Introduction to Machine Learning feat. TensorFlow

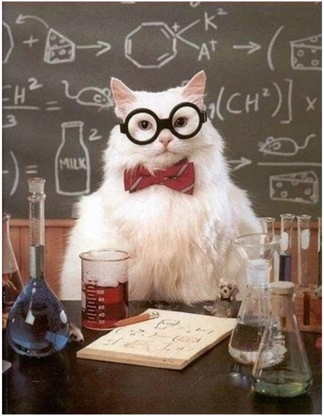
Peter Goldsborough

July 9, 2016

July 9, 2016

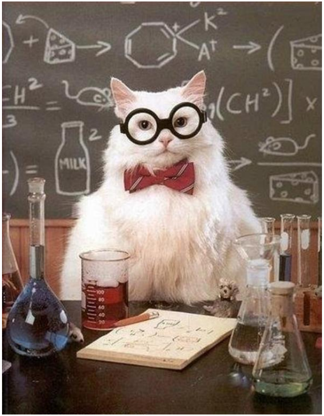
Table of Catents

Table of Catents

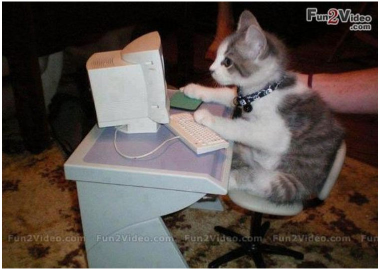


Theory

Table of Catents



Theory



Practice

Background

Background

- CS Student @ TUM

Background

- CS Student @ TUM
- Google & Bloomberg Intern

Background

- CS Student @ TUM
- Google & Bloomberg Intern
- I like cats

Background

- CS Student @ TUM
- Google & Bloomberg Intern
- I like cats

Seminar Topic: *Deep Learning With TensorFlow*

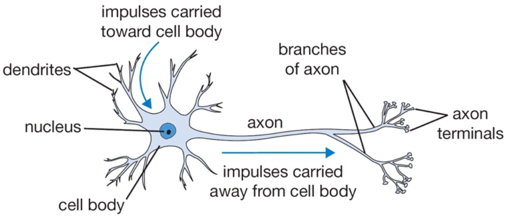
`github.com/peter-can-write/tensorflow-paper` `github.com/peter-can-talk/python-meetup-munich-2016`

Neural Networks

Neural Networks

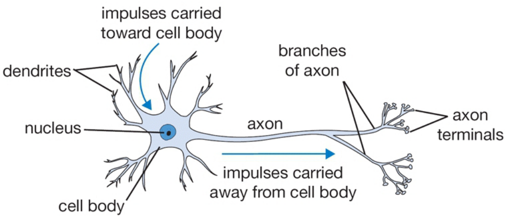
Neural Networks: Biological Motivation

Neural Networks: Biological Motivation



[?]

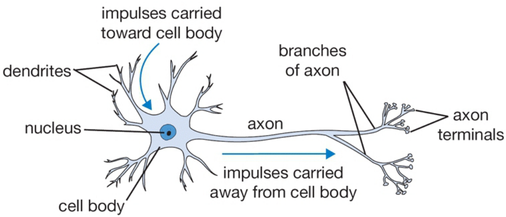
Neural Networks: Biological Motivation



[?]

- Receive electrochemical signals through *dendrites*

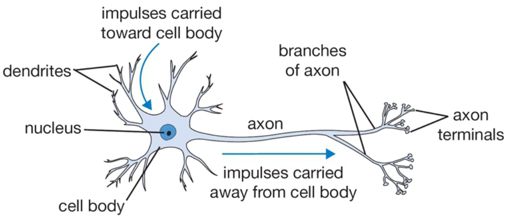
Neural Networks: Biological Motivation



[?]

- Receive electrochemical signals through *dendrites*
- Fire their own signal if the input exceeds some threshold

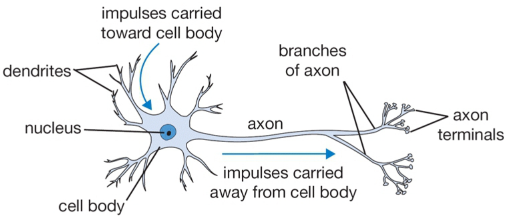
Neural Networks: Biological Motivation



[?]

- Receive electrochemical signals through *dendrites*
- Fire their own signal if the input exceeds some threshold
- Forward their signals via *axons*

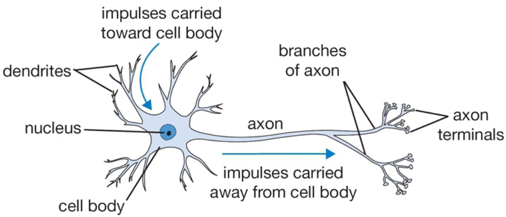
Neural Networks: Biological Motivation



[?]

- Receive electrochemical signals through *dendrites*
- Fire their own signal if the input exceeds some threshold
- Forward their signals via *axons*
- Connected via *synapses*, which control the strength of interaction

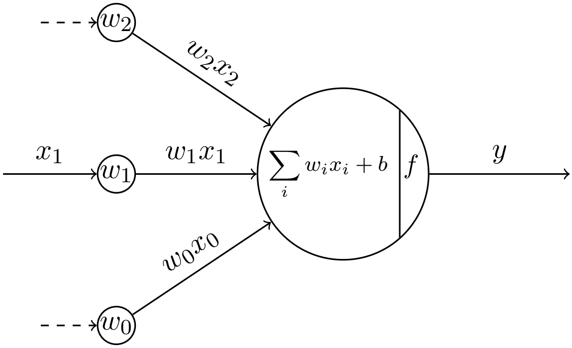
Neural Networks: Biological Motivation



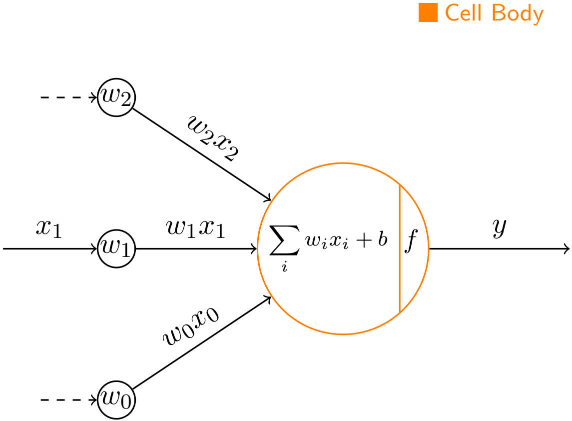
[?]

- Receive electrochemical signals through *dendrites*
- Fire their own signal if the input exceeds some threshold
- Forward their signals via *axons*
- Connected via *synapses*, which control the strength of interaction
- The dynamic alteration of synaptic strengths are the primary source of human *learning*

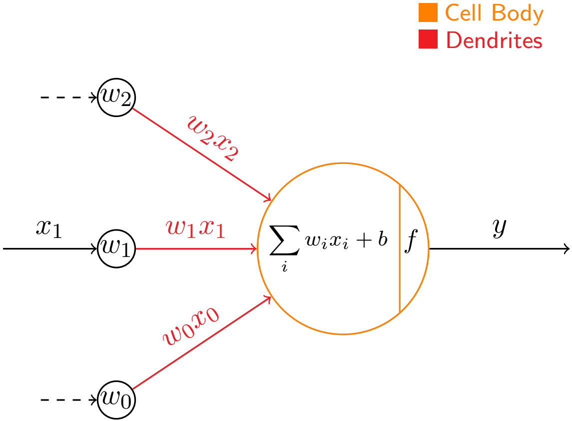
Neural Networks: Mathematical Model



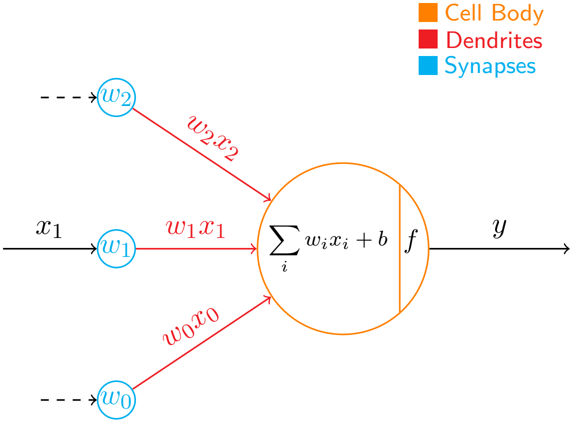
Neural Networks: Mathematical Model



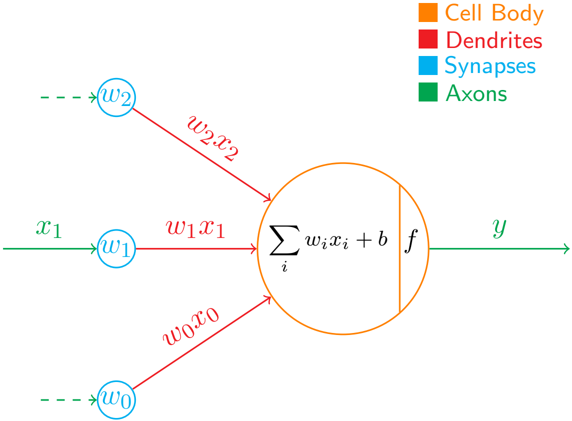
Neural Networks: Mathematical Model



Neural Networks: Mathematical Model



Neural Networks: Mathematical Model



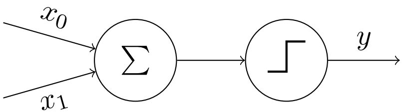
Artificial Neural Networks

Artificial Neural Networks

- First attempt at Artificial Neural Network (ANN) by McCulloch and Pitts in 1943

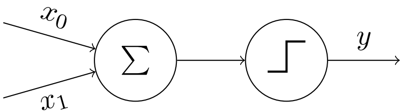
Artificial Neural Networks

- First attempt at Artificial Neural Network (ANN) by McCulloch and Pitts in 1943
- Summed binary inputs and thresholded them



Artificial Neural Networks

- First attempt at Artificial Neural Network (ANN) by McCulloch and Pitts in 1943
- Summed binary inputs and thresholded them
- Could learn AND, NOT and OR functions

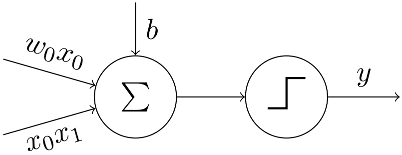


Artificial Neural Networks

- Frank Rosenblatt improved on this model in 1957

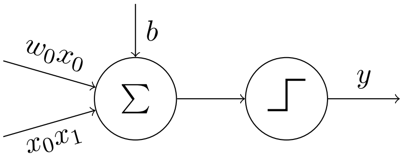
Artificial Neural Networks

- Frank Rosenblatt improved on this model in 1957
- He *weighted* the inputs and added a bias



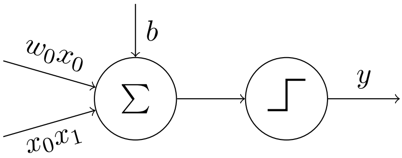
Artificial Neural Networks

- Frank Rosenblatt improved on this model in 1957
- He *weighted* the inputs and added a bias
- This models the synaptic strengths between axons and dendrites



Artificial Neural Networks

- Frank Rosenblatt improved on this model in 1957
- He *weighted* the inputs and added a bias
- This models the synaptic strengths between axons and dendrites
- He called this model a *Perceptron*



Artificial Neural Networks

Artificial Neural Networks

- Because the Perceptron has parameters, it can be trained

Artificial Neural Networks

- Because the Perceptron has parameters, it can be trained
- First supervised learning algorithm for ANNs:

Artificial Neural Networks

- Because the Perceptron has parameters, it can be trained
- First supervised learning algorithm for ANNs:

Algorithm Train Perceptron

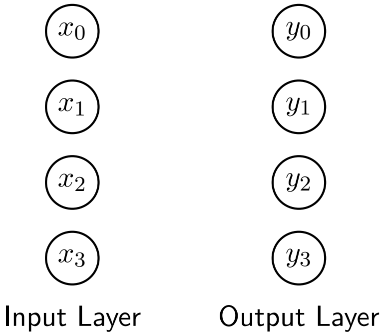
Input: A dataset of $(\mathbf{x}, \hat{y})$ pairs
Output: Trained Perceptron
for all $(\mathbf{x}, \hat{y})$ in dataset **do**:
    $y \leftarrow f(\mathbf{w}^\top \mathbf{x} + b)$
    if $y \neq \hat{y}$ **then**
        if $\hat{y} = 0 \land y = 1$ **then**
            Decrease all weights w_i where x_i was 1
        else if $\hat{y} = 1 \land y = 0$ **then**
            Increase all weights w_i where x_i was 1

Artificial Neural Networks

- Perceptrons even work for multi-class classification

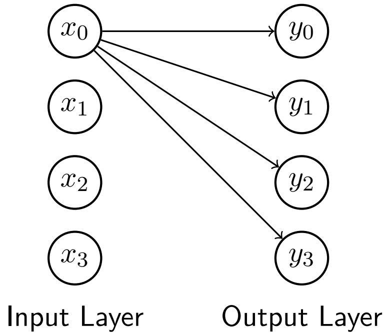
Artificial Neural Networks

- Perceptrons even work for multi-class classification
- Just use more perceptrons



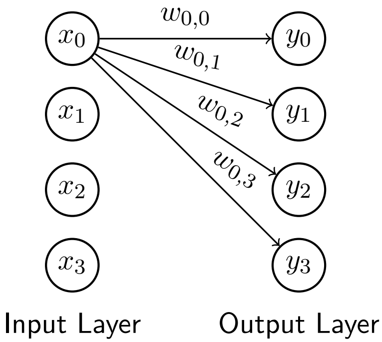
Artificial Neural Networks

- Perceptrons even work for multi-class classification
- Just use more perceptrons



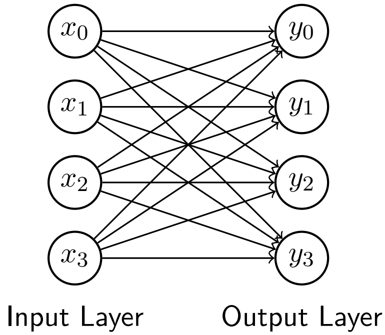
Artificial Neural Networks

- Perceptrons even work for multi-class classification
- Just use more perceptrons



Artificial Neural Networks

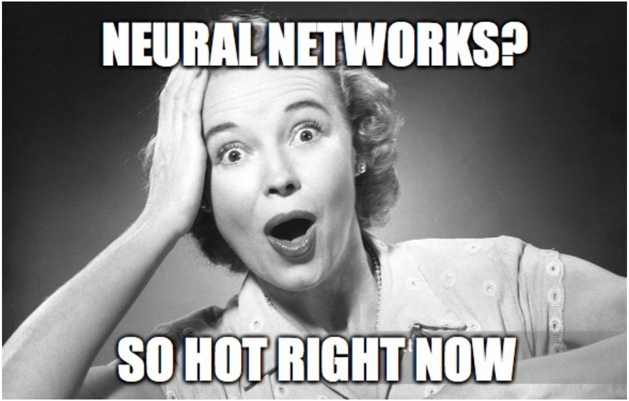
- Perceptrons even work for multi-class classification
- Just use more perceptrons
- This is now a *multilayer perceptron* (MLP)



NEW NAVY DEVICE LEARNS BY DOING

WASHINGTON, July 7 (UPI) – The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

[?]


NEURAL NETWORKS?
SO HOT RIGHT NOW

Artificial Neural Networks

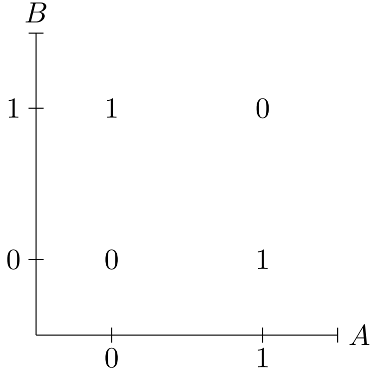
- Perceptrons have one fundamental problem

Artificial Neural Networks

- Perceptrons have one fundamental problem
- They can only learn linear functions

Artificial Neural Networks

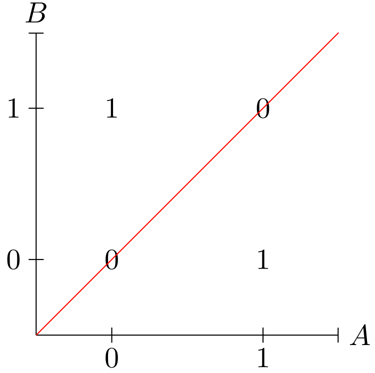
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

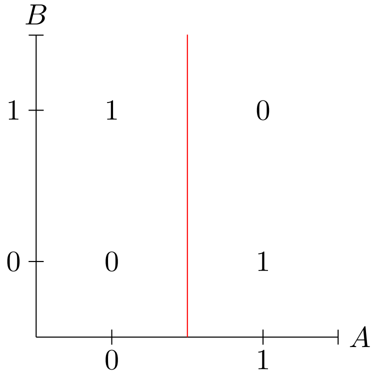
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

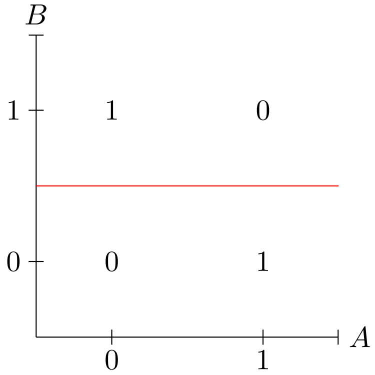
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

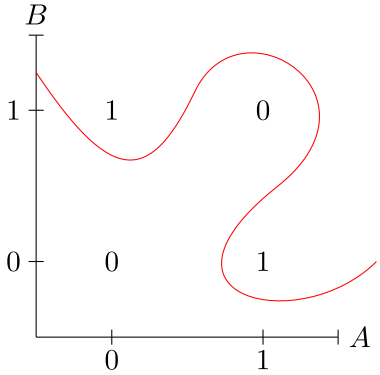
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

- The solution:

Artificial Neural Networks

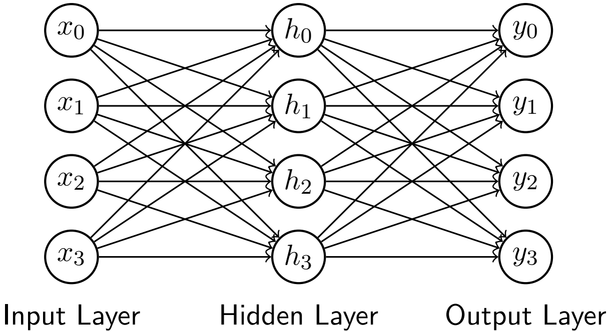
- The solution:
  - A hidden layer

Artificial Neural Networks

- The solution:
  - A hidden layer
  - With *non-linear* activation functions

Artificial Neural Networks

- The solution:
  - A hidden layer
  - With *non-linear* activation functions

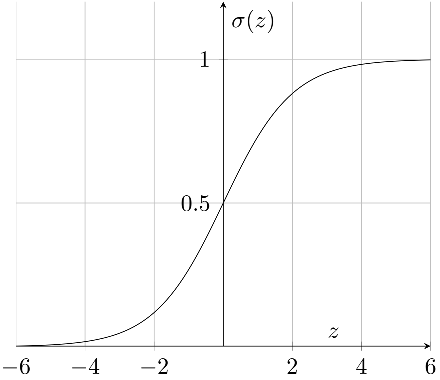


Activation Functions

- Three important activation functions

Activation Functions

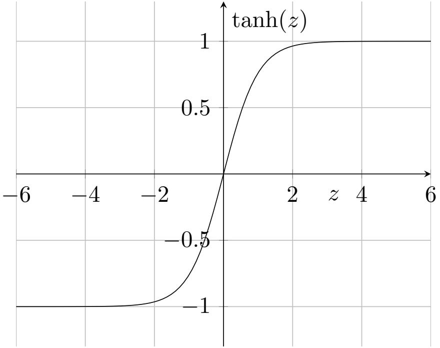
- Three important activation functions



Le *Sigmoid*

Activation Functions

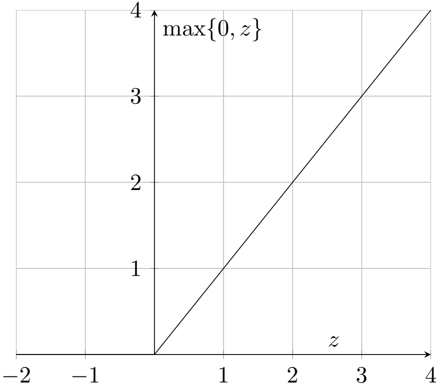
- Three important activation functions



El tanh

Activation Functions

- Three important activation functions



Das ReLu

Neural Networks by Foot

Neural Networks by Foot

- We want to classify programmers into one of two classes

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`
- Two input features:

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`
- Two input features:
  1. Maximum number of successive pointers to pointers maintained

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`
- Two input features:
  1. Maximum number of successive pointers to pointers maintained
  2. Liters of coffee per year

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`
- Two input features:
  1. Maximum number of successive pointers to pointers maintained
  2. Liters of coffee per year

$$\mathbf{D} = \begin{array}{c} \begin{array}{cc} p & c \end{array} \\ \begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix} \end{array}$$

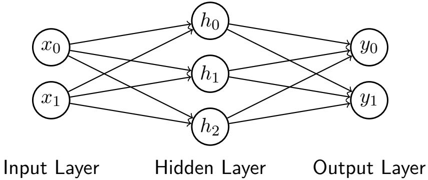
Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`
- Two input features:
  1. Maximum number of successive pointers to pointers maintained
  2. Liters of coffee per year
- Our labels are *one-hot* encoded

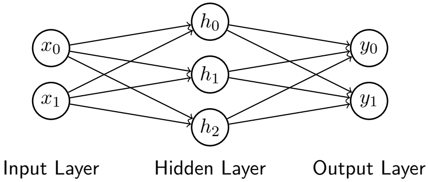
$$\mathbf{D} = \begin{array}{c} \begin{array}{cc} p & c \end{array} \\ \begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix} \end{array} \quad \hat{\mathbf{Y}} = \begin{array}{c} \begin{array}{cc} \texttt{l33t} & \texttt{n00b} \end{array} \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

Neural Networks by Foot

- We want to classify programmers into one of two classes
  1. `l33t h4x0r`
  2. `n00b`
- Two input features:
  1. Maximum number of successive pointers to pointers maintained
  2. Liters of coffee per year
- Our labels are *one-hot* encoded

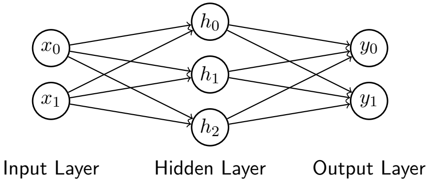


Neural Networks by Foot



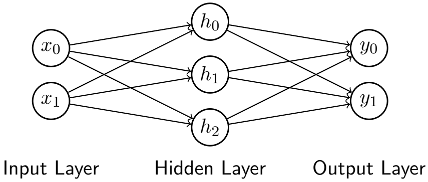
$$\underset{\mathbf{D}}{\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}}$$

Neural Networks by Foot



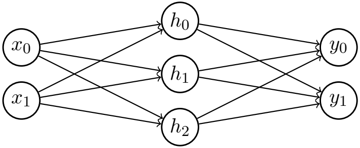
$$\underset{\mathbf{D}}{\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}} \times \underset{\mathbf{W_1}}{\begin{bmatrix} -0.04 & -0.43 & 0.57 \\ 0.04 & 0.52 & -0.6 \end{bmatrix}}$$

Neural Networks by Foot



$$\underbrace{\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}}_{\mathbf{D}} \times \underbrace{\begin{bmatrix} -0.04 & -0.43 & 0.57 \\ 0.04 & 0.52 & -0.6 \end{bmatrix}}_{\mathbf{W_1}} +_b \underbrace{\begin{bmatrix} -2.03 & -0.26 & 2.16 \end{bmatrix}}_{\mathbf{b_1}}$$

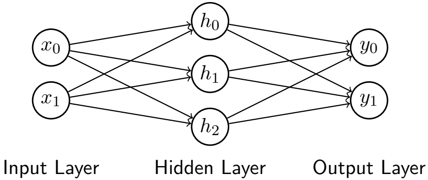
Neural Networks by Foot



$$\underbrace{\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}}_{\mathbf{D}} \times \underbrace{\begin{bmatrix} -0.04 & -0.43 & 0.57 \\ 0.04 & 0.52 & -0.6 \end{bmatrix}}_{\mathbf{W_1}} +_b \underbrace{\begin{bmatrix} -2.03 & -0.26 & 2.16 \end{bmatrix}}_{\mathbf{b_1}}$$

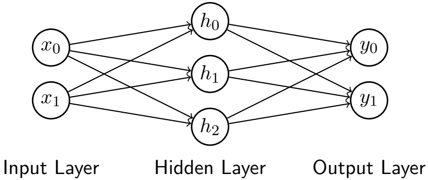
$$= \underbrace{\begin{bmatrix} 13.37 & 183.66 & -210.46 \\ 3.05 & 60.33 & -67.91 \\ 41.13 & 515.49 & -596.08 \end{bmatrix}}_{\mathbf{H}}$$

Neural Networks by Foot



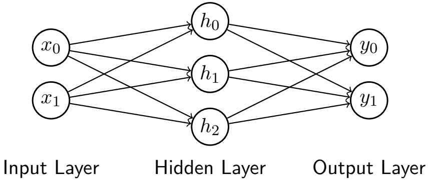
$$\mathrm{relu}(\mathbf{H})$$

Neural Networks by Foot



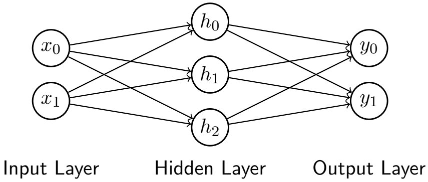
$$\mathrm{relu}(\mathbf{H}) = \max\{0, \mathbf{H}\}$$

Neural Networks by Foot



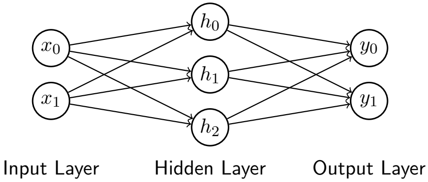
$$\mathrm{relu}(\mathbf{H}) = \max\{0, \mathbf{H}\} = \underbrace{\begin{bmatrix} 13.37 & 183.66 & 0.0 \\ 3.05 & 60.33 & 0.0 \\ 41.13 & 515.49 & 0.0 \end{bmatrix}}_{\mathbf{H}'}$$

Neural Networks by Foot



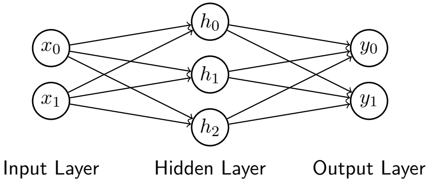
$\mathbf{H}'$

Neural Networks by Foot



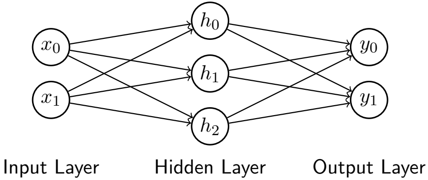
$$\mathbf{H}' \times \underbrace{\begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix}}_{\mathbf{W_2}}$$

Neural Networks by Foot



$$\mathbf{H}' \times \underbrace{\begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix}}_{\mathbf{W_2}} +_b \underbrace{\begin{bmatrix} -0.19 & -0.49 \end{bmatrix}}_{\mathbf{b_2}}$$

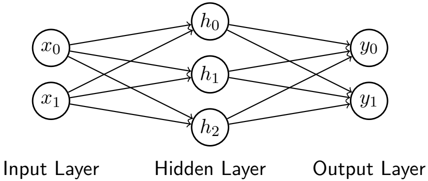
Neural Networks by Foot



$$\mathbf{H'} \times \underbrace{\begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix}}_{\mathbf{W_2}} +_b \underbrace{\begin{bmatrix} -0.19 & -0.49 \end{bmatrix}}_{\mathbf{b_2}}$$

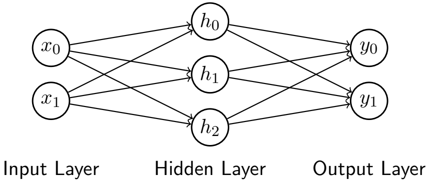
$$= \underbrace{\begin{bmatrix} -2.53 & 211.85 \\ -1.55 & 69.01 \\ -5.16 & 596.17 \end{bmatrix}}_{\mathbf{Y}}$$

Neural Networks by Foot



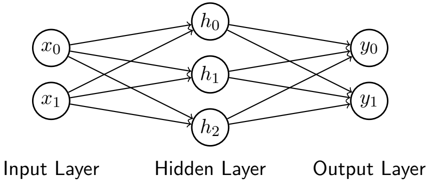
$$\text{softmax}(\mathbf{Y}) =$$

Neural Networks by Foot



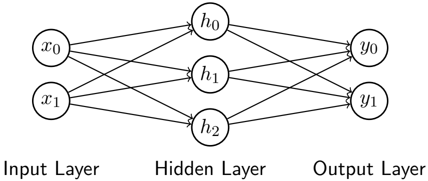
$$\text{softmax}(\mathbf{Y}) = \underbrace{\begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \\ 0.0 & 1.0 \end{bmatrix}}_{\mathbf{Y}}$$

Neural Networks By Foot



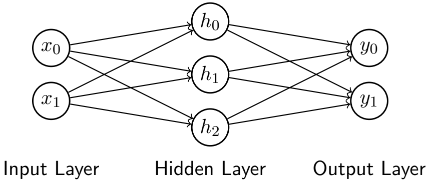
- Let $\mathcal{W}$ be $[\mathbf{W}_1, \mathbf{W}_2]$

Neural Networks By Foot



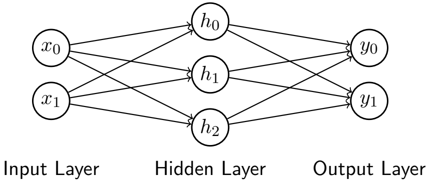
- Let $\mathcal{W}$ be $[\mathbf{W}_1, \mathbf{W}_2]$
- Then $J(\mathcal{W})$ is the network's loss

Neural Networks By Foot



- Let $\mathcal{W}$ be $[\mathbf{W}_1, \mathbf{W}_2]$
- Then $J(\mathcal{W})$ is the network's loss
- $\nabla J(\mathcal{W})$ are the gradients

Neural Networks By Foot



- Let $\mathcal{W}$ be $[\mathbf{W}_1, \mathbf{W}_2]$
- Then $J(\mathcal{W})$ is the network's loss
- $\nabla J(\mathcal{W})$ are the gradients
- The final step of this iteration would thus be:

$$\mathcal{W} \leftarrow \mathcal{W} - \nabla J(\mathcal{W})$$

Dropout

- One recent regularization technique is *Dropout*

Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks

Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:

Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p

Dropout

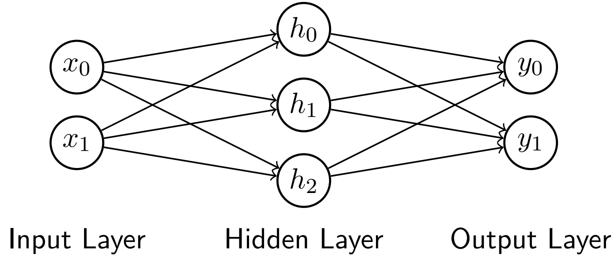
- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - Can be seen as sampling a new neural network each time
  - This prevents neurons from depending too much on each other

Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - Can be seen as sampling a new neural network each time
  - This prevents neurons from depending too much on each other

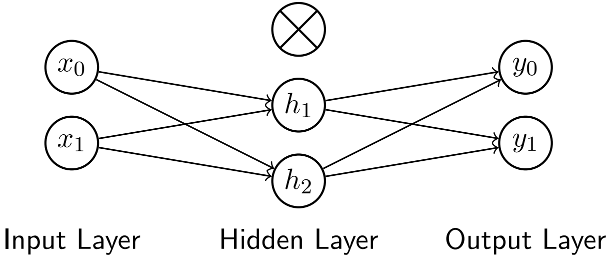
Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - Can be seen as sampling a new neural network each time
  - This prevents neurons from depending too much on each other



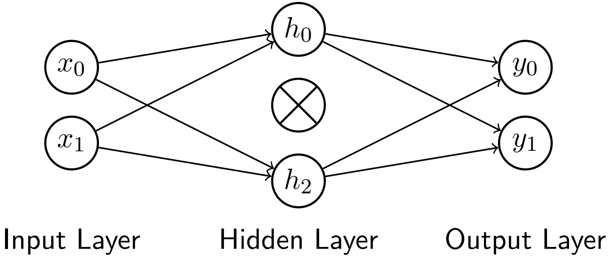
Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - Can be seen as sampling a new neural network each time
  - This prevents neurons from depending too much on each other



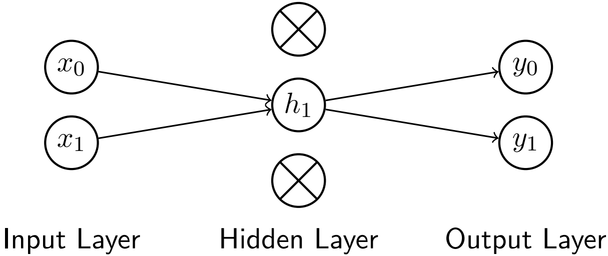
Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - Can be seen as sampling a new neural network each time
  - This prevents neurons from depending too much on each other



Dropout

- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - Can be seen as sampling a new neural network each time
  - This prevents neurons from depending too much on each other



References