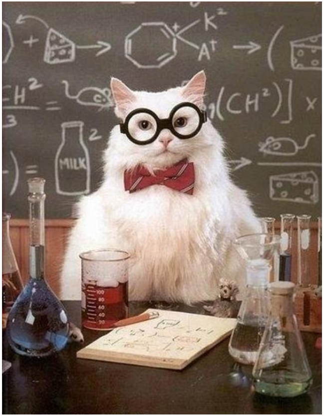
Introduction to Machine Learning feat. TensorFlow

Peter Goldsborough

July 12, 2016

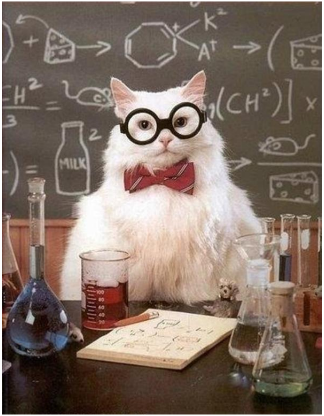
Table of Catents

Table of Catents

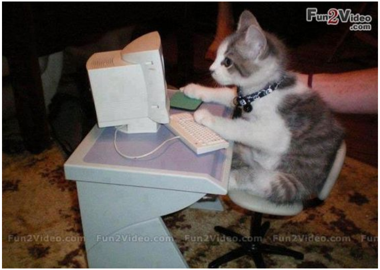


Theory

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

What is Machine Learning?

What is Machine Learning?

(and can I eat it?)

Definition I

Definition I

Machine Learning is cool.

Definition II

Machine Learning is *really* cool.

Definition III

Definition III

Machine learning is not magic; it can't get something from nothing.

Definition III

Machine learning is not magic; it can't get something from nothing.

What it does is get more from less.

Definition III

Machine learning is not magic; it can't get something from nothing.

What it does is get more from less.

Learning is like farming, which lets nature do most of the work. Farmers combine seeds with nutrients to grow crops. Learners combine knowledge with data to grow programs.

[?]

Definition IV

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

[**?**]

Definition IV

$$f : \text{Image} \rightarrow \{\text{cat, banana, spaceship, } \ldots\}$$

Definition IV

$$f : \text{Image} \rightarrow \{\text{cat, banana, spaceship, } \ldots\}$$

$$f^{\star}(x) \approx f(x)$$

The Task, T

The Task, T

- Discriminate by the way an algorithm processes an example $\mathbf{x} \in \mathbb{R}^n$

The Task, T

- Discriminate by the way an algorithm processes an example $\mathbf{x} \in \mathbb{R}^n$
- The output $\mathbf{y}$ can take on various forms

The Task, T

Classification

$$f : \mathbb{R}^n \to \{1, \ldots, k\}$$

The Task, T

Classification

$$f : \mathbb{R}^n \to \{1, \ldots, k\}$$

- Image classification

The Task, T

Classification

$$f : \mathbb{R}^n \to \{1, \ldots, k\}$$

- Image classification
- Predictive Policing

The Task, T

Regression

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

The Task, T

Regression

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

- Algorithmic Trading

The Task, T

Regression

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

- Algorithmic Trading
- Predicting the market price of a house

How do we make our algorithm learn?

How do we make our algorithm learn?

Unsupervised Learning Supervised Learning

How do I Machine Learning?

Methods

Methods

Methods

$$\mathbf{b} = (R, G, B)^\top$$

Methods

$$\mathbf{b} = (R, G, B)^\top$$

$$f : \mathbf{b} \mapsto \text{market price} \in \mathbb{R}$$

Methods: Features

Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**

Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space

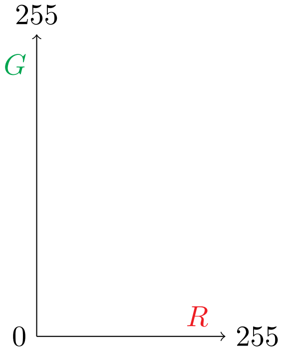
Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space

$$0 \xrightarrow{R} 255$$

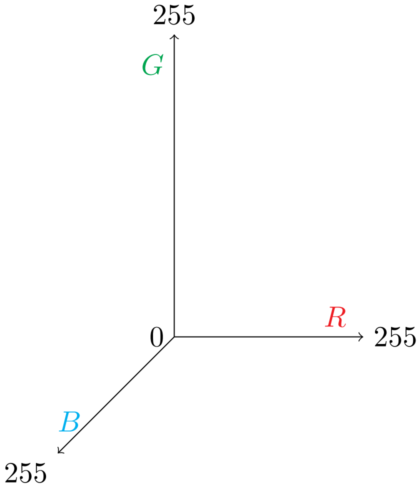
Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space



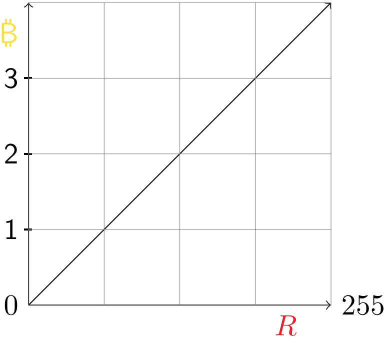
Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space



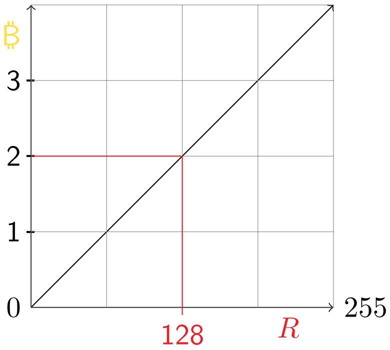
Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space



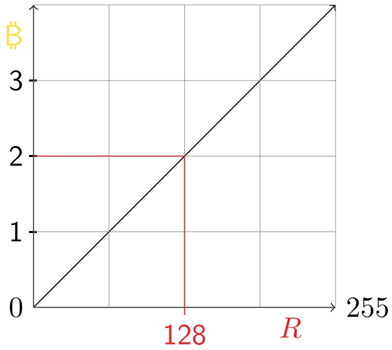
Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space



Methods: Features

- The components of each vector $\mathbf{b}$ are our **features**
- Each feature represents one axis in space
- Each feature should contribute to some extent to the output value



Methods: Features

- To model this, we apply a weight w_i to each component b_i

$$f(\mathbf{b}) = w_1 b_1 + w_2 b_2 + w_3 b_3$$

Methods: Features

- To model this, we apply a weight w_i to each component b_i
- We add a bias c as an offset

$$f(\mathbf{b}) = w_1 b_1 + w_2 b_2 + w_3 b_3 + c$$

Methods: Features

- To model this, we apply a weight w_i to each component b_i
- We add a bias c as an offset
- We expect our algorithm to learn $\mathbf{w}$ and c

$$f(\mathbf{b}) = \mathbf{w}^\top \mathbf{b} + c$$

Methods: Data

- To make our algorithm learn, we need to feed it (lots of) data

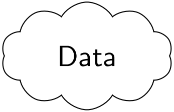
Methods: Data

- To make our algorithm learn, we need to feed it (lots of) data

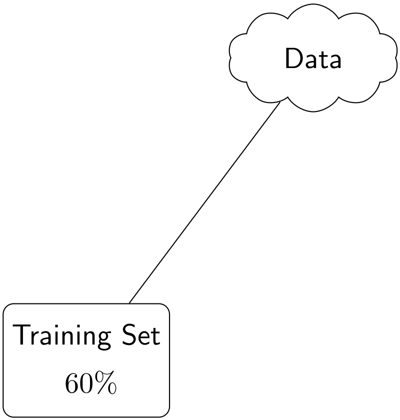
$$\begin{array}{cc} & \begin{array}{ccc} \text{R} & \text{G} & \text{B} \end{array} \\ \text{n} & \begin{bmatrix} 34 & 147 & 73 \\ 247 & 69 & 13 \\ 66 & 66 & 66 \end{bmatrix} \end{array}$$

Design Matrix

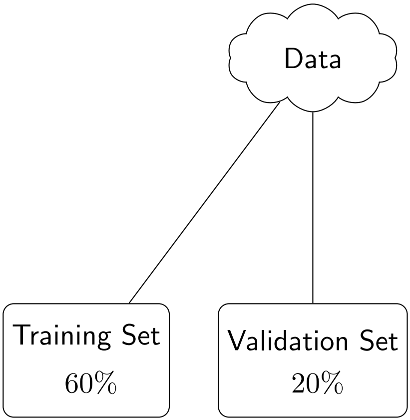
Methods: Data



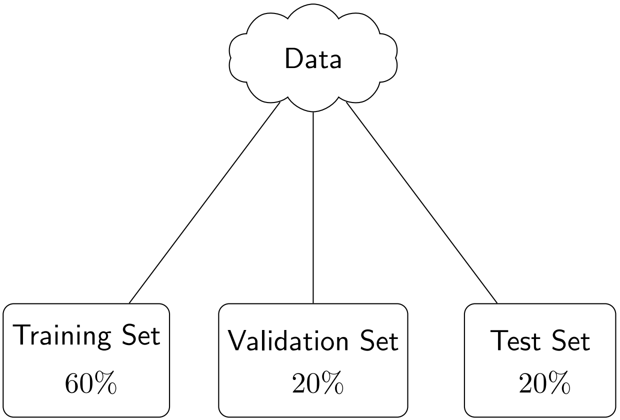
Methods: Data



Methods: Data



Methods: Data



Methods: Measuring Performance

- Need a way to evaluate the performance of our algorithm (P)

Methods: Measuring Performance

- Need a way to evaluate the performance of our algorithm (P)
- Once we know how well it is performing, we can update it

Methods: Measuring Performance

- Need a way to evaluate the performance of our algorithm (P)
- Once we know how well it is performing, we can update it

$$L(\mathbf{y}, \hat{\mathbf{y}}) \in \mathbb{R}$$

Methods: Measuring Performance

Regression

Methods: Measuring Performance

Regression

$$\underset{\mathbf{D}}{\begin{bmatrix} 34 & 147 & 73 \\ 247 & 69 & 13 \\ 66 & 66 & 66 \end{bmatrix}} \times \underset{\mathbf{w}}{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}} + c = \underset{\mathbf{y}}{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}} \xrightarrow{Target} \underset{\hat{\mathbf{y}}}{\begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \hat{y_3} \end{bmatrix}}$$

Methods: Measuring Performance

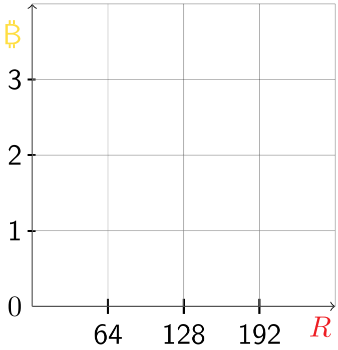
Regression

$$\underbrace{\begin{bmatrix} 34 & 147 & 73 \\ 247 & 69 & 13 \\ 66 & 66 & 66 \end{bmatrix}}_{\mathbf{D}} \times \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}}_{\mathbf{w}} + c = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{\mathbf{y}} \xrightarrow{Target} \underbrace{\begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \hat{y_3} \end{bmatrix}}_{\hat{\mathbf{y}}}$$

$$L(\mathbf{y}, \hat{\mathbf{y}}) = MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{k} \sum_{i=1}^{k} (y_i - \hat{y_i})^2$$

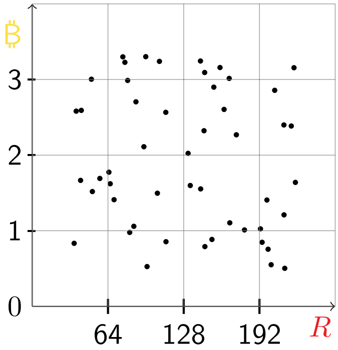
Methods: Measuring Performance

Regression



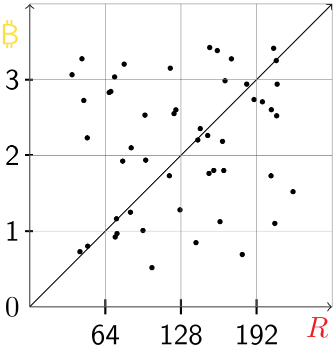
Methods: Measuring Performance

Regression



Methods: Measuring Performance

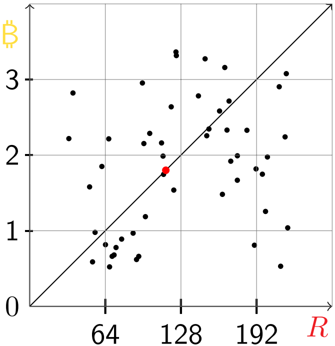
Regression



$$f(\mathbf{b}) = b_1 + b_2 + b_3$$

Methods: Measuring Performance

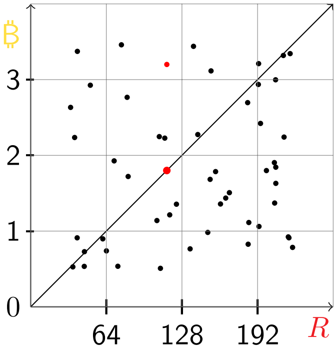
Regression



$$f(\mathbf{b}) = b_1 + b_2 + b_3$$

Methods: Measuring Performance

Regression



$$f(\mathbf{b}) = b_1 + b_2 + b_3$$

Methods: Weight Update

- Through $L(\mathbf{y}, \hat{\mathbf{y}})$ we know how our algorithm is performing

Methods: Weight Update

- Through $L(\mathbf{y}, \hat{\mathbf{y}})$ we know how our algorithm is performing
- We can compute its rate of change (derivative)

Methods: Weight Update

- Through $L(\mathbf{y}, \hat{\mathbf{y}})$ we know how our algorithm is performing
- We can compute its rate of change (derivative)
- Move our weights in the opposite direction

Methods: Weight Update

- Through $L(\mathbf{y}, \hat{\mathbf{y}})$ we know how our algorithm is performing
- We can compute its rate of change (derivative)
- Move our weights in the opposite direction

This is the core idea behind Machine Learning

Methods: Weight Update

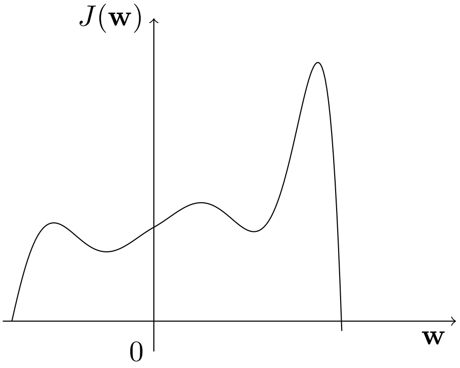
- $L(\mathbf{y}, \hat{\mathbf{y}})$ depends on $\mathbf{y}$ and $\hat{\mathbf{y}}$, but *also* $\mathbf{w}$: $L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w})$

Methods: Weight Update

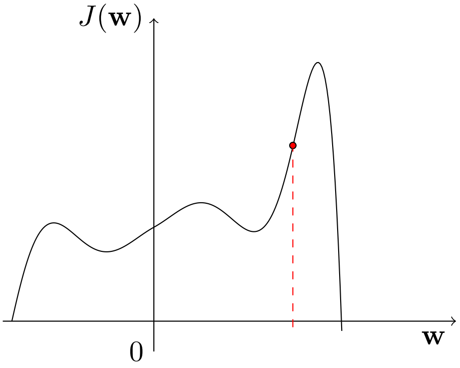
- $L(\mathbf{y}, \hat{\mathbf{y}})$ depends on $\mathbf{y}$ and $\hat{\mathbf{y}}$, but *also* $\mathbf{w}$: $L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w})$
- We can think of $\mathbf{Y}, \hat{\mathbf{Y}}$ as being *constant* and write:

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{Y}_i, \hat{\mathbf{Y}}_i; w)$$

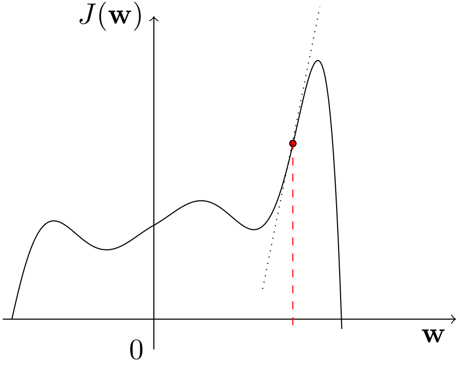
Methods: Weight Update



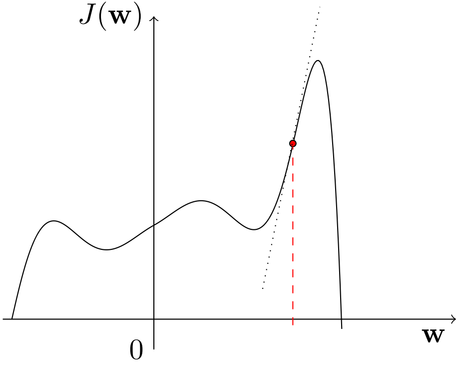
Methods: Weight Update



Methods: Weight Update



Methods: Weight Update

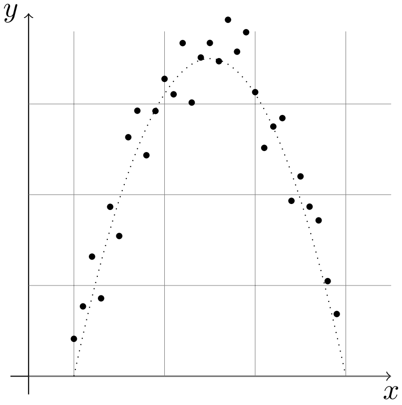


$$\mathbf{w} \leftarrow \mathbf{w} - \text{rate of change at } \mathbf{w}$$

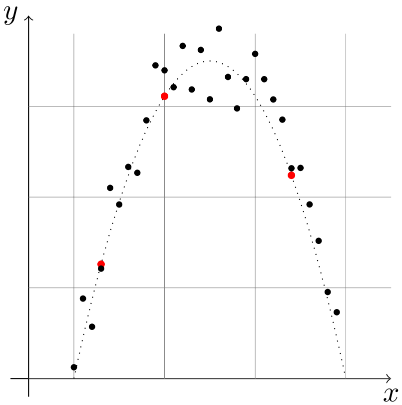
Methods: Regularization

- Generalization
- Regularization
- Capacity

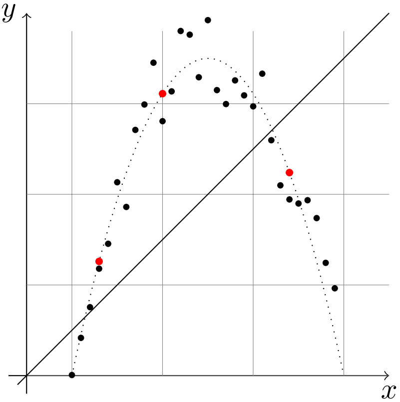
Concepts: Model Capacity



Concepts: Model Capacity

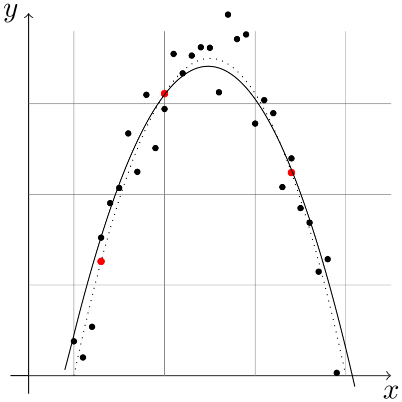


Concepts: Model Capacity



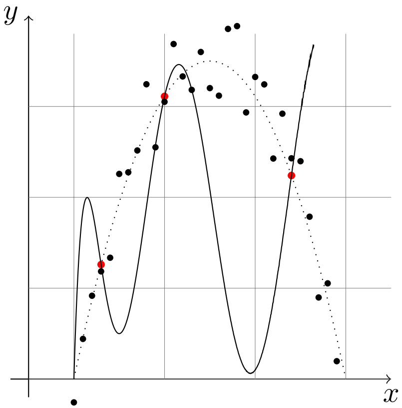
Underfitting

Concepts: Model Capacity



Just Right

Concepts: Model Capacity



Overfitting

Methods: Regularization

$$
\begin{aligned}
f(x) &= 0x^8 + 0x^7 + 0x^6 \\
&+ 0x^5 + 0x^4 + 0x^3 \\
&- 1.56x^2 + 4.67x + 0
\end{aligned}
$$

$$
\begin{aligned}
g(x) &= -0.69x^8 + 10.90x^7 - 69.52x^6 \\
&+ 229.12x^5 - 413.77x^4 + 399.50x^3 \\
&- 185.95x^2 + 33.51x + 7.17 \cdot 10^{-8}
\end{aligned}
$$

Methods: Regularization

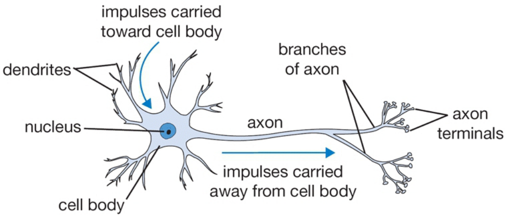
$$J(\mathbf{w}) = MSE(\mathbf{y}, \hat{\mathbf{y}}) + \lambda(\mathbf{w}^\top \mathbf{w})$$

Neural Networks

Neural Networks

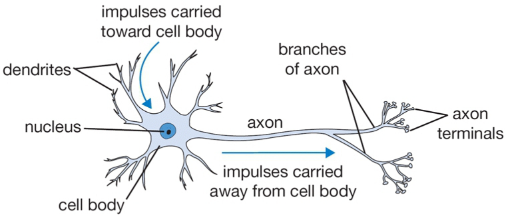
Neural Networks: Biological Motivation

Neural Networks: Biological Motivation



[?]

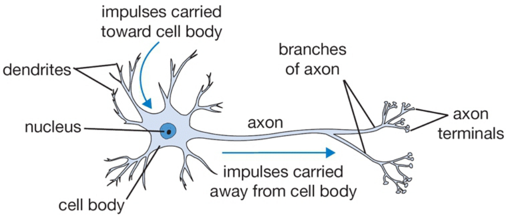
Neural Networks: Biological Motivation



[?]

- Receive electrochemical signals through *dendrites*

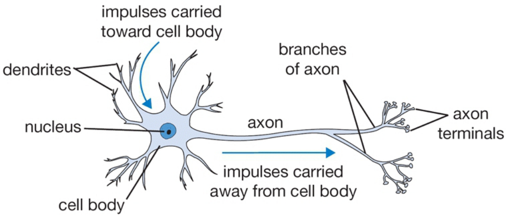
Neural Networks: Biological Motivation



[?]

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

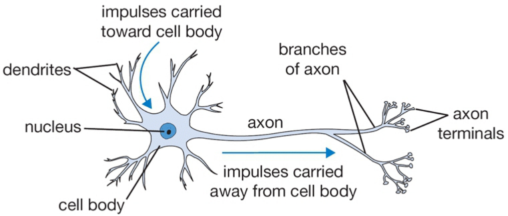
Neural Networks: Biological Motivation



[?]

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

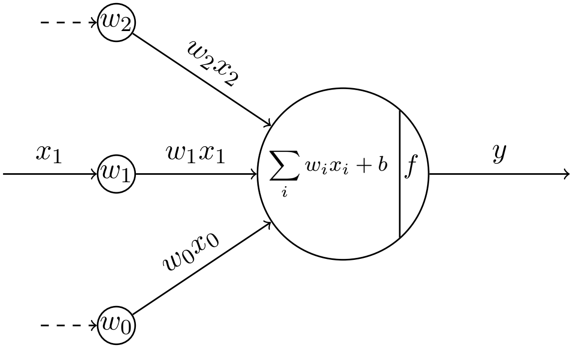
Neural Networks: Biological Motivation



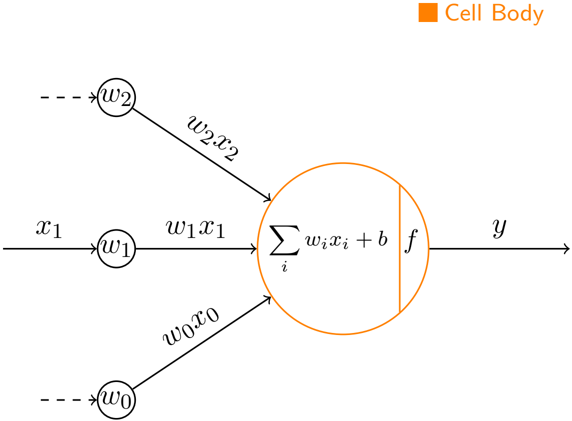
[?]

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

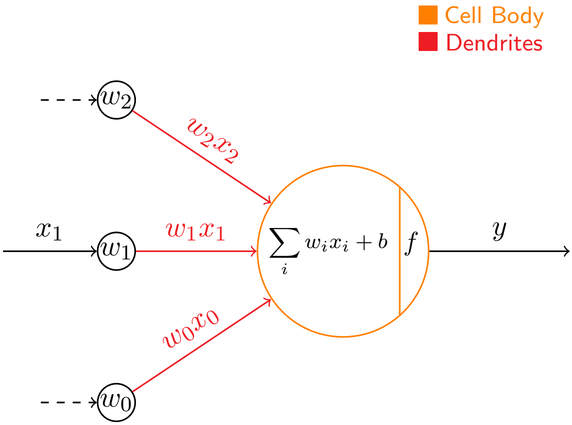
Neural Networks: Mathematical Model



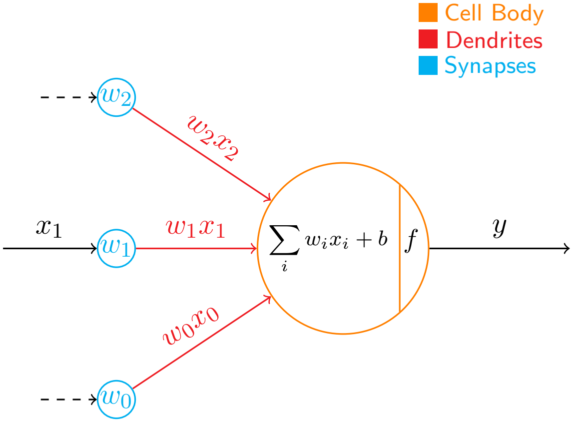
Neural Networks: Mathematical Model



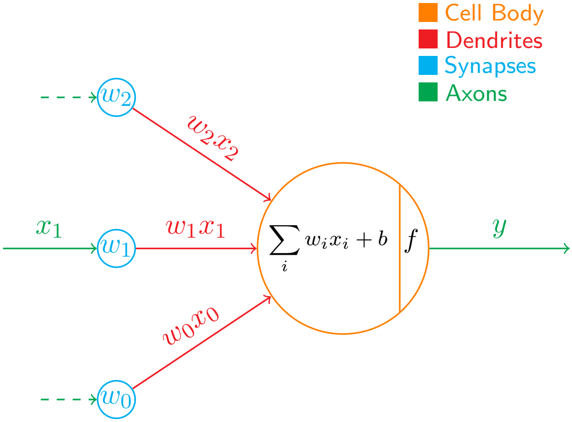
Neural Networks: Mathematical Model



Neural Networks: Mathematical Model



Neural Networks: Mathematical Model



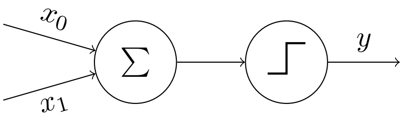
Artificial Neural Networks

Artificial Neural Networks

- First attempt by McCulloch and Pitts in 1943

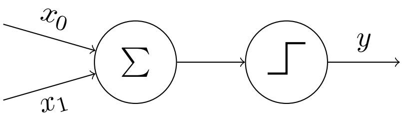
Artificial Neural Networks

- First attempt by McCulloch and Pitts in 1943
- Sum binary inputs and threshold them



Artificial Neural Networks

- First attempt by McCulloch and Pitts in 1943
- Sum binary inputs and threshold them
- Can learn AND, NOT and OR functions

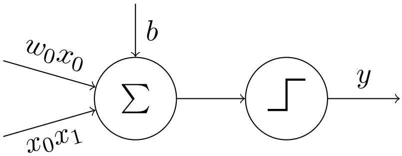


Artificial Neural Networks

- Frank Rosenblatt improved on this model in 1957

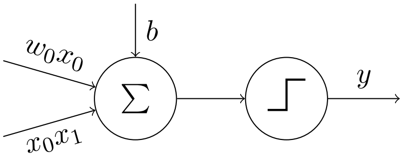
Artificial Neural Networks

- 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
- 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

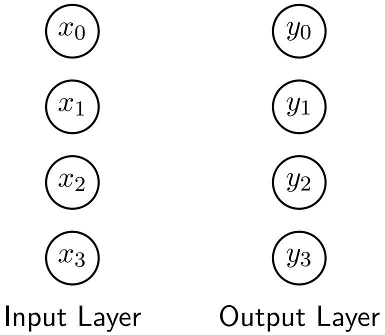
Input: A dataset of $(\mathbf{x}, \hat{y})$ pairs
Output: Trained Perceptron
for all $(\mathbf{x}, \hat{y})$ in dataset **do**:
    $y \leftarrow f(\mathbf{w}^\top \mathbf{x} + b)$
    if $y \neq \hat{y}$ **then**
        if $\hat{y} = 0 \wedge y = 1$ **then**
            Decrease all weights w_i where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

Artificial Neural Networks

- Perceptrons even work for multi-class classification

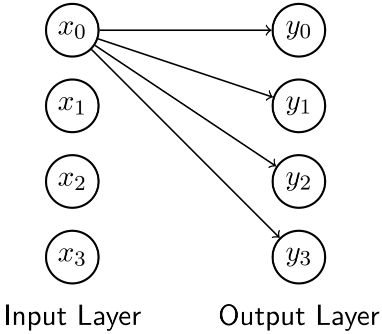
Artificial Neural Networks

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



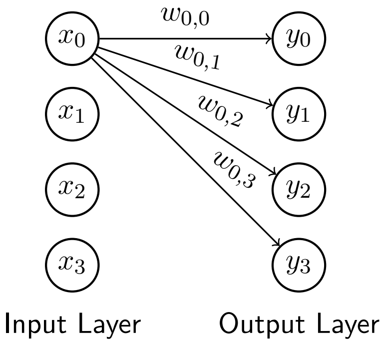
Artificial Neural Networks

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



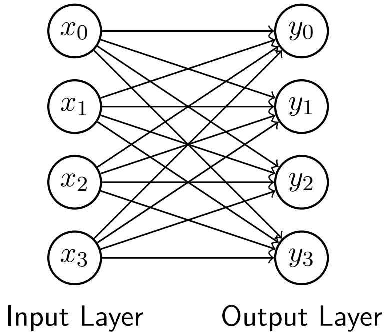
Artificial Neural Networks

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



Artificial Neural Networks

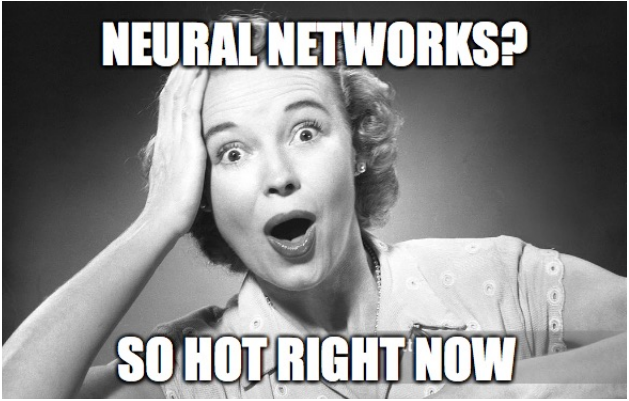
- Perceptrons even work for multi-class classification
- Just use more perceptrons
- This is now a *multilayer perceptron* (MLP)



NEW NAVY DEVICE LEARNS BY DOING

WASHINGTON, July 7, 1958 (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

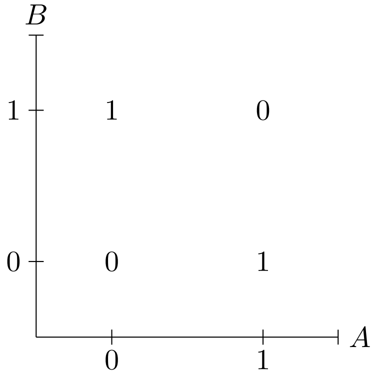
- Perceptrons have one fundamental problem

Artificial Neural Networks

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

Artificial Neural Networks

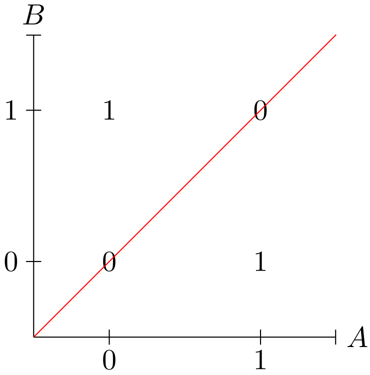
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

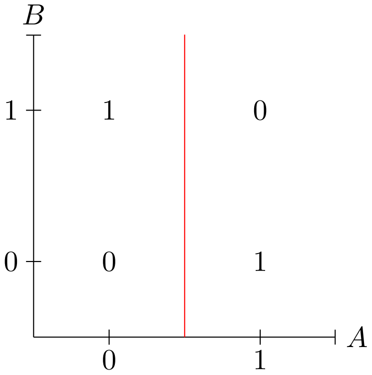
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

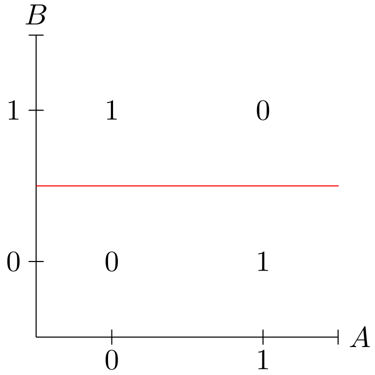
- Perceptrons have one fundamental problem
- They can only learn linear functions



A XOR B

Artificial Neural Networks

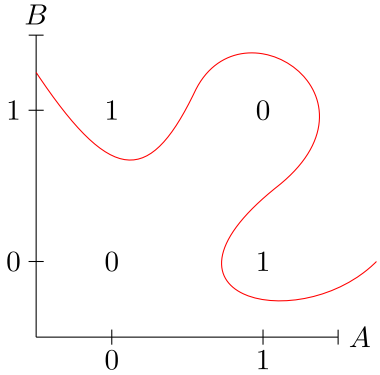
- Perceptrons have one fundamental problem
- They can only learn linear functions



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

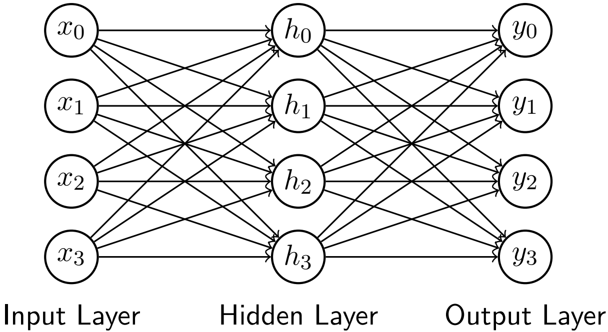
- The solution:
  - A hidden layer

Artificial Neural Networks

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

Artificial Neural Networks

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

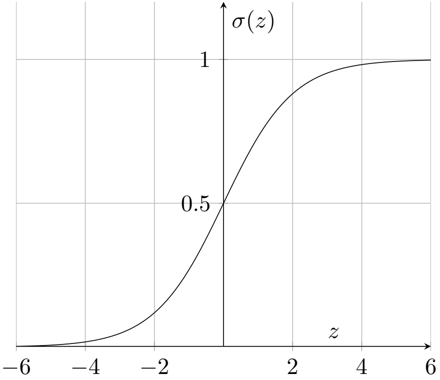


Activation Functions

- Three important activation functions

Activation Functions

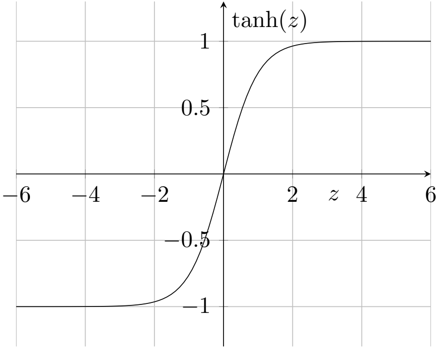
- Three important activation functions



Le *Sigmoid*

Activation Functions

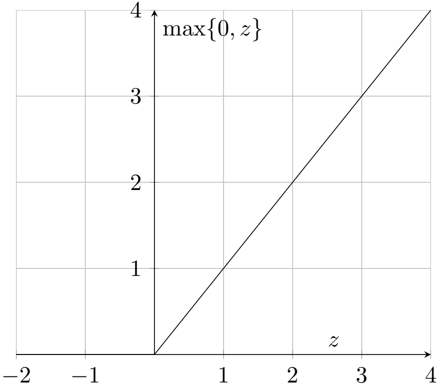
- Three important activation functions



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

$$\mathbf{D} = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{cc} p & c \\ \end{array}$$

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

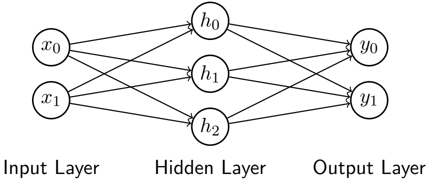
Neural Networks by Foot

- 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

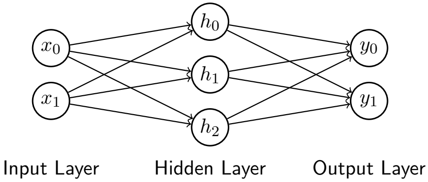
$$\mathbf{D} = \begin{array}{c} \begin{array}{cc} p & c \end{array} \\ \begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix} \end{array} \quad \hat{\mathbf{Y}} = \begin{array}{c} \begin{array}{cc} \texttt{l33t} & \texttt{n00b} \end{array} \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}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

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

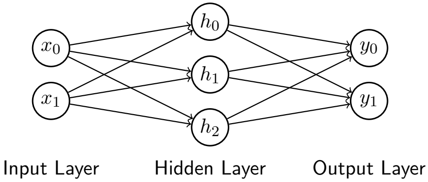


Neural Networks by Foot



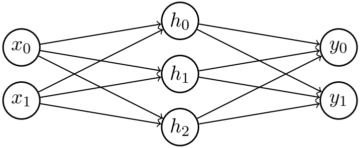
$$\underset{\mathbf{D}}{\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}}$$

Neural Networks by Foot



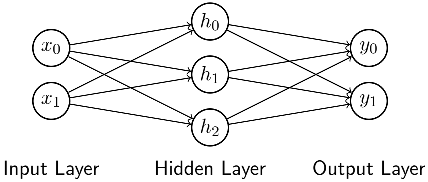
$$\underbrace{\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}}_{\mathbf{D}} \times \underbrace{\begin{bmatrix} -0.04 & -0.43 & 0.57 \\ 0.04 & 0.52 & -0.6 \end{bmatrix}}_{\mathbf{W_1}}$$

Neural Networks by Foot



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

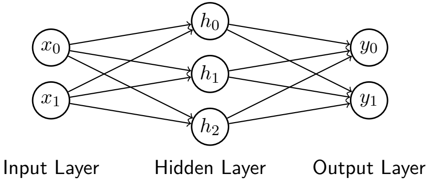
Neural Networks by Foot



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

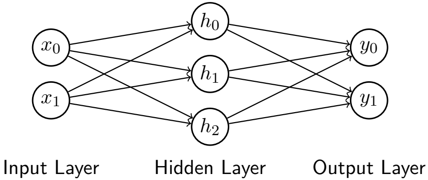
$$= \underbrace{\begin{bmatrix} 13.37 & 183.66 & -210.46 \\ 3.05 & 60.33 & -67.91 \\ 41.13 & 515.49 & -596.08 \end{bmatrix}}_{\mathbf{H}}$$

Neural Networks by Foot



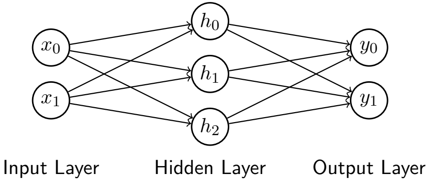
$$\mathrm{relu}(\mathbf{H})$$

Neural Networks by Foot



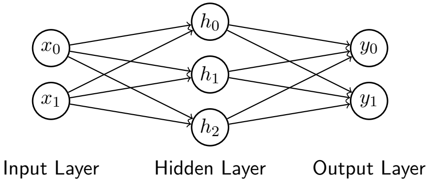
$$\mathrm{relu}(\mathbf{H}) = \max\{0, \mathbf{H}\}$$

Neural Networks by Foot



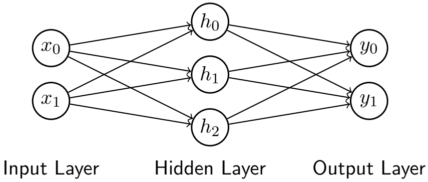
$$\mathrm{relu}(\mathbf{H}) = \max\{0, \mathbf{H}\} = \underbrace{\begin{bmatrix} 13.37 & 183.66 & 0.0 \\ 3.05 & 60.33 & 0.0 \\ 41.13 & 515.49 & 0.0 \end{bmatrix}}_{\mathbf{H}'}$$

Neural Networks by Foot



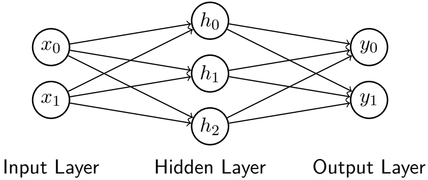
$$\mathbf{H}'$$

Neural Networks by Foot



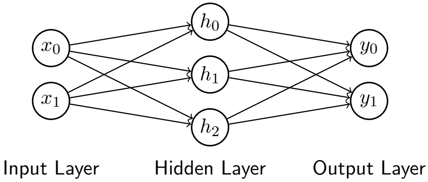
$$\mathbf{H}' \times \underbrace{\begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix}}_{\mathbf{W_2}}$$

Neural Networks by Foot



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

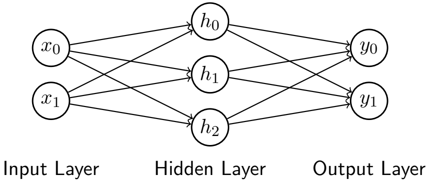
Neural Networks by Foot



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

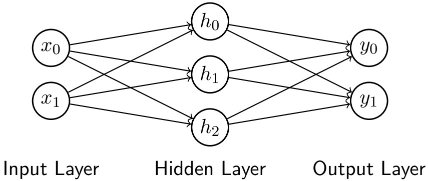
$$= \underbrace{\begin{bmatrix} -2.53 & 211.85 \\ -1.55 & 69.01 \\ -5.16 & 596.17 \end{bmatrix}}_{\mathbf{Y}}$$

Neural Networks by Foot



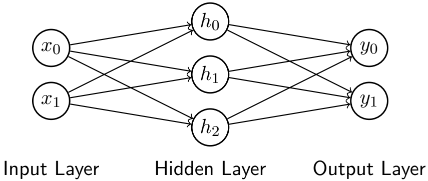
$$\text{softmax}(\mathbf{Y}) =$$

Neural Networks by Foot



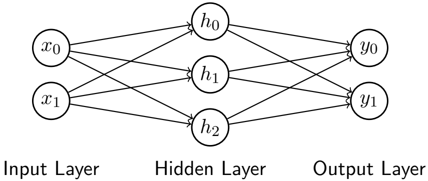
$$\text{softmax}(\mathbf{Y}) = \underbrace{\begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \\ 0.0 & 1.0 \end{bmatrix}}_{\mathbf{Y}}$$

Neural Networks By Foot



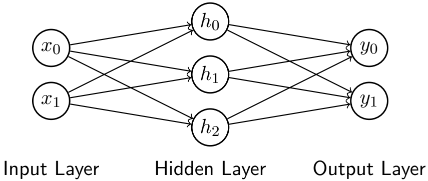
- Let $\mathcal{W}$ be $[\mathbf{W}_1, \mathbf{W}_2]$

Neural Networks By Foot



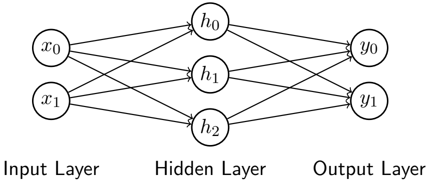
- Let $\mathcal{W}$ be $[\mathbf{W}_1, \mathbf{W}_2]$
- Then $J(\mathcal{W})$ is the network's loss

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

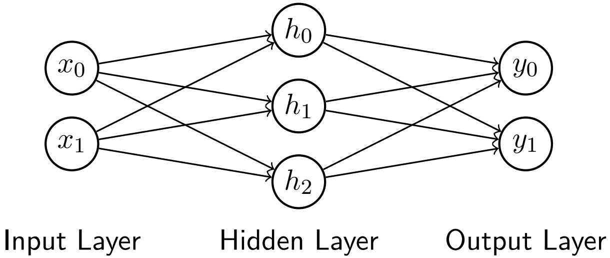
- One recent regularization technique is *Dropout*
- It is used especially for deep neural networks
- Basic idea:
  - We deactivate neurons on each activation with probability p
  - This prevents neurons from depending too much on each other

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
  - This prevents neurons from depending too much on each other

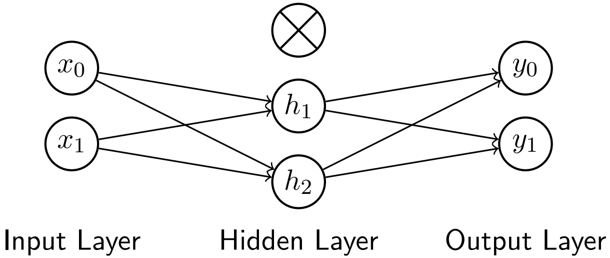
Dropout

- 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
  - This prevents neurons from depending too much on each other



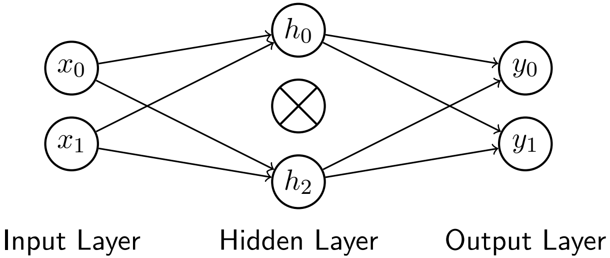
Dropout

- 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
  - This prevents neurons from depending too much on each other



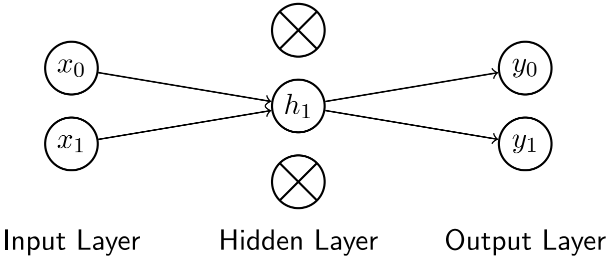
Dropout

- 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
  - 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
  - This prevents neurons from depending too much on each other



Time to go Deeper

What if the meaning of life is to spend your time thinking about the meaning of life?

Deep Learning

Deep Learning

The why

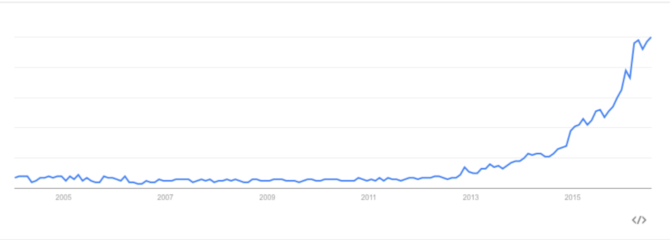
Deep Learning

The why, the what

Deep Learning

The why, the what and the ugly.

Deep Learning



The Curse of Dimensionality

Deep Learning assumes that data is structured

Deep Learning assumes that data is structured

hierarchically

Convolutional Neural Networks

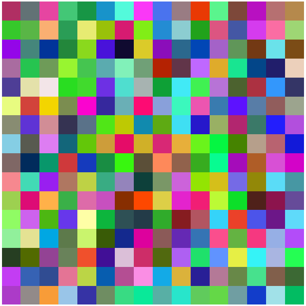
- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition

Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images

Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images



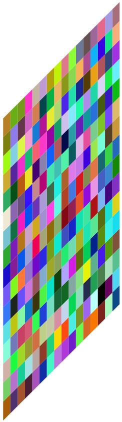
Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images

50	44	59	84	35	86	26	46	13	86	7	79	47	58	15	90
36	78	16	96	98	21	82	0	41	93	35	72	82	34	51	94
98	88	74	20	59	6	31	92	13	69	28	94	32	21	90	36
78	9	67	58	85	50	89	19	14	36	75	60	94	7	72	62
81	3	72	2	50	86	50	17	67	8	96	78	28	4	32	95
18	12	99	99	71	98	42	63	95	22	32	89	24	8	38	67
96	63	97	99	81	46	81	88	70	3	98	24	37	87	64	29
61	81	82	37	53	12	66	25	21	34	24	28	89	64	20	10
57	70	82	63	2	78	46	70	12	7	99	99	18	42	9	65
67	24	63	1	29	61	71	43	41	2	53	64	36	40	13	65
56	24	17	14	30	47	66	82	29	37	48	68	15	47	98	32
62	19	43	73	47	98	38	6	22	10	33	98	7	69	66	12
60	65	60	60	91	60	69	91	77	30	31	91	10	44	81	60
30	98	6	46	83	62	94	7	28	28	64	93	59	89	28	32
34	33	42	64	72	31	18	29	68	96	92	26	9	4	3	91
23	88	12	79	84	97	79	78	46	47	10	99	6	40	88	12

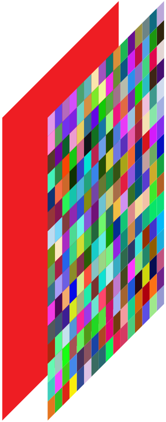
Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images



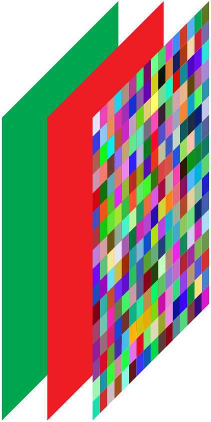
Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images



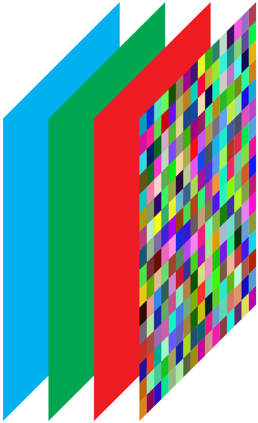
Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images



Convolutional Neural Networks

- Convolutional Neural Networks (CNNs; ConvNets) are used in image recognition
- They assume their input are images



Convolutional Neural Network

- We want to classify images into one of k classes

Convolutional Neural Network

- We want to classify images into one of k classes
- The model should learn to extract features

Convolutional Neural Network

- We want to classify images into one of k classes
- The model should learn to extract features
- We expect it to exploit the hierarchical nature of the data

Convolutional Neural Network

- We want to classify images into one of k classes
- The model should learn to extract features
- We expect it to exploit the hierarchical nature of the data
  1. Lines and edges
  2. Corners and contours
  3. Abstract components (e.g. noses, ears, feet)
  4. Entire objects (e.g. faces, spaceship)

Convolutional Neural Network

- We want to classify images into one of k classes
- The model should learn to extract features
- We expect it to exploit the hierarchical nature of the data
  1. Lines and edges
  2. Corners and contours
  3. Abstract components (e.g. noses, ears, feet)
  4. Entire objects (e.g. faces, spaceship)
- First idea: just feed the pixels into a neural network

$$\underset{R}{\begin{bmatrix}116 & 80 \\ 170 & 194\end{bmatrix}} \quad \underset{G}{\begin{bmatrix}82 & 78 \\ 5 & 236\end{bmatrix}} \quad \underset{B}{\begin{bmatrix}76 & 139 \\ 245 & 236\end{bmatrix}}$$

$$\underset{R}{\begin{bmatrix} 116 & 80 \\ 170 & 194 \end{bmatrix}} \quad \underset{G}{\begin{bmatrix} 82 & 78 \\ 5 & 236 \end{bmatrix}} \quad \underset{B}{\begin{bmatrix} 76 & 139 \\ 245 & 236 \end{bmatrix}}$$

$\texttt{concatenate(}R\texttt{.flatten, }G\texttt{.flatten, }B\texttt{.flatten)}$

Convolutional Neural Network

$$\underset{R}{\begin{bmatrix} 116 & 80 \\ 170 & 194 \end{bmatrix}} \quad \underset{G}{\begin{bmatrix} 82 & 78 \\ 5 & 236 \end{bmatrix}} \quad \underset{B}{\begin{bmatrix} 76 & 139 \\ 245 & 236 \end{bmatrix}}$$

$$R.\texttt{flatten} = \begin{bmatrix} 116 & 80 & 170 & 194 \end{bmatrix}$$

$$\underset{R}{\begin{bmatrix} 116 & 80 \\ 170 & 194 \end{bmatrix}} \quad \underset{G}{\begin{bmatrix} 82 & 78 \\ 5 & 236 \end{bmatrix}} \quad \underset{B}{\begin{bmatrix} 76 & 139 \\ 245 & 236 \end{bmatrix}}$$

$$R.\texttt{flatten} = \begin{bmatrix} 116 & 80 & 170 & 194 \end{bmatrix}$$
$$G.\texttt{flatten} = \begin{bmatrix} 82 & 78 & 5 & 236 \end{bmatrix}$$

Convolutional Neural Network

$$\underset{R}{\begin{bmatrix} 116 & 80 \\ 170 & 194 \end{bmatrix}} \qquad \underset{G}{\begin{bmatrix} 82 & 78 \\ 5 & 236 \end{bmatrix}} \qquad \underset{B}{\begin{bmatrix} 76 & 139 \\ 245 & 236 \end{bmatrix}}$$

$$R.\texttt{flatten} = \begin{bmatrix} 116 & 80 & 170 & 194 \end{bmatrix}$$
$$G.\texttt{flatten} = \begin{bmatrix} 82 & 78 & 5 & 236 \end{bmatrix}$$
$$B.\texttt{flatten} = \begin{bmatrix} 76 & 139 & 245 & 236 \end{bmatrix}$$

Convolutional Neural Network

$$\underset{R}{\begin{bmatrix} 116 & 80 \\ 170 & 194 \end{bmatrix}} \qquad \underset{G}{\begin{bmatrix} 82 & 78 \\ 5 & 236 \end{bmatrix}} \qquad \underset{B}{\begin{bmatrix} 76 & 139 \\ 245 & 236 \end{bmatrix}}$$

$$\begin{aligned} R.\texttt{flatten} &= \begin{bmatrix} 116 & 80 & 170 & 194 \end{bmatrix} \\ G.\texttt{flatten} &= \begin{bmatrix} 82 & 78 & 5 & 236 \end{bmatrix} \\ B.\texttt{flatten} &= \begin{bmatrix} 76 & 139 & 245 & 236 \end{bmatrix} \end{aligned}$$

$$\mathbf{x} = \left[116, 80, 170, 194, 82, 78, 5, 236, 76, 139, 245, 236\right]$$

Convolutional Neural Networks

- Why is this a bad idea?

Convolutional Neural Networks

- Why is this a bad idea?
  1. Fully connected NNs scale badly for images

Convolutional Neural Networks

- Why is this a bad idea?
  1. Fully connected NNs scale badly for images
     - $200 \times 200 \times 3 = 120,000$ features

Convolutional Neural Networks

- Why is this a bad idea?
  1. Fully connected NNs scale badly for images
     - $200 \times 200 \times 3 = 120,000$ features
     - With 100 hidden units: $100 \times 120,000 = 12,000,000$

Convolutional Neural Networks

- Why is this a bad idea?
  1. Fully connected NNs scale badly for images
     - $200 \times 200 \times 3 = 120,000$ features
     - With 100 hidden units: $100 \times 120,000 = 12,000,000$
  2. It assumes every pixel has entirely new information

Convolutional Neural Networks

This is a cat ♥

Convolutional Neural Networks

Still a cat ♥♥

Convolutional Neural Networks

Half cat / half salad ♥♥♥

Convolutional Neural Networks

Minecraft cat ♥♥♥♥

Convolutional Neural Networks

- Our network learns to detect a feature in one part of the image

Convolutional Neural Networks

- Our network learns to detect a feature in one part of the image
- Wouldn't it make sense to reuse that information?

Convolutional Neural Networks

- Our network learns to detect a feature in one part of the image
- Wouldn't it make sense to reuse that information?
- Yes!

Weight Sharing

Recipe for a Convolutional Neural Network

Recipe for a Convolutional Neural Network

- Ingredients

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$
- Cooking

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$
- Cooking
  - Put the image into the oven at 150°C

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$
- Cooking
  - Don't put the image into the oven at 150°C

Convolutional Neural Network: Mechanics

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$
- Cooking
  - Don't put the image into the oven at 150°C
  - Slide the kernel across the image

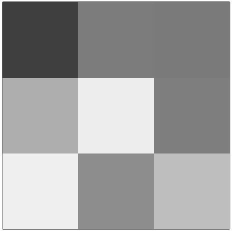
Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$
- Cooking
  - Don't put the image into the oven at 150°C
  - Slide the kernel across the image
  - Compute the "dot product" for each configuration

Recipe for a Convolutional Neural Network

- Ingredients
  1. Image I with dimension $w \times h \times d$
  2. A kernel (filter) K of size $k \times k \times d$
- Cooking
  - Don't put the image into the oven at 150°C
  - Slide the kernel across the image
  - Compute the "dot product" for each configuration
  - (This is a convolution $I * K$)

Convolutional Neural Network: Mechanics



Image

Convolutional Neural Network: Mechanics

0.4	0.9	0.1
0.7	0.2	0.6
0.8	0.3	0.5

Image

Convolutional Neural Network: Mechanics

0.4	0.9	0.1
0.7	0.2	0.6
0.8	0.3	0.5

Image

5.7	2.4
3.1	0.9

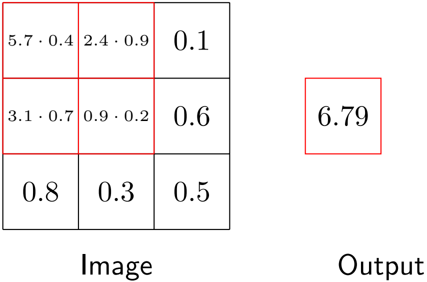
Kernel

Convolutional Neural Network: Mechanics

$5.7 \cdot 0.4$	$2.4 \cdot 0.9$	0.1
$3.1 \cdot 0.7$	$0.9 \cdot 0.2$	0.6
0.8	0.3	0.5

Image

Convolutional Neural Network: Mechanics



Convolutional Neural Network: Mechanics

0.4	$5.7 \cdot 0.9$	$2.4 \cdot 0.1$
0.7	$3.1 \cdot 0.2$	$0.9 \cdot 0.6$
0.8	0.3	0.5

Image

6.79

Output

Convolutional Neural Network: Mechanics

0.4	$5.7 \cdot 0.9$	$2.4 \cdot 0.1$
0.7	$3.1 \cdot 0.2$	$0.9 \cdot 0.6$
0.8	0.3	0.5

Image

6.79	6.53

Output

Convolutional Neural Network: Mechanics

0.4	0.9	0.1
$5.7 \cdot 0.7$	$2.4 \cdot 0.2$	0.6
$3.1 \cdot 0.8$	$0.9 \cdot 0.3$	0.5

Image

6.79	6.53

Output

Convolutional Neural Network: Mechanics

0.4	0.9	0.1
$5.7 \cdot 0.7$	$2.4 \cdot 0.2$	0.6
$3.1 \cdot 0.8$	$0.9 \cdot 0.3$	0.5

Image

6.79	6.53
7.67	

Output

Convolutional Neural Network: Mechanics

0.4	0.9	0.1
0.7	$5.7 \cdot 0.2$	$2.4 \cdot 0.6$
0.8	$3.1 \cdot 0.3$	$0.9 \cdot 0.5$

Image

6.79	6.53
7.67	

Output

Convolutional Neural Network: Mechanics

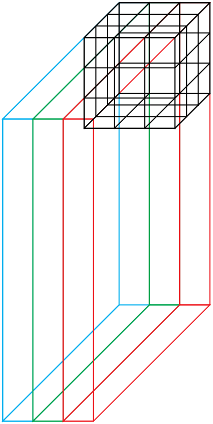
0.4	0.9	0.1
0.7	$5.7 \cdot 0.2$	$2.4 \cdot 0.6$
0.8	$3.1 \cdot 0.3$	$0.9 \cdot 0.5$

Image

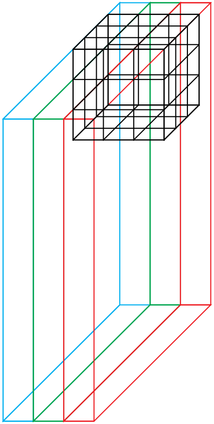
6.79	6.53
7.67	3.96

Output

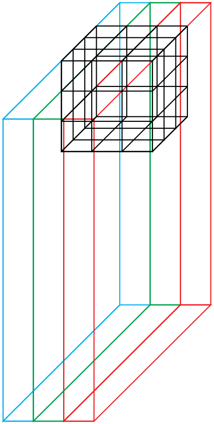
Convolutional Neural Networks: Mechanics



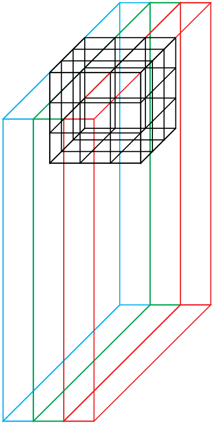
Convolutional Neural Networks: Mechanics



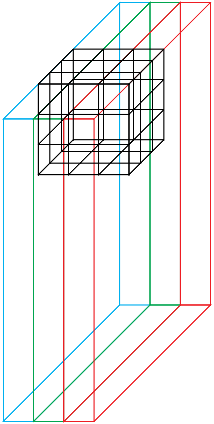
Convolutional Neural Networks: Mechanics



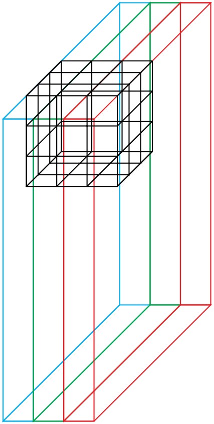
Convolutional Neural Networks: Mechanics



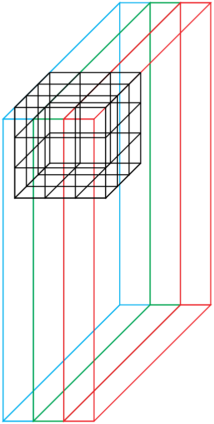
Convolutional Neural Networks: Mechanics



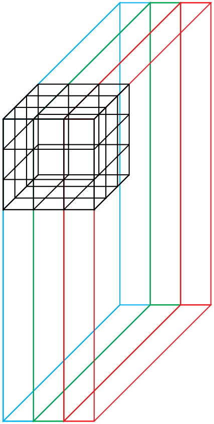
Convolutional Neural Networks: Mechanics



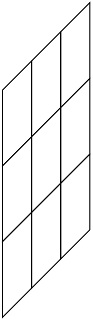
Convolutional Neural Networks: Mechanics



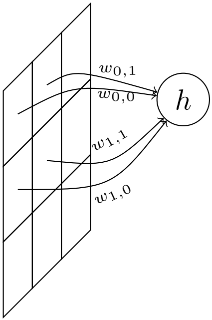
Convolutional Neural Networks: Mechanics



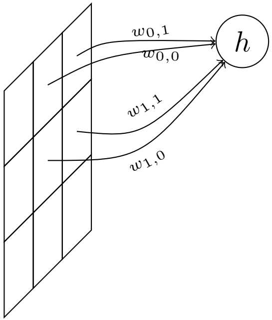
Convolutional Neural Network: Mechanics



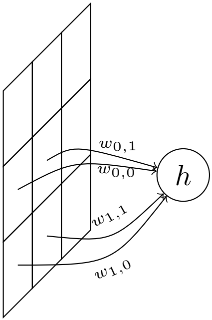
Convolutional Neural Network: Mechanics



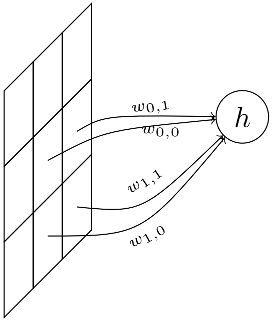
Convolutional Neural Network: Mechanics



Convolutional Neural Network: Mechanics



Convolutional Neural Network: Mechanics



Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters

Convolutional Neural Network: Hyperparameters

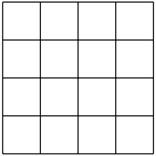
- Convolutional Neural Networks have three hyperparameters
    1. Kernel size

Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride

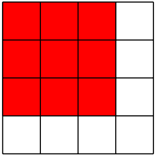
Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)



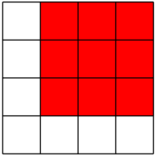
Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)



Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)



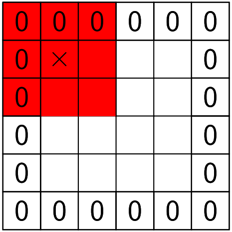
Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)

0	0	0	0	0	0
0					0
0					0
0					0
0					0
0	0	0	0	0	0

Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)



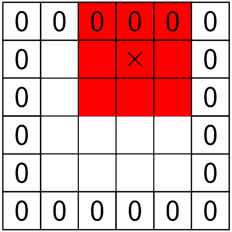
Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)

0	0	0	0	0	0
0		×			0
0					0
0					0
0					0
0	0	0	0	0	0

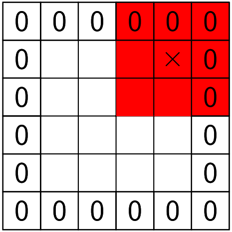
Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)



Convolutional Neural Network: Hyperparameters

- Convolutional Neural Networks have three hyperparameters
  1. Kernel size
  2. Kernel stride
  3. Padding (valid or same)



Convolutional Neural Networks: Pooling

- *Pooling* achieves translational invariance

Convolutional Neural Networks: Pooling

- *Pooling* achieves translational invariance
- A form of downsampling

Convolutional Neural Networks: Pooling

- *Pooling* achieves translational invariance
- A form of downsampling
- The maximum stays the maximum

66	2
6	32

Convolutional Neural Networks: Pooling

- *Pooling* achieves translational invariance
- A form of downsampling
- The maximum stays the maximum

6	2
66	32

Convolutional Neural Networks: Pooling

- *Pooling* achieves translational invariance
- A form of downsampling
- The maximum stays the maximum

2	66
6	32

Convolutional Neural Networks: Pooling

- *Pooling* achieves translational invariance
- A form of downsampling
- The maximum stays the maximum

32	6
2	66

Convolutional Neural Network: Architecture

```
INPUT ->
[[CONV -> RELU]*N -> POOL?]*M ->
[FC -> RELU]*K -> FC
```

[?]

Convolutional Neural Networks: Intuition

Convolutional Neural Networks: Intuition

- Each layer of a ConvNet learns to detect features

Convolutional Neural Networks: Intuition

- Each layer of a ConvNet learns to detect features
- Later layers combine features of earlier layers

Convolutional Neural Networks: Intuition

- Each layer of a ConvNet learns to detect features
- Later layers combine features of earlier layers
- For early layers, we can still gain an intuition of what they do

What's in a kernel?

0	1	0
0	1	0
0	1	0

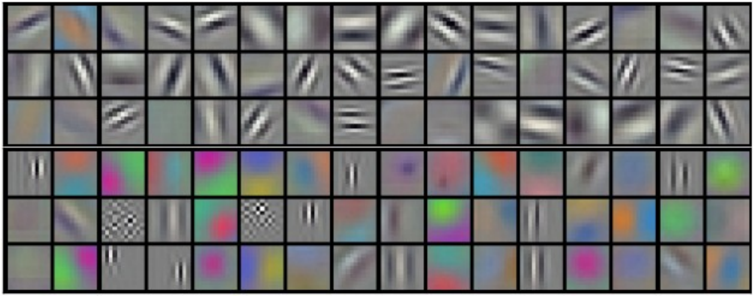
Patterns

What's in a kernel?

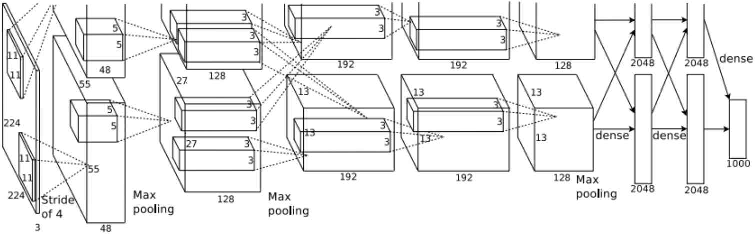
0	0	0
-1	1	0
0	0	0

Features

Convolutional Neural Networks: Intuition



Case Study: AlexNet



Sequences

- Humans think in sequences

Sequences

- Humans think in sequences
- Simple neural networks don't

Sequences

- Humans think in sequences
- Simple neural networks don't
- Sequences give single entities context

I eat people

Sequences

- Humans think in sequences
- Simple neural networks don't
- Sequences give single entities context

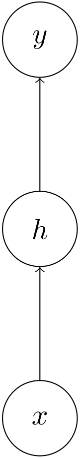
I enjoy eating dinner with people

Sequences

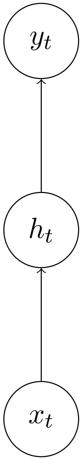
- Humans think in sequences
- Simple neural networks don't
- Sequences give single entities context
- The key to understanding sequences is memory

I enjoy eating dinner with people

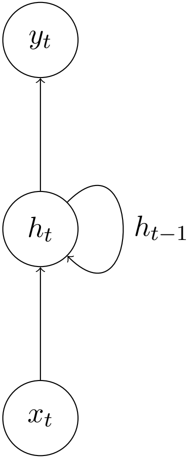
Machine Memory



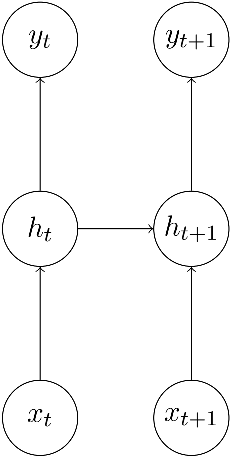
Machine Memory



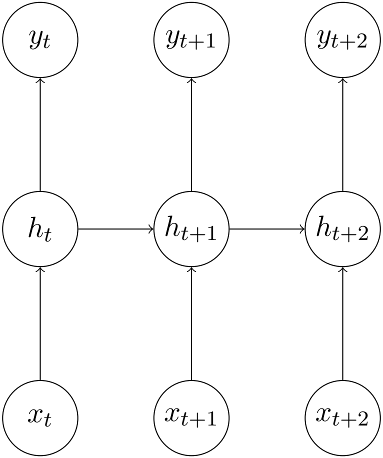
Machine Memory



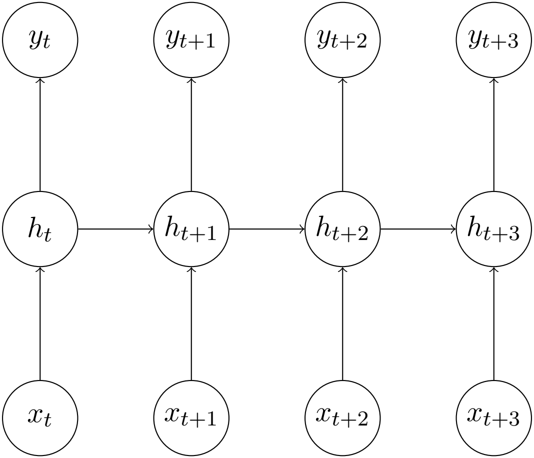
Machine Memory



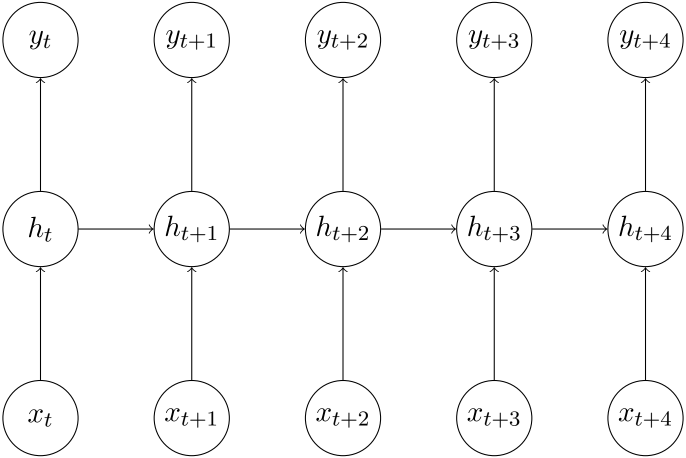
Machine Memory



Machine Memory



Machine Memory



Machine Memory

$$h_t = f(w \cdot x + b)$$

Machine Memory

$$h_t = f(\mathbf{w}^\top [h_{t-1}, x] + b)$$

Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) share weights through time
- They have *memory*
- And they have a problem:

The Vanishing Gradient Problem

LSTMs

- RNN's are forgetful

LSTMs

- RNN's are forgetful
- *Long Short Term Memory* (LSTM) Units solve this problem

LSTMs

- RNN's are forgetful
- *Long Short Term Memory* (LSTM) Units solve this problem
- Developed by Schmidhuber and Hochreiter at TUM in 1997

LSTMs

LSTMs

- LSTMs are a lot like flip-flops

LSTMs

- LSTMs are a lot like flip-flops
- They have three *gates*

LSTMs

- LSTMs are a lot like flip-flops
- They have three *gates*
  - Input gate $G_i(x, h_{t-1}) = \sigma(\mathbf{w}_i^\top [x, h_{t-1}] + b_i)$

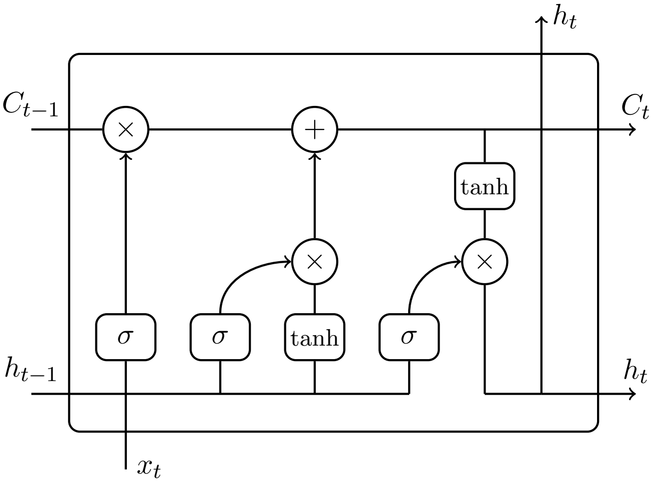
LSTMs

- LSTMs are a lot like flip-flops
- They have three *gates*
  - Input gate $G_i(x, h_{t-1}) = \sigma(\mathbf{w}_i^\top [x, h_{t-1}] + b_i)$
  - Forget gate $G_f(x, h_{t-1}) = \sigma(\mathbf{w}_f^\top [x, h_{t-1}] + b_f)$

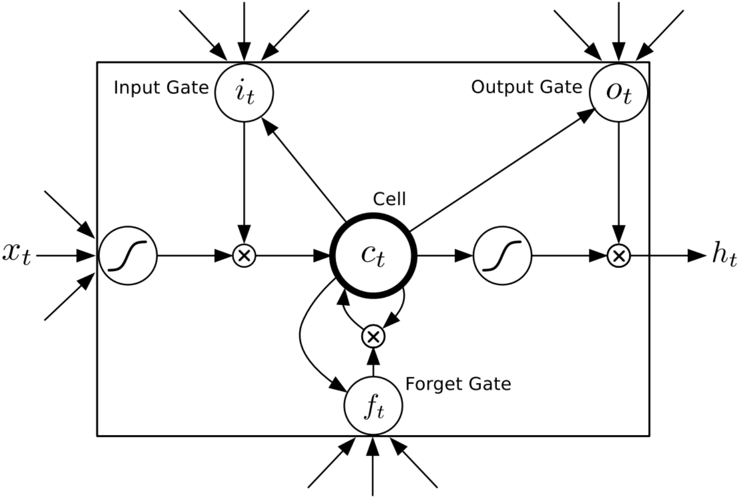
LSTMs

- LSTMs are a lot like flip-flops
- They have three *gates*
  - Input gate $G_i(x, h_{t-1}) = \sigma(\mathbf{w}_i^\top [x, h_{t-1}] + b_i)$
  - Forget gate $G_f(x, h_{t-1}) = \sigma(\mathbf{w}_f^\top [x, h_{t-1}] + b_f)$
  - Output gate $G_o(x, h_{t-1}) = \sigma(\mathbf{w}_o^\top [x, h_{t-1}] + b_o)$

LSTMs



LSTMs



;

LSTMs: What can they do?

So, what can LSTMs actually do?

LSTMs: What can they do?

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tklrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

Iteration: 100

[?]

LSTMs: What can they do?

"Tmont thithey" fomesscerliund Keushey. Thom here sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

Iteration: 300

[**?**]

LSTMs: What can they do?

we counter. He stutn co des. His stanted out one ofler that concossions and was to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn

Iteration: 500

[**?**]

LSTMs: What can they do?

Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort how, and Gogition is so overelical and ofter.

Iteration: 700

[**?**]

LSTMs: What can they do?

“Why do what that day,” replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him. Pierre aking his soul came to the packs and drove up his father-in-law women.

Iteration: 2000

[**?**]

LSTMs: What can they do?

They can write Linux kernel code!

Deep Learning

- Why the recent success of deep learning?

Deep Learning

- Why the recent success of deep learning?
- Three reasons

Deep Learning

- Why the recent success of deep learning?
- Three reasons
    1. Better hardware

Deep Learning

- Why the recent success of deep learning?
- Three reasons
    1. Better hardware
    2. More data

Deep Learning

- Why the recent success of deep learning?
- Three reasons
  1. Better hardware
  2. More data
  3. Better methods

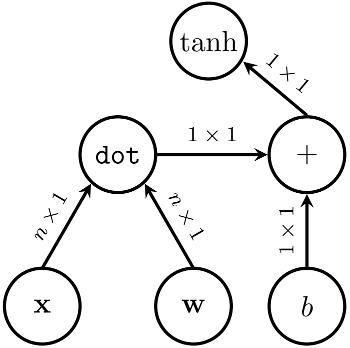
The Ugly

TensorFlow

Feel the TensorFlow

Computational Paradigms

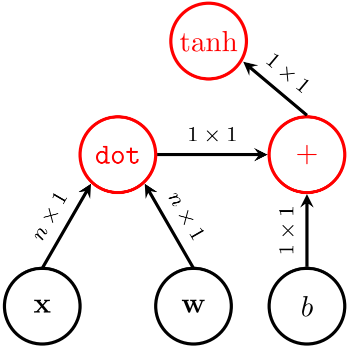
Computational Paradigms



$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

Computational Paradigms

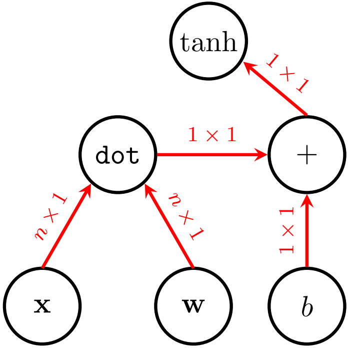


$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

1. Operations

Computational Paradigms

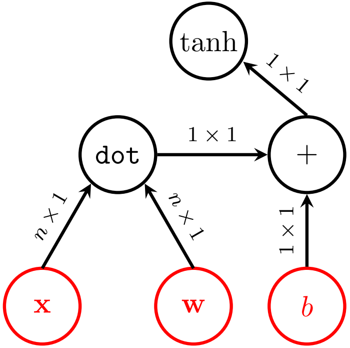


$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

1. Operations
2. Tensors

Computational Paradigms

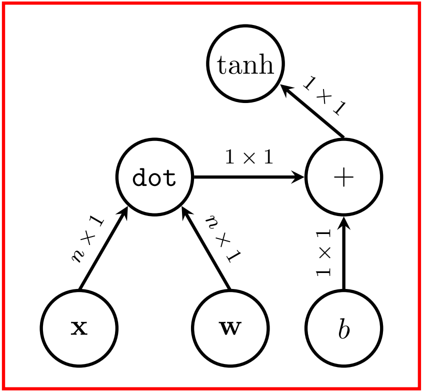


$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

1. Operations
2. Tensors
3. Variables

Computational Paradigms



$$\hat{y} = \texttt{session.run}(\tanh(\mathbf{x}^\top \mathbf{w} + b))$$

Computational Graphs

1. Operations
2. Tensors
3. Variables
4. Sessions

Execution Model

Execution Model

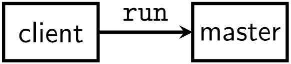
Actors

Execution Model

Actors

1. Client

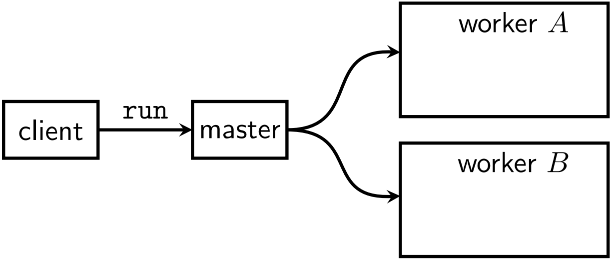
Execution Model



Actors

1. Client
2. Master

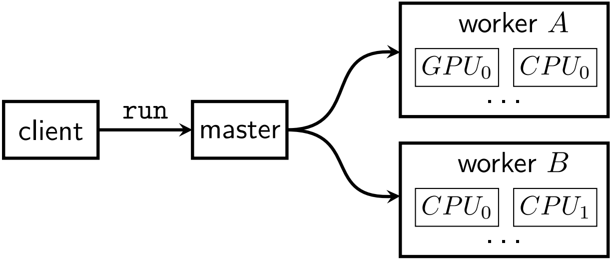
Execution Model



Actors

1. Client
2. Master
3. Workers

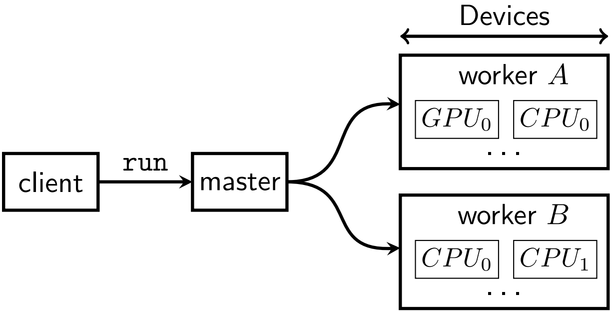
Execution Model



Actors

1. Client
2. Master
3. Workers
4. Devices

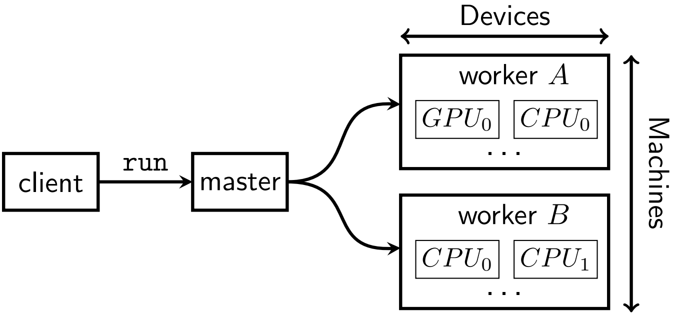
Execution Model



Actors

1. Client
2. Master
3. Workers
4. Devices

Execution Model



Actors

1. Client
2. Master
3. Workers
4. Devices

Visualization Tools

Visualization Tools

- Deep Neural Networks have the tendency of being . . . deep

Visualization Tools

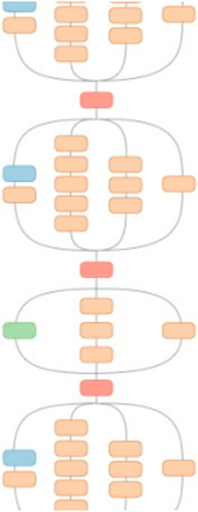
- Deep Neural Networks have the tendency of being ... deep
- Easy to drown in the complexity of an architecture

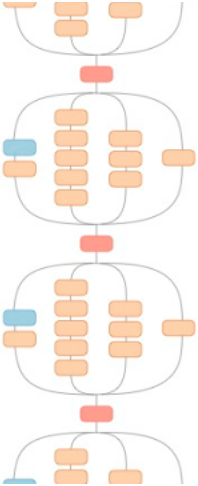
Visualization Tools

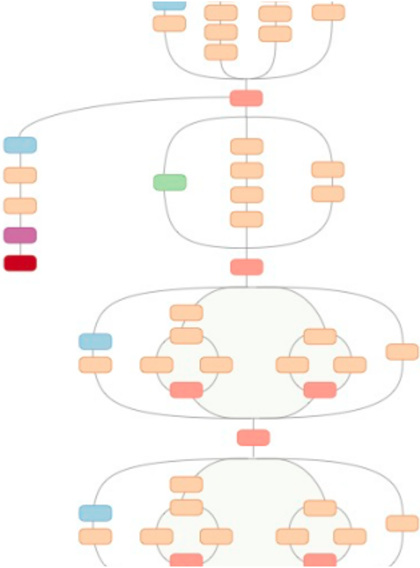
- Deep Neural Networks have the tendency of being . . . deep
- Easy to drown in the complexity of an architecture
- $>$ 36,000 nodes for Google's *Inception* model

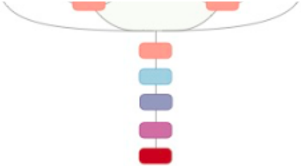
Visualization Tools

- Deep Neural Networks have the tendency of being . . . deep
- Easy to drown in the complexity of an architecture
- $>$ 36,000 nodes for Google's *Inception* model

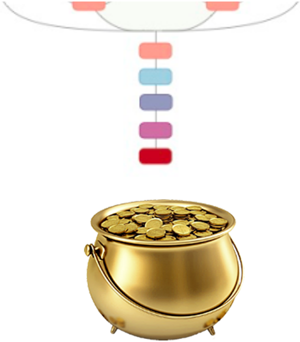








Source: http://googleresearch.blogspot.de/2016/03/train-your-own-image-classifier-with.html



Source: http://googleresearch.blogspot.de/2016/03/train-your-own-image-classifier-with.html

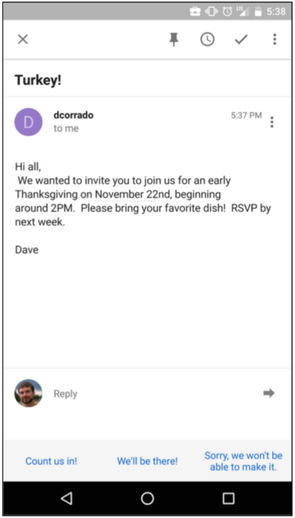
TensorBoard to the Rescue

Use Cases

Source: http://googleresearch.blogspot.de/2015/11/computer-respond-to-this-email.html

Use Cases

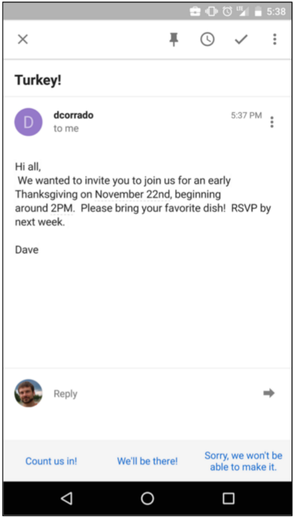
- Smart email replies in Google *Inbox*



Source: http://googleresearch.blogspot.de/2015/11/computer-respond-to-this-email.html

Use Cases

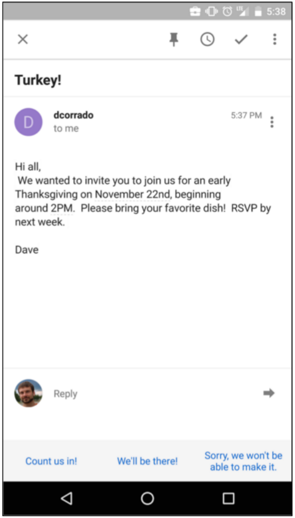
- Smart email replies in Google *Inbox*
- Emails mapped to "thought vectors"



Source: http://googleresearch.blogspot.de/2015/11/computer-respond-to-this-email.html

Use Cases

- Smart email replies in Google *Inbox*
- Emails mapped to "thought vectors"
- LSTMs synthesize valid replies



Source: http://googleresearch.blogspot.de/2015/11/computer-respond-to-this-email.html

Use Cases

Use Cases

- Google DeepMind now using TensorFlow

Use Cases

- Google DeepMind now using TensorFlow
- Already for *AlphaGo*

Source: https://deepmind.com/css/images/opengraph/alphago-logo.png

Use Cases

- Google DeepMind now using TensorFlow
- Already for *AlphaGo*
- According to a DeepMind SWE reasons are:

Source: https://deepmind.com/css/images/opengraph/alphago-logo.png

Use Cases

- Google DeepMind now using TensorFlow
- Already for *AlphaGo*
- According to a DeepMind SWE reasons are:
  - Python,

Source: https://deepmind.com/css/images/opengraph/alphago-logo.png

Use Cases

- Google DeepMind now using TensorFlow
- Already for *AlphaGo*
- According to a DeepMind SWE reasons are:
  - Python,
  - Integration with Google Cloud Platform,

Source: https://deepmind.com/css/images/opengraph/alphago-logo.png

Use Cases

- Google DeepMind now using TensorFlow
- Already for *AlphaGo*
- According to a DeepMind SWE reasons are:
  - Python,
  - Integration with Google Cloud Platform,
  - Support for TPUs,

Source: https://deepmind.com/css/images/opengraph/alphago-logo.png

Use Cases

- Google DeepMind now using TensorFlow
- Already for *AlphaGo*
- According to a DeepMind SWE reasons are:
  - Python,
  - Integration with Google Cloud Platform,
  - Support for TPUs,
  - Ability to run on many GPUs.

Source: https://deepmind.com/css/images/opengraph/alphago-logo.png

Walkthrough

How do I continue?

Resources

Resources

- MOOCs

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io
  - http://cs231n.github.io

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io
  - http://cs231n.github.io
  - http://karpathy.github.io

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io
  - http://cs231n.github.io
  - http://karpathy.github.io
  - http://www.deeplearningbook.org

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io
  - http://cs231n.github.io
  - http://karpathy.github.io
  - http://www.deeplearningbook.org
  - http://neuralnetworksanddeeplearning.com

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io
  - http://cs231n.github.io
  - http://karpathy.github.io
  - http://www.deeplearningbook.org
  - http://neuralnetworksanddeeplearning.com
  - https://www.kaggle.com

Resources

- MOOCs
  - Machine Learning by Andrew Ng @ Coursera
  - Deep Learning by Google @ Udacity
  - Machine Learning Nanodegree @ Udacity
- Websites
  - http://colah.github.io
  - http://cs231n.github.io
  - http://karpathy.github.io
  - http://www.deeplearningbook.org
  - http://neuralnetworksanddeeplearning.com
  - https://www.kaggle.com
  - https://www.tensorflow.org

Meetups

Deep Learning in Action #4

ACM Munich Student Chapter

Deep Learning in Action #4

ACM Munich Student Chapter

- Monday, August 1, 2016

Meetups

Deep Learning in Action #4

ACM Munich Student Chapter

- Monday, August 1, 2016
- Geometric Deep Learning by Prof. Dr. Michael M. Bronstein

Meetups

Deep Learning in Action #4

ACM Munich Student Chapter

- Monday, August 1, 2016
- Geometric Deep Learning by Prof. Dr. Michael M. Bronstein
- Introduction to TensorFlow

Deep Learning in Action #4

ACM Munich Student Chapter

- Monday, August 1, 2016
- Geometric Deep Learning by Prof. Dr. Michael M. Bronstein
- Introduction to TensorFlow
- More to come

Stay in Touch

Stay in Touch

- `peter@goldsborough.me`

Stay in Touch

- `peter@goldsborough.me`
- `linkedin.com/in/petergoldsborough`

Stay in Touch

- peter@goldsborough.me
- linkedin.com/in/petergoldsborough
- github.com/goldsborough

Stay in Touch

- peter@goldsborough.me
- linkedin.com/in/petergoldsborough
- github.com/goldsborough
- @peterawks

Q & A

References

Pedro Domingos, *A few useful things to know about machine learning*, Commun. ACM **55** (2012), no. 10, 78–87.

Andrej Karpathy, *The unreasonable effectiveness of recurrent neural networks*, May 21 2015 (accessed Jul 10, 2016), http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

———, *Neural networks part 1: Setting up the architecture*, CS231n: Convolutional Neural Networks for Visual Recognition, 2016 (accessed July 9, 2016), http://cs231n.github.io/neural-networks-1/.

Thomas M. Mitchell, *Machine learning*, 1 ed., McGraw-Hill, Inc., New York, NY, USA, 1997.

New navy device learns by doing, July 08 1958 (accessed Jul 9, 2016), http://query.nytimes.com/gst/abstract.html?res=9D01E4D8173DE53BBC4