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, $\boldsymbol{x}$ is mapped to category y ie. $y = f^*(\boldsymbol{x})$**
  - **A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ 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, $\boldsymbol{x}$ is mapped to category y ie. $y = f^*(\boldsymbol{x})$**
  - **A feedforward network maps $y = f(\boldsymbol{x}; \theta)$ 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**

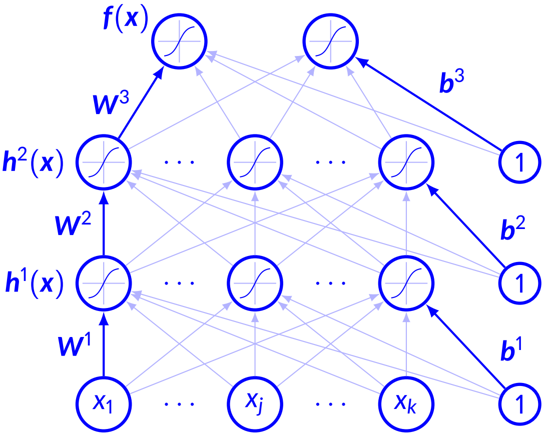
Deep feedforward networks

- **Also known as feedforward neural network or multilayer perceptron**
- **Goal of such network is to approximate some function f^***
  - For classifier, $\boldsymbol{x}$ is mapped to category y ie. $y = f^*(\boldsymbol{x})$
  - A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ 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, $\boldsymbol{x}$ is mapped to category y ie. $y = f^*(\boldsymbol{x})$**
  - **A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ 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 $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ 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 $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ 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 $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ 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 $y = f(\mathbf{x}; \boldsymbol{\theta}, \mathbf{w}) = \phi(\mathbf{x}; \boldsymbol{\theta})^T \mathbf{w}$
- We use $\boldsymbol{\theta}$ to learn ϕ
- $\mathbf{w}$ 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 $\phi(\mathbf{x}, \boldsymbol{\theta})$
  - $\boldsymbol{\theta}$ can be determined by solving optimization problem
- Advantages
  - ϕ can be very generic
  - Human practitioner can encode their knowledge to designing $\phi(\mathbf{x}; \boldsymbol{\theta})$

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

Example

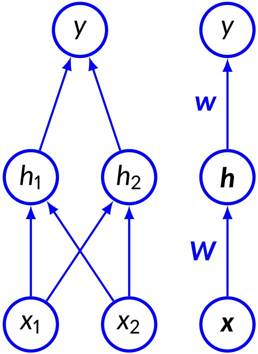
- **Let us choose XOR function**
- **Target function is** $y = f^*(\boldsymbol{x})$ **and our model provides** $y = f(\boldsymbol{x}; \boldsymbol{\theta})$
- **Learning algorithm will choose the parameters** $\boldsymbol{\theta}$ **to make** f **close to** f^*
- **Target is to fit output for** $\boldsymbol{X}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

Example

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

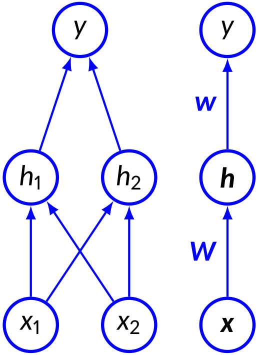
Simple FFN with hidden layer

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



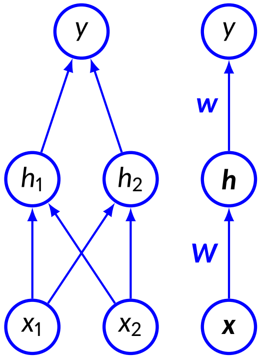
Simple FFN with hidden layer

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



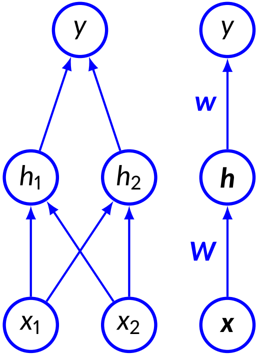
Simple FFN with hidden layer

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



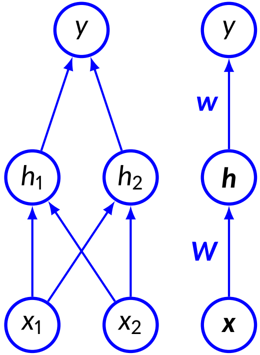
Simple FFN with hidden layer

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



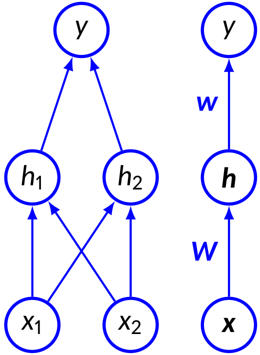
Simple FFN with hidden layer

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



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



Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

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
  - $\mathbf{X}$

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

Simple FFN with hidden layer (contd.)

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

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 $f(x - \epsilon \mathbf{sign}(f'(x)))$ is less than $f(x)$
- For multiple inputs partial derivatives are used ie. $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$
- Gradient vector is represented as $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
- Gradient descent proposes a new point as $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(x)$ where ϵ is the learning rate

Stochastic gradient descent

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

Stochastic gradient descent

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

Stochastic gradient descent

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

Cost function

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

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 $p_{data}(\boldsymbol{x})$**
- **Let $p_{model}(\boldsymbol{x}; \theta)$ 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 $p_{data}(\boldsymbol{x})$
- Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

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

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 $p_{data}(\boldsymbol{x})$
- Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

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

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

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 $p_{data}(\boldsymbol{x})$**
- **Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution**
- **Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as**

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

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

Maximum likelihood estimation (cont.)

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

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

Maximum likelihood estimation (cont.)

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

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

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

Conditional log-likelihood

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

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

Linear regression as maximum likelihood

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

Linear regression as maximum likelihood

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

Linear regression as maximum likelihood

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

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

Linear regression as maximum likelihood

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

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

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 $J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$
- 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 $p(\boldsymbol{y}|\boldsymbol{x};\theta)$, we want to learn one conditional statistics of $\boldsymbol{y}$ given $\boldsymbol{x}$**
  - **For a predicting function $f(\boldsymbol{x};\theta)$, we would like to predict the mean of $\boldsymbol{y}$**

Learning conditional statistics

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

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

Learning conditional statistics

- **Need to solve the optimization problem**

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

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

Learning conditional statistics

- **Need to solve the optimization problem**

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

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

Learning conditional statistics

- **Need to solve the optimization problem**

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

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

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**

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,** $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx$

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,** $\Phi'(0) \equiv \left.\frac{d\Phi}{d\varepsilon}\right|_{\varepsilon=0} = \int_{x_1}^{x_2} \left.\frac{dL}{d\varepsilon}\right|_{\varepsilon=0} dx = 0$

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, $\frac{dL}{d\varepsilon}$

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, $\frac{dL}{d\varepsilon} = \frac{\partial L}{\partial y}\eta + \frac{\partial L}{\partial y'}\eta'$

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**

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

Calculus of variations (contd.)

- **Now we have**

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

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

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

- **Euler-Lagrange equation** $\frac{\partial L}{\partial f} - \frac{d}{dx}\frac{\partial L}{\partial f'} = 0$

Example

- Let us consider distance between two points $A[y] = \int_{x_1}^{x_2} \sqrt{1 + [y'(x)]^2}\, dx$
  - $y'(x) = \frac{dy}{dx}\,, \;\; y_1 = f(x_1)\,, \;\; y_2 = f(x_2)$

Example

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

Example

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

Example

- Let us consider distance between two points $A[y] = \int_{x_1}^{x_2} \sqrt{1 + [y'(x)]^2}\, dx$
  - $y'(x) = \dfrac{dy}{dx}, \; y_1 = f(x_1), \; y_2 = f(x_2)$
- 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** $\frac{d^2f}{dx^2} \cdot \frac{1}{\left[\sqrt{1 + [f'(x)]^2}\right]^3} = 0$

Example

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

Example

- **Taking derivative we get** $\dfrac{d^2f}{dx^2} \cdot \dfrac{1}{\left[\sqrt{1+[f'(x)]^2}\right]^3} = 0$
- **Therefore we have,** $\dfrac{d^2f}{dx^2} = 0$
- **Hence we have** $f(x) = mx + b$ **with** $m = \dfrac{y_2 - y_1}{x_2 - x_1}$ **and** $b = \dfrac{x_2y_1 - x_1y_2}{x_2 - x_1}$

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 $\boldsymbol{h}$, linear output unit produces $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$
- This can be treated as conditional probability $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$
- 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** $p(y=1|\boldsymbol{x})$
  - **If linear unit has been chosen,** $p(y=1|\boldsymbol{x}) = \max\left\{0, \min\{1, \boldsymbol{W}^T\boldsymbol{h}+\boldsymbol{b}\}\right\}$
  - **Gradient?**
- **Model should have strong gradient whenever the answer is wrong**
- **Let us assume unnormalized log probability is linear with** $z = \boldsymbol{W}^T\boldsymbol{h}+\boldsymbol{b}$
- **Therefore,** $\log \tilde{P}(y) = yz \Rightarrow \tilde{P}(y) = \exp(yz) \Rightarrow P(y) = \frac{\exp(yz)}{\sum_{y'\in\{0,1\}}\exp(y'z)}$
  - **It can be written as** $P(y) = \sigma((2y-1)z)$
- **The loss function for maximum likelihood is** $J(\boldsymbol{\theta}) = -\log P(y|\boldsymbol{x}) = -\log\sigma((2y-1)z) = \zeta((1-2y)z)$

Softmax unit

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

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**

Generalization of ReLU

- **ReLU is defined as** $g(z) = \max\{0, z\}$
- **Using non-zero slope,** $h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$
  - **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** $g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j$
  - **Suitable for learning piecewise linear function**

Logistic sigmoid & hyperbolic tangent

- **Logistic sigmoid** $g(z) = \sigma(z)$
- **Hyperbolic tangent** $g(z) = tanh(z)$
  - $tanh(z) = 2\sigma(2z) - 1$
- **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** $h = \cos(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ **performs well for MNIST data set**
- **Sometimes no activation function helps in reducing the number of parameters**
- **Radial Basis Function -** $\phi(\boldsymbol{x}, \boldsymbol{c}) = \phi(\|\boldsymbol{x} - \boldsymbol{c}\|)$
  - **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**

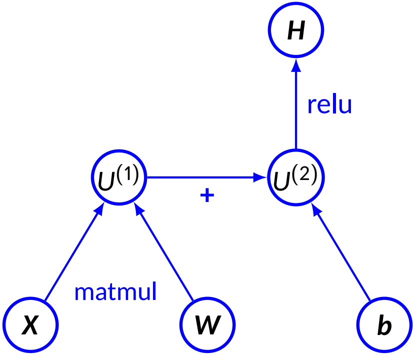
Architecture design

- 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

Back propagation

- In a feedforward network, an input x is read and produces an output $\hat{y}$
  - This is forward propagation
- During training forward propagation continues until it produces cost $J(\theta)$
- Back-propagation algorithm allows the information to flow backward in the network to compute the gradient
- Computation of analytical expression for gradient is easy
- We need to find out gradient of the cost function with respect to the parameters ie. $\nabla_\theta J(\theta)$

Computational graph



Chain rule of calculus

- **Back-propagation algorithm heavily depends on it**
- **Let x be a real number and $y = g(x)$ and $z = f(g(x)) = f(y)$**
- **Chain rule says $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$**
- **This can be generalized: Let $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R} \to \mathbb{R}$ and $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$ then $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$**
- **In vector notation it will be where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g**

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z$$

Application of chain rule

- Let us consider $u^{(n)}$ be the loss quantity. Need to find out the gradient for this.
- Let $u^{(1)}$ to $u^{(n_i)}$ are the inputs
- Therefore, we wish to compute $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ where $i = 1, 2, \ldots, n_i$
- Let us assume the nodes are ordered so that we can compute one after another
- Each $u^{(i)}$ is associated with an operation $f^{(i)}$ ie. $u^{(i)} = f(\mathbb{A}^{(i)})$

Algorithm for forward pass

for $i = 1, \ldots, n_i$ **do**

$u^{(i)} \leftarrow x_i$

end for

for $i = n_i + 1, \ldots, n$ **do**

$\mathbb{A}^{(i)} \leftarrow \{u^{(j)} | j \in Pa(u^{(i)})\}$

$u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$

end for

return $u^{(n)}$

Algorithm for backward pass

$\texttt{grad\_table}[u^{(n)}] \leftarrow 1$

for $j = n - 1$ **down to 1 do**

$$\texttt{grad\_table}[u^{(j)}] \leftarrow \sum_{i:j \in Pa(u^{(i)})} \texttt{grad\_table}[u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$$

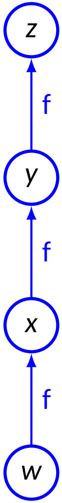
end for

return $\texttt{grad\_table}$

Computational graph & subexpression

- We have $x = f(w)$, $y = f(x)$, $z = f(y)$

$$
\begin{aligned}
\frac{\partial z}{\partial w} \\
&= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\
&= f'(y)f'(x)f'(w) \\
&= f'(f(f(w)))f'(f(w))f'(w)
\end{aligned}
$$



Forward propagation in MLP

- Input
  - $\boldsymbol{h}^{(0)} = \boldsymbol{x}$
- Computation for each layer $k = 1, \ldots, l$
  - $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}$
  - $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
- Computation of output and loss function
  - $\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
  - $J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda\Omega(\theta)$

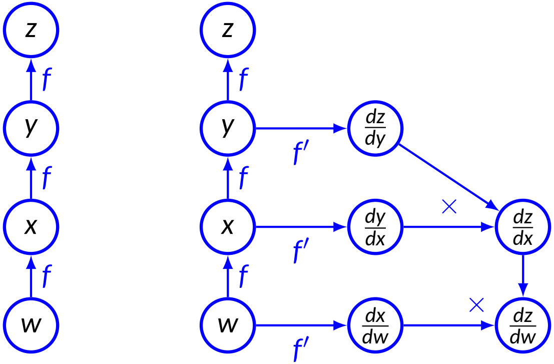
Backward computation in MLP

- **Compute gradient at the output**
  - $\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
- **Convert the gradient at output layer into gradient of pre-activation**
  - $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
- **Compute gradient on weights and biases**
  - $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
  - $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\boldsymbol{h}^{(k-1)T} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
- **Propagate the gradients wrt the next lower level activation**
  - $g \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)T} \boldsymbol{g}$

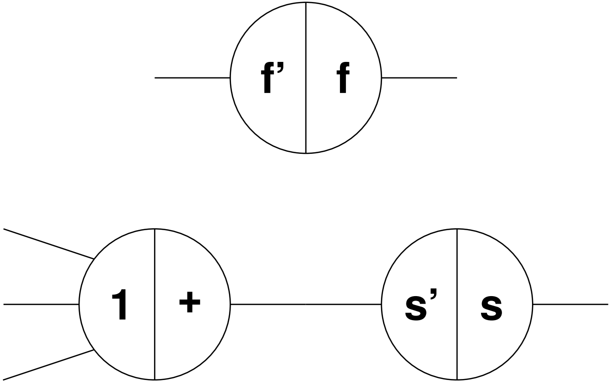
Computation of derivatives

- **Takes a computational graph and a set of numerical values for the inputs, then return a set of numerical values**
  - Symbol-to-number differentiation
  - Torch, Caffe
- **Takes computational graph and add additional nodes to the graph that provide symbolic description of derivative**
  - Symbol-to-symbol derivative
  - Theano, TensorFlow

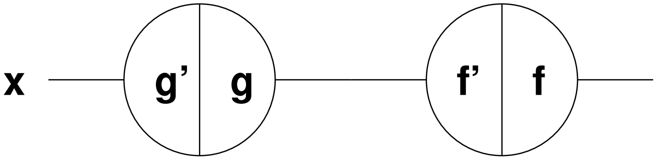
Example



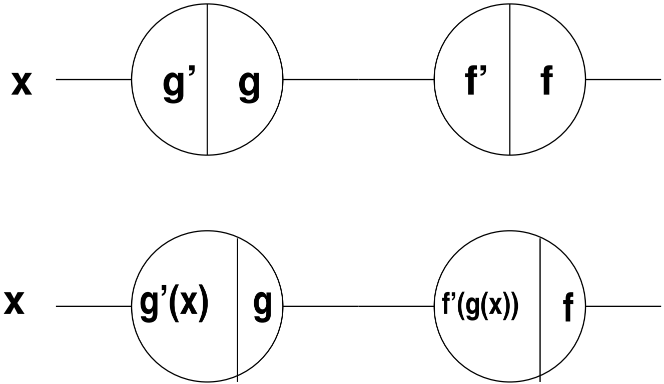
Back propagation



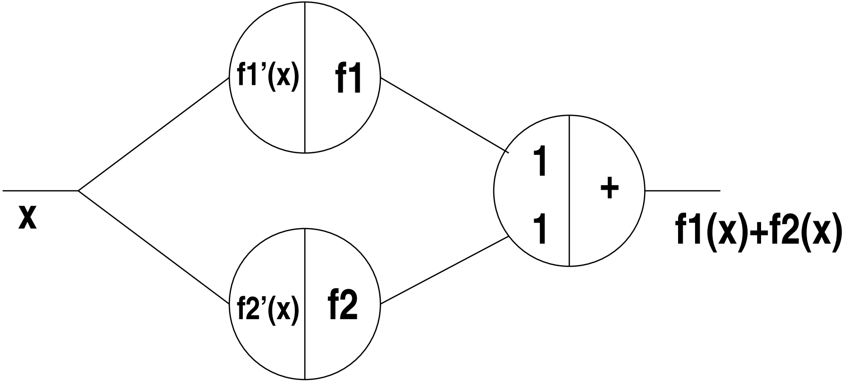
Back propagation



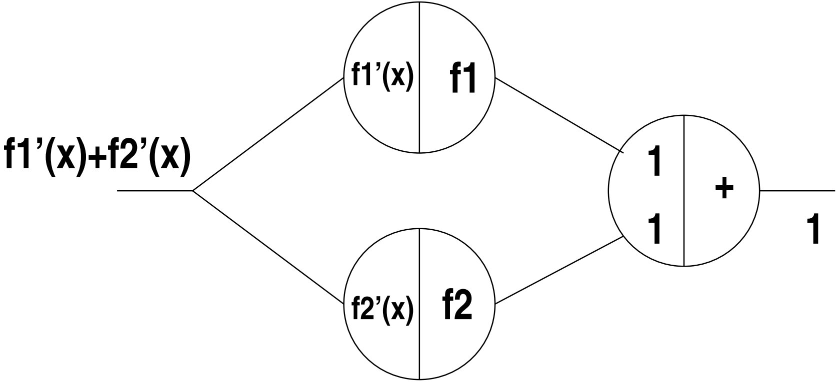
Back propagation



Back propagation



Back propagation

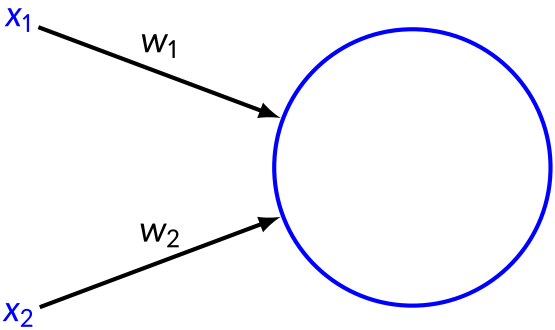


Backpropagation

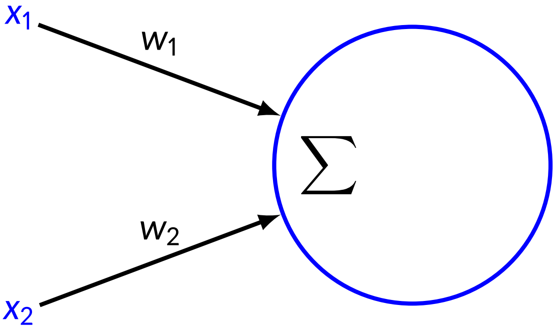
x_1

x_2

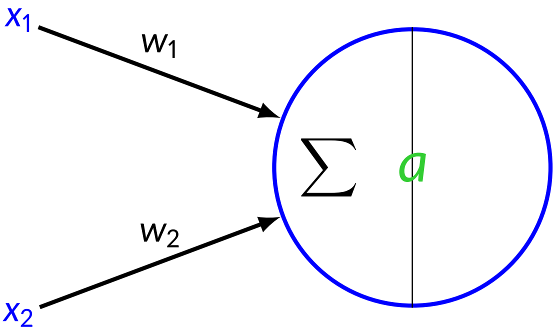
Backpropagation



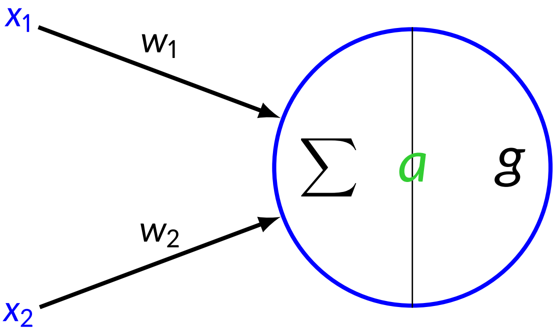
Backpropagation



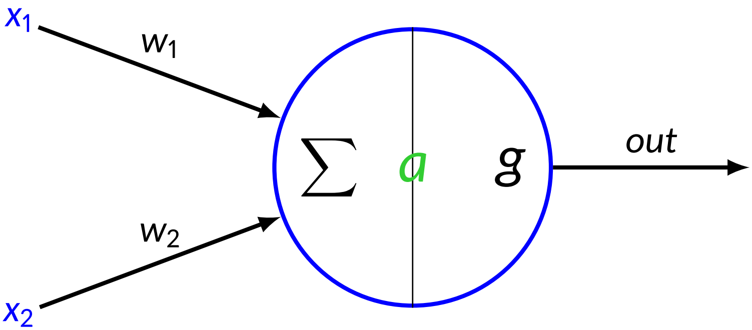
Backpropagation



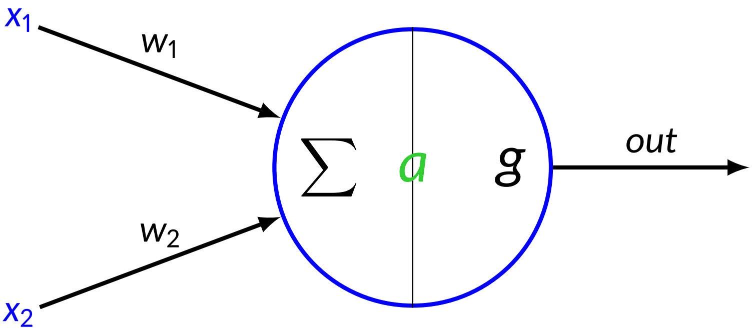
Backpropagation



Backpropagation

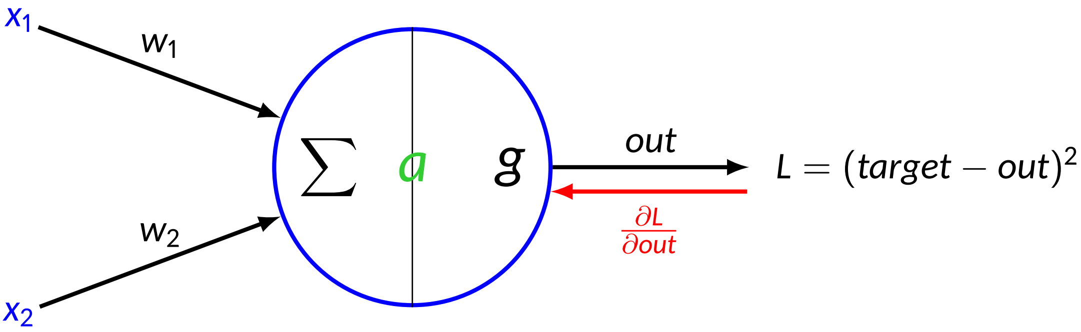


Backpropagation

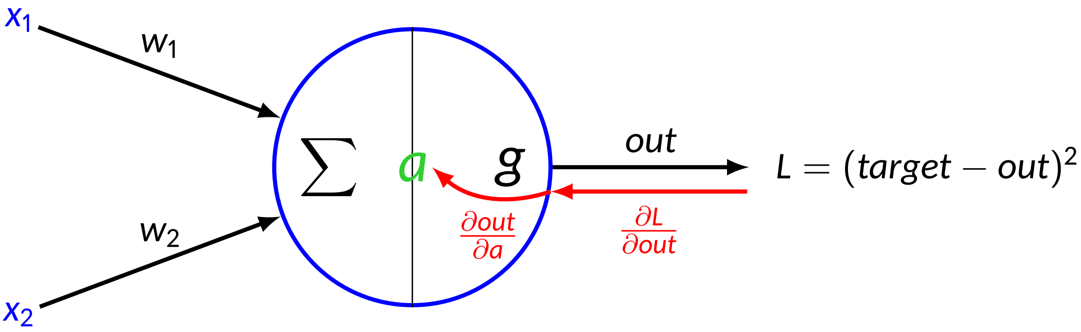


$$L = (target - out)^2$$

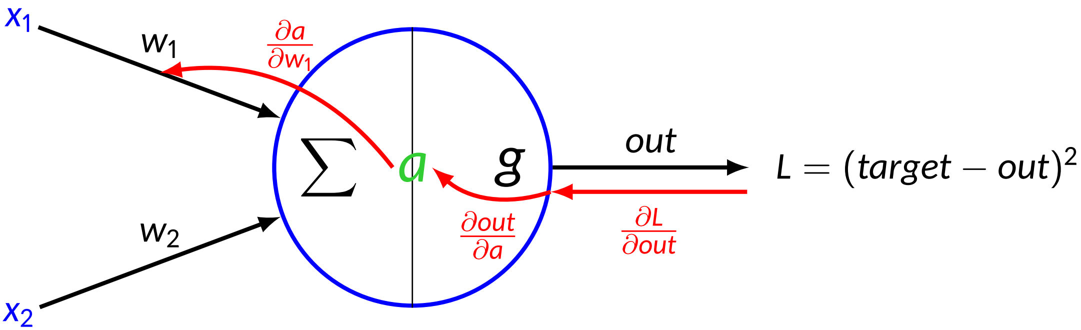
Backpropagation



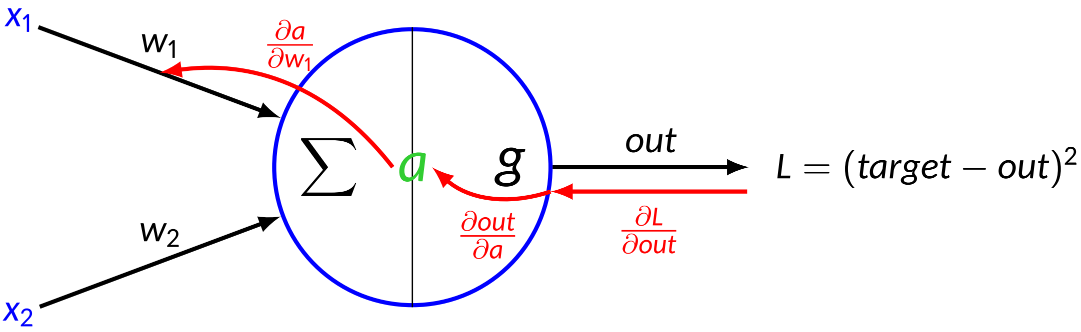
Backpropagation



Backpropagation



Backpropagation

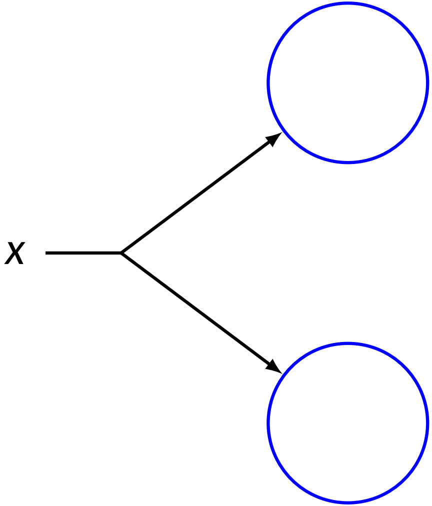


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial out} \frac{\partial out}{\partial a} \frac{\partial a}{\partial w_1}$$

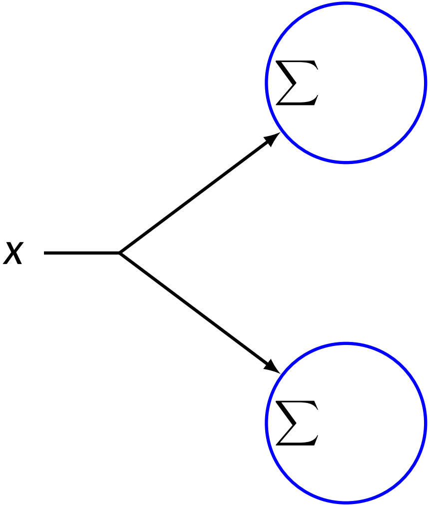
Backpropagation

x ——

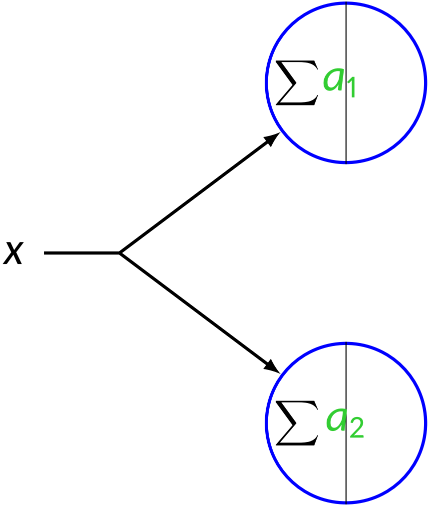
Backpropagation



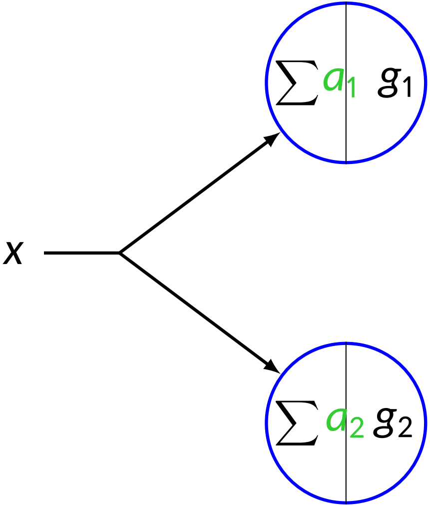
Backpropagation



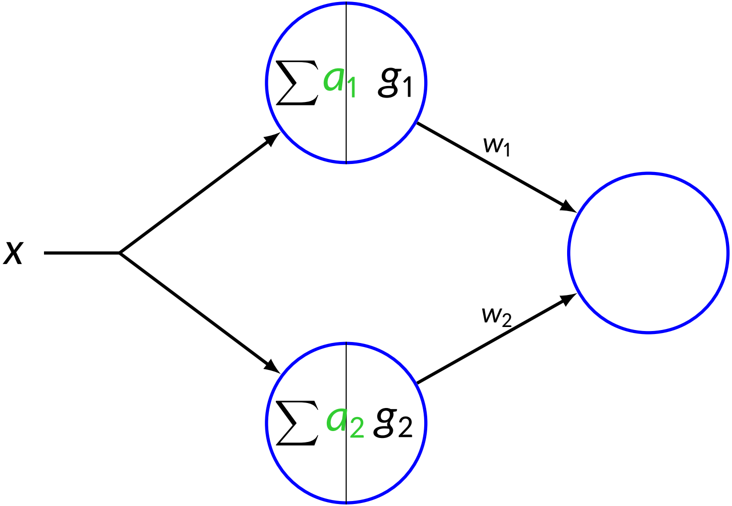
Backpropagation



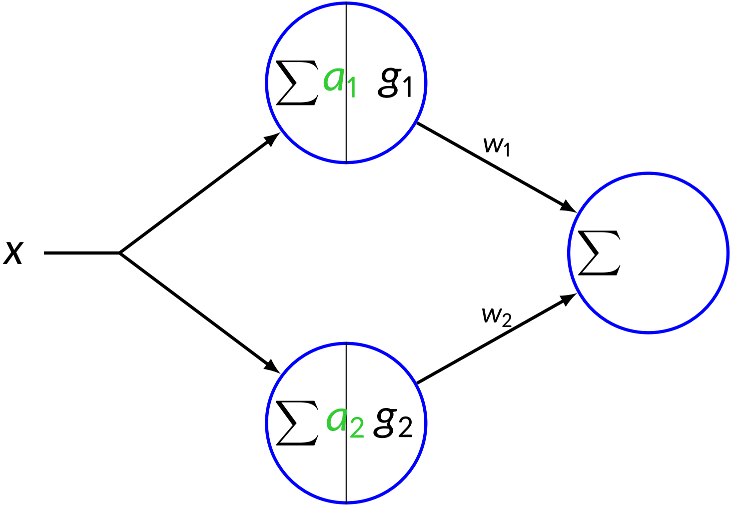
Backpropagation



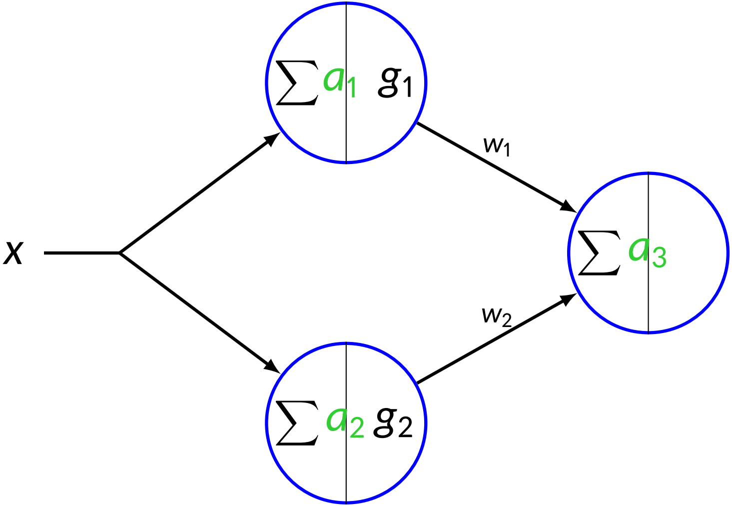
Backpropagation



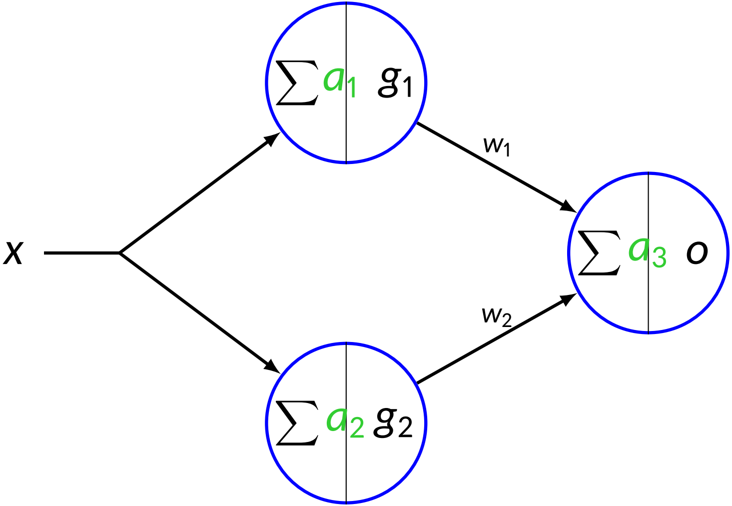
Backpropagation



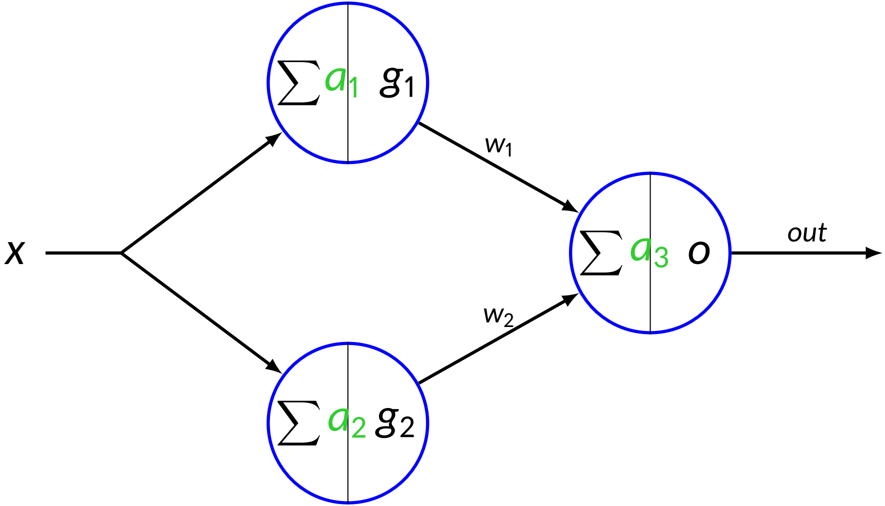
Backpropagation



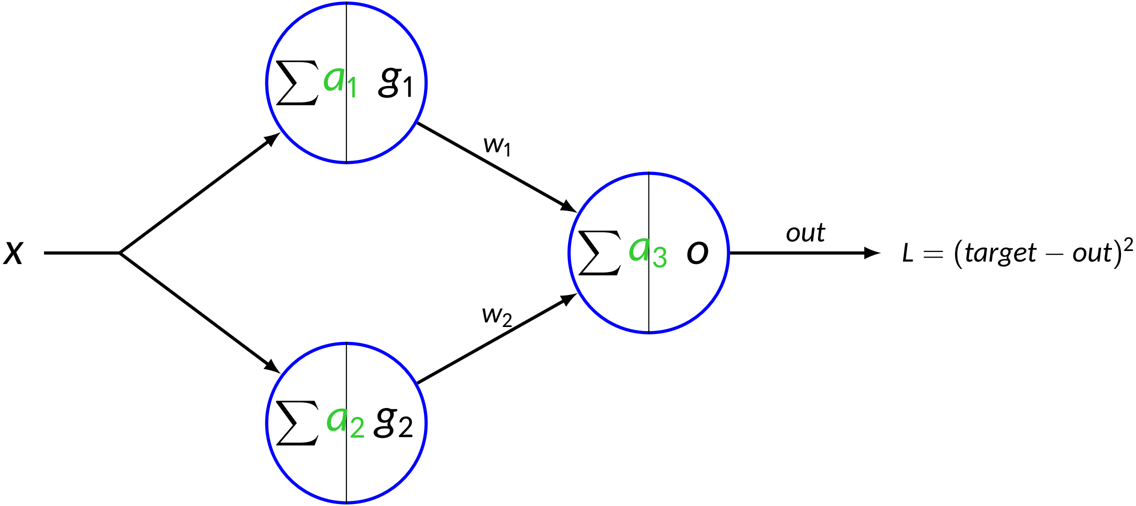
Backpropagation



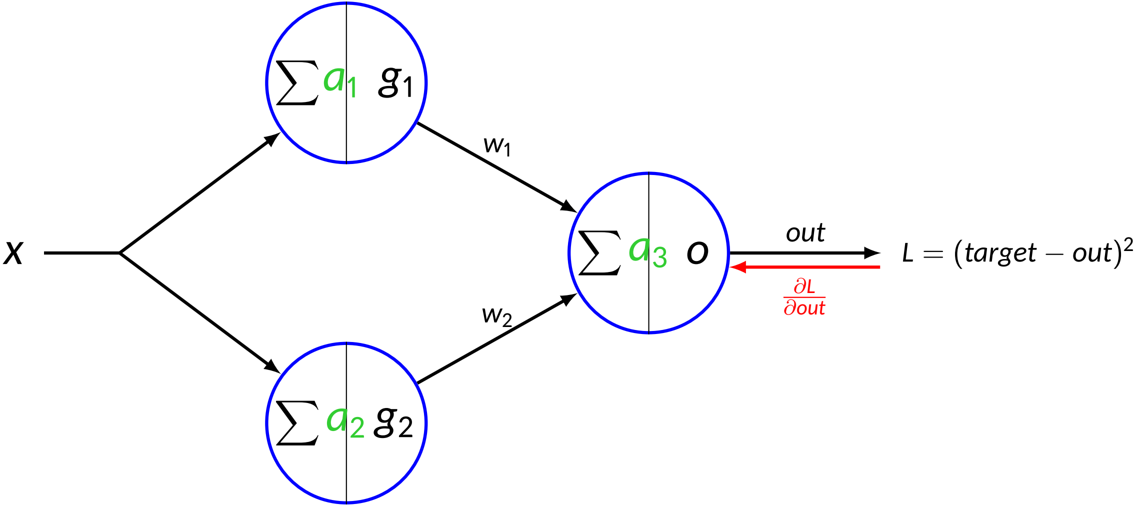
Backpropagation



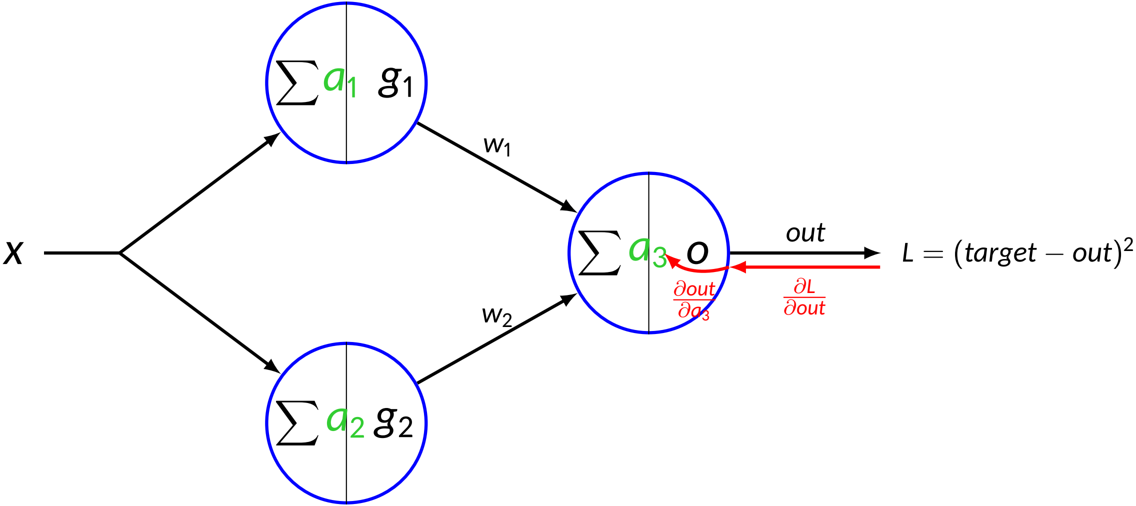
Backpropagation



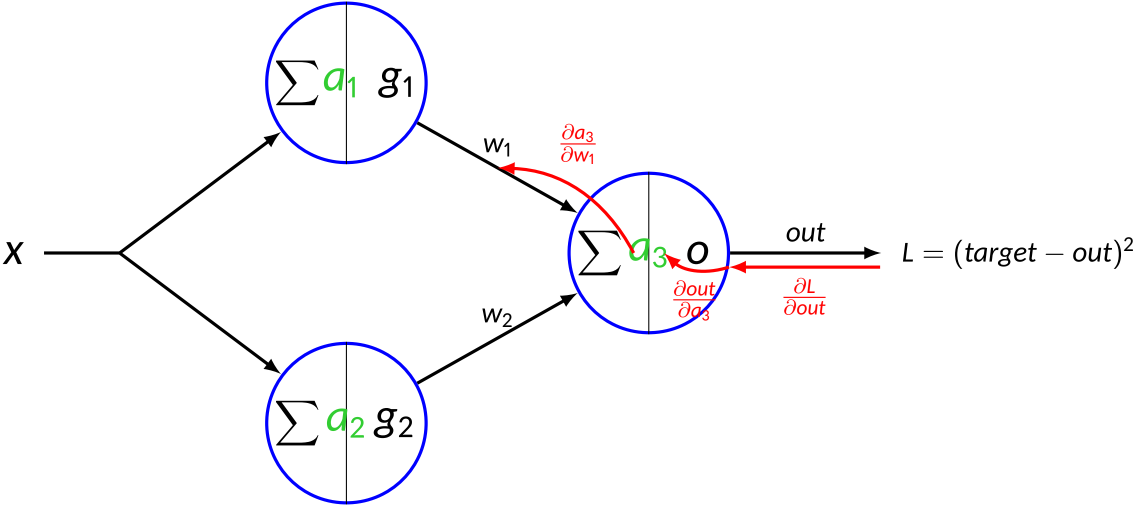
Backpropagation



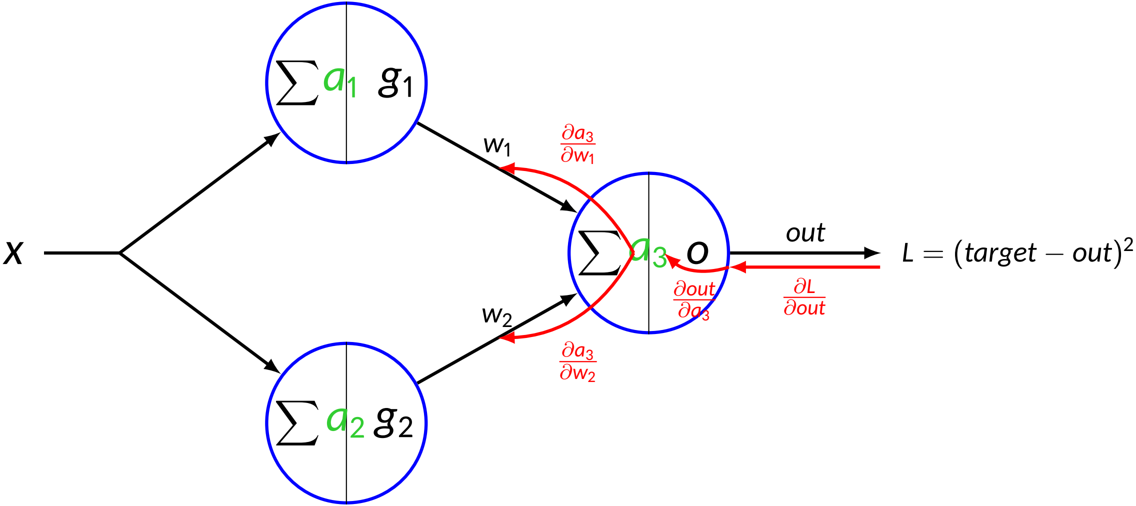
Backpropagation



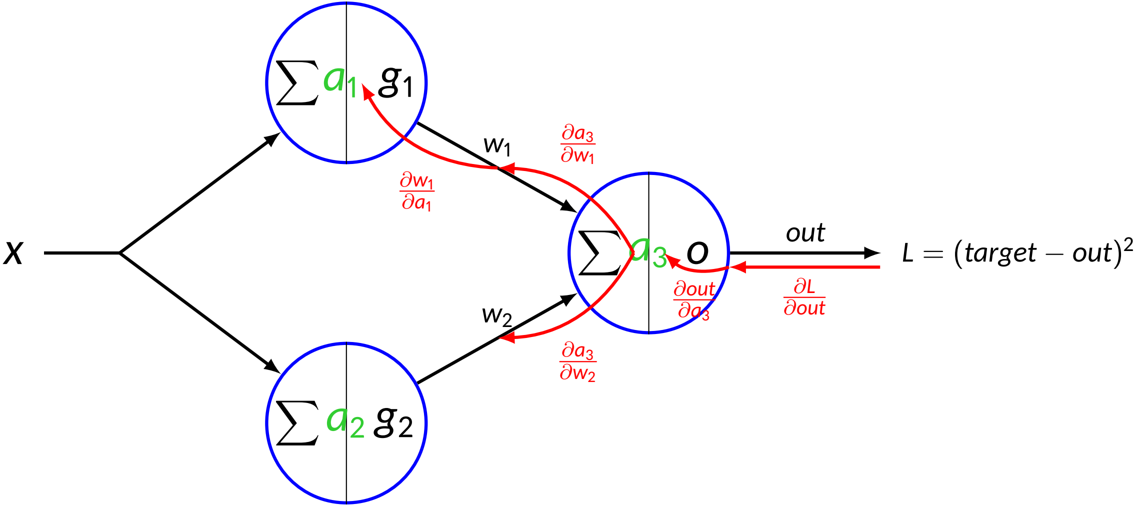
Backpropagation



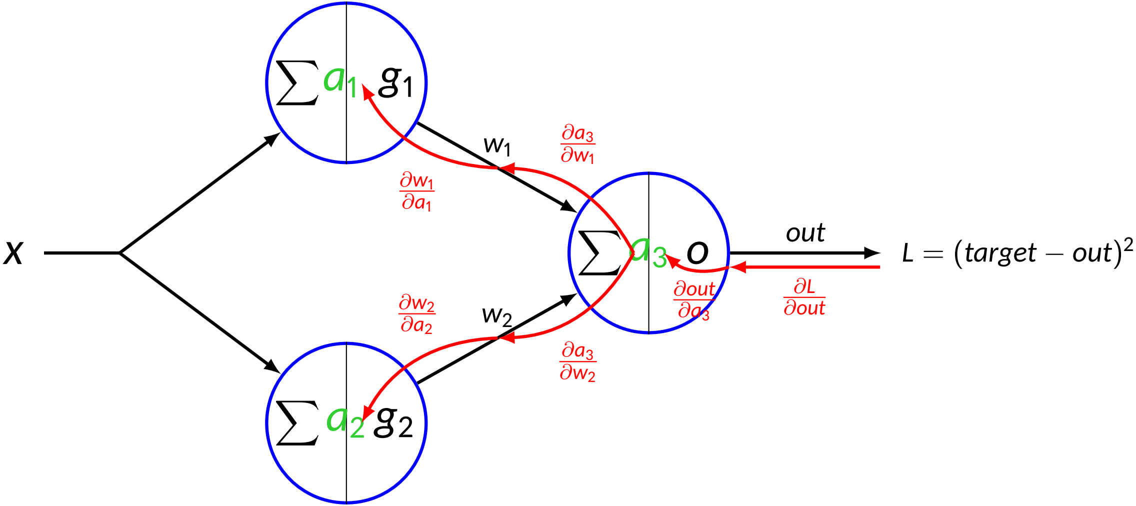
Backpropagation



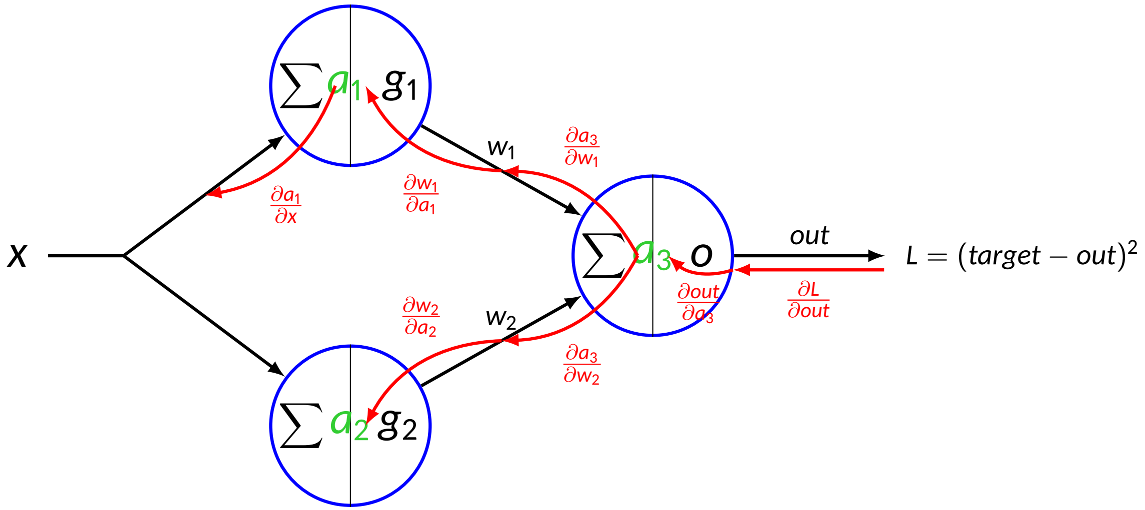
Backpropagation



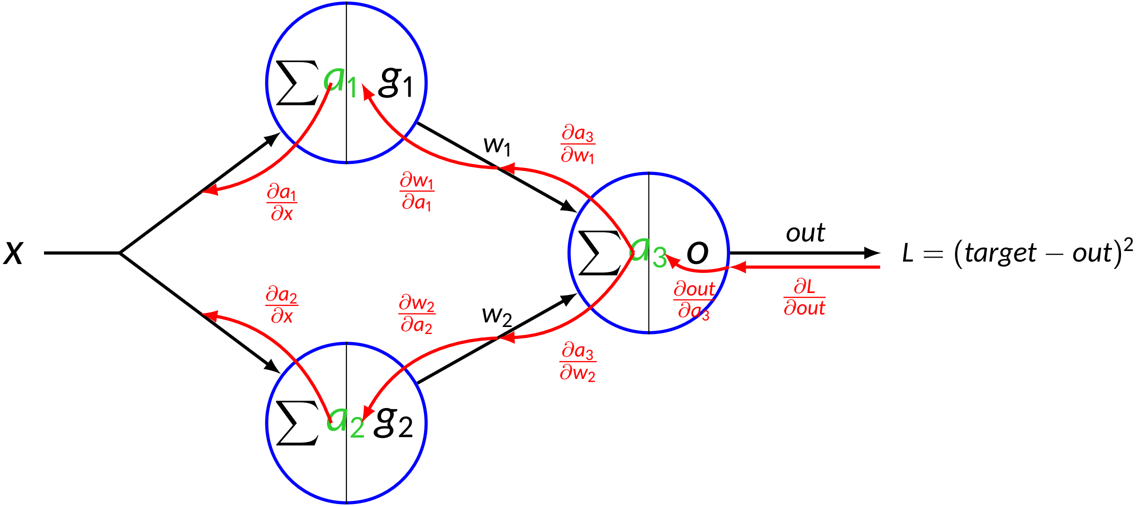
Backpropagation



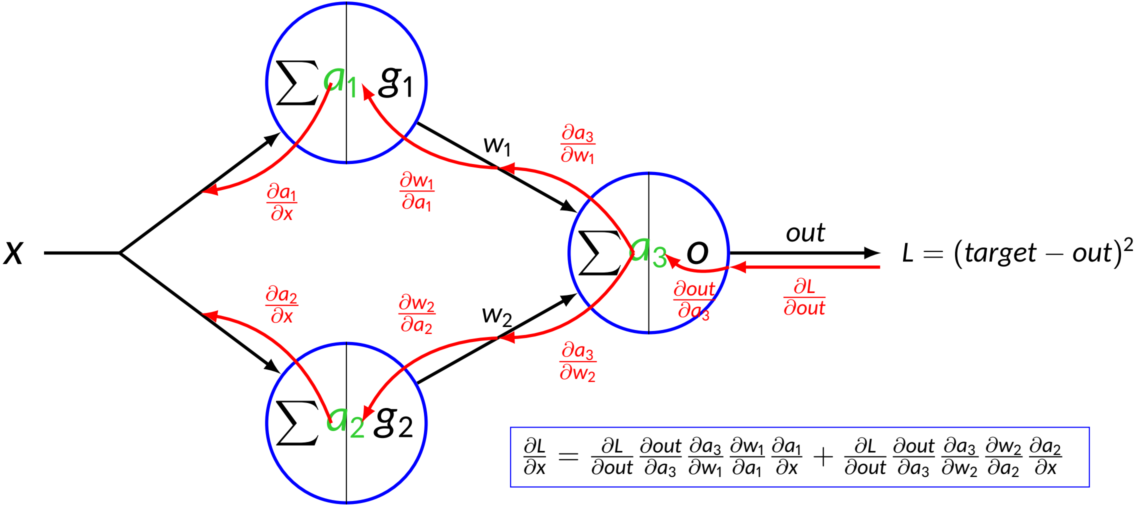
Backpropagation



Backpropagation

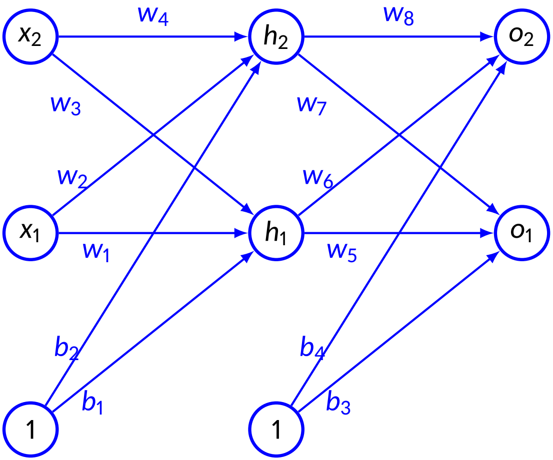


Backpropagation

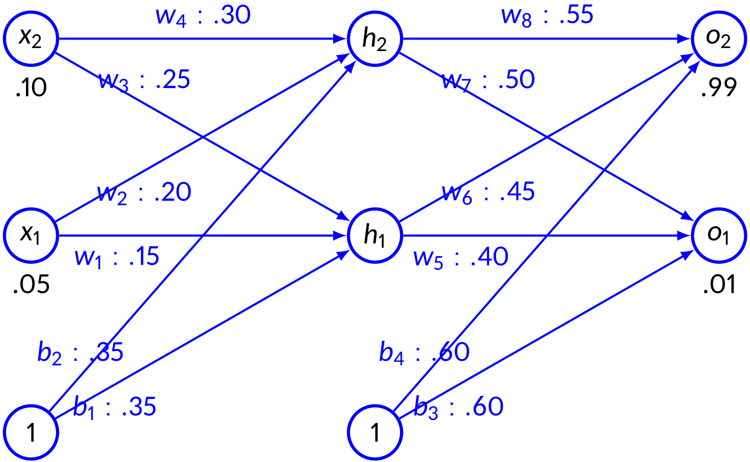


$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial a_3}\frac{\partial a_3}{\partial w_1}\frac{\partial w_1}{\partial a_1}\frac{\partial a_1}{\partial x} + \frac{\partial L}{\partial out}\frac{\partial out}{\partial a_3}\frac{\partial a_3}{\partial w_2}\frac{\partial w_2}{\partial a_2}\frac{\partial a_2}{\partial x}$$

Example



Example



x_2
.10
x_1
.05
1
$w_4 : .30$
$w_3 : .25$
$w_2 : .20$
$w_1 : .15$
$b_2 : .35$
$b_1 : .35$
h_2
h_1
1
$w_8 : .55$
$w_7 : .50$
$w_6 : .45$
$w_5 : .40$
$b_4 : .60$
$b_3 : .60$
o_2
.99
o_1
.01