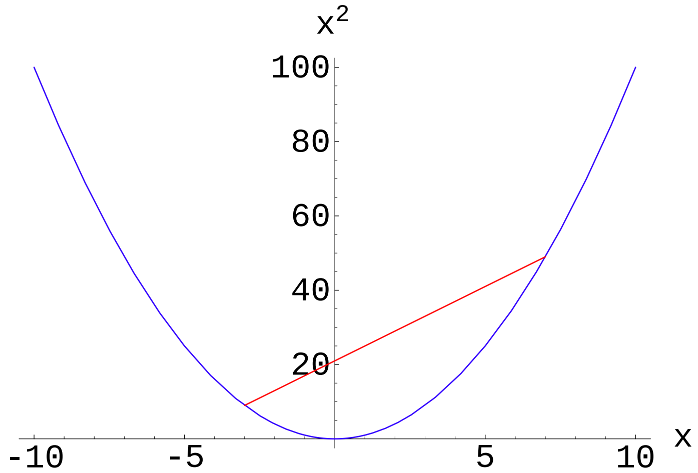
Convex Functions

Intuition:



Convex Functions

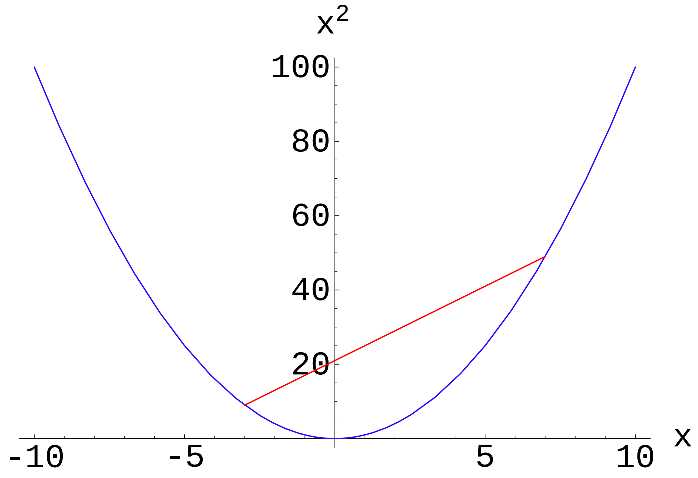
Intuition:



- A function is convex if it lies below the line between any two of its points. For example, $f(-3)$ and $f(7)$.

Convex Functions

Intuition:



- A function is convex if it lies below the line between any two of its points. For example, $f(-3)$ and $f(7)$.

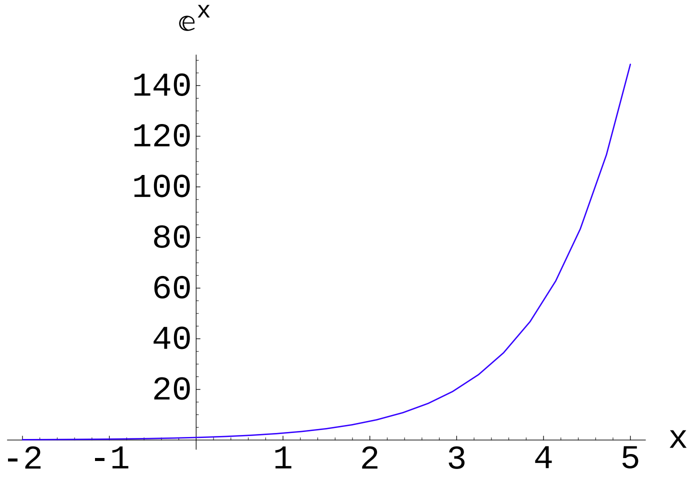
Definition: A function $f(x)$ is **convex** if

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2)$$

for any two inputs x_1, x_2 and any $0 \leq \alpha \leq 1$.

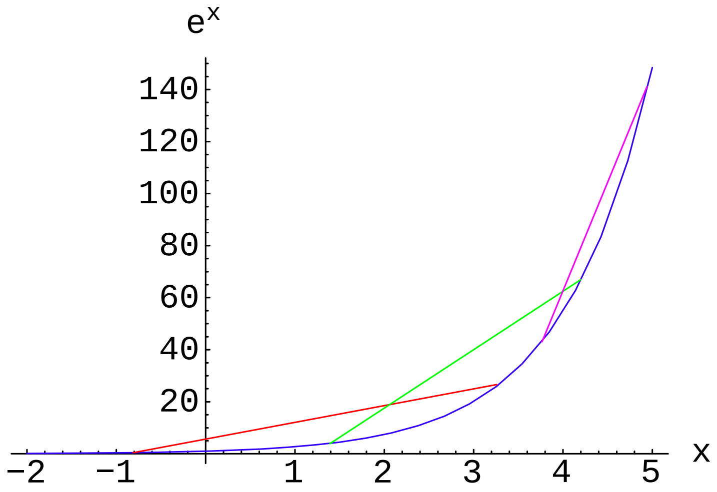
Examples

Convex:



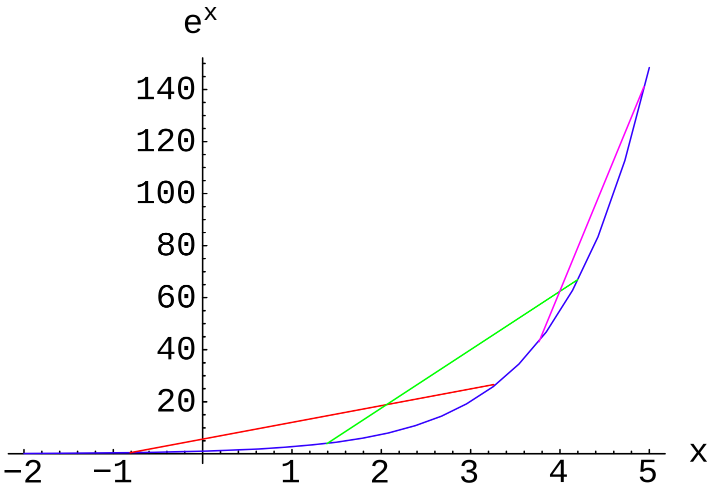
Examples

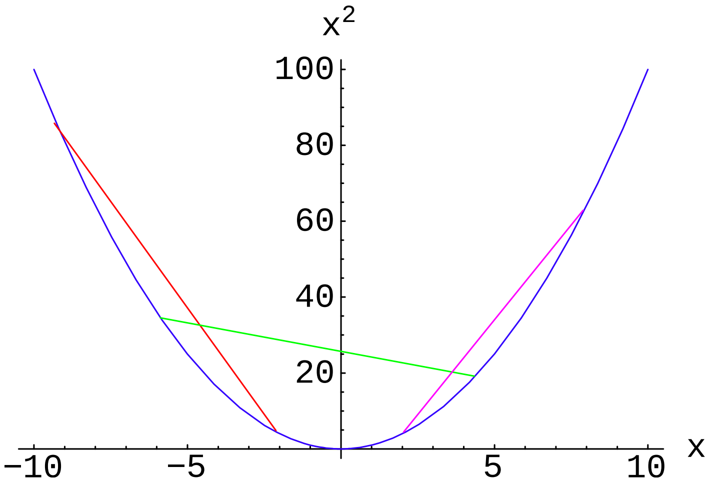
Convex:



Examples

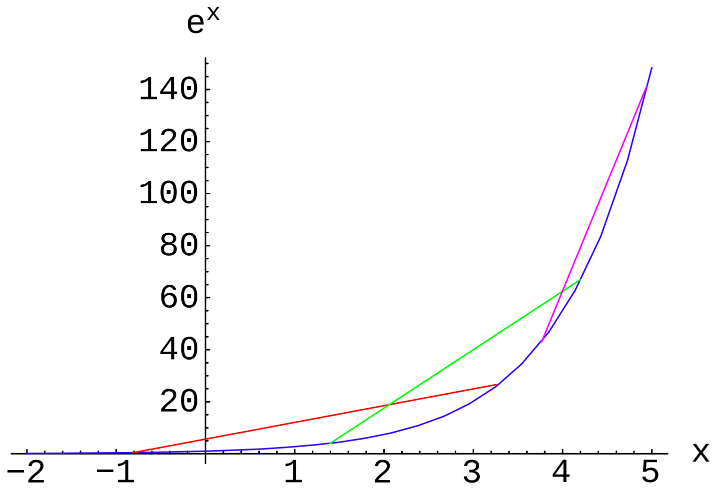
Convex:

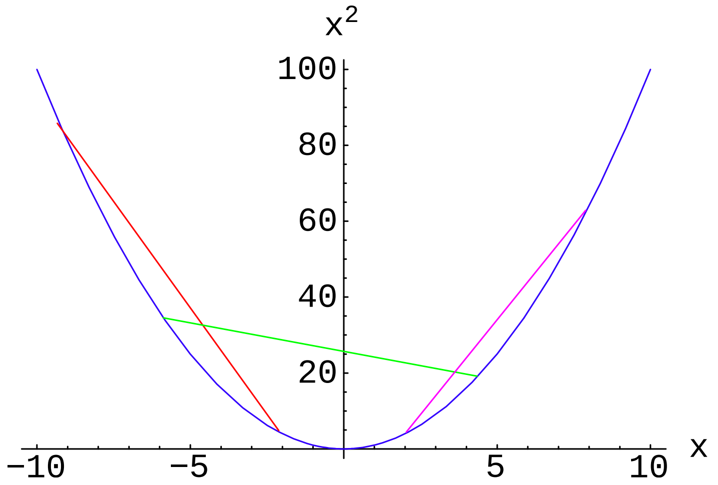




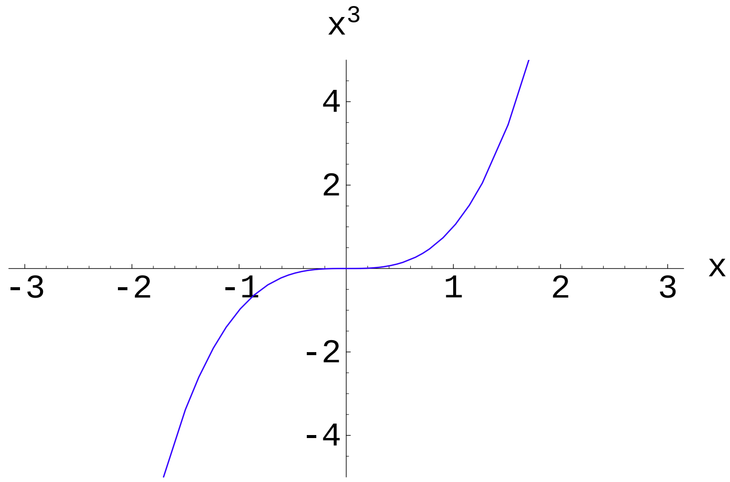
Examples

Convex:



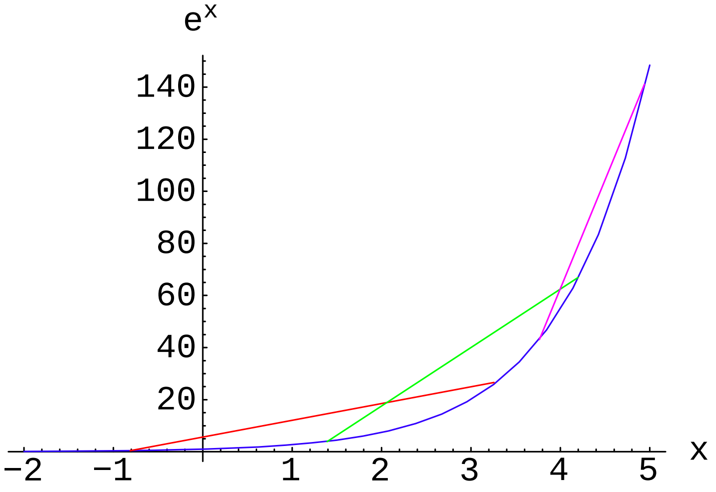


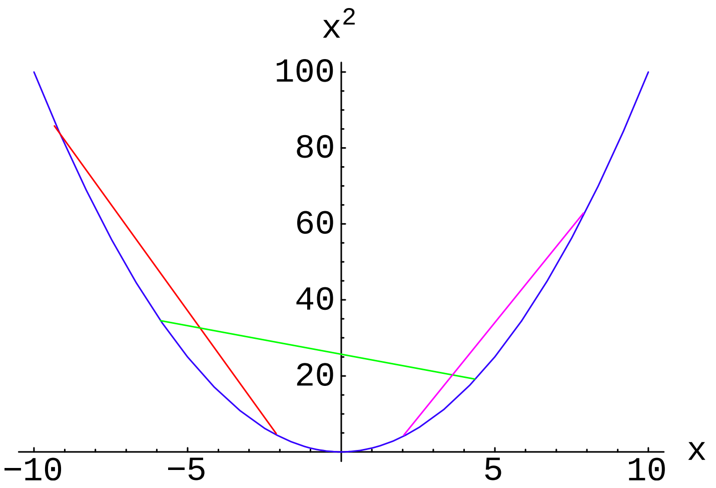
Not Convex:



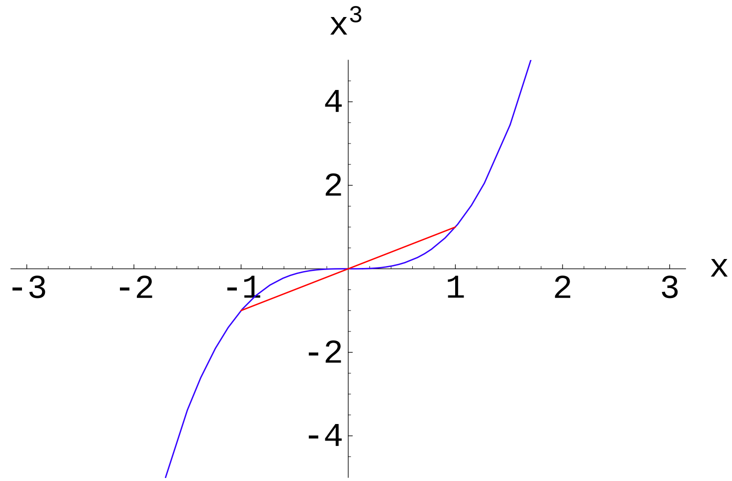
Examples

Convex:



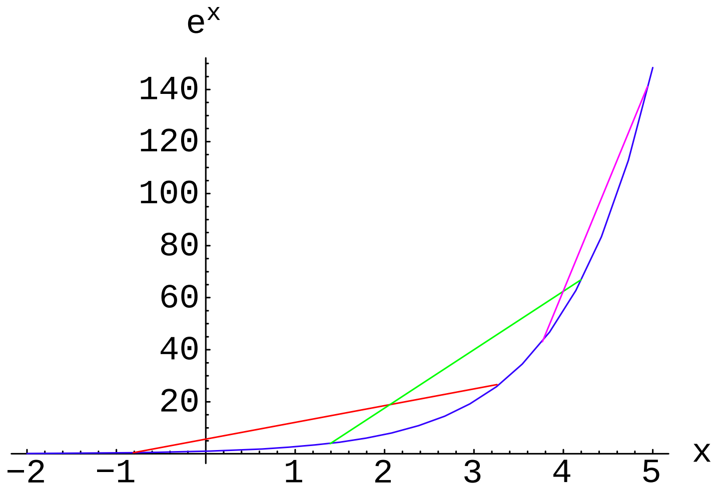


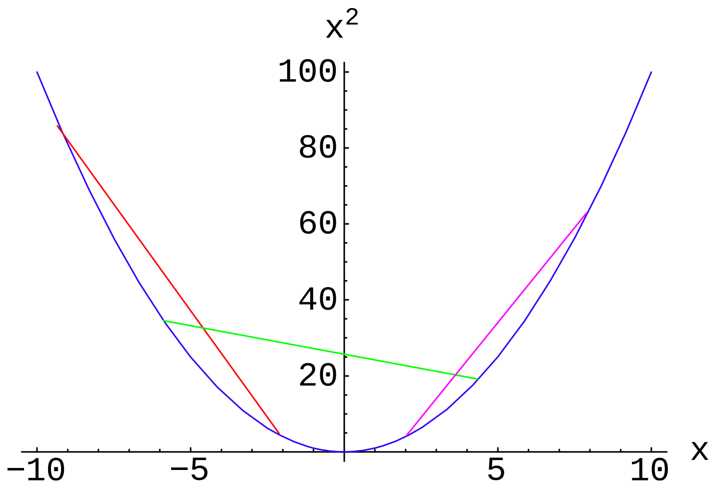
Not Convex:



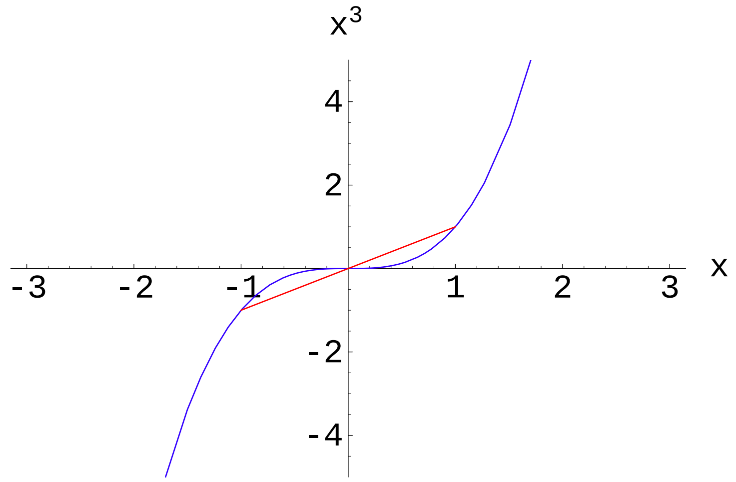
Examples

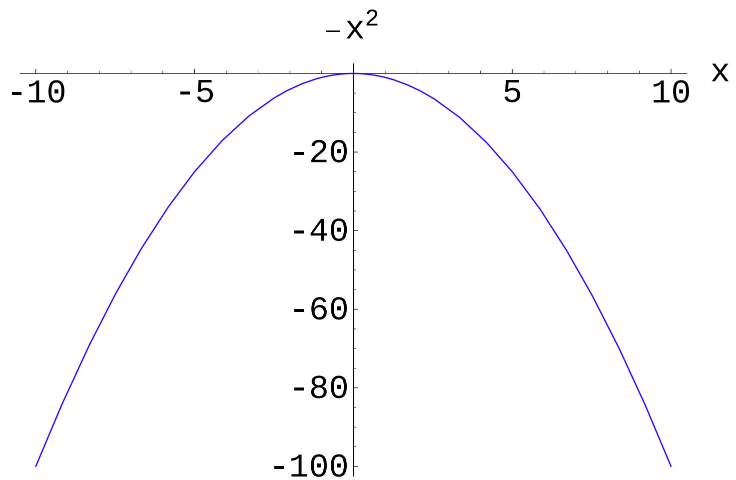
Convex:





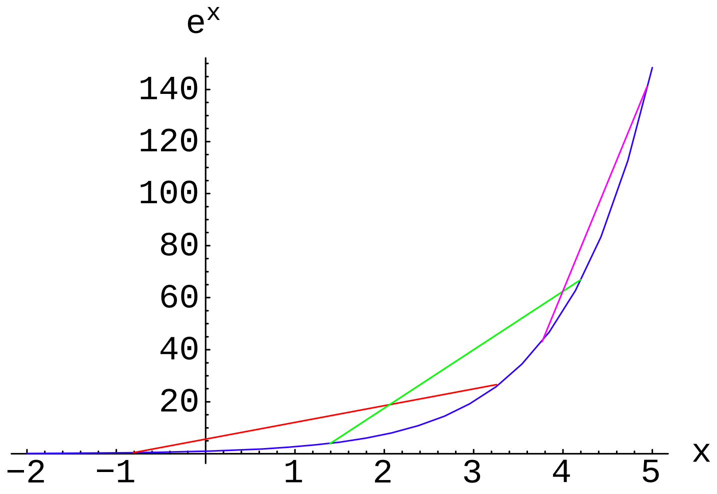
Not Convex:

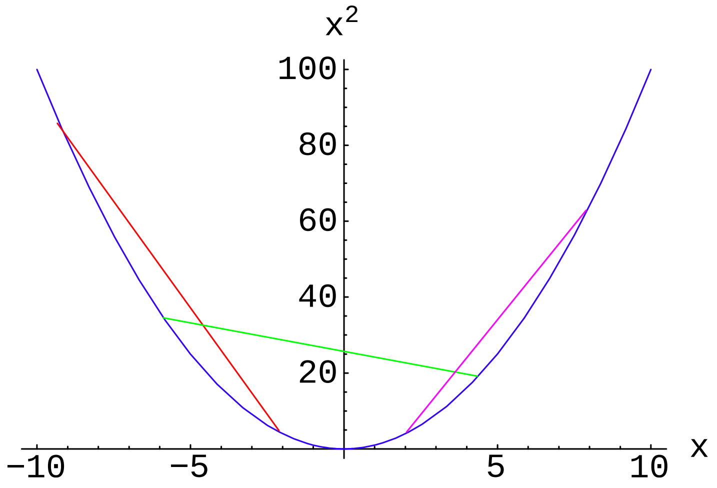




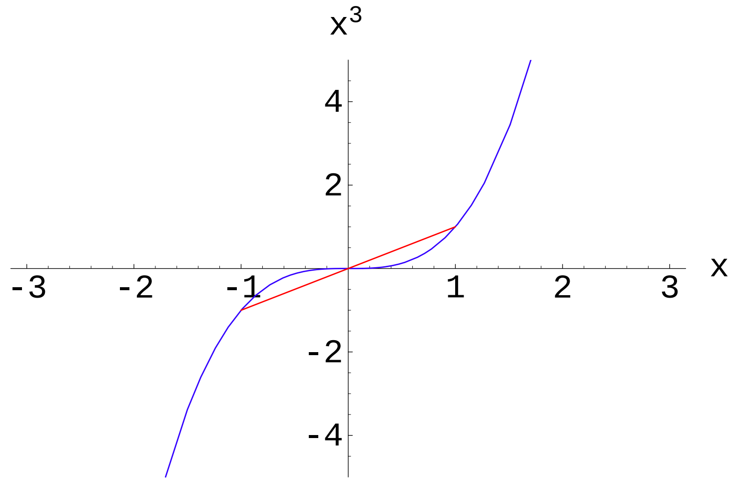
Examples

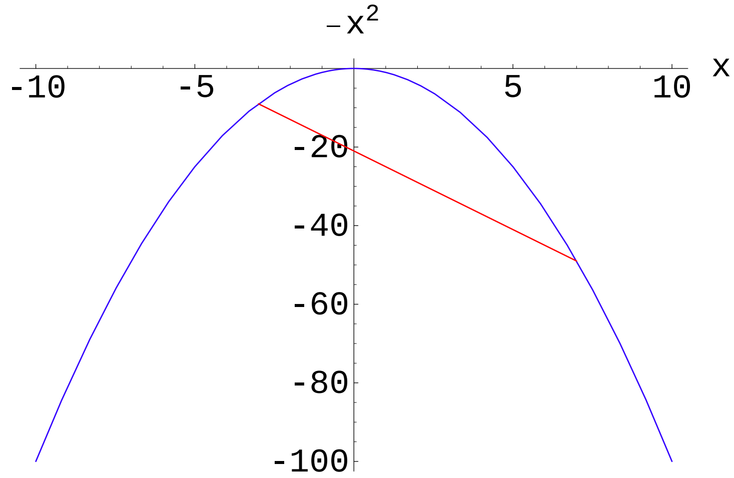
Convex:





Not Convex:





Overview

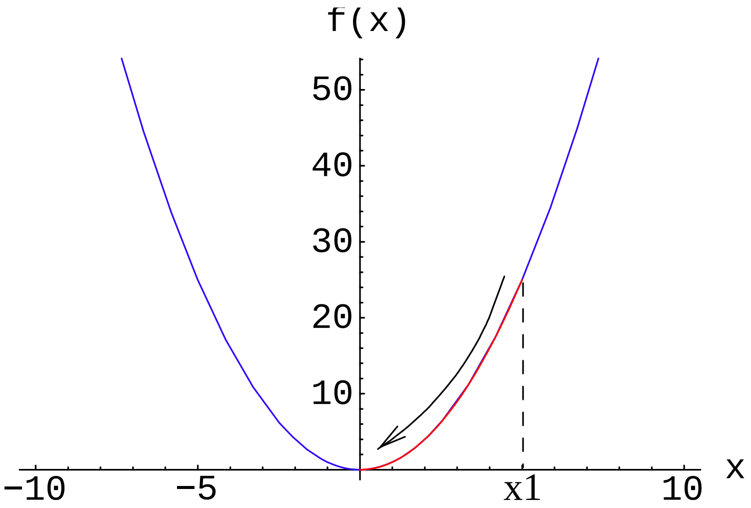
- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - **Gradient Descent in One Variable**
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

Gradient Descent

- Gradient descent is a method to find the minimum of a function: $\min_x f(x)$.
- It works for convex functions, but not for some other functions.

Gradient Descent

- Gradient descent is a method to find the minimum of a function: $\min_x f(x)$.
- It works for convex functions, but not for some other functions.



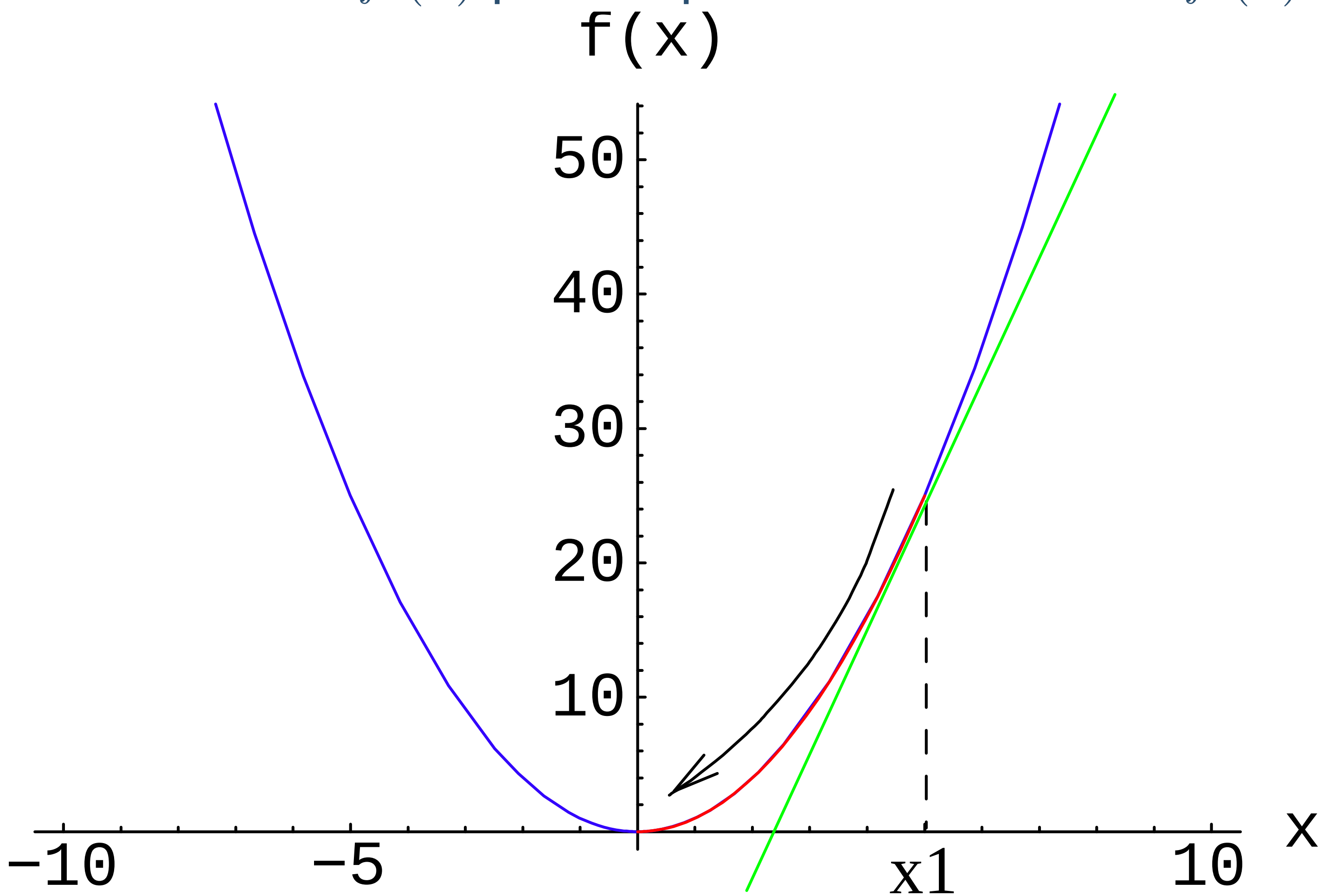
General Idea:

1. Pick some starting point x_1.
2. Keep taking small steps downhill:
   $f(x_1) > f(x_2) > f(x_3) > \ldots$
3. Stop at the minimum.

Gradient Descent More Precisely

What is Downhill?

The derivative $f'(x)$ points uphill, so downhill is $-f'(x)$.



Gradient Descent More Precisely

What is Downhill?

The derivative $f'(x)$ points uphill, so downhill is $-f'(x)$.

Step Size:

- We multiply $-f'(x_n)$ by the **learning rate** η.
- This controls the size of our steps.
- If η is too big, we will walk past the minimum.
- If η is too small, it will take very long before we get to the minimum.

Gradient Descent More Precisely

What is Downhill?

The derivative $f'(x)$ points uphill, so downhill is $-f'(x)$.

Step Size:

- We multiply $-f'(x_n)$ by the **learning rate** η.
- This controls the size of our steps.
- If η is too big, we will walk past the minimum.
- If η is too small, it will take very long before we get to the minimum.
- There exist more advanced methods to choose your step size.

Gradient Descent More Precisely

What is Downhill?

The derivative $f'(x)$ points uphill, so downhill is $-f'(x)$.

Step Size:

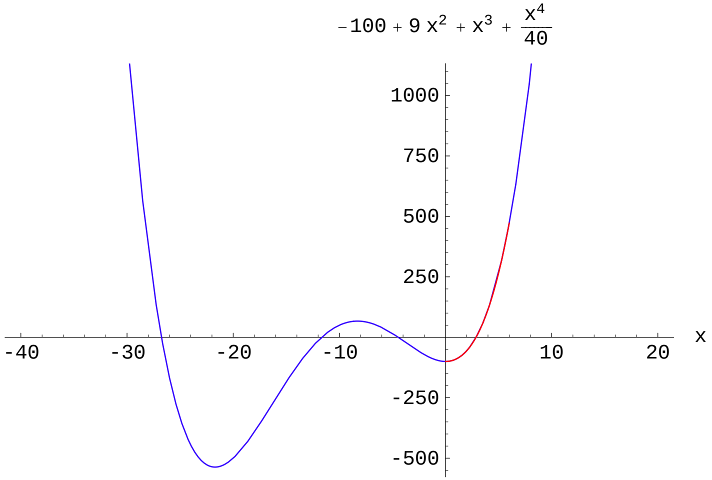
- We multiply $-f'(x_n)$ by the **learning rate** η.
- This controls the size of our steps.
- If η is too big, we will walk past the minimum.
- If η is too small, it will take very long before we get to the minimum.
- There exist more advanced methods to choose your step size.

The Gradient Descent Algorithm:

1. Pick some starting point x_1.
2. $x_{n+1} = x_n + \Delta x_n$, where $\Delta x_n = -\eta \cdot f'(x_n)$.
3. Stop when Δx_n is very small.

What Can Go Wrong?

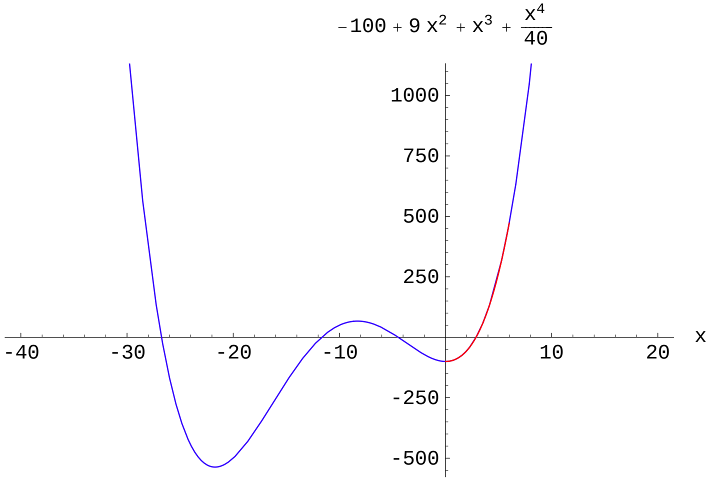
Local minima:



- For some starting points, we may get stuck at a local minimum ($x = 0$ in figure).
- Most important problem for gradient descent.
- **Convex** functions do not have local minima!

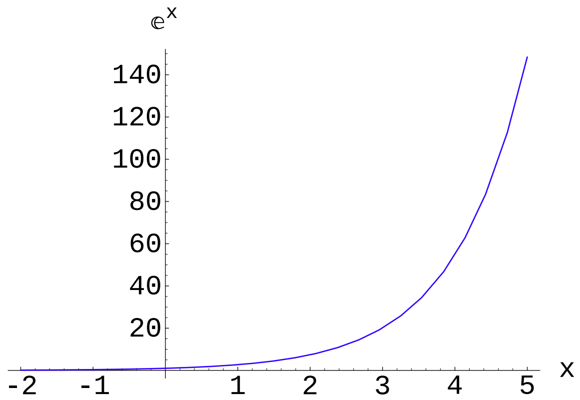
What Can Go Wrong?

Local minima:



- For some starting points, we may get stuck at a local minimum ($x = 0$ in figure).
- Most important problem for gradient descent.
- **Convex** functions do not have local minima!

No minimum exists:



- The function may have no minima at all.
- In that case gradient descent cannot find a minimum (of course).

Overview

- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - **Gradient Descent in More Variables**
  - Optimizing Perceptron Weights

The Gradient in Two Variables

One Variable:

- Suppose $g(x)$ is a function in one variable x.
- Then we can take the derivative $\frac{\partial}{\partial x} g$.

Two Variables:

- But suppose $f(\mathbf{x})$ is a function that takes a 2-dimensional vector $\mathbf{x}$ as input and outputs a scalar.
- Does there exist something like the derivative of f with respect to $\mathbf{x}$?

The Gradient in Two Variables

One Variable:

- Suppose $g(x)$ is a function in one variable x.
- Then we can take the derivative $\frac{\partial}{\partial x} g$.

Two Variables:

- But suppose $f(\mathbf{x})$ is a function that takes a 2-dimensional vector $\mathbf{x}$ as input and outputs a scalar.
- Does there exist something like the derivative of f with respect to $\mathbf{x}$?
- Yes, it is called the **gradient**:

Gradient: $\nabla f = \begin{pmatrix} \frac{\partial}{\partial x_1} f \\ \frac{\partial}{\partial x_2} f \end{pmatrix}$ **Example:** $\nabla x_1^2 x_2 + x_2 = \begin{pmatrix} 2x_1 x_2 \\ x_1^2 + 1 \end{pmatrix}$

- Note that ∇f is a function that takes $\mathbf{x}$ as input (like f), but outputs a vector!

The Gradient in d Variables

Definition:

Suppose f is a function that takes an d-dimensional vector $\mathbf{x}$ as input and outputs a scalar, then the gradient of f is

$$\nabla f = \begin{pmatrix} \frac{\partial}{\partial x_1} f \\ \vdots \\ \frac{\partial}{\partial x_d} f \end{pmatrix}$$

The Gradient in d Variables

Definition:

Suppose f is a function that takes an d-dimensional vector $\mathbf{x}$ as input and outputs a scalar, then the gradient of f is

$$\nabla f = \begin{pmatrix} \frac{\partial}{\partial x_1} f \\ \vdots \\ \frac{\partial}{\partial x_d} f \end{pmatrix}$$

- ∇f is a **function** that takes an d-dimensional vector $\mathbf{x}$ as input, just like f.
- But ∇f also outputs an d-dimensional vector, unlike f.

The Gradient in d Variables

Definition:

Suppose f is a function that takes an d-dimensional vector $\mathbf{x}$ as input and outputs a scalar, then the gradient of f is

$$\nabla f = \begin{pmatrix} \frac{\partial}{\partial x_1} f \\ \vdots \\ \frac{\partial}{\partial x_d} f \end{pmatrix}$$

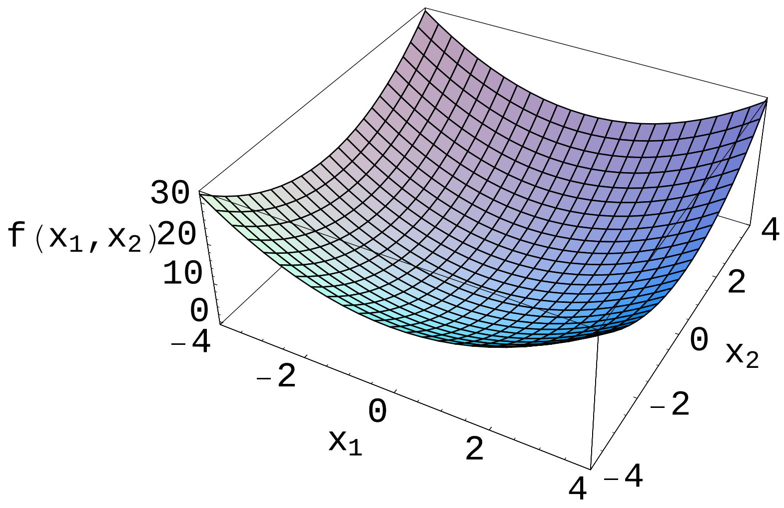
- ∇f is a **function** that takes an d-dimensional vector $\mathbf{x}$ as input, just like f.
- But ∇f also outputs an d-dimensional vector, unlike f.
- For $d = 1$ the gradient is just the derivative.
- The gradient is a generalisation of the derivative to higher dimensional inputs.

Gradient Examples

Examples on 3-dimensional input vector $\mathbf{x}$:

Functions		Functions at $\mathbf{x} = (1, 2, 3)^\top$	
f	∇f	$f\left(\begin{pmatrix}1\\2\\3\end{pmatrix}\right)$	$\nabla f\left(\begin{pmatrix}1\\2\\3\end{pmatrix}\right)$
$x_1 + 2x_2^2 - x_3$	$\begin{pmatrix}1\\4x_2\\-1\end{pmatrix}$	6	$\begin{pmatrix}1\\8\\-1\end{pmatrix}$
$x_1x_2x_3^2$	$\begin{pmatrix}x_2x_3^2\\x_1x_3^2\\2x_1x_2x_3\end{pmatrix}$	18	$\begin{pmatrix}18\\9\\12\end{pmatrix}$

Gradient Descent in More Dimensions

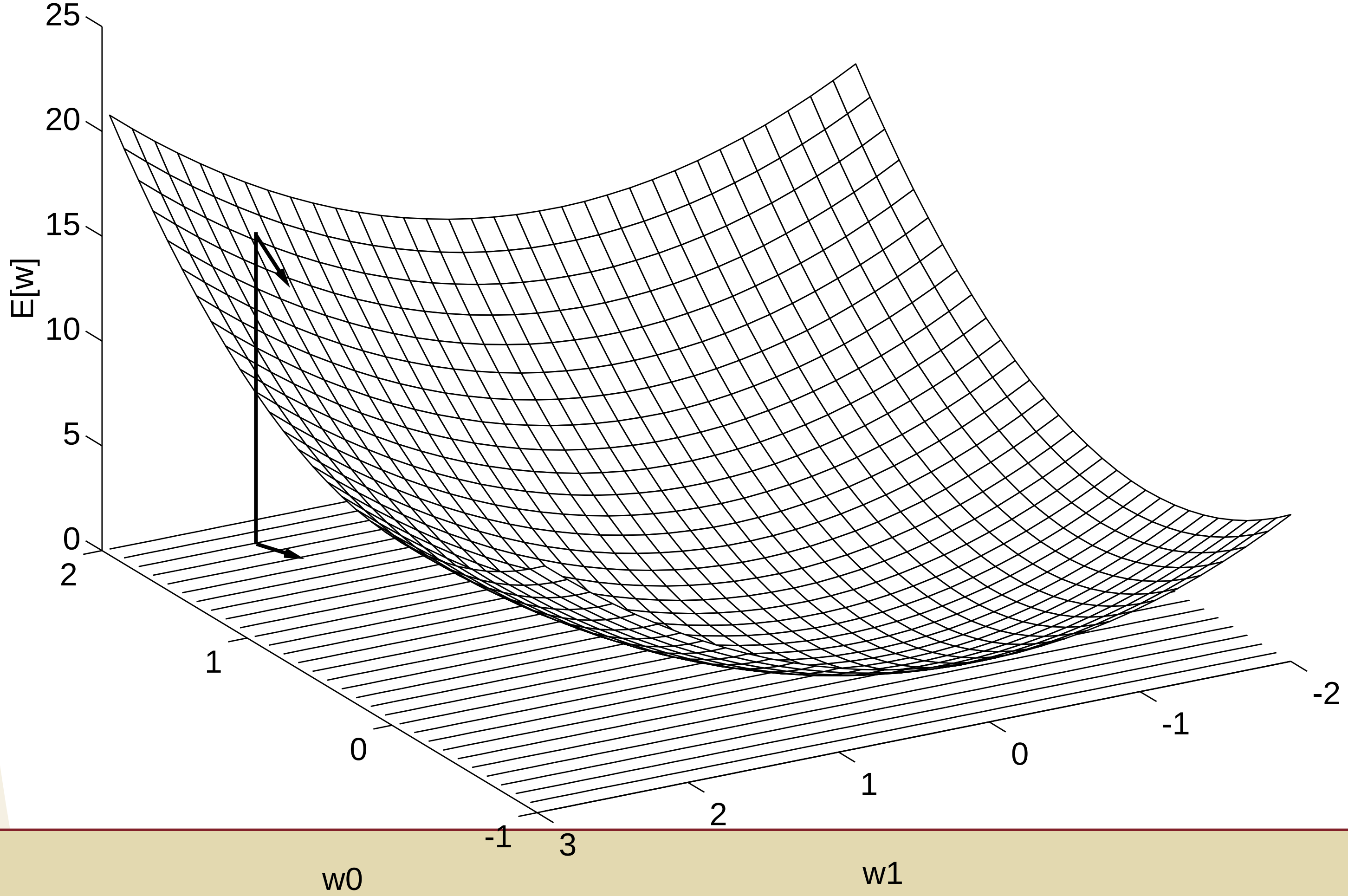


- We can also use gradient descent to find the minimum of a function that takes a vector as input: $\min_{\mathbf{x}} f(\mathbf{x})$.
- It is called gradient descent because it walks in the direction of minus the gradient.
- It works for convex functions, but not for some other functions.

Gradient Descent in More Dimensions

What is Downhill?

It can be shown that the gradient $\nabla f(\mathbf{x})$ points in the direction of the steepest ascent at $\mathbf{x}$, and that $-\nabla f(\mathbf{x})$ points in the direction of the steepest descent.



Gradient Descent in More Dimensions

What is Downhill?

It can be shown that the gradient $\nabla f(\mathbf{x})$ points in the direction of the steepest ascent at $\mathbf{x}$, and that $-\nabla f(\mathbf{x})$ points in the direction of the steepest descent.

Step Size:

- We multiply $-\nabla f(\mathbf{x})$ by the **learning rate** η.
- This controls the size of our steps.

Gradient Descent in More Dimensions

What is Downhill?

It can be shown that the gradient $\nabla f(\mathbf{x})$ points in the direction of the steepest ascent at $\mathbf{x}$, and that $-\nabla f(\mathbf{x})$ points in the direction of the steepest descent.

Step Size:

- We multiply $-\nabla f(\mathbf{x})$ by the **learning rate** η.
- This controls the size of our steps.

The Gradient Descent Algorithm:

1. Pick some starting point $\mathbf{x}_1$.
2. $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$, where $\Delta\mathbf{x}_n = -\eta \cdot \nabla f(\mathbf{x}_n)$.
3. Stop when $\Delta\mathbf{x}_n$ is a very small vector.

Gradient Descent in More Dimensions

What is Downhill?

It can be shown that the gradient $\nabla f(\mathbf{x})$ points in the direction of the steepest ascent at $\mathbf{x}$, and that $-\nabla f(\mathbf{x})$ points in the direction of the steepest descent.

Step Size:

- We multiply $-\nabla f(\mathbf{x})$ by the **learning rate** η.
- This controls the size of our steps.

The Gradient Descent Algorithm:

1. Pick some starting point $\mathbf{x}_1$.
2. $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$, where $\Delta\mathbf{x}_n = -\eta \cdot \nabla f(\mathbf{x}_n)$.
3. Stop when $\Delta\mathbf{x}_n$ is a very small vector.

- Do not confuse Δ (delta) and ∇ (the gradient).

Overview

- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - **Optimizing Perceptron Weights**

The Delta Rule

The idea: Given data D, use gradient descent to find perceptron weights that minimize the number of wrongly classified training examples in D.

The Delta Rule

The idea: Given data D, use gradient descent to find perceptron weights that minimize the number of wrongly classified training examples in D.

A Problem:

- The perceptron applies a threshold to a linear function.
- This threshold makes the derivative/gradient undefined for some inputs.

Solution:

- Minimize the sum of squared errors on D for the perceptron **without the threshold**.
- Note that D is considered fixed: We are minimizing $\mathrm{SSE}(\mathbf{w}, D)$ as a function of $\mathbf{w}$.

The Delta Rule

The idea: Given data D, use gradient descent to find perceptron weights that minimize the number of wrongly classified training examples in D.

A Problem:

- The perceptron applies a threshold to a linear function.
- This threshold makes the derivative/gradient undefined for some inputs.

Solution:

- Minimize the sum of squared errors on D for the perceptron **without the threshold**.
- Note that D is considered fixed: We are minimizing SSE$(\mathbf{w}, D)$ as a function of $\mathbf{w}$.
- The perceptron without the threshold is just a linear function $h_{\mathbf{w}}(\mathbf{x})$ (also called **linear unit** in NNs).
- This is just linear regression!

Gradient Descent for Perceptron Weights

Remarks:

- $\text{SSE}(\mathbf{w}, D)$ is a convex function of $\mathbf{w}$.

Gradient Descent for Perceptron Weights

Remarks:

- $\text{SSE}(\mathbf{w}, D)$ is a convex function of $\mathbf{w}$.
- To apply gradient descent we need to compute the gradient.
- It will be convenient to minimize $\frac{1}{2}\text{SSE}(\mathbf{w}, D)$ instead of $\text{SSE}(\mathbf{w}, D)$.

Gradient Descent for Perceptron Weights

Remarks:

- $\mathrm{SSE}(\mathbf{w}, D)$ is a convex function of $\mathbf{w}$.
- To apply gradient descent we need to compute the gradient.
- It will be convenient to minimize $\frac{1}{2}\mathrm{SSE}(\mathbf{w}, D)$ instead of $\mathrm{SSE}(\mathbf{w}, D)$.

Computing The Gradient:

We can compute the ith component of the gradient as follows (see Mitchell, Equation 4.6):

$$\begin{aligned}\frac{\partial}{\partial w_i}\frac{1}{2}\mathrm{SSE}(\mathbf{w}, D) &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{(y,\mathbf{x})^\top \in D}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\ &= \sum_{(y,\mathbf{x})^\top \in D}(y - h_{\mathbf{w}}(\mathbf{x}))\cdot(-x_i)\end{aligned}$$

Overview

- Organisational Matters
- Linear Functions as Inner Products
- Neural Networks
  - The Perceptron
  - General Neural Networks
- Gradient Descent
  - Convex Functions
  - Gradient Descent in One Variable
  - Gradient Descent in More Variables
  - Optimizing Perceptron Weights

References

- S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge University Press, 2004
- T.M. Mitchell, “Machine Learning”, McGraw-Hill, 1997