Introduction to Deep Learning

Arijit Mondal

Dept. of Computer Science & Engineering

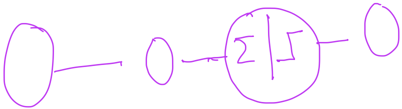
Indian Institute of Technology Patna

arijit@iitp.ac.in

Deep Feedforward Networks

Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron

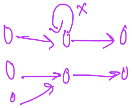


Deep feedforward networks

- 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

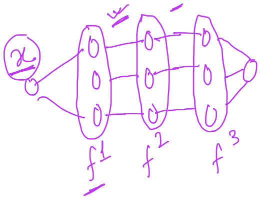
Deep feedforward networks

- 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

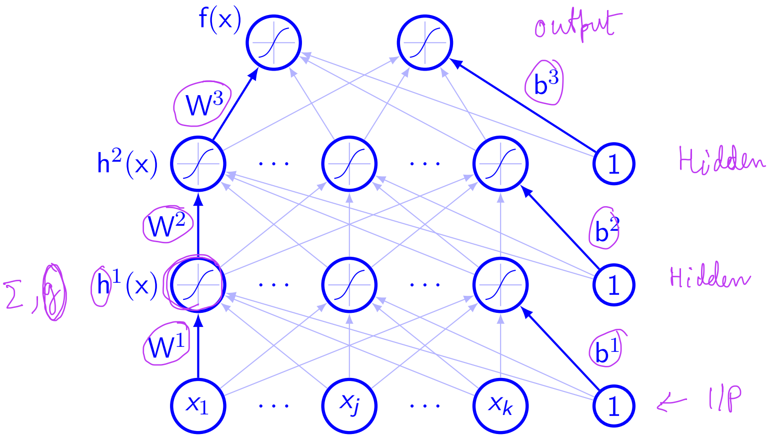
- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function f^*
  - For classifier, x is mapped to category y ie. $y = f^*(\mathrm{x})$
  - A feedforward network maps $y = f(\mathrm{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$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

Multilayer neural network

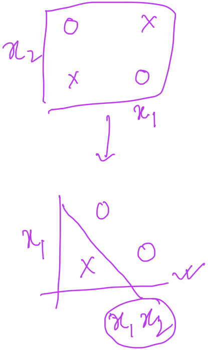


Issues with linear FFN

- 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 ϕ(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

$x+\epsilon$

Overcome issues of linear FFN

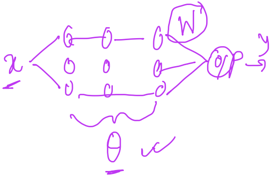
- Transform x (input) into $\phi(x)$ where ϕ is nonlinear transformation
- How to choose ϕ?
  - Use a very generic ϕ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough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

Goal of deep learning

- We have a model $y = f(\mathrm{x}; \boldsymbol{\theta}, \mathrm{w}) = \phi(\mathrm{x}; \boldsymbol{\theta})^T \mathrm{w}$
- 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 $\phi(\mathrm{x}, \boldsymbol{\theta})$
    - $\boldsymbol{\theta}$ can be determined by solving optimization problem
- Advantages
    - ϕ can be very generic
    - Human practitioner can encode their knowledge to designing $\phi(\mathrm{x}; \boldsymbol{\theta})$



Design issues of feedforward network

- Choice of optimizer
- Cost function
- The form of output unit
- Choice of activation function
- Design of architecture - number of layers, number of units in each layer
- Computation of gradients

Example

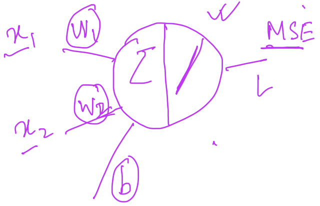
- Let us choose XOR function
- Target function is $y = f^*(\mathrm{x})$ and our model provides $y = f(\mathrm{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make f close to f^*

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 $\left(J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\mathrm{x}\in\mathrm{X}} (f^*(\mathrm{x}) - f(\mathrm{x};\boldsymbol{\theta}))^2\right)$
- We need to choose $f(\mathrm{x};\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ depends on w and b
- Let us consider a linear model $f(\mathrm{x};\mathrm{w},b) = \mathrm{x}^T\mathrm{w} + b$

-1.5

0 1 1 0



$(0 - b)^2$

$$J(\theta) = \frac{1}{4}\Big(\big(0 - \underbrace{(w_1x_1 + w_2x_2 + b)}_{x_1=0,\ x_2=0}\big)^2 + \big(1 - \underbrace{(w_1x_1 + w_2x_2 + b)}_{x_1=0,\ x_2=1}\big)^2 + 2 + 2 \Big)$$

$x_1 = 0,\ x_2 = 0$

$(1 - w_2 - b)^2$

10 11

$-\infty < W < \infty$

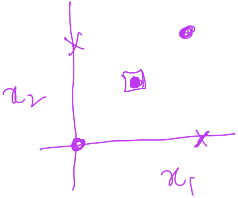
$$\frac{\partial J}{\partial w_1} = 0 \qquad \frac{\partial J}{\partial w_2} = 0 \qquad \frac{\partial J}{\partial b} = 0$$

$\theta = [w_1\ w_2\ b]$

CS551

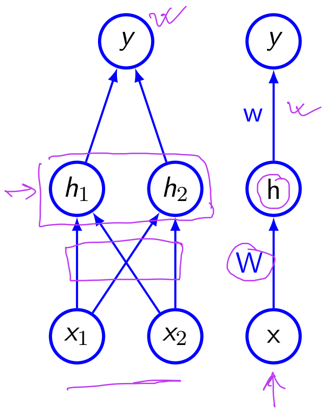
Example

- 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 $\left(J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\mathrm{x}\in\mathrm{X}}(f^*(\mathrm{x}) - f(\mathrm{x};\boldsymbol{\theta}))^2\right)$
- 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 $b = \frac{1}{2}$



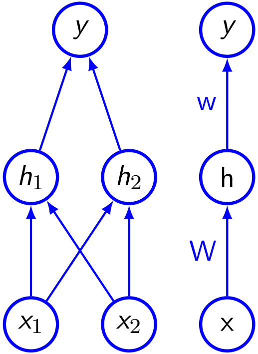
Simple FFN with hidden layer

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



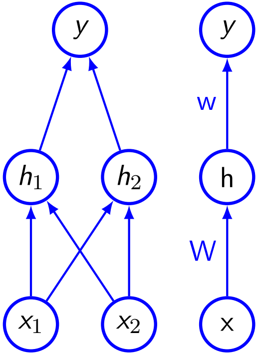
Simple FFN with hidden layer

- 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



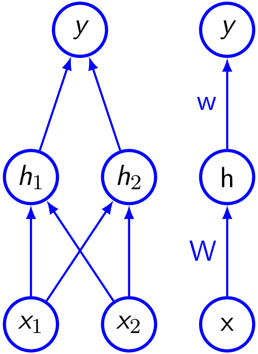
Simple FFN with hidden layer

- 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

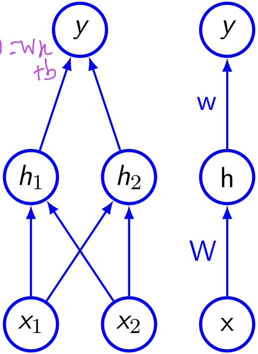
- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x}; \mathrm{W}, \mathrm{c})$
- In the next layer $y = f^{(2)}(\mathrm{h}; \mathrm{w}, b)$ is computed
- Complete model $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = f^{(2)}(f^{(1)}(\mathrm{x}))$
- Suppose $f^{(1)}(\mathrm{x})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



Simple FFN with hidden layer

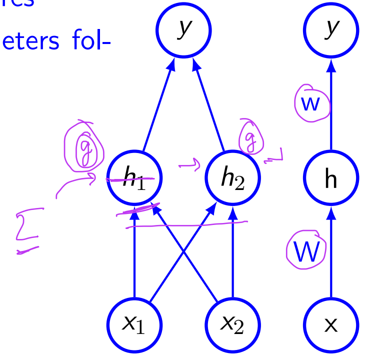
- Let us assume that the hidden unit h computes $f^{(1)}(\mathrm{x};\mathrm{W},\mathrm{c})$
- In the next layer $y = f^{(2)}(\mathrm{h};\mathrm{w},b)$ is computed
- Complete model $f(\mathrm{x};\mathrm{W},\mathrm{c},\mathrm{w},b) = f^{(2)}(f^{(1)}(\mathrm{x}))$
- Suppose $f^{(1)}(\mathrm{x}) = \mathrm{W}^T\mathrm{x}$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

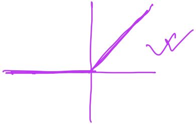
$f(x) = wx + b$



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 $\mathbf{h} = g(\mathbf{W}^T\mathbf{x} + \mathbf{c})$
- Let us use ReLU as activation function $g(z) = \max\{0, z\}$
- g is chosen element wise $h_i = g(\mathbf{x}^T\mathbf{W}_{:,i} + c_i)$





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

XOR

0	0	→	0
1	0	→	1
0	1	→	1
1	1	→	0

CS551

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 $f(\mathsf{x}; \mathsf{W}, \mathsf{c}, \mathsf{w}, b) = \mathsf{w}^T \max\{0, \mathsf{W}^T\mathsf{x} + \mathsf{c}\} + b$
- 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
  - $\mathsf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, XW

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
  - $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$ $+ c$,

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

Simple FFN with hidden layer (contd.)

- Complete network is $f(\mathrm{x}; \mathrm{W}, \mathrm{c}, \mathrm{w}, b) = \mathrm{w}^T \underbrace{\max\{0, \mathrm{W}^T\mathrm{x} + \mathrm{c}\}}_{\text{ReLU}} + b$
- 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 $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$,

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 $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, apply h

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 $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, apply h $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$,

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 $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, apply h $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, multiply

    with 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

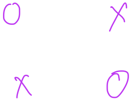
$x \xrightarrow{W} h \xrightarrow{w} o/p$

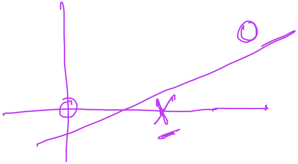
- $\mathrm{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathrm{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathrm{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $b = 0$

h

- 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 $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, apply h $\begin{array}{c} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \\ x_1 \quad x_2 \end{array}$, 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)

y

$x \to W \to o/p$

error

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 $f(x - \epsilon\,\mathrm{sign}(f'(x)))$ is less than $f(x)$

  sgn$(f'(x))$
- 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 $\mathrm{x}' = \mathrm{x} - \epsilon \nabla_{\mathrm{x}} f(\mathrm{x})$ where ϵ is the learning rate

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 $O(m)$

m

$\rightarrow m'$ $\quad m' \lesssim m$

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 $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$ 10L
  - Computation cost is $O(m)$
- For SGD, gradient is an expectation estimated from a small sample known as minibatch ($\mathbb{B} = \{x^{(1)}, \ldots, x^{(m')}\}$) 100
- Estimated gradient is $\mathrm{g} = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\mathrm{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- New point will be $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \mathrm{g}$

$m' \leq m$

CS551

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

batch size - 1

- MSE as cost function. Derivative will be $x(w \times x - y)$

Step	Point	Derivative		New w
1	(1,2)	1(4×1−2) = 2	?	$\cancel{2}$
2	(2,4)	2(3·8×2−4) = 7·2		

$4 - 0.1 \times 2 = 3.8$

$3.8 - 0.1 \times 7.2 =$

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 - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$ 2x
- Let us try to fit a curve as follows $y = w \times x$ where w is initialized with 4, learning rate as 0.1 2
- 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	(4,8)	1*(2.0*1-2)=0.0	2.00

(1,2) (2,4)

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 $\frac{1}{4}\sum_i x_i(w \times x_i - y_i)$

batch 4
2

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

(1,2) (2,4) - 1
(3,6) (4,8) - 2

20 → (1,2) (3,6) - 3
→ (2,4) (4,8) - 4

100

10

10

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

$MSE + \lambda w^2$

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 $p_{model}(\mathrm{x}; \theta)$ 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

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

$0 \leq p \leq 1$

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

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

CS551

Maximum likelihood estimation (cont.)

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

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

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

$$\boxed{\arg\max_{\theta} \mathbb{E}_{x \sim p_{data}} \log p_{model}(x, \theta)}$$

x

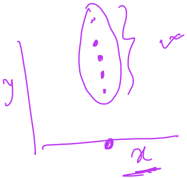
Conditional log-likelihood

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

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

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

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

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, $p(y|\mathrm{x}) = \mathcal{N}(y; \hat{y}(\mathrm{x}; \mathrm{w}), \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}}\ e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$
- Since the examples are assumed to be i.i.d, conditional log-likelihood is given by

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

σ^{-2}

$\arg\max_{\theta} - \{\text{MSE}\}$

$\arg\min_{\theta} \text{MSE}$

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

e^{-x}

$x \gg 1$

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

$\log p()$

$\log \sigma$

$\log (\exp) \rightarrow x$

Learning conditional statistics

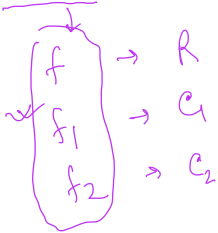
- Instead of learning the whole distribution $p(y|x;\boldsymbol{\theta})$, we want to learn one conditional statistics of y given x
  - For a predicting function $f(x;\boldsymbol{\theta})$, we would like to predict the mean of y

Learning conditional statistics

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

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

- 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 $f^* = \arg\min_{f} \mathbb{E}_{\mathsf{X},\mathsf{Y}\sim p_{data}} \|\underline{\mathsf{y} - f(\mathsf{x})}\|_1$ ✓ L^1

Learning conditional statistics

- Need to solve the optimization problem

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

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

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} L(x, y(x), y'(x))\, dx$

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} L(x, y(x), y'(x))\, dx$
- Let $J[y]$ has local minima at f. Therefore, we can say $J[f] \leq J[f + \varepsilon\eta]$
  - η is an arbitrary function of x such that $\eta(x_1) = \eta(x_2) = 0$ and differentiable

CS551

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} \underline{L(x, y(x), y'(x))}\, dx$
- Let $J[y]$ has local minima at f. Therefore, we can say $J[f] \leq J[f + \varepsilon\eta]$
  - η is an arbitrary function of x such that $\eta(x_1) = \eta(x_2) = 0$ and differentiable

$\underline{\epsilon = 0}$

- Let us assume $\Phi(\varepsilon) = J[f + \varepsilon\eta]$. Therefore, $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx$

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} L(x, y(x), y'(x))\, dx$
- Let $J[y]$ has local minima at f. Therefore, we can say $J[f] \leq J[f + \varepsilon\eta]$
  - η is an arbitrary function of x such that $\eta(x_1) = \eta(x_2) = 0$ and differentiable
- Let us assume $\Phi(\varepsilon) = J[f + \varepsilon\eta]$. Therefore, $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx = 0$

CS551

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} L(x, y(x), y'(x))\, dx$
- Let $J[y]$ has local minima at f. Therefore, we can say $J[f] \leq J[f + \varepsilon\eta]$
  - η is an arbitrary function of x such that $\eta(x_1) = \eta(x_2) = 0$ and differentiable
- Let us assume $\Phi(\varepsilon) = J[f + \varepsilon\eta]$. Therefore, $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx = 0$
- Now we can say, $\frac{dL}{d\varepsilon} = \frac{\partial L}{\partial y}\frac{dy}{d\varepsilon} + \frac{\partial L}{\partial y'}\frac{dy'}{d\varepsilon}$

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} L(x, y(x), y'(x))\, dx$
- Let $J[y]$ has local minima at f. Therefore, we can say $J[f] \leq J[f + \varepsilon\eta]$
  - η is an arbitrary function of x such that $\eta(x_1) = \eta(x_2) = 0$ and differentiable
- Let us assume $\Phi(\varepsilon) = J[f + \varepsilon\eta]$. Therefore, $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx = 0$
- Now we can say, $\frac{dL}{d\varepsilon} = \frac{\partial L}{\partial y}\frac{dy}{d\varepsilon} + \frac{\partial L}{\partial y'}\frac{dy'}{d\varepsilon}$
- As we have $y = f + \varepsilon\eta$ and $y' = f' + \varepsilon\eta'$, therefore, $\frac{dL}{d\varepsilon}$

  x

Calculus of variations

- Let us consider functional $J[y] = \int_{x_1}^{x_2} L(x, y(x), y'(x))\, dx$
- Let $J[y]$ has local minima at f. Therefore, we can say $J[f] \leq J[f + \varepsilon\eta]$
  - η is an arbitrary function of x such that $\eta(x_1) = \eta(x_2) = 0$ and differentiable
- Let us assume $\Phi(\varepsilon) = J[f + \varepsilon\eta]$. Therefore, $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \boxed{\int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx = 0}$
- Now we can say, $\frac{dL}{d\varepsilon} = \frac{\partial L}{\partial y}\frac{dy}{d\varepsilon} + \frac{\partial L}{\partial y'}\frac{dy'}{d\varepsilon}$
- As we have $y = f + \varepsilon\eta$ and $y' = f' + \varepsilon\eta'$, therefore, $\frac{dL}{d\varepsilon} = \frac{\partial L}{\partial y}\eta + \frac{\partial L}{\partial y'}\eta'$

Calculus of variations (contd.)

- Now we have

$$\int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx = \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta + \frac{\partial L}{\partial f'}\eta' \right) dx$$

Calculus of variations (contd.)

- Now we have

$$\begin{aligned}
\int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx &= \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta + \frac{\partial L}{\partial f'}\eta' \right) dx \\
&= \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta - \eta \frac{d}{dx}\frac{\partial L}{\partial f'} \right) dx + \left.\frac{\partial L}{\partial f'}\eta\right|_{x_1}^{x_2}
\end{aligned}$$

Calculus of variations (contd.)

- Now we have

$$\begin{aligned}\int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx &= \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta + \frac{\partial L}{\partial f'}\eta'\right) dx \\ &= \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta - \eta\frac{d}{dx}\frac{\partial L}{\partial f'}\right) dx + \left.\frac{\partial L}{\partial f'}\eta\right|_{x_1}^{x_2}\end{aligned}$$

- Hence $\int_{x_1}^{x_2} \eta \left(\frac{\partial L}{\partial f} - \frac{d}{dx}\frac{\partial L}{\partial f'}\right) dx = 0$

Calculus of variations (contd.)

- Now we have

$$\begin{aligned}\int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx &= \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta + \frac{\partial L}{\partial f'}\eta'\right) dx \\ &= \int_{x_1}^{x_2} \left(\frac{\partial L}{\partial f}\eta - \eta\frac{d}{dx}\frac{\partial L}{\partial f'}\right) dx + \left.\frac{\partial L}{\partial f'}\eta\right|_{x_1}^{x_2}\end{aligned}$$

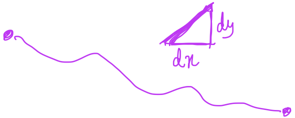
- Hence $\displaystyle\int_{x_1}^{x_2} \eta\left(\frac{\partial L}{\partial f} - \frac{d}{dx}\frac{\partial L}{\partial f'}\right) dx = 0$
- Euler-Lagrange equation $\displaystyle\boxed{\frac{\partial L}{\partial f} - \frac{d}{dx}\frac{\partial L}{\partial f'} = 0}$

Example

- Let us consider distance between two points $A[y] = \int_{x_1}^{x_2} \sqrt{1 + [y'(x)]^2}\, dx$

  y $\quad 1 + \left(\frac{dy}{dx}\right)^2$

  - $y'(x) = \dfrac{dy}{dx}\,, \quad y_1 = f(x_1)\,, \quad y_2 = f(x_2)$



Example

- Let us consider distance between two points $A[y] = \int_{x_1}^{x_2} \sqrt{1 + [y'(x)]^2}\, dx$
  - $y'(x) = \dfrac{dy}{dx}\,, \quad y_1 = f(x_1)\,, \quad y_2 = f(x_2)$
- We have, $\dfrac{\partial L}{\partial f} - \dfrac{d}{dx}\dfrac{\partial L}{\partial f'} = 0$ where $L = \sqrt{1 + [f'(x)]^2}$

Example

- Let us consider distance between two points $A[y] = \int_{x_1}^{x_2} \sqrt{1 + [y'(x)]^2}\, dx$
  - $y'(x) = \dfrac{dy}{dx}\,, \quad y_1 = f(x_1)\,, \quad y_2 = f(x_2)$
- We have, $\dfrac{\partial L}{\partial f} - \dfrac{d}{dx}\dfrac{\partial L}{\partial f'} = 0$ where $L = \sqrt{1 + [f'(x)]^2}$
- As f does not appear explicitly in L, hence $\dfrac{d}{dx}\dfrac{\partial L}{\partial f'} = 0$

Example

- Let us consider distance between two points $A[y] = \int_{x_1}^{x_2} \sqrt{1 + [y'(x)]^2}\, dx$
  - $y'(x) = \dfrac{dy}{dx}\,, \quad y_1 = f(x_1)\,, \quad y_2 = f(x_2)$
- We have, $\dfrac{\partial L}{\partial f} - \dfrac{d}{dx}\dfrac{\partial L}{\partial f'} = 0$ where $L = \sqrt{1 + [f'(x)]^2}$
- As f does not appear explicitly in L, hence $\dfrac{d}{dx}\dfrac{\partial L}{\partial f'} = 0$
- Now we have, $\dfrac{d}{dx}\dfrac{f'(x)}{\sqrt{1 + [f'(x)]^2}} = 0$

Example

- Taking derivative we get $\dfrac{d^2 f}{dx^2} \cdot \dfrac{1}{\left[\sqrt{1+[f'(x)]^2}\,\right]^3} = 0$

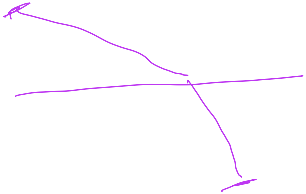
Example

- Taking derivative we get $\frac{d^2 f}{dx^2} \cdot \frac{1}{\left[\sqrt{1+[f'(x)]^2}\,\right]^3} = 0$

- Therefore we have, $\frac{d^2 f}{dx^2} = 0$

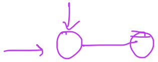
Example

- Taking derivative we get $\frac{d^2 f}{dx^2} \cdot \frac{1}{\left[\sqrt{1+[f'(x)]^2}\right]^3} = 0$
- Therefore we have, $\frac{d^2 f}{dx^2} = 0$
- Hence we have $f(x) = mx + b$ with $m = \frac{y_2 - y_1}{x_2 - x_1}$ and $b = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$



Output units

- Choice of cost function is directly related with the choice of output function
- In most cases cost function is determined by cross entropy between data and model distribution
- Any kind of output unit can be used as hidden unit



Linear units

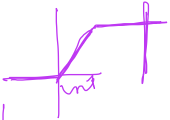
- Suited for Gaussian output distribution
- Given features h, linear output unit produces $\hat{\mathrm{y}} = \mathrm{W}^T\mathrm{h} + \mathrm{b}$ x
- This can be treated as conditional probability $p(\mathrm{y}|\mathrm{x}) = \mathcal{N}(\mathrm{y}; \hat{\mathrm{y}}, \mathbf{I})$ MSE
- Maximizing log-likelihood is equivalent to minimizing mean square error

Sigmoid unit

- Mostly suited for binary classification problem that is Bernoulli output distribution
- The neural networks need to predict $p(y = 1|\mathrm{x})$ $p(y=0|x)$
  - If linear unit has been chosen, $p(y = 1|\mathrm{x}) = \max\left\{0, \min\{1, \mathrm{W}^T\mathrm{h} + \mathrm{b}\}\right\}$
  - Gradient? $\rightarrow$

$0 < x < 1$

- Model should have strong gradient whenever the answer is wrong



- Let us assume unnormalized log probability is linear with $\boxed{z = \mathrm{W}^T\mathrm{h} + \mathrm{b}}$
- Therefore, $\log \tilde{P}(y) = yz \Rightarrow \tilde{P}(y) = \exp(yz) \Rightarrow P(y) = \frac{\exp(yz)}{\sum_{y' \in \{0,1\}} \exp(y'z)}$

$0-1$

  - It can be written as $P(y) = \sigma((2y - 1)z)$

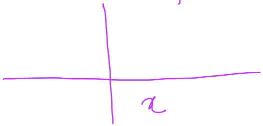
$y \in 0, 1$

- The loss function for maximum likelihood is
$J(\boldsymbol{\theta}) = -\log P(y|\mathrm{x}) = -\log \sigma((2y-1)z) = \zeta((1-2y)z)$

soft plus

$\downarrow$

$\zeta(x) = \log(1 + \exp(x))$



Softmax unit

- Similar to sigmoid. Mostly suited for multinoulli distribution
- We need to predict a vector $\hat{y}$ such that $\hat{y}_i = P(Y = i|x)$
- A linear layer predicts unnormalized probabilities $z = W^T h + b$ that is $z_i = \log \tilde{P}(y = i|x)$
- Formally, $\text{softmax}(z)_i = \frac{\exp z_i}{\sum_j \exp(z_j)}$
- Log in log-likelihood can undo exp $\log \text{softmax}(z)_i = z_i - \log \sum_j \exp(z_j)$
  - Does it saturate?
  - What about incorrect prediction?
- Invariant to addition of some scalar to all input variables ie. $\text{softmax}(z) = \text{softmax}(z + c)$

≈ 0

$i = j$

$\exp(z_i)$ z_i

$z_1\ z_2\ z_i\ z \ldots$

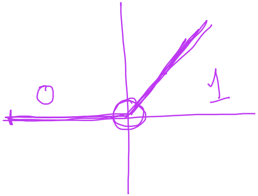
$\sum_j \exp(z_j) \approx \exp(z_i)$

Hidden units

- Active area of research and does not have good guiding theoretical principle
- Usually rectified linear unit (ReLU) is chosen in most of the cases
- Design process consists of trial and error, then the suitable one is chosen
- Some of the activation functions are not differentiable (eg. ReLU)
  - Still gradient descent performs well
  - Neural network does not converge to local minima but reduces the value of cost function to a very small value

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

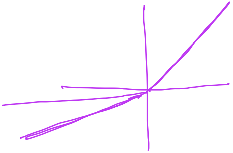
$$\boxed{\frac{\partial \sigma}{\partial x} = \sigma(x)\left(1-\sigma(x)\right)}$$



$\max\{0, x\}$

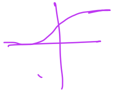
Generalization of ReLU

- ReLU is defined as $g(z) = \max\{0, z\}$
- Using non-zero slope, $h_i = g(\mathrm{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$
  - Absolute value rectification will make $\alpha_i = -1$ and $g(z) = |z|$
- Leaky ReLU assumes very small values for α_i
- Parametric ReLU tries to learn α_i parameters
- Maxout unit $g(\mathrm{z})_i = \max\limits_{j \in \mathbb{G}^{(i)}} z_j$
  - Suitable for learning piecewise linear function

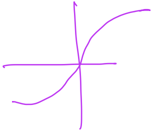


Logistic sigmoid & hyperbolic tangent

- Logistic sigmoid $g(z) = \sigma(z)$
- Hyperbolic tangent $g(z) = tanh(z)$ ← RNN
  - $tanh(z) = 2\sigma(2z) - 1$

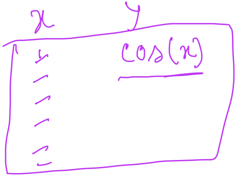


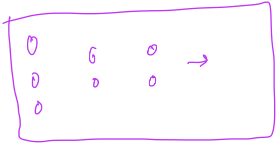
- Widespread saturation of sigmoidal unit is an issue for gradient based learning
  - Usually discouraged to use as hidden units
- Usually, hyperbolic tangent function performs better where sigmoidal function must be used
  - Behaves linearly at 0
  - Sigmoidal activation function are more common in settings other than feedforward network



Other hidden units

- Differentiable functions are usually preferred
- Activation function $h = \cos(\mathsf{W}\mathsf{x} + \mathsf{b})$ performs well for MNIST data set
- Sometimes no activation function helps in reducing the number of parameters
- Radial Basis Function - $\phi(\mathsf{x}, \mathsf{c}) = \phi(\|\mathsf{x} - \mathsf{c}\|)$
  - Gaussian - $\exp(-(\varepsilon r)^2)$
- Softplus - $g(x) = \zeta(x) = \log(1 + exp(x))$
- Hard tanh - $g(x) = \max(-1, \min(1, x))$
- Hidden unit design is an active area of research





CS551

Architecture design

- Structure of neural network (chain based architecture)
  - Number of layers
  - Number of units in each layer
  - Connectivity of those units
- Single hidden layer is sufficient to fit the training data
- Often deeper networks are preferred
  - Fewer number of units
  - Fewer number of parameters
  - Difficult to optimize

Next Quiz - 16 th Feb