CS551: Introduction to Deep Learning

Deep Feedforward Network

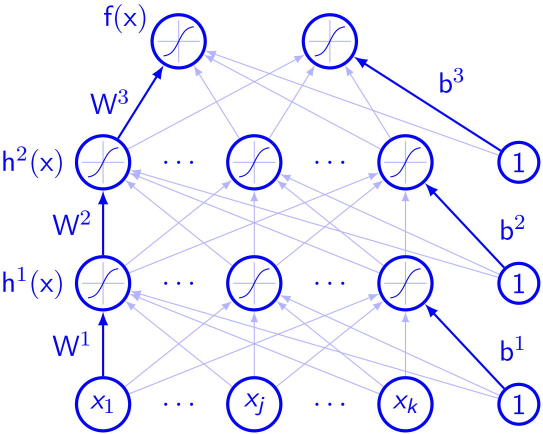
Arijit Mondal

Dept. of Computer Science & Engineering

Indian Institute of Technology Patna

arijit@iitp.ac.in

Multilayer neural network



Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron

Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function f^*
  - For classifier, x is mapped to category y ie. $y = f^*(\mathrm{x})$
  - A feedforward network maps $y = f(\mathrm{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation

Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function f^*
  - For classifier, x is mapped to category y ie. $y = f^*(\mathrm{x})$
  - A feedforward network maps $y = f(\mathrm{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation
- Information flows from input to intermediate to output
  - No feedback, directed acyclic graph
  - For general model, it can have feedback and known as recurrent neural network

Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function f^*
  - For classifier, x is mapped to category y ie. $y = f^*(\mathrm{x})$
  - A feedforward network maps $y = f(\mathrm{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation
- Information flows from input to intermediate to output
  - No feedback, directed acyclic graph
  - For general model, it can have feedback and known as recurrent neural network
- Typically it represents composition of functions
  - Three functions $f^{(1)}, f^{(2)}, f^{(3)}$ are connected in chain
  - Overall function realized is $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
  - The number of layers provides the depth of the model

Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function f^*
  - For classifier, x is mapped to category y ie. $y = f^*(\mathrm{x})$
  - A feedforward network maps $y = f(\mathrm{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation
- Information flows from input to intermediate to output
  - No feedback, directed acyclic graph
  - For general model, it can have feedback and known as recurrent neural network
- Typically it represents composition of functions
  - Three functions $f^{(1)}, f^{(2)}, f^{(3)}$ are connected in chain
  - Overall function realized is $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
  - The number of layers provides the depth of the model
- Goal of NN is not to model brain accurately!

Issues with linear FFN

- Fit well for linear and logistic regression
- Convex optimization technique may be used
- Capacity of such function is limited
- Model cannot understand interaction between any two variables

Overcome issues of linear FFN

- Transform x (input) into $\phi(x)$ where ϕ is nonlinear transformation

Overcome issues of linear FFN

- Transform x (input) into $\phi(x)$ where ϕ is nonlinear transformation
- How to choose ϕ?

Overcome issues of linear FFN

- Transform x (input) into $\phi(x)$ where ϕ is nonlinear transformation
- How to choose ϕ?
  - Use a very generic ϕ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough prior information

Overcome issues of linear FFN

- Transform x (input) into $\phi(x)$ where ϕ is nonlinear transformation
- How to choose ϕ?
  - Use a very generic ϕ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough prior information
  - Manually design ϕ
    - Require domain knowledge

Overcome issues of linear FFN

- Transform x (input) into $\phi(x)$ where ϕ is nonlinear transformation
- How to choose ϕ?
  - Use a very generic ϕ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough prior information
  - Manually design ϕ
    - Require domain knowledge
  - Strategy of deep learning is to learn ϕ

Goal of deep learning

- We have a model $y = f(\mathsf{x}; \boldsymbol{\theta}, \mathsf{w}) = \phi(\mathsf{x}; \boldsymbol{\theta})^T \mathsf{w}$
- We use $\boldsymbol{\theta}$ to learn ϕ
- w and ϕ determines the output. ϕ defines the hidden layer
- It looses the convexity of the training problem but benefits a lot
- Representation is parameterized as $\phi(\mathsf{x}, \boldsymbol{\theta})$
  - $\boldsymbol{\theta}$ can be determined by solving optimization problem
- Advantages
  - ϕ can be very generic
  - Human practitioner can encode their knowledge to designing $\phi(\mathsf{x}; \boldsymbol{\theta})$

Example

- Let us choose XOR function
- Target function is $y = f^*(\mathrm{x})$ and our model provides $y = f(\mathrm{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make f close to f^*

Example

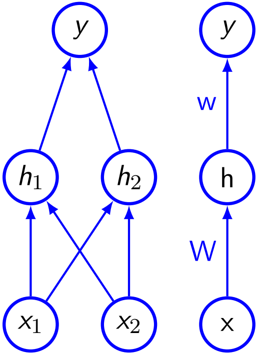
- Let us choose XOR function
- Target function is $y = f^*(\mathrm{x})$ and our model provides $y = f(\mathrm{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make f close to f^*
- Target is to fit output for $\mathrm{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
- This can be treated as regression problem and MSE error can be chosen as loss function ($J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\mathrm{x}\in\mathrm{X}}(f^*(\mathrm{x}) - f(\mathrm{x}; \boldsymbol{\theta}))^2$)
- We need to choose $f(\mathrm{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ depends on w and b
- Let us consider a linear model $f(\mathrm{x}; \mathrm{w}, b) = \mathrm{x}^T\mathrm{w} + b$

Example

- Let us choose XOR function
- Target function is $y = f^*(\mathrm{x})$ and our model provides $y = f(\mathrm{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make f close to f^*
- Target is to fit output for $\mathrm{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
- This can be treated as regression problem and MSE error can be chosen as loss function ($J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\mathrm{x}\in\mathrm{X}}(f^*(\mathrm{x}) - f(\mathrm{x}; \boldsymbol{\theta}))^2$)
- We need to choose $f(\mathrm{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ depends on w and b
- Let us consider a linear model $f(\mathrm{x}; \mathrm{w}, b) = \mathrm{x}^T\mathrm{w} + b$
- Solving these, we get $\mathrm{w} = 0$ and $b = \frac{1}{2}$

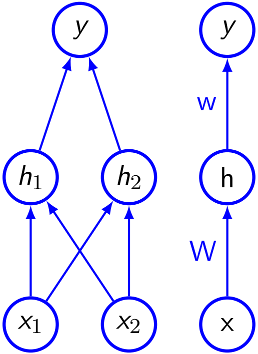
Simple FFN with hidden layer

- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x}; \mathrm{W}, \mathrm{c})$



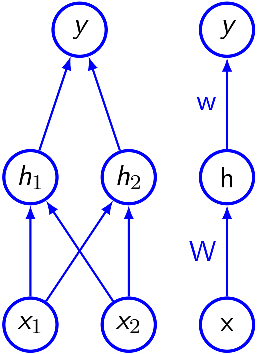
Simple FFN with hidden layer

- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x}; \mathrm{W}, \mathrm{c})$
- In the next layer $y = f^{(2)}(\mathrm{h}; \mathrm{w}, b)$ is computed



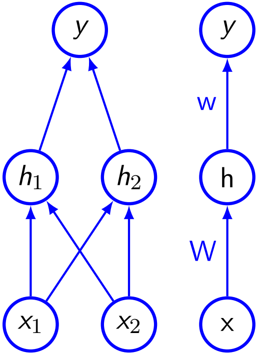
Simple FFN with hidden layer

- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x}; \mathrm{W}, \mathrm{c})$
- In the next layer $y = f^{(2)}(\mathrm{h}; \mathrm{w}, b)$ is computed
- Complete model $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = f^{(2)}(f^{(1)}(\mathrm{x}))$



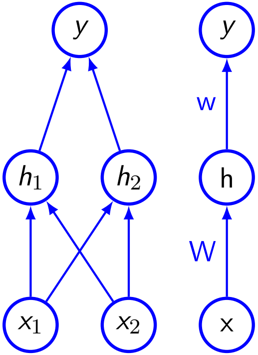
Simple FFN with hidden layer

- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x}; \mathrm{W}, \mathrm{c})$
- In the next layer $y = f^{(2)}(\mathrm{h}; \mathrm{w}, b)$ is computed
- Complete model $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = f^{(2)}(f^{(1)}(\mathrm{x}))$
- Suppose $f^{(1)}(\mathrm{x}) = \mathrm{W}^T\mathrm{x}$ and $f^2(\mathrm{h}) = \mathrm{h}^T\mathrm{w}$



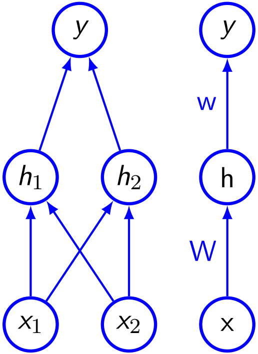
Simple FFN with hidden layer

- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x}; \mathrm{W}, \mathrm{c})$
- In the next layer $y = f^{(2)}(\mathrm{h}; \mathrm{w}, b)$ is computed
- Complete model $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = f^{(2)}(f^{(1)}(\mathrm{x}))$
- Suppose $f^{(1)}(\mathrm{x}) = \mathrm{W}^T\mathrm{x}$ and $f^2(\mathrm{h}) = \mathrm{h}^T\mathrm{w}$ then $f(\mathrm{x}) = \mathrm{w}^T\mathrm{W}^T\mathrm{x}$



Simple FFN with hidden layer (contd.)

- We need to have nonlinear function to describe the features
- Usually NN have affine transformation of learned parameters followed by nonlinear activation function
- Let us use $\mathrm{h} = g(\mathrm{W}^T\mathrm{x} + \mathrm{c})$
- Let us use ReLU as activation function $g(z) = \max\{0, z\}$
- g is chosen element wise $h_i = g(\mathrm{x}^T\mathrm{W}_{:,i} + c_i)$



Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = \mathrm{w}^T \max\{0, \mathrm{W}^T\mathrm{x} + \mathrm{c}\} + b$

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathsf{x}; \mathsf{W}, \mathsf{c}, \mathsf{w}, b) = \mathsf{w}^T \max\{0, \mathsf{W}^T\mathsf{x} + \mathsf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathsf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathsf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathsf{x}; \mathsf{W}, \mathsf{c}, \mathsf{w}, b) = \mathsf{w}^T \max\{0, \mathsf{W}^T\mathsf{x} + \mathsf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathsf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathsf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - X

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathsf{x}; \mathsf{W}, \mathsf{c}, \mathsf{w}, b) = \mathsf{w}^T \max\{0, \mathsf{W}^T\mathsf{x} + \mathsf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathsf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathsf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathsf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$,

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\{0, \mathbf{W}^T\mathbf{x} + \mathbf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{XW}$

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathsf{x}; \mathsf{W}, \mathsf{c}, \mathsf{w}, b) = \mathsf{w}^T \max\{0, \mathsf{W}^T\mathsf{x} + \mathsf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathsf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathsf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathsf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$,

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\{0, \mathbf{W}^T\mathbf{x} + \mathbf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, add bias c

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathsf{x}; \mathsf{W}, \mathsf{c}, \mathsf{w}, b) = \mathsf{w}^T \max\{0, \mathsf{W}^T\mathsf{x} + \mathsf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathsf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathsf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathsf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathsf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, add bias c $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$,

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = \mathrm{w}^T \max\{0, \mathrm{W}^T\mathrm{x} + \mathrm{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathrm{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathrm{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathrm{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathrm{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, add bias c $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, compute h

CS551

Simple FFN with hidden layer (contd.)

- Complete network is $f(x; W, c, w, b) = w^T \max\{0, W^T x + c\} + b$
- A solution for XOR problem can be as follows
  - $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $X = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, add bias c $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, compute h $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$,

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\{0, \mathbf{W}^T\mathbf{x} + \mathbf{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, add bias $\mathbf{c}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, compute $\mathbf{h}$ $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, multiply

    with $\mathbf{w}$

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = \mathrm{w}^T \max\{0, \mathrm{W}^T \mathrm{x} + \mathrm{c}\} + b$
- A solution for XOR problem can be as follows
  - $\mathrm{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathrm{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathrm{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$
- Now we have
  - $\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathrm{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, add bias c $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, compute h $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, multiply

    with w $\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

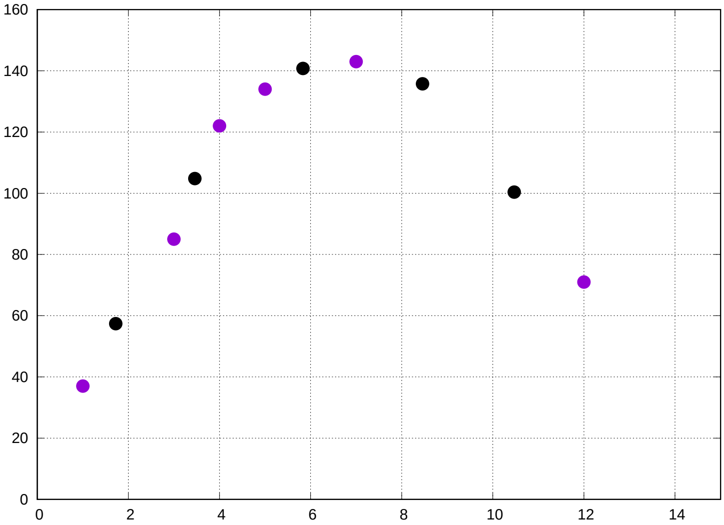
Gradient based learning

- Similar to machine learning tasks, gradient descent based learning is used
  - Need to specify optimization procedure, cost function and model family
- For NN, model is nonlinear and function becomes nonconvex
  - Usually trained by iterative, gradient based optimizer
- Solved by using gradient descent or stochastic gradient descent (SGD)

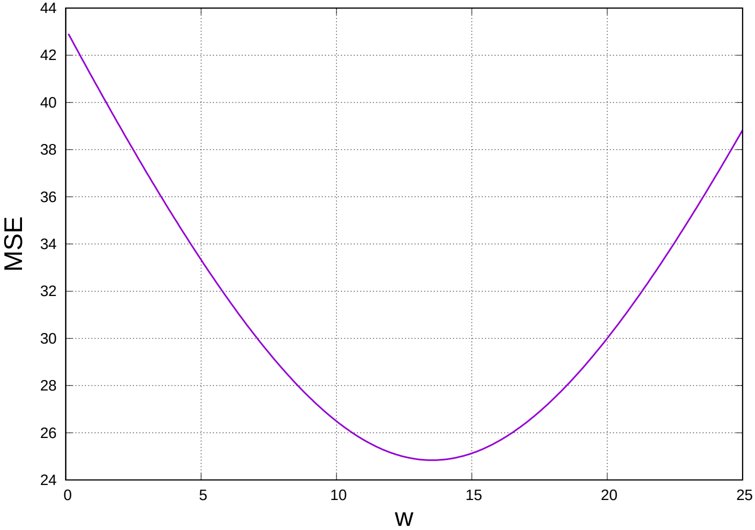
Gradient descent

- For a function $y = f(x)$, derivative (slope at point x) of it is $f'(x) = \frac{dy}{dx}$
- A small change in the input can cause output to move to a value given by $f(x+\epsilon) \approx f(x)+\epsilon f'(x)$
- We need to take a jump so that y reduces (assuming minimization problem)
- We can say that $f(x - \epsilon \text{sign}(f'(x)))$ is less than $f(x)$
- For multiple inputs partial derivatives are used ie. $\frac{\partial}{\partial x_i} f(\mathrm{x})$
- Gradient vector is represented as $\nabla_{\mathrm{x}} f(\mathrm{x})$
- Gradient descent proposes a new point as $\mathrm{x}' = \mathrm{x} - \epsilon \nabla_{\mathrm{x}} f(x)$ where ϵ is the learning rate

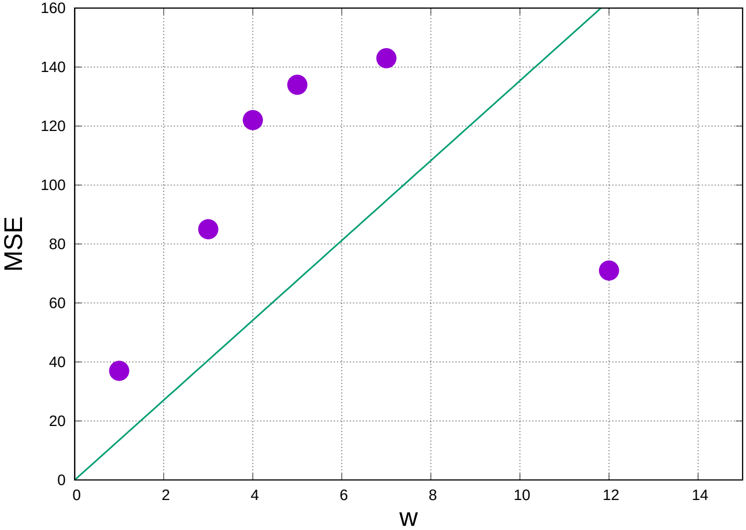
Example



Example: Variation of MSE wrt *w*



Example: Best fit



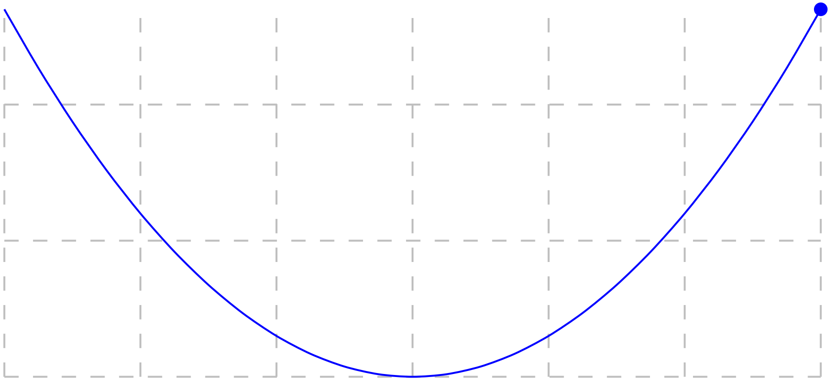
Minimization of MSE: Gradient descent

- Assuming $\text{MSE}_{(\text{train})} = J(w_1, w_2)$
- Target is to $\min_{w_1, w_2} J(w_1, w_2)$
- Approach
  - Start with some w_1, w_2
  - Keep modifying w_1, w_2 so that $J(w_1, w_2)$ reduces till the desired accuracy is achieved

Minimization of MSE: Gradient descent

- Assuming $\text{MSE}_{(\text{train})} = J(w_1, w_2)$
- Target is to $\min_{w_1, w_2} J(w_1, w_2)$
- Approach
  - Start with some w_1, w_2
  - Keep modifying w_1, w_2 so that $J(w_1, w_2)$ reduces till the desired accuracy is achieved
- Algorithm
  - Repeat the following until convergence $w_j = w_j - \dfrac{\partial}{\partial w_j} J(w_1, w_2)$
- Gradient descent proposes a new point as $\mathrm{w}' = \mathrm{w} - \epsilon \nabla_{\mathrm{w}} f(\mathrm{w})$ where ϵ is the learning rate

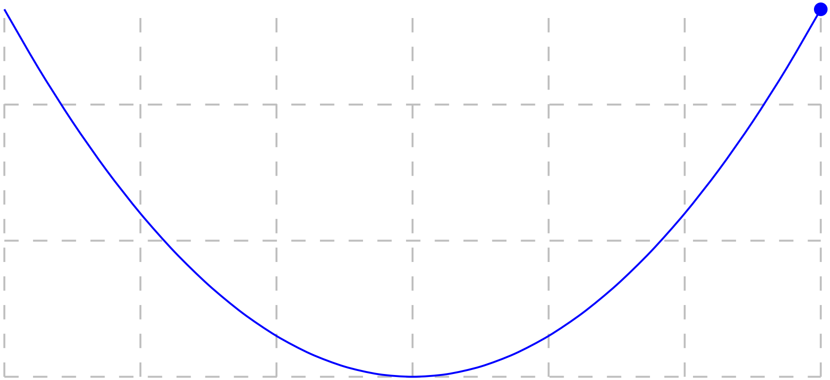
Gradient descent



$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=1.80002

Gradient descent

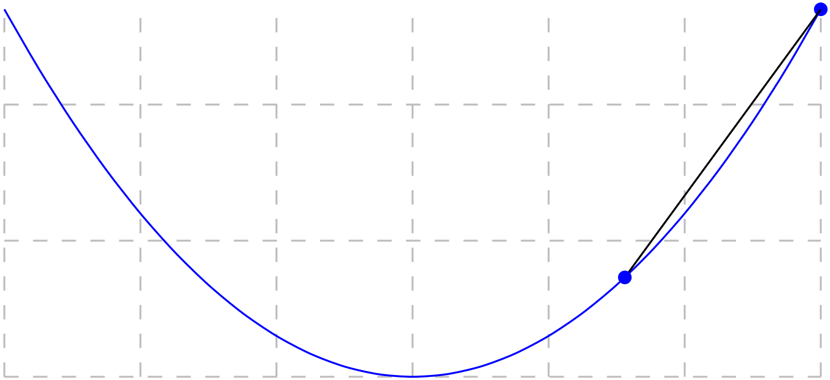


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=1.80002

x_{new}=1.56001

Gradient descent

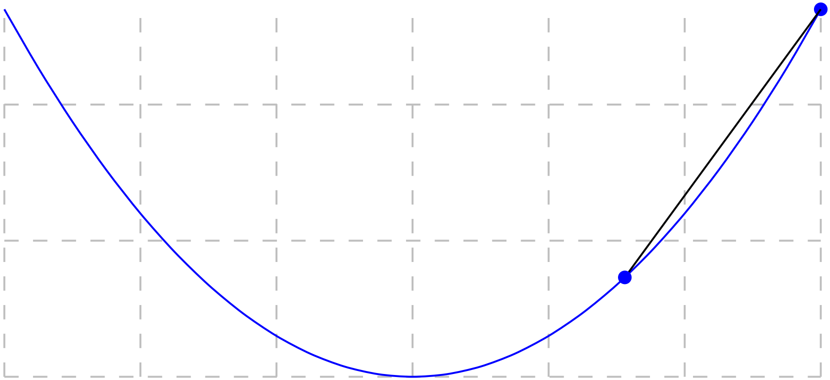


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=1.80002

x_{new}=1.56001

Gradient descent

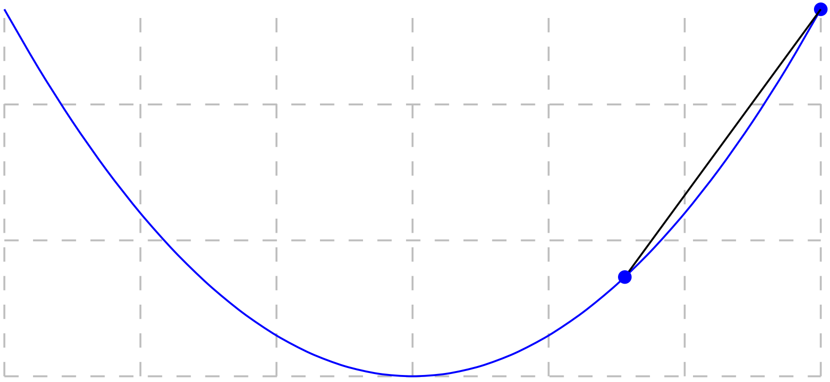


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.936

x_{new}=1.56001

Gradient descent

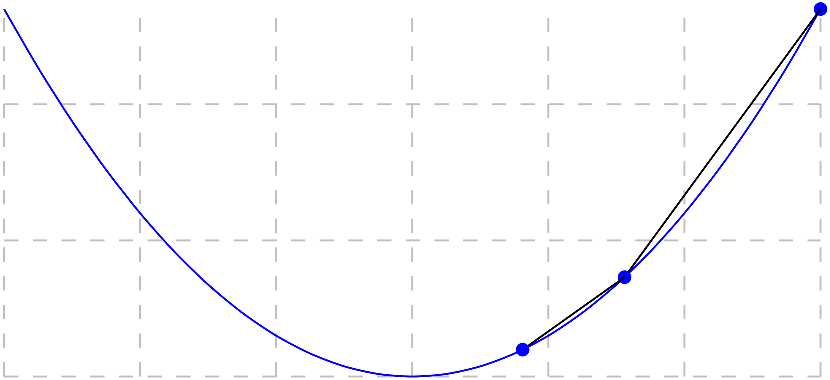


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.936

x_{new}=0.81122

Gradient descent

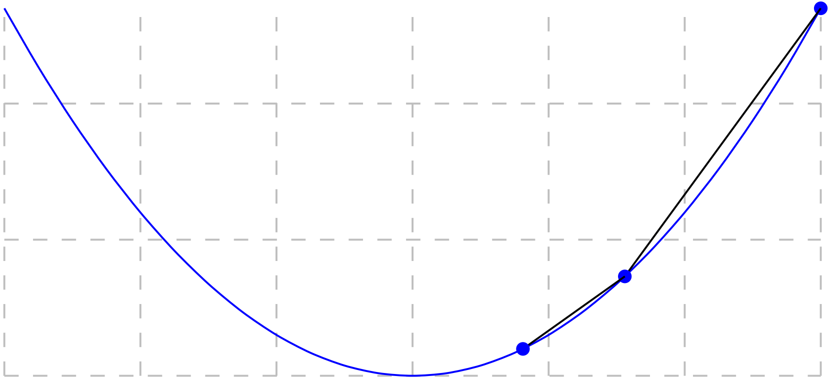


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.936

x_{new}=0.81122

Gradient descent

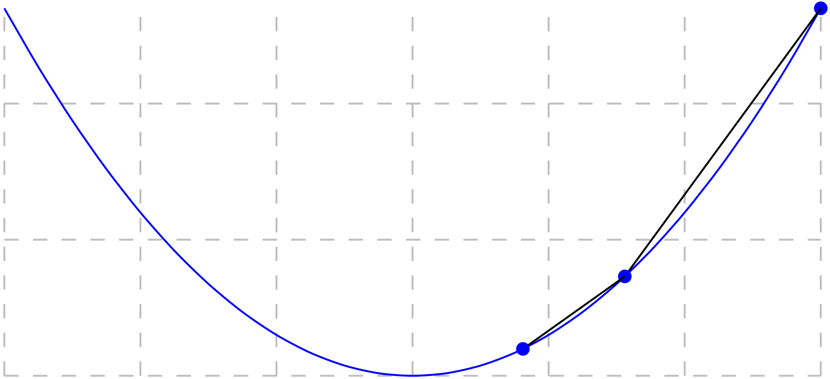


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.48672

x_{new}=0.81122

Gradient descent

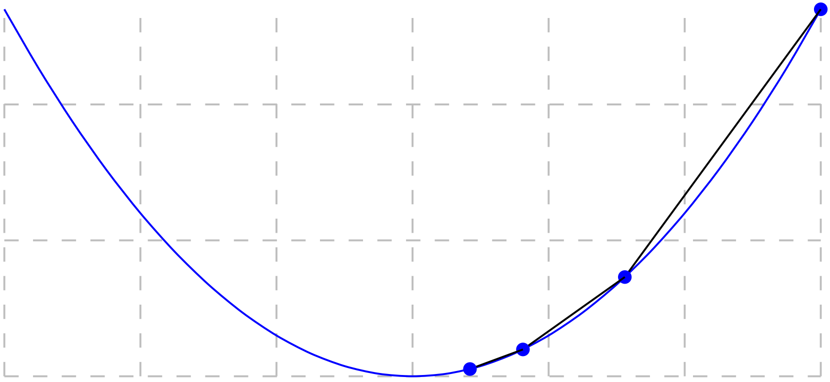


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.48672

x_{new}=0.42184

Gradient descent

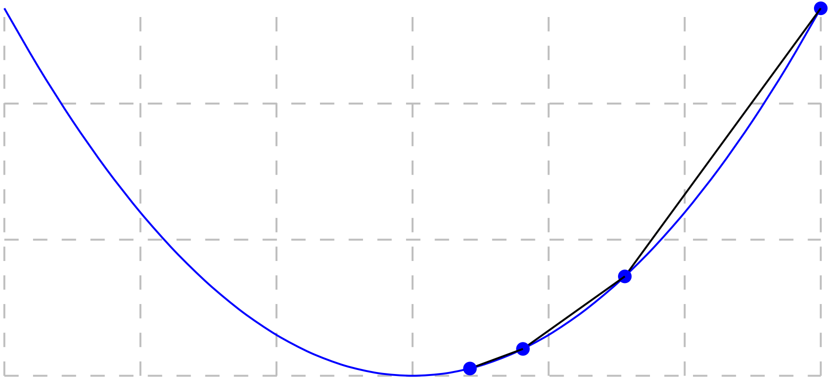


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.48672

x_{new}=0.42184

Gradient descent

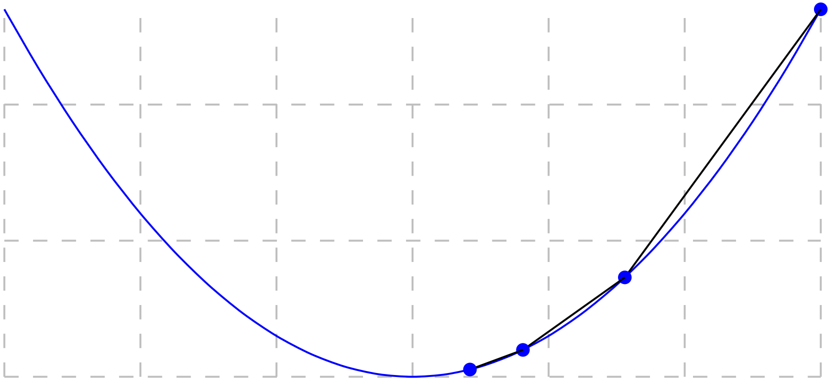


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.2531

x_{new}=0.42184

Gradient descent

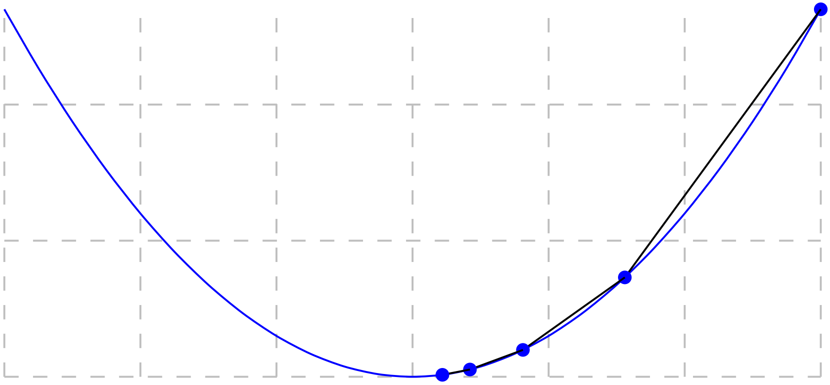


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.2531

x_{new}=0.21938

Gradient descent

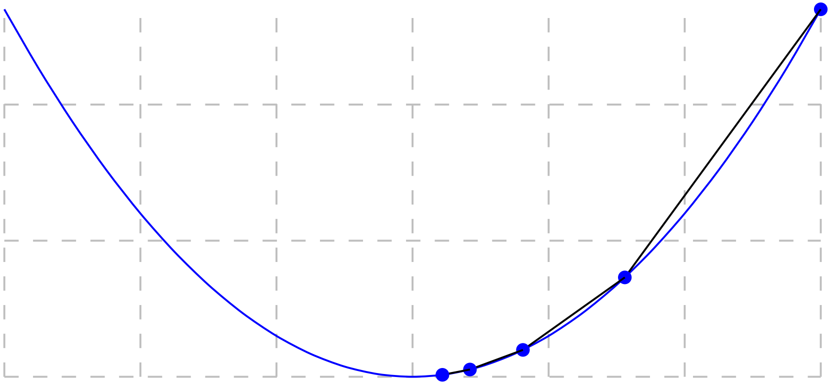


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.2531

x_{new}=0.21938

Gradient descent

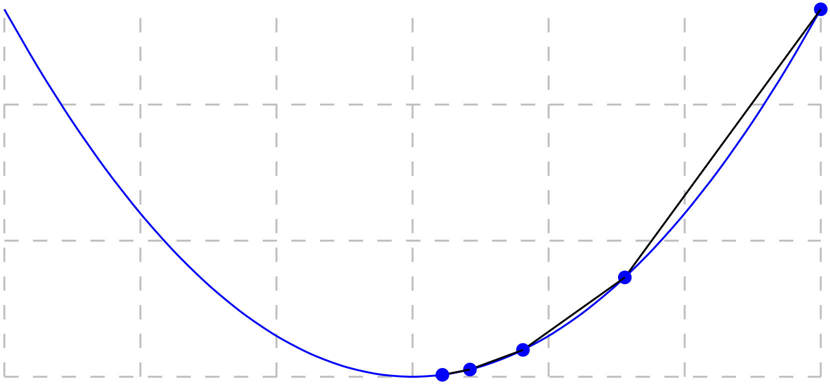


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.13162

x_{new}=0.21938

Gradient descent

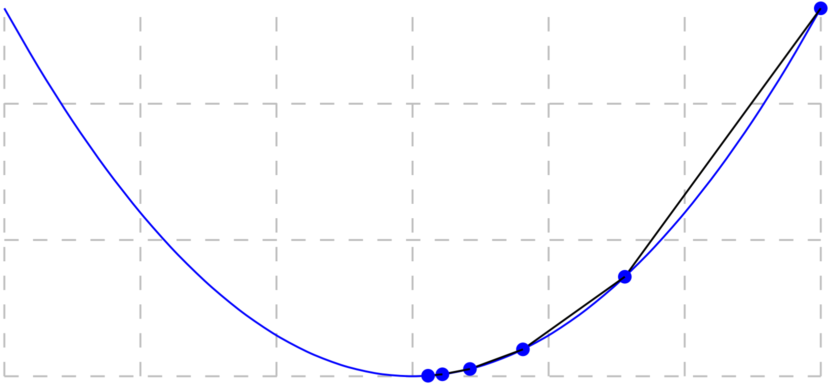


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.13162

x_{new}=0.11409

Gradient descent

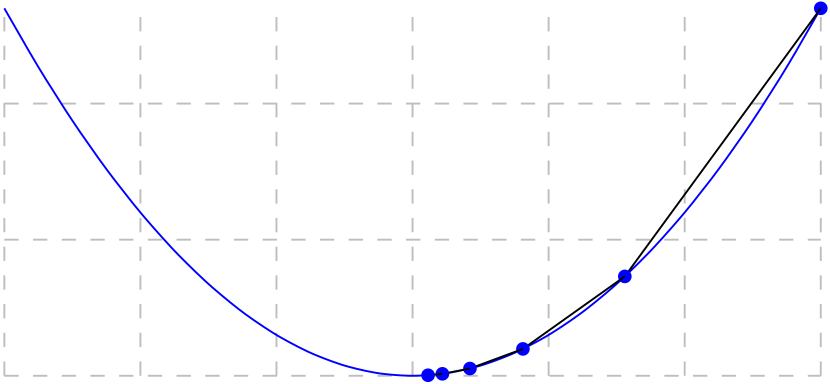


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.13162

x_{new}=0.11409

Gradient descent

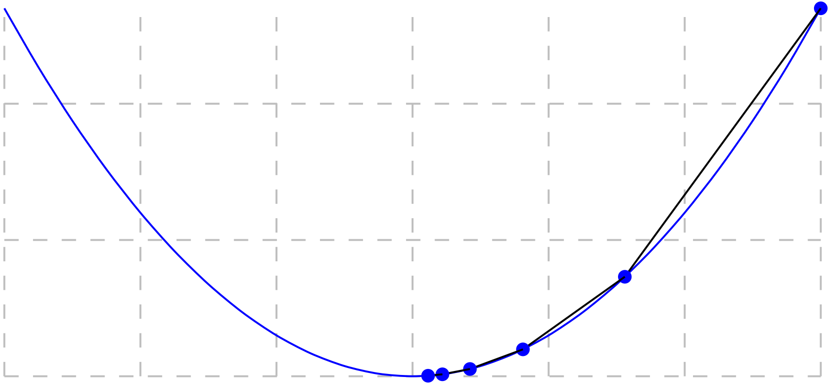


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.06845

x_{new}=0.11409

Gradient descent

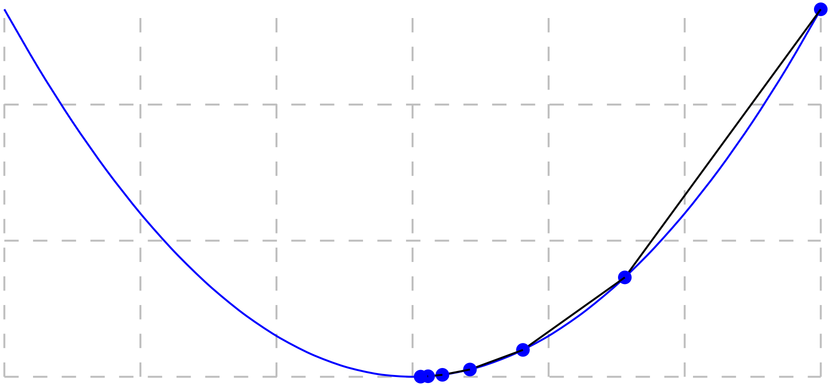


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.06845

x_{new}=0.05934

Gradient descent

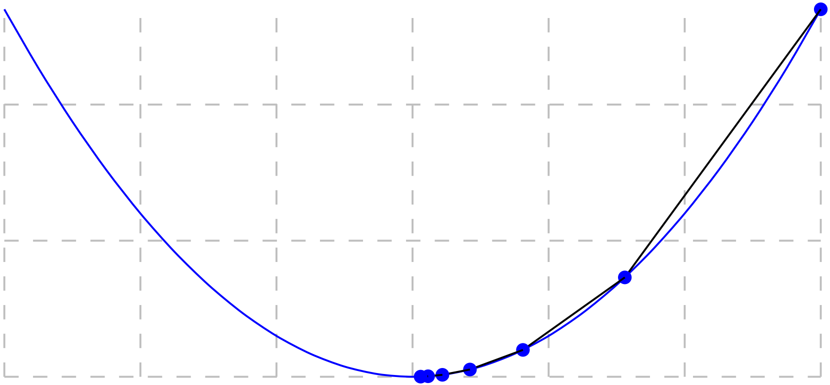


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.06845

x_{new}=0.05934

Gradient descent

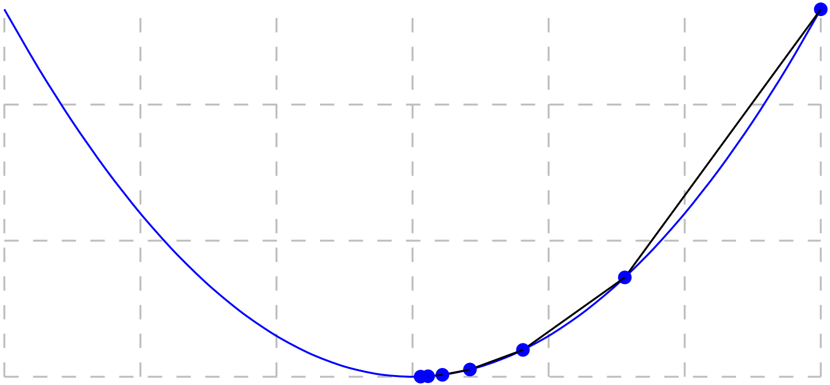


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.0356

x_{new}=0.05934

Gradient descent



$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.0356

x_{new}=0.03087

Stochastic gradient descent

- Large training set are necessary for good generalization
- Cost function used for optimization is $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Gradient descent requires $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$

Stochastic gradient descent

- Large training set are necessary for good generalization
- Cost function used for optimization is $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Gradient descent requires $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
  - Computation cost is $O(m)$

Stochastic gradient descent

- Large training set are necessary for good generalization
- Cost function used for optimization is $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Gradient descent requires $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
    - Computation cost is $O(m)$
- For SGD, gradient is an expectation estimated from a small sample known as minibatch $(\mathbb{B} = \{x^{(1)}, \ldots, x^{(m')}\})$
- Estimated gradient is $\mathrm{g} = \dfrac{1}{m'}\displaystyle\sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- New point will be $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \mathrm{g}$

SGD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $x(w \times x - y)$

Step	Point	Derivative	New w

SGD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $x(w \times x - y)$

```
Step    Point   Derivative          New w
1       (1,2)   1*(4.0*1-2)=2.0     3.80
```

SGD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $x(w \times x - y)$

Step	Point	Derivative	New w
1	(1,2)	1*(4.0*1-2)=2.0	3.80
2	(2,4)	2*(3.8*2-4)=7.2	3.08

SGD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $x(w \times x - y)$

Step	Point	Derivative	New w
1	(1,2)	1*(4.0*1-2)=2.0	3.80
2	(2,4)	2*(3.8*2-4)=7.2	3.08
3	(3,6)	3*(3.1*3-6)=9.7	2.11
4	(4,8)	4*(2.1*4-8)=1.7	1.94
5	(1,2)	1*(1.9*1-2)=-0.1	1.94
6	(2,4)	2*(1.9*2-4)=-0.2	1.97
7	(3,6)	3*(2.0*3-6)=-0.3	1.99
8	(4,8)	4*(2.0*4-8)=-0.1	2.00
9	(1,2)	1*(2.0*1-2)=0.0	2.00

GD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $\frac{1}{4} \sum_i x_i (w \times x_i - y_i)$

```
Step     Derivative   New w
```

GD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $\frac{1}{4}\sum_i x_i(w \times x_i - y_i)$

Step	Derivative	New w
1	15	2.5

GD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $\frac{1}{4} \sum_i x_i(w \times x_i - y_i)$

Step	Derivative	New w
1	15	2.5
2	3.75	2.13

GD example

- Consider the following pair (x, y) of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1
- MSE as cost function. Derivative will be $\frac{1}{4}\sum_i x_i(w \times x_i - y_i)$

Step	Derivative	New w
1	15	2.5
2	3.75	2.13
3	0.94	2.03
4	0.23	2.01
5	0.06	2.00

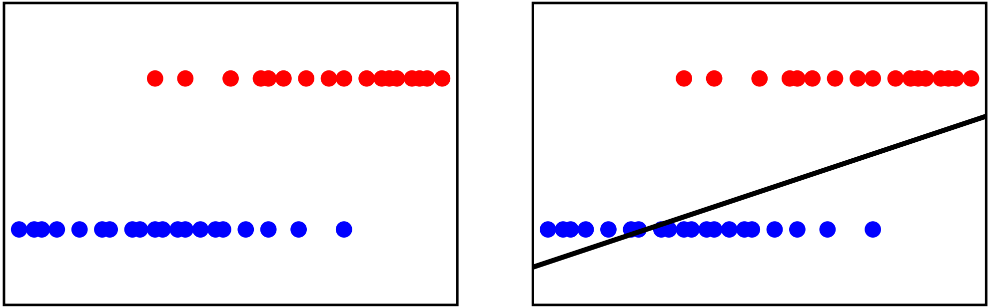
Cost function

- Similar to other parametric model like linear models
- Parametric model defines distribution $p(\mathrm{y}|\mathrm{x};\boldsymbol{\theta})$
- Principle of maximum likelihood is used (cross entropy between training data and model prediction)
- Instead of predicting the whole distribution of y, some statistic of y conditioned on x is predicted
- It can also contain regularization term

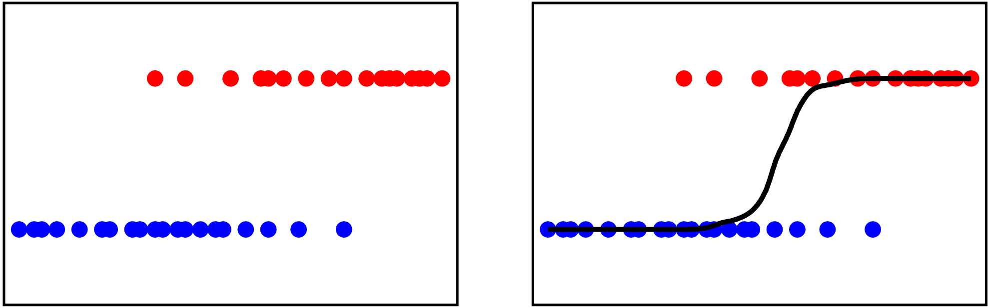
Logistic regression

- Responses may be qualitative (categorical)
  - Example: $\langle$Hours of study, pass/fail$\rangle$, $\langle$MRI scan, benign/malignant$\rangle$
  - Output should be 0 or 1
- Predicting qualitative response is known as classification
- Linear regression does not help

Issues with linear regression



Logistic regression



Logistic model

- Linear regression model to represent non-normalized probability $p'(x) = w_0 + w_1 x$
- To avoid problem, we use function $p(x) = \dfrac{e^{w_0 + w_1 x}}{1 + e^{w_0 + w_1 x}}$
- Quantity $\frac{p(x)}{1-p(x)} = e^{w_0 + w_1 x}$ is known as odds
- Taking $\log$ on both the sides, we get $\log\left(\dfrac{p(x)}{1 - p(x)}\right) = w_0 + w_1 x$
- Coefficient can be determined using maximum likelihood
  - $l(w_0, w_1) = \prod_{i:y_i=1} p(x_i) \prod_{j:y_j=0} p(x_j)$
- Similar to linear regression except the output is mapped between 0 and 1 ie.

$$p(y|\mathrm{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T \mathrm{x})$$

where $\sigma(x) = \dfrac{1}{1 + \exp(-x)}$ (Sigmoid function)

Maximum likelihood estimation

- Consider a set of m examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\mathrm{x})$
- Let $p_{model}(\mathrm{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution

Maximum likelihood estimation

- Consider a set of m examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\mathrm{x})$
- Let $p_{model}(\mathrm{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\mathrm{x}^{(i)}; \boldsymbol{\theta})$$

Maximum likelihood estimation

- Consider a set of m examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\mathrm{x})$
- Let $p_{model}(\mathrm{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\mathrm{x}^{(i)}; \boldsymbol{\theta})$$

- It can be written as $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{model}(\mathrm{x}^{(i)}; \boldsymbol{\theta})$

Maximum likelihood estimation

- Consider a set of m examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\mathrm{x})$
- Let $p_{model}(\mathrm{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\mathrm{x}^{(i)}; \boldsymbol{\theta})$$

- It can be written as $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{model}(\mathrm{x}^{(i)}; \boldsymbol{\theta})$
- By dividing m we get $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathrm{x} \sim p_{data}} \log p_{model}(\mathrm{x}; \boldsymbol{\theta})$

Maximum likelihood estimation (cont.)

- Minimizing dissimilarity between the empirical $\hat{p}_{data}$ and model distribution p_{model} and it is measured by KL divergence

$$D_{KL}(\hat{p}_{data} \| p_{model}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathsf{x} \sim \hat{p}_{data}} [\log \hat{p}_{data}(\mathsf{x}) - \log p_{model}(\mathsf{x}; \boldsymbol{\theta})]$$

Maximum likelihood estimation (cont.)

- Minimizing dissimilarity between the empirical $\hat{p}_{data}$ and model distribution p_{model} and it is measured by KL divergence

  $$D_{KL}(\hat{p}_{data} \| p_{model}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathsf{x} \sim \hat{p}_{data}} [\log \hat{p}_{data}(\mathsf{x}) - \log p_{model}(\mathsf{x}; \boldsymbol{\theta})]$$

- We need to minimize $-\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathsf{x} \sim \hat{p}_{data}} \log p_{model}(\mathsf{x}; \boldsymbol{\theta})$

Conditional log-likelihood

- In most of the supervised learning we estimate $P(\mathrm{y}|\mathrm{x};\boldsymbol{\theta})$
- If X be the all inputs and Y be observed targets then conditional maximum likelihood estimator is $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} P(\mathrm{Y}|\mathrm{X};\boldsymbol{\theta})$
- If the examples are assumed to be i.i.d then we can say

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\mathrm{y}^{(i)}|\mathrm{x}^{(i)};\boldsymbol{\theta})$$

Linear regression as maximum likelihood

- Instead of producing single prediction $\hat{y}$ for a given x, we assume the model produces conditional distribution $p(y|\mathrm{x})$
- For infinitely large training set, we can observe multiple examples having the same x but different values of y
- Goal is to fit the distribution $p(y|\mathrm{x})$

Linear regression as maximum likelihood

- Instead of producing single prediction $\hat{y}$ for a given x, we assume the model produces conditional distribution $p(y|\mathrm{x})$
- For infinitely large training set, we can observe multiple examples having the same x but different values of y
- Goal is to fit the distribution $p(y|\mathrm{x})$
- Let us assume, $p(y|\mathrm{x}) = \mathcal{N}(y; \hat{y}(\mathrm{x}; \mathrm{w}), \sigma^2)$

Linear regression as maximum likelihood

- Instead of producing single prediction $\hat{y}$ for a given x, we assume the model produces conditional distribution $p(y|\mathrm{x})$
- For infinitely large training set, we can observe multiple examples having the same x but different values of y
- Goal is to fit the distribution $p(y|\mathrm{x})$
- Let us assume, $p(y|\mathrm{x}) = \mathcal{N}(y; \hat{y}(\mathrm{x}; \mathrm{w}), \sigma^2)$
- Since the examples are assumed to be i.i.d, conditional log-likelihood is given by

$$\sum_{i=1}^{m} \log p(\mathrm{y}^{(i)}|\mathrm{x}^{(i)}; \boldsymbol{\theta})$$

Linear regression as maximum likelihood

- Instead of producing single prediction $\hat{y}$ for a given x, we assume the model produces conditional distribution $p(y|\mathrm{x})$
- For infinitely large training set, we can observe multiple examples having the same x but different values of y
- Goal is to fit the distribution $p(y|\mathrm{x})$
- Let us assume, $p(y|\mathrm{x}) = \mathcal{N}(y; \hat{y}(\mathrm{x}; \mathrm{w}), \sigma^2)$
- Since the examples are assumed to be i.i.d, conditional log-likelihood is given by

$$\sum_{i=1}^{m} \log p(\mathrm{y}^{(i)}|\mathrm{x}^{(i)}; \boldsymbol{\theta}) = -m \log \sigma - \frac{m}{2}\log(2\pi) - \sum_{i=1}^{m} \frac{\|\hat{y}^{(i)} - y^{(i)}\|^2}{2\sigma^2}$$

Learning conditional distributions

- Usually neural networks are trained using maximum likelihood. Therefore the cost function is negative log-likelihood. Also known as cross entropy between training data and model distribution
- Cost function $J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathsf{X},\mathsf{Y}\sim\hat{p}_{data}} \log p_{model}(\mathsf{y}|\mathsf{x}, \boldsymbol{\theta})$
- Uniform across different models
- Gradient of cost function is very much crucial
  - Large and predictable gradient can serve good guide for learning process
  - Function that saturates will have small gradient
    - Activation function usually produces values in a bounded zone (saturates)
  - Negative log-likelihood can overcome some of the problems
    - Output unit having exp function can saturate for high negative value
    - Log-likelihood cost function undoes the exp of some output functions

Learning conditional statistics

- Instead of learning the whole distribution $p(\mathrm{y}|\mathrm{x};\boldsymbol{\theta})$, we want to learn one conditional statistics of y given x
  - For a predicting function $f(\mathrm{x};\boldsymbol{\theta})$, we would like to predict the mean of y

Learning conditional statistics

- Instead of learning the whole distribution $p(\mathrm{y}|\mathrm{x};\boldsymbol{\theta})$, we want to learn one conditional statistics of y given x
  - For a predicting function $f(\mathrm{x};\boldsymbol{\theta})$, we would like to predict the mean of y
- Neural network can represent any function f from a very wide range of functions
- Range of function is limited by features like continuity, boundedness, etc.

Learning conditional statistics

- Instead of learning the whole distribution $p(\mathrm{y}|\mathrm{x};\boldsymbol{\theta})$, we want to learn one conditional statistics of y given x
  - For a predicting function $f(\mathrm{x};\boldsymbol{\theta})$, we would like to predict the mean of y
- Neural network can represent any function f from a very wide range of functions
- Range of function is limited by features like continuity, boundedness, etc.
- Cost function becomes functional rather than a function

Learning conditional statistics

- Need to solve the optimization problem

$$f^* = \arg\min_f \mathbb{E}_{\mathsf{X},\mathsf{Y}\sim p_{data}} \|\mathsf{y} - f(\mathsf{x})\|^2$$

Learning conditional statistics

- Need to solve the optimization problem

  $$f^* = \arg\min_f \mathbb{E}_{\mathsf{X},\mathsf{Y}\sim p_{data}} \|\mathsf{y} - f(\mathsf{x})\|^2$$

- Using calculus of variation, it gives $f^*(\mathsf{x}) = \mathbb{E}_{\mathsf{Y}\sim p_{data}(\mathsf{y}|\mathsf{x})}[y]$
  - Mean of y for each value of x

Learning conditional statistics

- Need to solve the optimization problem

  $f^* = \arg\min_f \mathbb{E}_{\mathsf{X},\mathsf{Y}\sim p_{data}} \|\mathsf{y} - f(\mathsf{x})\|^2$

- Using calculus of variation, it gives $f^*(\mathsf{x}) = \mathbb{E}_{\mathsf{Y}\sim p_{data}(\mathsf{y}|\mathsf{x})}[y]$
  - Mean of y for each value of x
- Using a different cost function $f^* = \arg\min_f \mathbb{E}_{\mathsf{X},\mathsf{Y}\sim p_{data}} \|\mathsf{y} - f(\mathsf{x})\|_1$

Learning conditional statistics

- Need to solve the optimization problem

  $$f^* = \arg\min_f \mathbb{E}_{\mathrm{X,Y} \sim p_{data}} \|\mathrm{y} - f(\mathrm{x})\|^2$$

- Using calculus of variation, it gives $f^*(\mathrm{x}) = \mathbb{E}_{\mathrm{Y} \sim p_{data}(\mathrm{y}|\mathrm{x})}[y]$
  - Mean of y for each value of x
- Using a different cost function $f^* = \arg\min_f \mathbb{E}_{\mathrm{X,Y} \sim p_{data}} \|\mathrm{y} - f(\mathrm{x})\|_1$
  - Median of y for each value of x

Thank you!