

Introduction to Machine Learning feat. TensorFlow

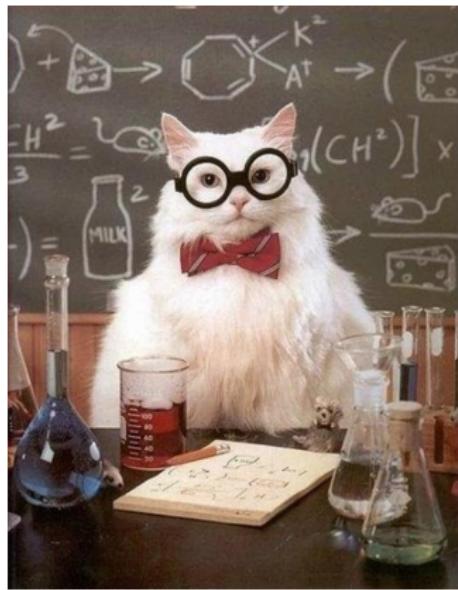


Peter Goldsborough

July 11, 2016

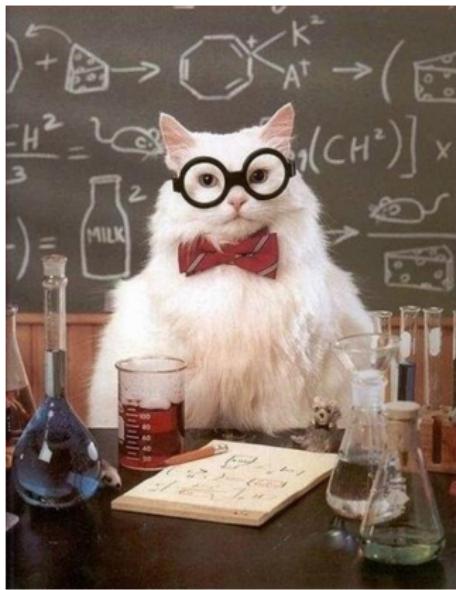
Table of Catents

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.

[Dom12]

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 .

[Mit97]

Definition IV

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

Definition IV

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

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

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
 $x \in \mathbb{R}^n$
- ▶ x contains features, such as (lunch, dinner)

The Task, T

- ▶ Discriminate by the way an algorithm processes an example
 $x \in \mathbb{R}^n$
- ▶ x contains features, such as (lunch, dinner)
- ▶ The output y can take on various forms

The Task, T

Classification

$$f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

The Task, T

Classification

$$f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

- ▶ Categorize an input \mathbf{x} into one of k classes

The Task, T

Classification

$$f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

- ▶ Categorize an input \mathbf{x} into one of k classes
- ▶ Image classification

The Task, T

Classification

$$f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

- ▶ Categorize an input \mathbf{x} into one of k classes
- ▶ Image classification
- ▶ Recognizing handwritten digits

The Task, T

Classification

$$f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

- ▶ Categorize an input \mathbf{x} into one of k classes
- ▶ Image classification
- ▶ Recognizing handwritten digits
- ▶ Predictive Policing

The Task, T

Regression

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

The Task, T

Regression

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

- ▶ Predict a scalar value given input features

The Task, T

Regression

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

- ▶ Predict a scalar value given input features
- ▶ Algorithmic Trading

The Task, T

Regression

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

- ▶ Predict a scalar value given input features
- ▶ Algorithmic Trading
- ▶ Predicting the market price of a house

The Task, T

Regression

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

- ▶ Predict a scalar value given input features
- ▶ Algorithmic Trading
- ▶ Predicting the market price of a house
- ▶ Predicting the amount of rain in a season

The Task, T

Transcription

$$f : \text{Language} \rightarrow \text{Text}$$

The Task, T

Transcription

$$f : \text{Language} \rightarrow \text{Text}$$

- ▶ Transform language from an unstructured representation into text

The Task, T

Transcription

$$f : \text{Language} \rightarrow \text{Text}$$

- ▶ Transform language from an unstructured representation into text
- ▶ Recognizing street addresses in Google Street View

The Task, T

Transcription

$$f : \text{Language} \rightarrow \text{Text}$$

- ▶ Transform language from an unstructured representation into text
- ▶ Recognizing street addresses in Google Street View
- ▶ Transcribe speech into text

The Task, T

Transcription

$$f : \text{Language} \rightarrow \text{Text}$$

- ▶ Transform language from an unstructured representation into text
- ▶ Recognizing street addresses in Google Street View
- ▶ Transcribe speech into text
- ▶ Similar to *Translation*

The Task, T

Synthesis

$f : \text{Stuff} \rightarrow \text{More Stuff}$

The Task, T

Synthesis

$f : \text{Stuff} \rightarrow \text{More Stuff}$

- Generate new data similar to the input

The Task, T

Synthesis

$f : \text{Stuff} \rightarrow \text{More Stuff}$

- ▶ Generate new data similar to the input
- ▶ Reproduce patterns of the sky for video games

The Task, T

Synthesis

$f : \text{Stuff} \rightarrow \text{More Stuff}$

- ▶ Generate new data similar to the input
- ▶ Reproduce patterns of the sky for video games
- ▶ Synthesize speech

The Task, T

Synthesis

$f : \text{Stuff} \rightarrow \text{More Stuff}$

- ▶ Generate new data similar to the input
- ▶ Reproduce patterns of the sky for video games
- ▶ Synthesize speech
- ▶ Generate more data to train our algorithm to generate more data ...

Experience E

How do we make our algorithm learn?

Experience E

How do we make our algorithm learn?

Unsupervised Learning

Experience E

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

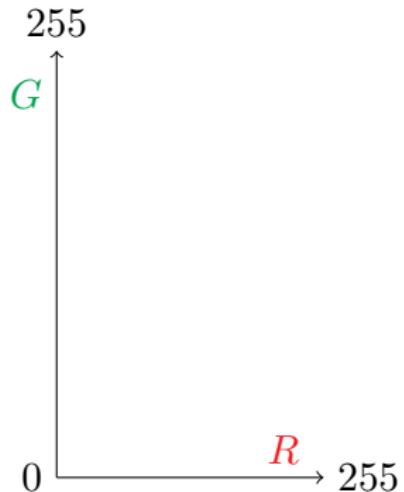
Methods: Features

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

$$0 \xrightarrow{R} 255$$

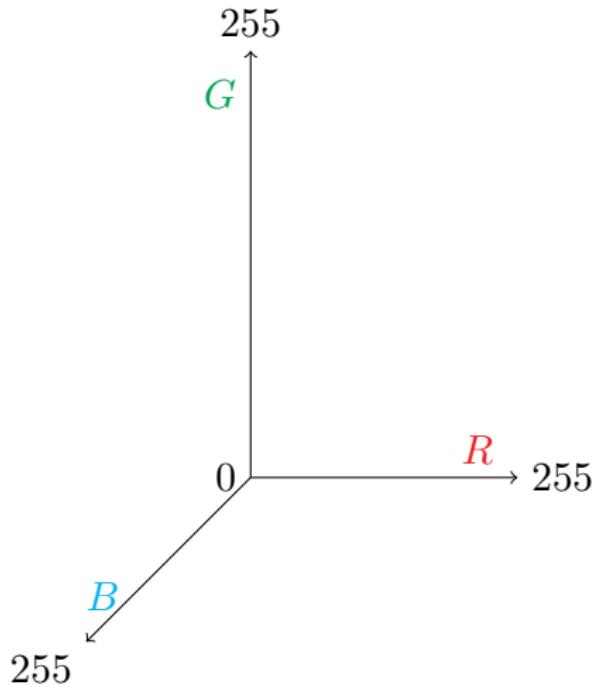
Methods: Features

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



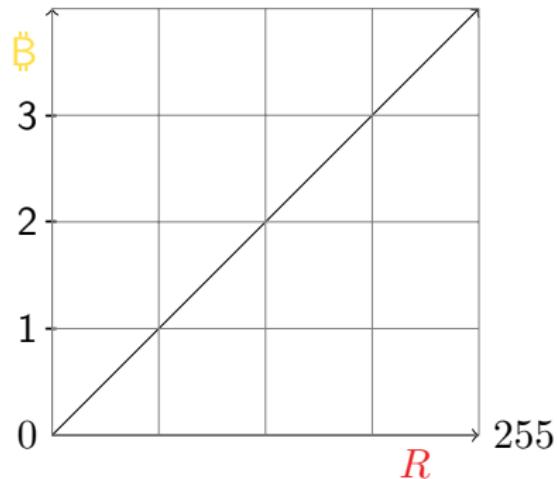
Methods: Features

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



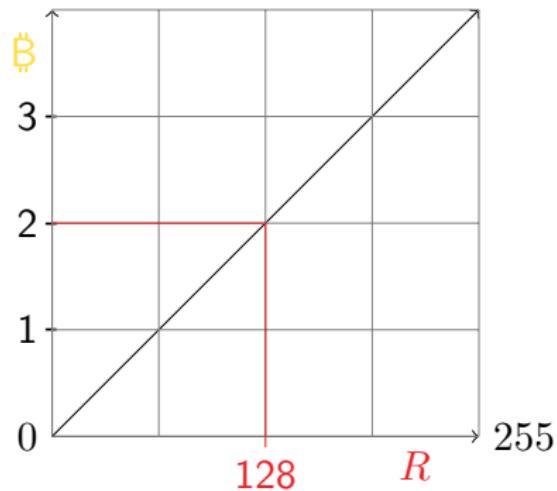
Methods: Features

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



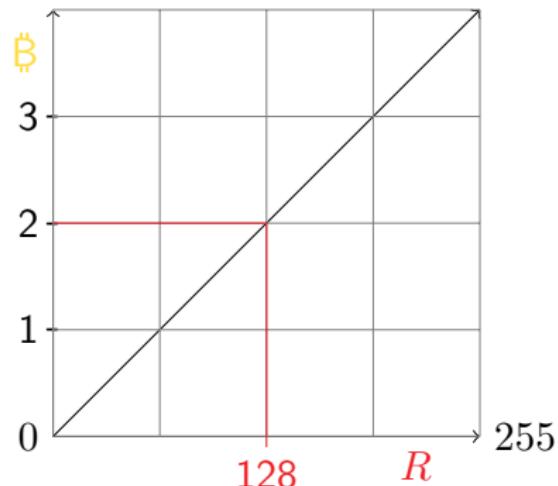
Methods: Features

- ▶ 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 (market price)



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
- ▶ Positive values of w_i encourage positive values of 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
- ▶ Positive values of w_i encourage positive values of b_i
- ▶ Negative values of w_i inhibit 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
- ▶ Positive values of w_i encourage positive values of b_i
- ▶ Negative values of w_i inhibit 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
- ▶ Positive values of w_i encourage positive values of b_i
- ▶ Negative values of w_i inhibit 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
- ▶ In practice, we organize our data into a matrix $\mathbf{D} \in \mathbb{R}^{n \times k}$

Methods: Data

- ▶ To make our algorithm learn, we need to feed it (lots of) data
- ▶ In practice, we organize our data into a matrix $\mathbf{D} \in \mathbb{R}^{n \times k}$

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

Design Matrix

Methods: Data

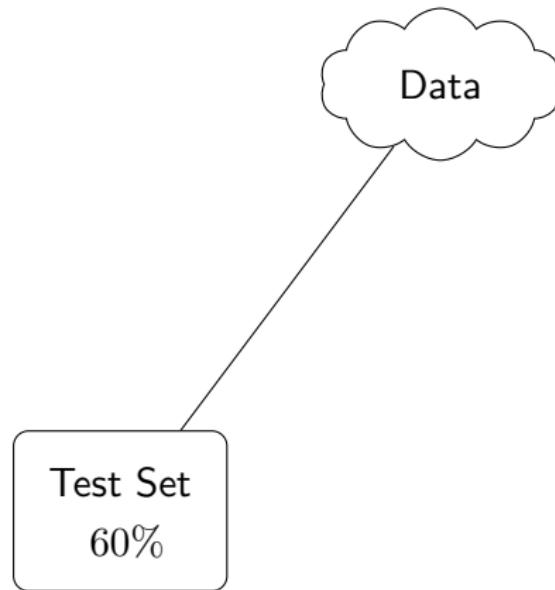
- ▶ To make our algorithm learn, we need to feed it (lots of) data
- ▶ In practice, we organize our data into a matrix $\mathbf{D} \in \mathbb{R}^{n \times k}$

```
D = np.array([[34, 147, 37], [247, 69, 13], [66, 66, 66]])
w = np.random.rand(3, 1)
b = np.random.randn()
y = D.dot(w) + b
```

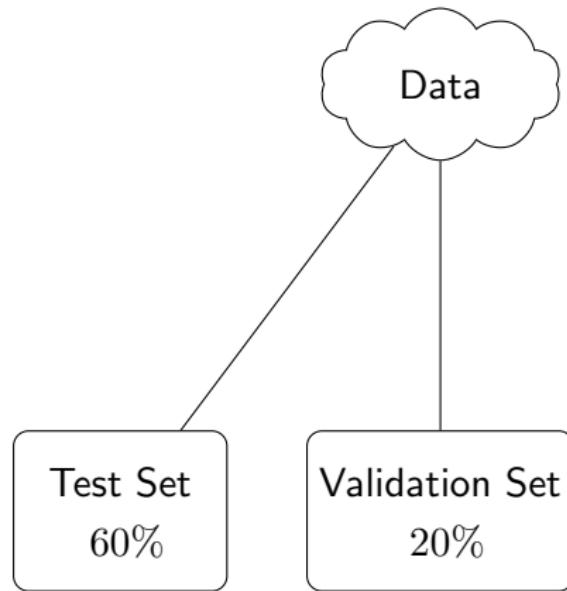
Methods: Data



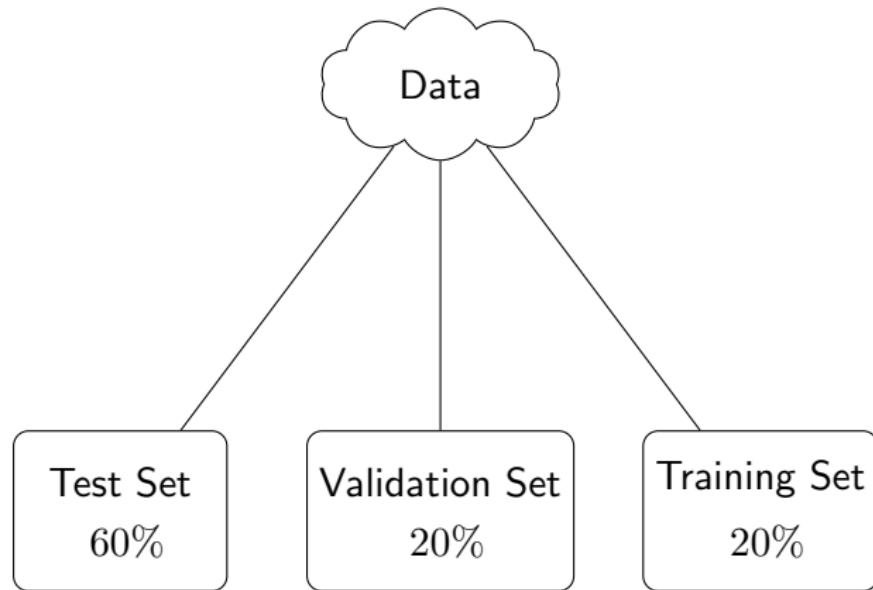
Methods: Data



Methods: Data



Methods: Data



Methods: Measuring Performance

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
- ▶ The loss function we use depends on the learning task

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
- ▶ The loss function we use depends on the learning task

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

Methods: Measuring Performance

Regression

Methods: Measuring Performance

Regression

$$\begin{bmatrix} 34 & 147 & 73 \\ 247 & 69 & 13 \\ 66 & 66 & 66 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + c = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \xrightarrow{\text{Target}} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}$$

D **w** **y** **\hat{y}**

Methods: Measuring Performance

Regression

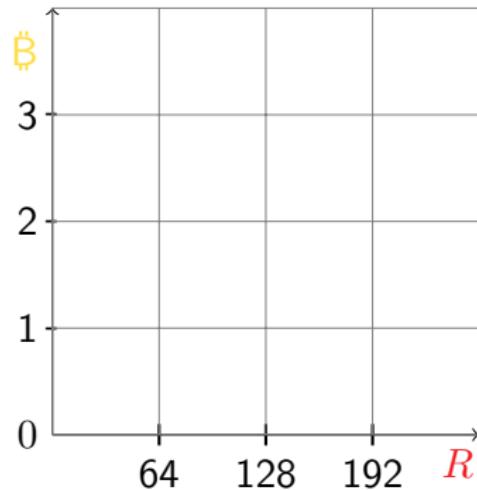
$$\begin{bmatrix} 34 & 147 & 73 \\ 247 & 69 & 13 \\ 66 & 66 & 66 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + c = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \xrightarrow{\text{Target}} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}$$

D **w** **y** **\hat{y}**

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

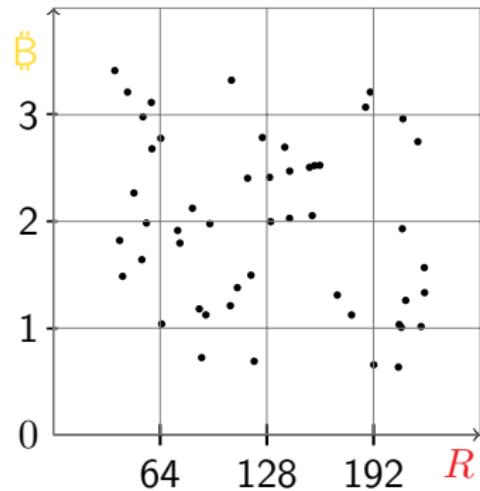
Methods: Measuring Performance

Regression



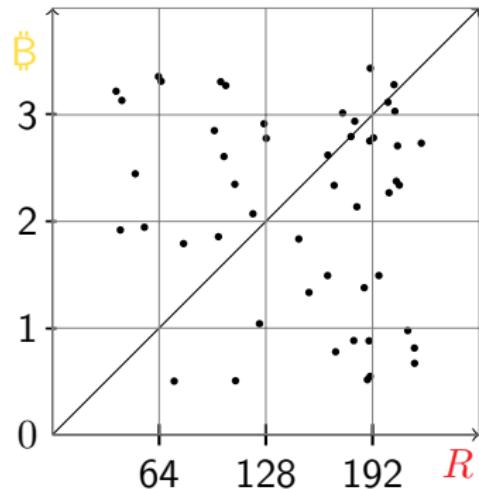
Methods: Measuring Performance

Regression



Methods: Measuring Performance

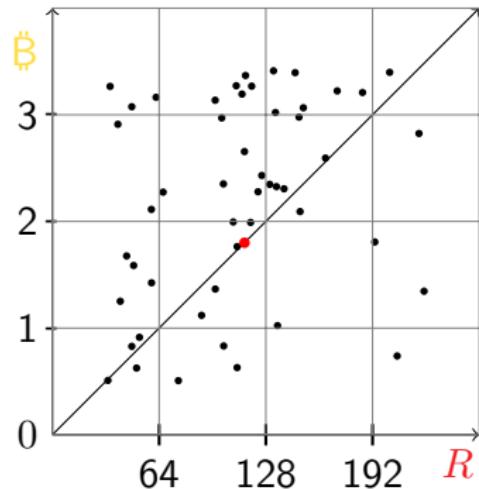
Regression



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

Methods: Measuring Performance

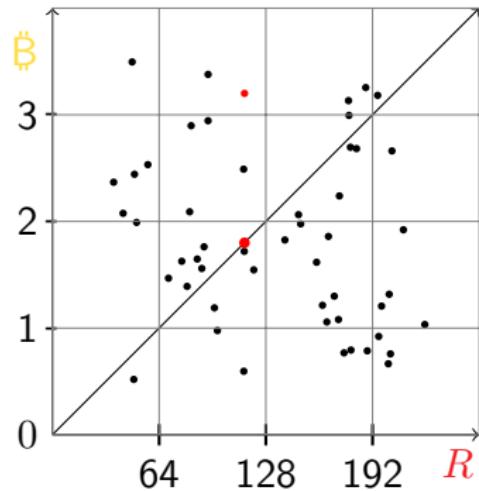
Regression



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

Methods: Measuring Performance

Regression



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

Methods: Weight Update

Methods: Weight Update

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

Methods: Weight Update

- ▶ Through $L(y, \hat{y})$ 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
- ▶ Next time, the loss will be smaller

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
- ▶ Next time, the loss will be smaller
- ▶ We repeat this, until we're happy

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
- ▶ Next time, the loss will be smaller
- ▶ We repeat this, until we're happy

This is the core idea behind Machine Learning

Methods: Weight Update

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 are interested in the way the loss changes w.r.t to the *weights* \mathbf{w} (and bias b)

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 are interested in the way the loss changes w.r.t to the *weights* \mathbf{w} (and bias b)
- ▶ Moreover, we think in terms of entire batches, not single examples

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 are interested in the way the loss changes w.r.t to the *weights* \mathbf{w} (and bias b)
- ▶ Moreover, we think in terms of entire batches, not single examples
- ▶ We need new notation:

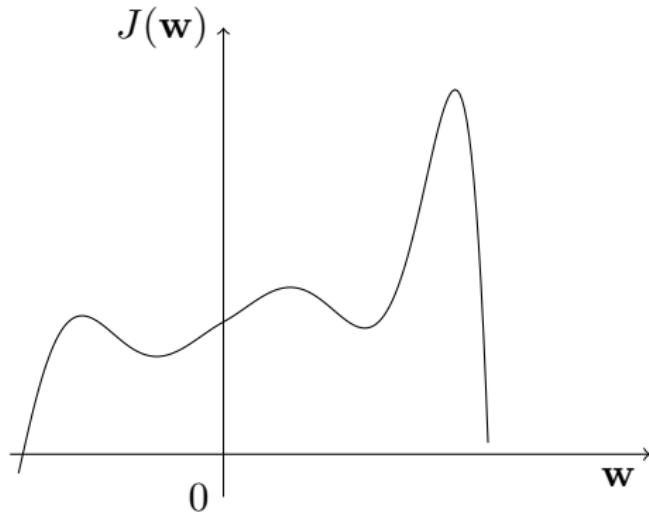
Methods: Weight Update

- ▶ $L(\mathbf{y}, \hat{\mathbf{y}})$ depends on \mathbf{y} and $\hat{\mathbf{y}}$, but *also* \mathbf{w} : $L(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w})$
- ▶ We are interested in the way the loss changes w.r.t to the *weights* \mathbf{w} (and bias b)
- ▶ Moreover, we think in terms of entire batches, not single examples
- ▶ We need new notation:

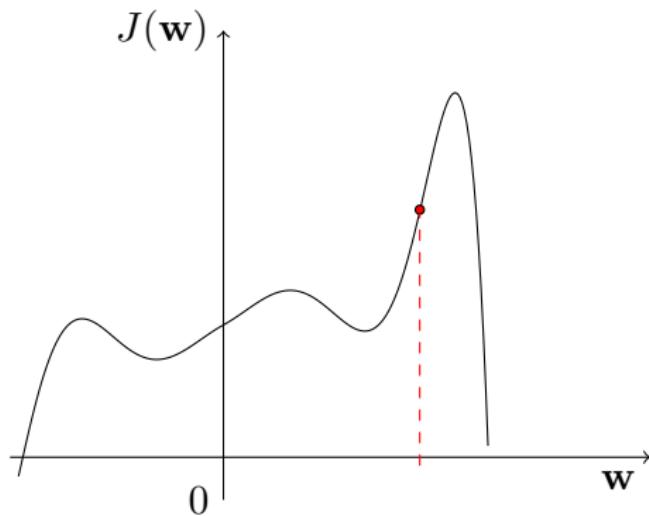
We can think of $\mathbf{Y}, \hat{\mathbf{Y}}$ as being *constant* and write:

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

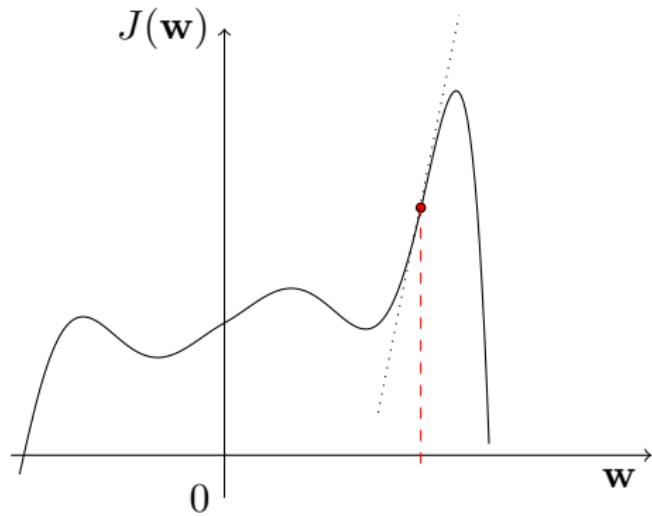
Methods: Weight Update



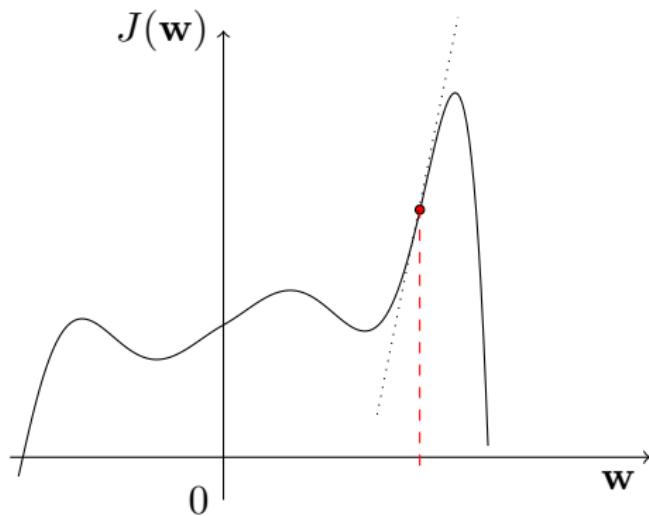
Methods: Weight Update



Methods: Weight Update



Methods: Weight Update


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

Digression: Calculus Recap

Digression: Calculus Recap

- ▶ Let's talk about f

Digression: Calculus Recap

- ▶ Let's talk about f
- ▶ When f depends on one variable x ,

$$f'(x) = \frac{df}{dx}$$

is the *derivative* of f w.r.t. x

Digression: Calculus Recap

- ▶ Let's talk about f
- ▶ When f depends on one variable x ,

$$f'(x) = \frac{df}{dx}$$

is the *derivative* of f w.r.t. x

- ▶ It tells us how f is changing in dependence of x

Digression: Calculus Recap

Digression: Calculus Recap

- In $f(x, y) = x + y$ both x and y are influencing f 's output

Digression: Calculus Recap

- ▶ In $f(x, y) = x + y$ both x and y are influencing f 's output
- ▶ We want to know *how* they are influencing f independently

Digression: Calculus Recap

- ▶ In $f(x, y) = x + y$ both x and y are influencing f 's output
- ▶ We want to know *how* they are influencing f independently
- ▶ The partial derivative $\frac{\partial f}{\partial x}$ gives us a *view* of f

Digression: Calculus Recap

- ▶ In $f(x, y) = x + y$ both x and y are influencing f 's output
- ▶ We want to know *how* they are influencing f independently
- ▶ The partial derivative $\frac{\partial f}{\partial x}$ gives us a *view* of f
- ▶ In this view, all variables but x are constant

Digression: Calculus Recap

- ▶ In $f(x, y) = x + y$ both x and y are influencing f 's output
- ▶ We want to know *how* they are influencing f independently
- ▶ The partial derivative $\frac{\partial f}{\partial x}$ gives us a *view* of f
- ▶ In this view, all variables but x are constant
- ▶ Thus, we can extract the change of f w.r.t. to f

Digression: Calculus Recap

Digression: Calculus Recap

- ▶ Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar-valued function *taking* a vector \mathbf{x} as input.

Digression: Calculus Recap

- ▶ Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar-valued function *taking* a vector \mathbf{x} as input.
- ▶ Then

$$\frac{\partial f}{\partial v_i}$$

is a partial derivative, and

Digression: Calculus Recap

- ▶ Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar-valued function *taking* a vector \mathbf{x} as input.
- ▶ Then

$$\frac{\partial f}{\partial v_i}$$

is a partial derivative, and

- ▶ ∇f the vector containing the partial derivatives of w :

$$\nabla f = \left(\frac{\partial f}{\partial v_1}, \dots, \frac{\partial f}{\partial v_n} \right)^\top$$

Digression: Calculus Recap

- ▶ Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar-valued function *taking* a vector \mathbf{x} as input.
- ▶ Then

$$\frac{\partial f}{\partial v_i}$$

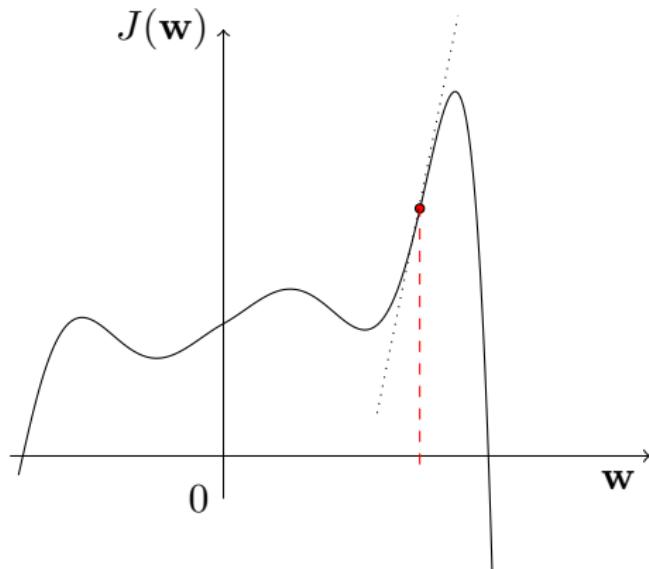
is a partial derivative, and

- ▶ ∇f the vector containing the partial derivatives of w :

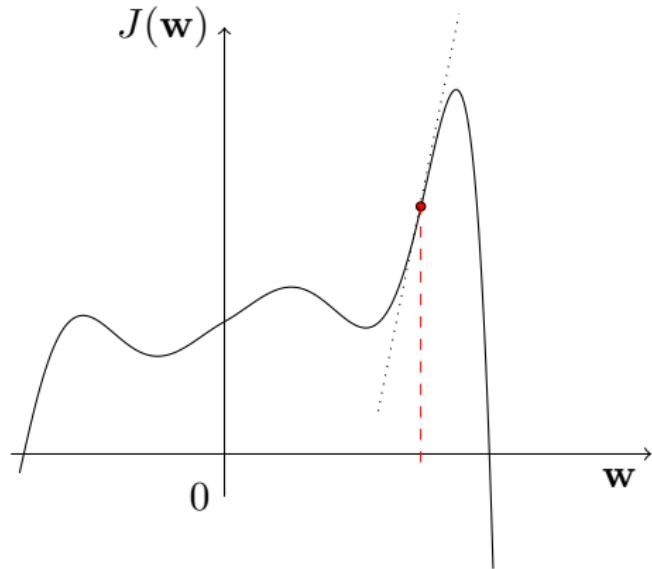
$$\nabla f = \left(\frac{\partial f}{\partial v_1}, \dots, \frac{\partial f}{\partial v_n} \right)^\top$$

- ▶ ∇f is called the *gradient* of f

Methods: Weight Update



Methods: Weight Update



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

Methods: Weight Update

- ▶ This method of updating weights is called *Gradient Descent* (GD)

Methods: Weight Update

- ▶ This method of updating weights is called *Gradient Descent* (GD)
- ▶ It adds a *hyperparameter* α as a knob to tweak:

$$w \leftarrow w - \alpha \nabla J(w)$$

Methods: Weight Update

- ▶ This method of updating weights is called *Gradient Descent* (GD)
- ▶ It adds a *hyperparameter* α as a knob to tweak:

$$w \leftarrow w - \alpha \nabla J(w)$$

- ▶ α is called the *learning rate*

Methods: Weight Update

- ▶ This method of updating weights is called *Gradient Descent* (GD)
- ▶ It adds a *hyperparameter* α as a knob to tweak:

$$w \leftarrow w - \alpha \nabla J(w)$$

- ▶ α is called the *learning rate*
- ▶ Often, we will apply a decay to it

Methods: Weight Update

- ▶ The goal of GD is to find the weights that minimize the loss for our training data

Methods: Weight Update

- ▶ The goal of GD is to find the weights that minimize the loss for our training data
- ▶ The theoretical goal is to find the global minimum of $J(w)$

Methods: Weight Update

- ▶ The goal of GD is to find the weights that minimize the loss for our training data
- ▶ The theoretical goal is to find the global minimum of $J(w)$
- ▶ In practice, we most often use *Stochastic Gradient Descent*

Methods: Weight Update

- ▶ The goal of GD is to find the weights that minimize the loss for our training data
- ▶ The theoretical goal is to find the global minimum of $J(w)$
- ▶ In practice, we most often use *Stochastic Gradient Descent*
- ▶ Many variations and alternatives exist

Methods: Weight Update

- ▶ The goal of GD is to find the weights that minimize the loss for our training data
- ▶ The theoretical goal is to find the global minimum of $J(w)$
- ▶ In practice, we most often use *Stochastic Gradient Descent*
- ▶ Many variations and alternatives exist

$$\nu = \gamma\nu + \alpha\nabla J(w)$$

$$w \leftarrow w - \nu$$

Momentum Optimizer

Methods: Regularization

Methods: Regularization

- We want our models to *generalize*

Methods: Regularization

- ▶ We want our models to *generalize*
- ▶ Parameters should not specialize too much to training data

Methods: Regularization

- ▶ We want our models to *generalize*
- ▶ Parameters should not specialize too much to training data
- ▶ *Regularization* is the art of finding a tradeoff between

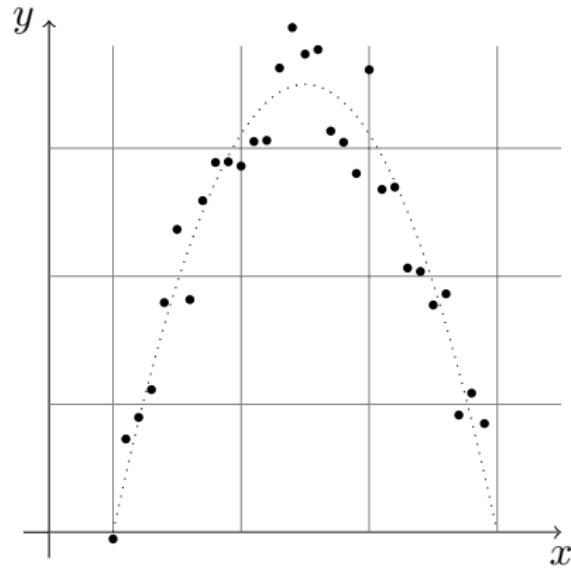
Methods: Regularization

- ▶ We want our models to *generalize*
- ▶ Parameters should not specialize too much to training data
- ▶ *Regularization* is the art of finding a tradeoff between
 - ▶ Training error
 - ▶ Generalization error

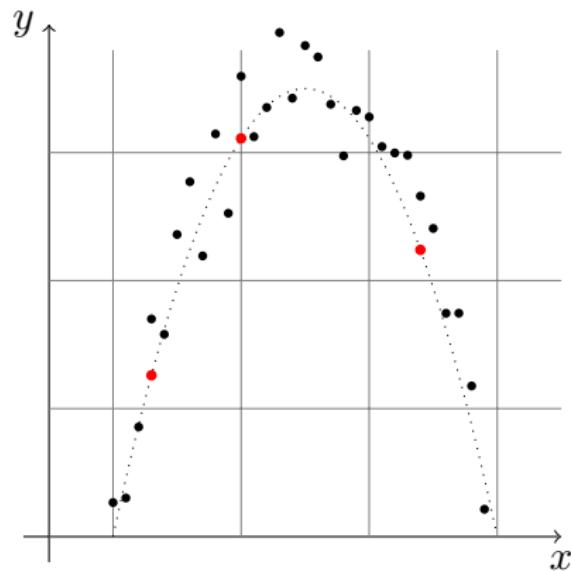
Methods: Regularization

- ▶ We want our models to *generalize*
- ▶ Parameters should not specialize too much to training data
- ▶ *Regularization* is the art of finding a tradeoff between
 - ▶ Training error
 - ▶ Generalization error
- ▶ We try to influence a model's *capacity*

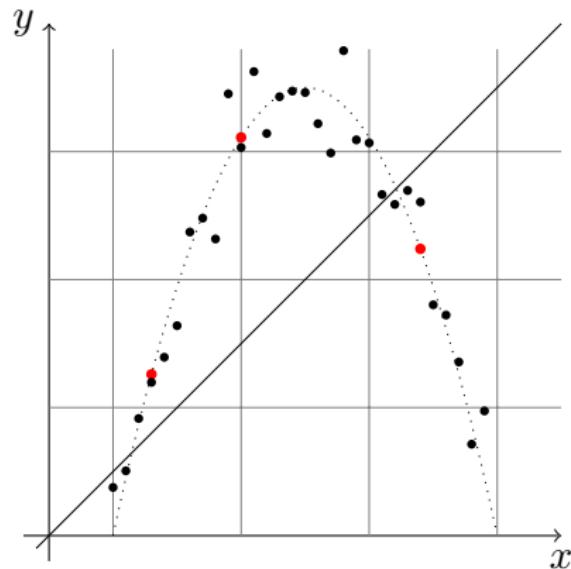
Concepts: Model Capacity



Concepts: Model Capacity

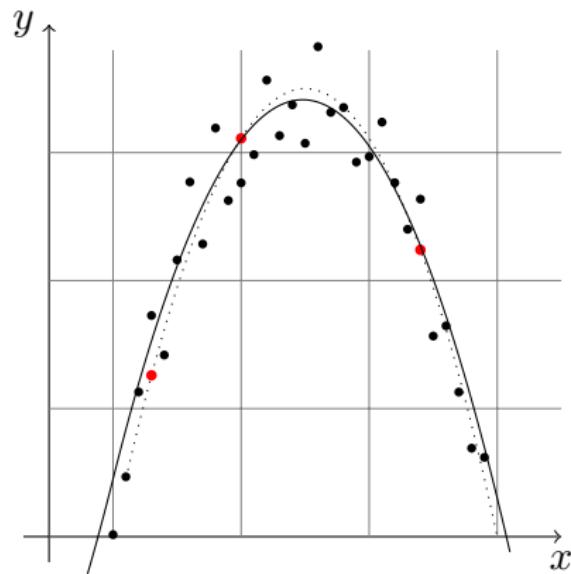


Concepts: Model Capacity



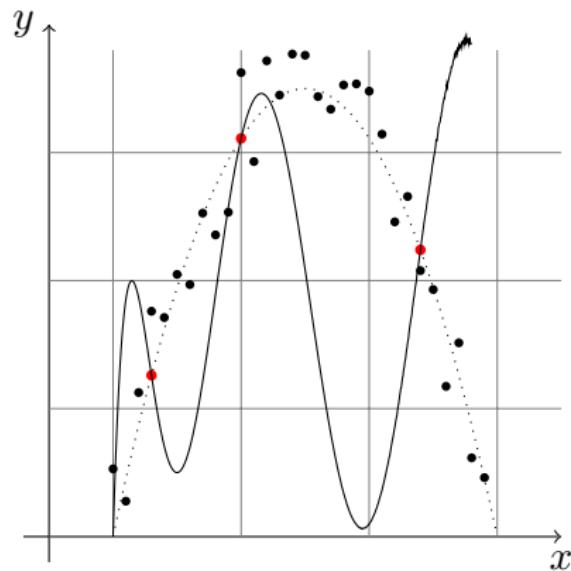
Underfitting (low capacity)

Concepts: Model Capacity



Just Right (optimal capacity)

Concepts: Model Capacity



Overfitting (too high capacity)

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

- We can keep our polynomial from getting too funky by keeping weights small

Methods: Regularization

- ▶ We can keep our polynomial from getting too funky by keeping weights small
- ▶ We do so by *adding the weights to the cost*

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

Methods: Regularization

- ▶ We can keep our polynomial from getting too funky by keeping weights small
- ▶ We do so by *adding the weights to the cost*

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

- ▶ Our algorithm must make a tradeoff between

Methods: Regularization

- ▶ We can keep our polynomial from getting too funky by keeping weights small
- ▶ We do so by *adding the weights to the cost*

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

- ▶ Our algorithm must make a tradeoff between
 - ▶ Minimizing the training error (make weights large)
 - ▶ Keeping the cost small (make weights small)

Methods: Regularization

- ▶ We can keep our polynomial from getting too funky by keeping weights small
- ▶ We do so by *adding the weights to the cost*

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

- ▶ Our algorithm must make a tradeoff between
 - ▶ Minimizing the training error (make weights large)
 - ▶ Keeping the cost small (make weights small)
- ▶ We thus reduce the capacity by *favoring* certain functions over others

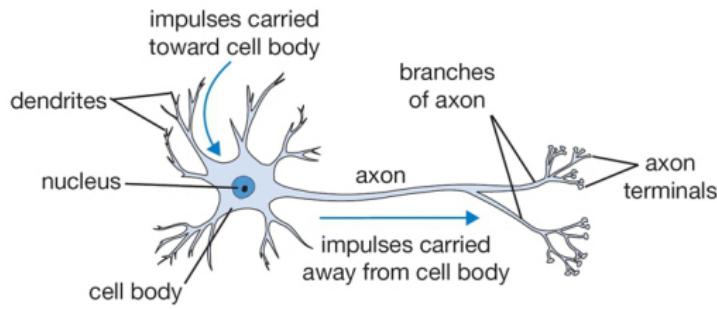
Neural Networks

Neural Networks



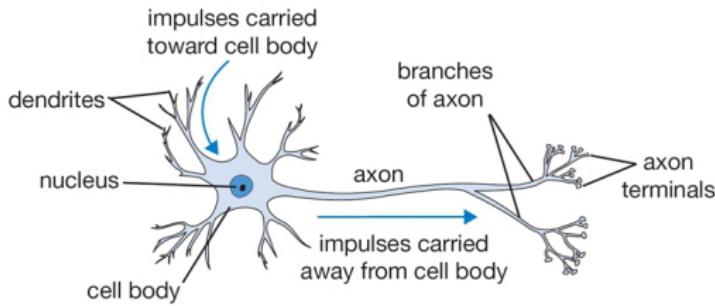
Neural Networks: Biological Motivation

Neural Networks: Biological Motivation



[Kar16b]

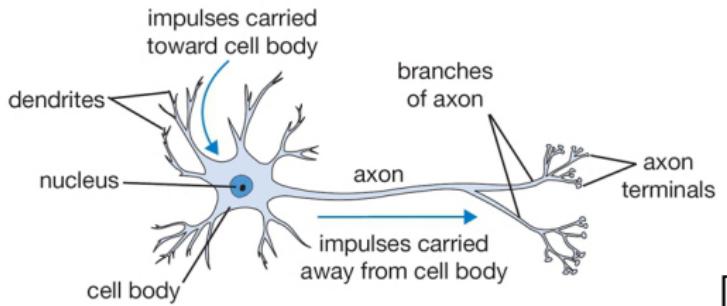
Neural Networks: Biological Motivation



[Kar16b]

- ▶ Receive electrochemical signals through *dendrites*

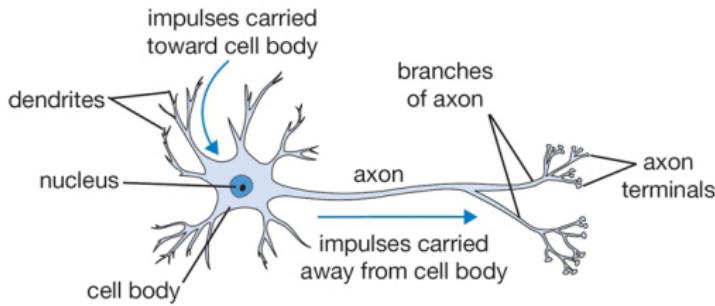
Neural Networks: Biological Motivation



[Kar16b]

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

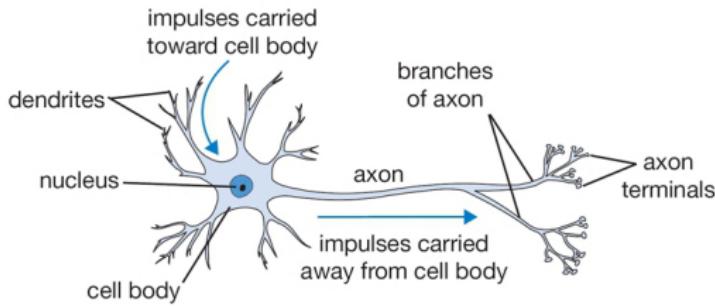
Neural Networks: Biological Motivation



[Kar16b]

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

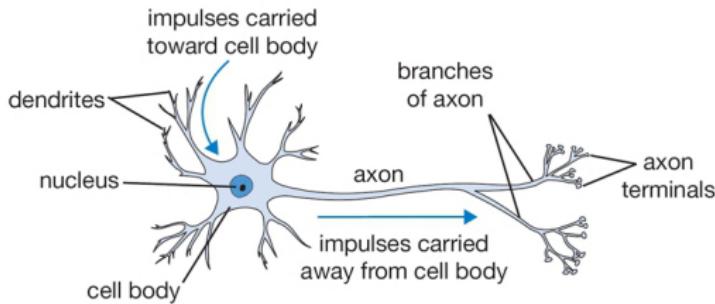
Neural Networks: Biological Motivation



[Kar16b]

- ▶ 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

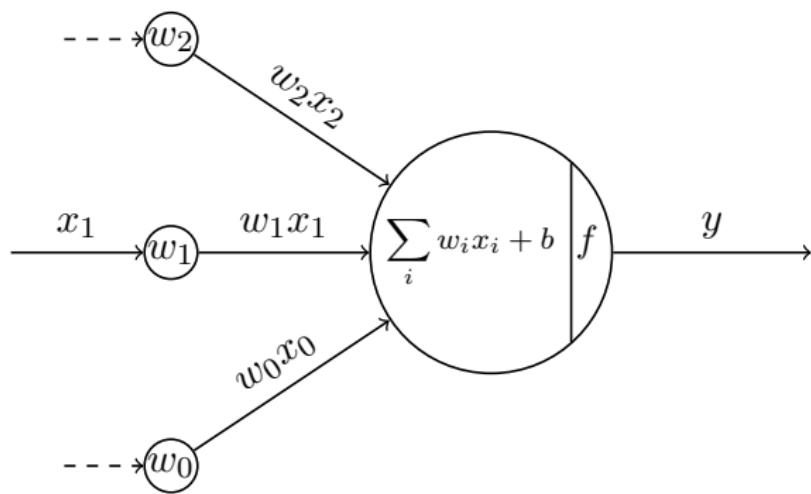
Neural Networks: Biological Motivation



[Kar16b]

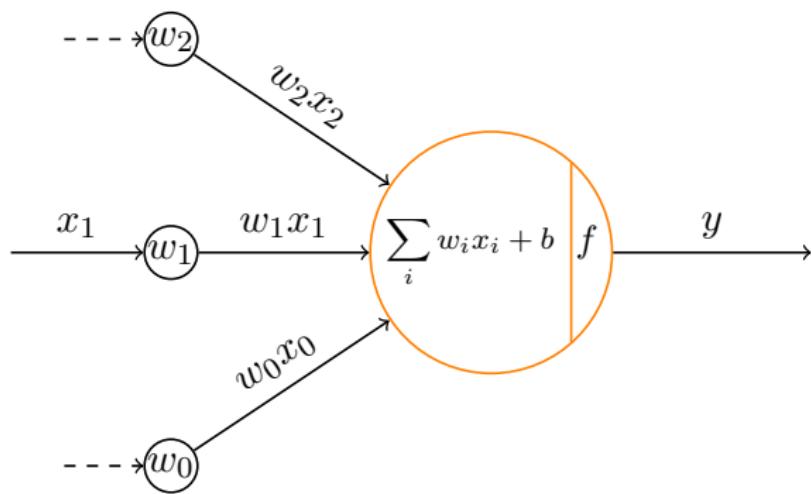
- ▶ Receive electrochemical signals through *dendrites*
- ▶ Fire their own signal if the input exceeds some threshold
- ▶ Forward their signals via *axons*
- ▶ Connected via *synapses*, which control the strength of interaction
- ▶ The dynamic alteration of synaptic strengths are the primary source of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

Neural Networks: Mathematical Model



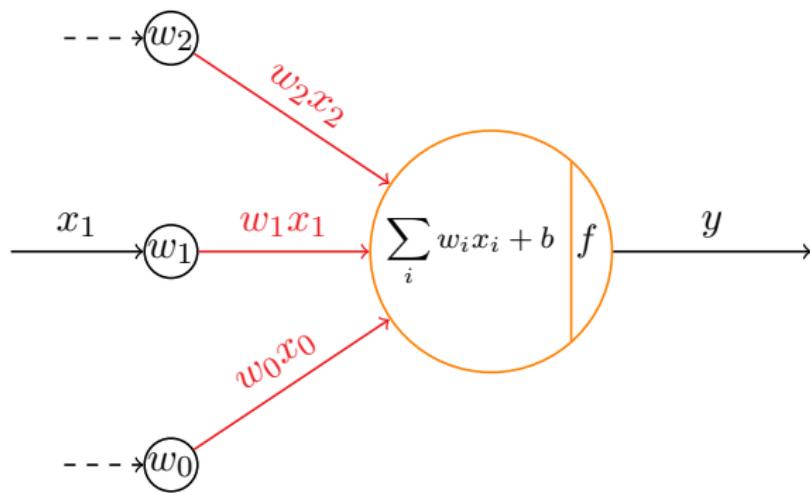
Neural Networks: Mathematical Model

■ Cell Body

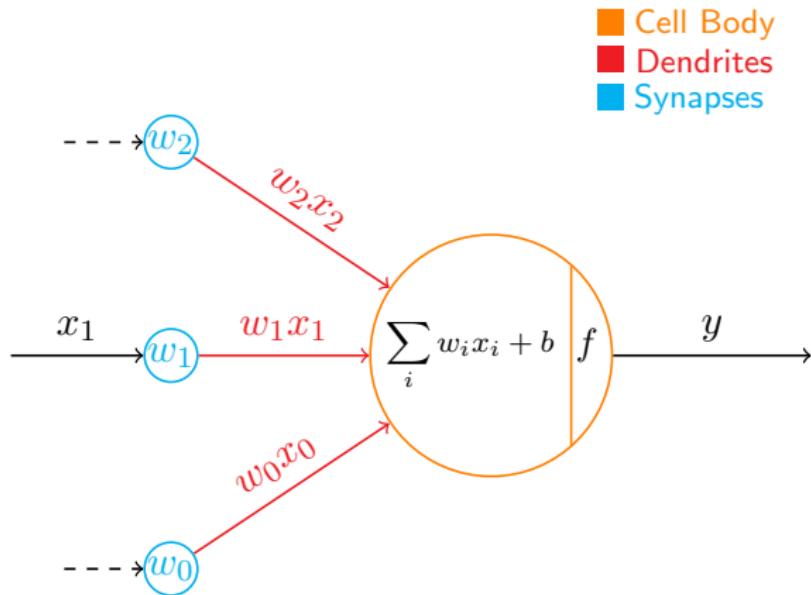


Neural Networks: Mathematical Model

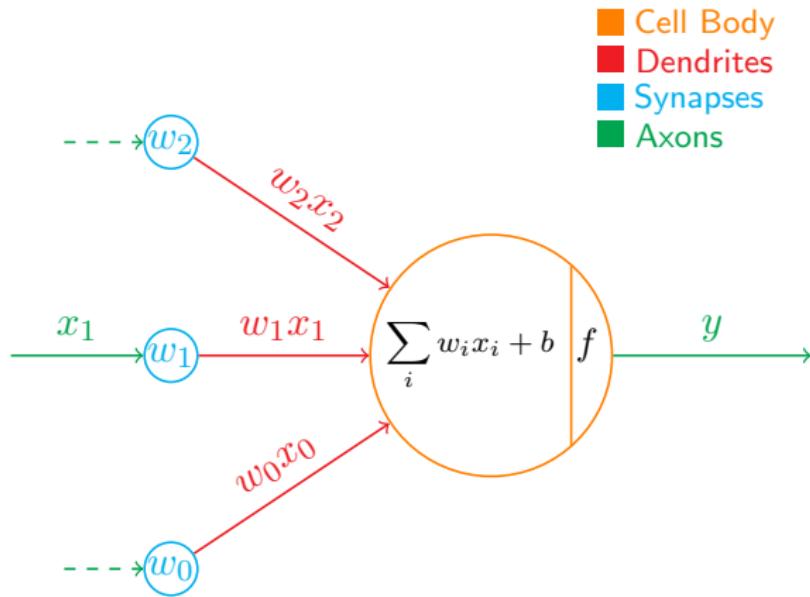
Cell Body
Dendrites



Neural Networks: Mathematical Model



Neural Networks: Mathematical Model



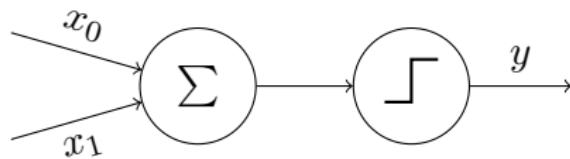
Artificial Neural Networks

Artificial Neural Networks

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

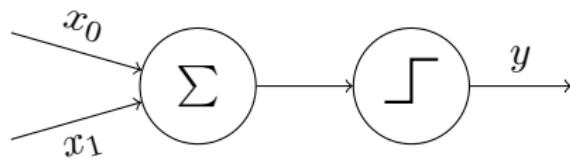
Artificial Neural Networks

- ▶ 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

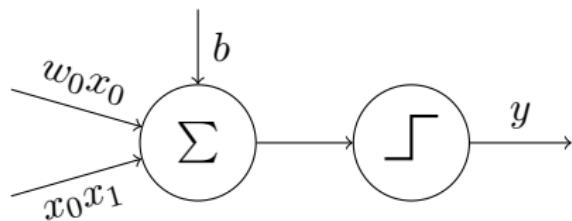


Artificial Neural Networks

- Frank Rosenblatt improved on this model in 1957

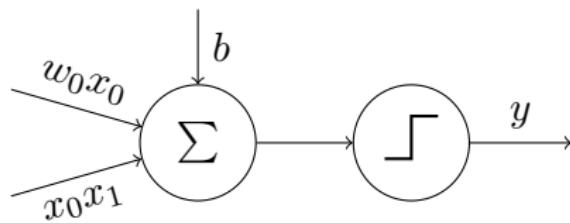
Artificial Neural Networks

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



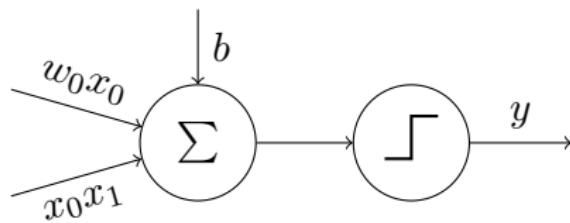
Artificial Neural Networks

- ▶ 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 $\hat{y} = 0 \wedge y = 1$ **then**

Decrease all weights w_i where x_i was 1

else if $\hat{y} = 1 \wedge y = 0$ **then**

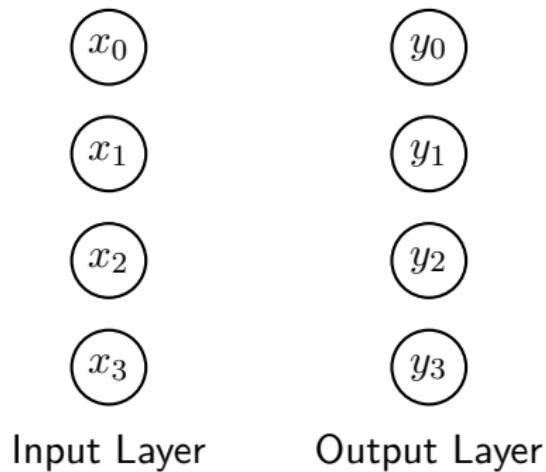
Increase all weights w_i where x_i was 1

Artificial Neural Networks

- ▶ Perceptrons even work for multi-class classification

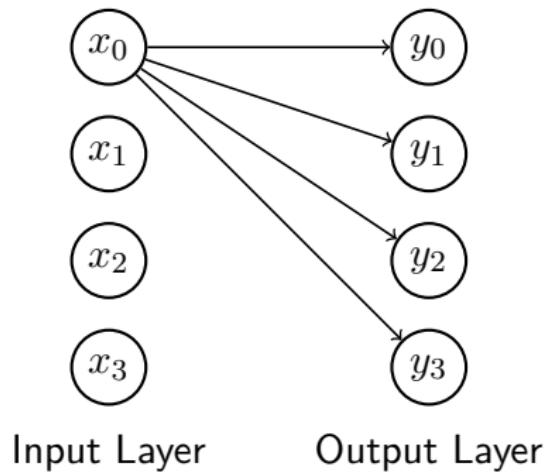
Artificial Neural Networks

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



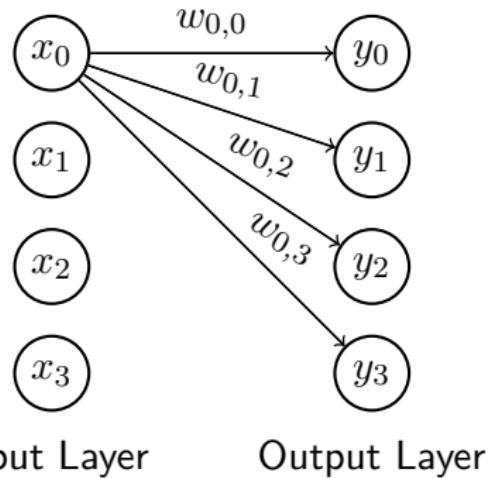
Artificial Neural Networks

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



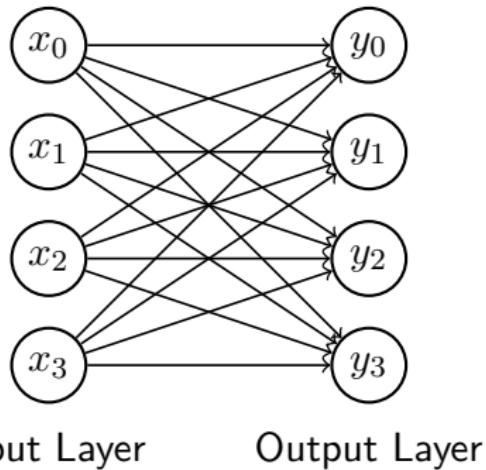
Artificial Neural Networks

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



Artificial Neural Networks

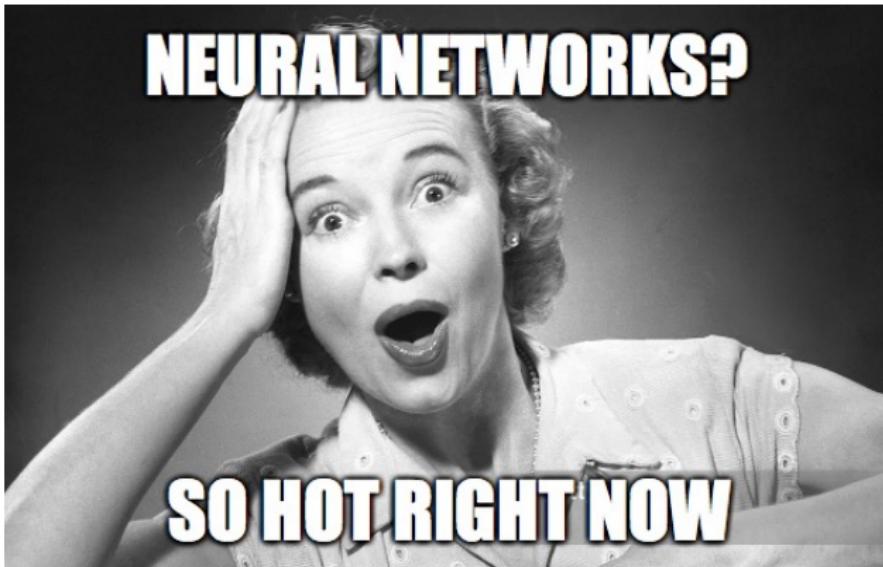
- ▶ Perceptrons even work for multi-class classification
- ▶ Just use more perceptrons
- ▶ This is now a *multilayer perceptron* (MLP)



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.

[nyt16]



Artificial Neural Networks

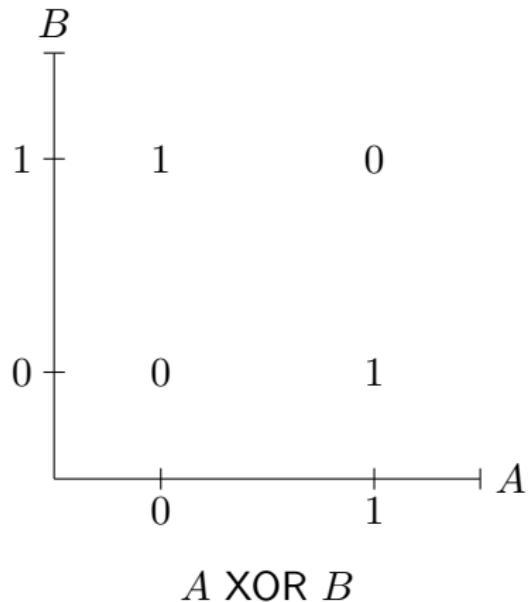
- ▶ Perceptrons have one fundamental problem

Artificial Neural Networks

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

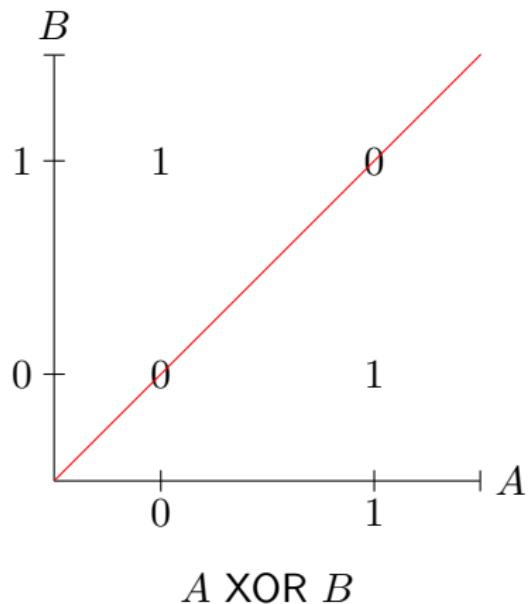
Artificial Neural Networks

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



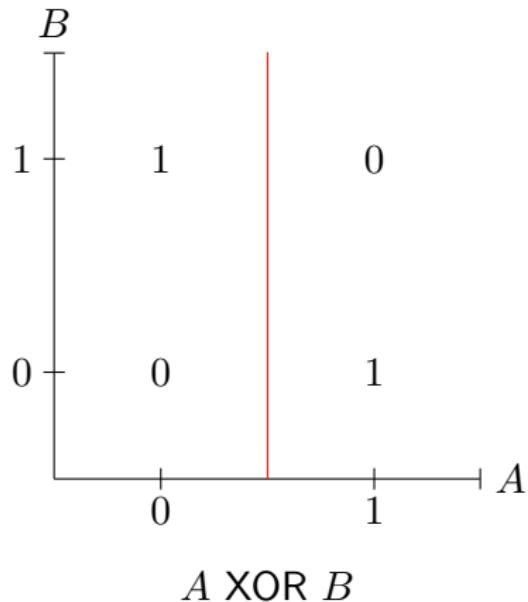
Artificial Neural Networks

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



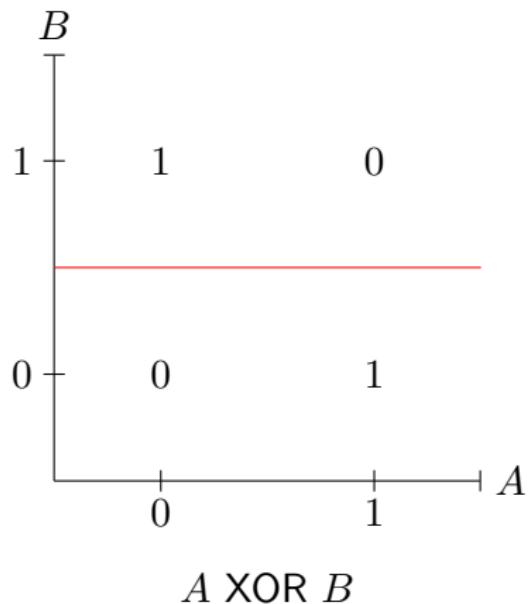
Artificial Neural Networks

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



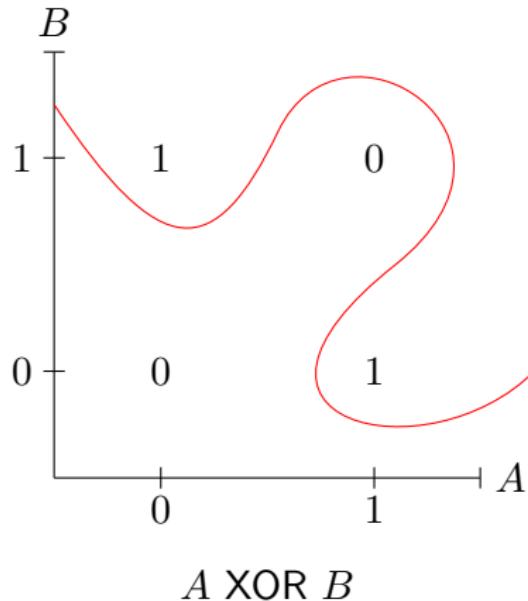
Artificial Neural Networks

- ▶ 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



Artificial Neural Networks

- The solution:

Artificial Neural Networks

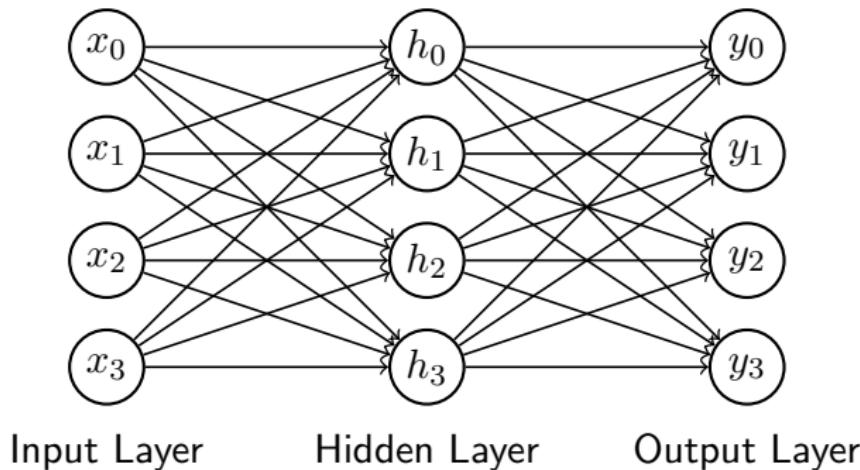
- ▶ The solution:
 - ▶ A hidden layer

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

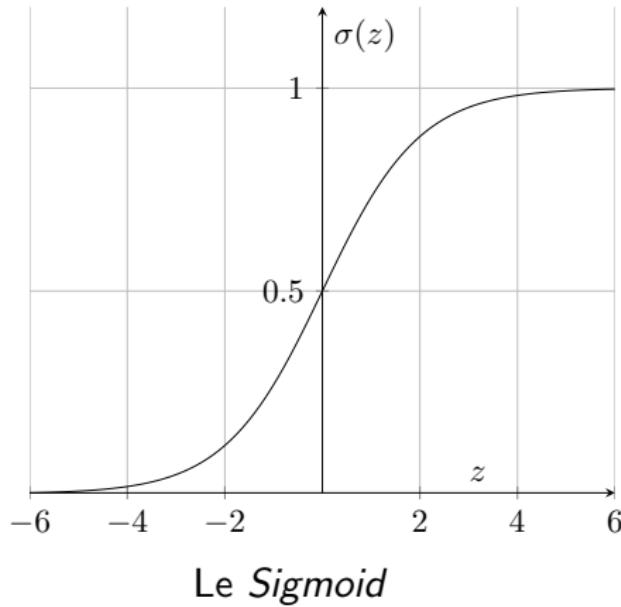


Activation Functions

- ▶ Three important activation functions

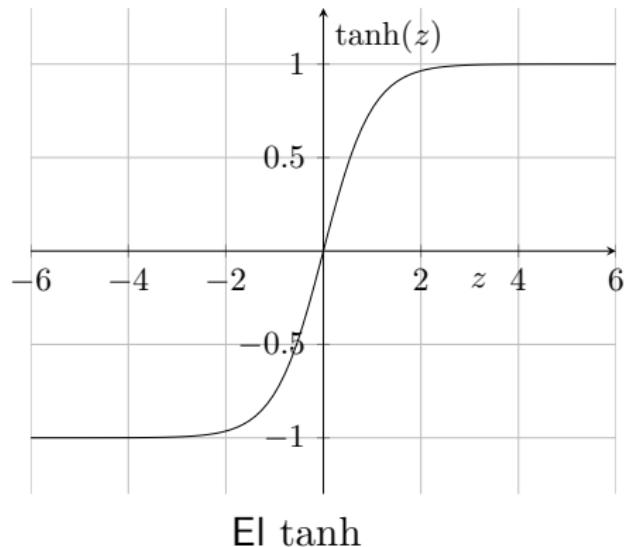
Activation Functions

- ▶ Three important activation functions



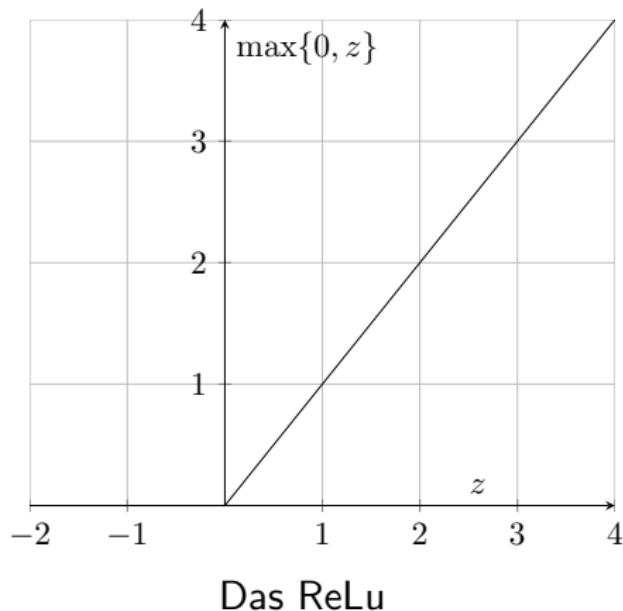
Activation Functions

- ▶ Three important activation functions



Activation Functions

- Three important activation functions



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. 133t h4x0r

Neural Networks by Foot

- We want to classify programmers into one of two classes
 1. 133t h4x0r
 2. n00b

Neural Networks by Foot

- We want to classify programmers into one of two classes
 1. 133t 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. 133t 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. 133t h4x0r
 2. n00b
- ▶ Two input features:
 1. Maximum number of successive pointers to pointers maintained
 2. Liters of coffee per year

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

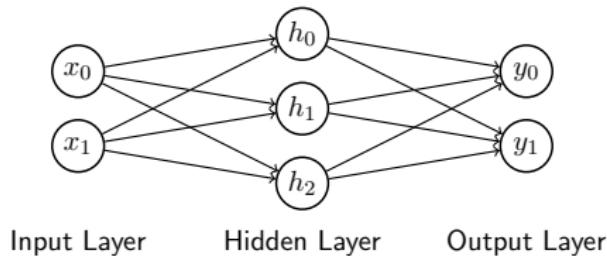
Neural Networks by Foot

- We want to classify programmers into one of two classes
 1. 133t 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

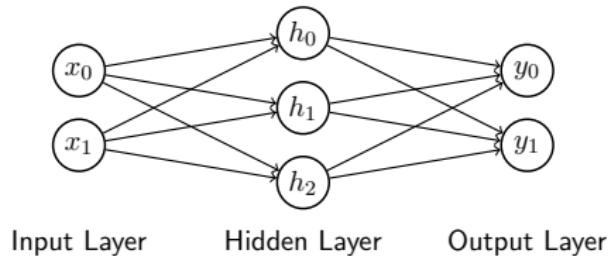
$$\mathbf{D} = \begin{bmatrix} p & c \\ 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix} \quad \hat{\mathbf{Y}} = \begin{bmatrix} 133t & n00b \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Neural Networks by Foot

- ▶ We want to classify programmers into one of two classes
 1. 133t 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

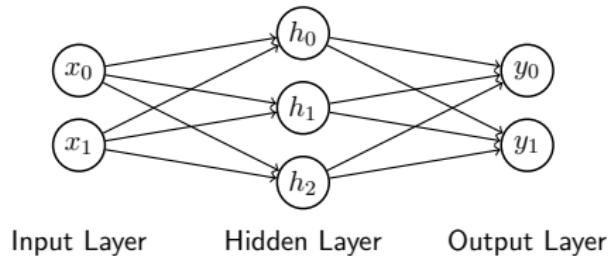


Neural Networks by Foot



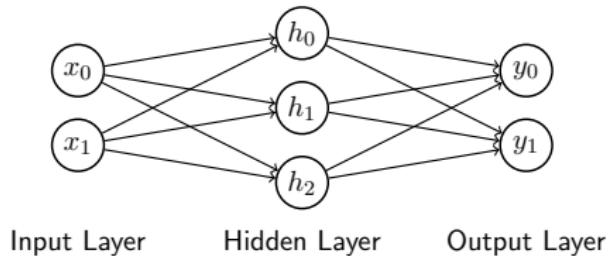
$$\mathbf{D} = \begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix}$$

Neural Networks by Foot



$$\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix} \times \begin{bmatrix} -0.04 & -0.43 & 0.57 \\ 0.04 & 0.52 & -0.6 \end{bmatrix} \mathbf{W}_1$$

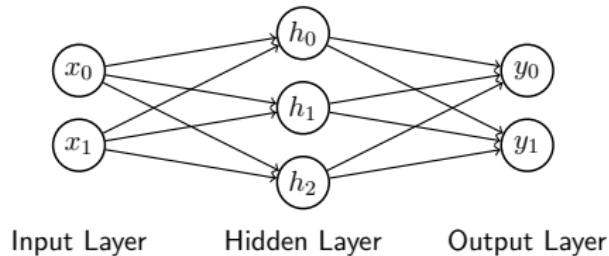
Neural Networks by Foot



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

D **W₁** **b₁**

Neural Networks by Foot



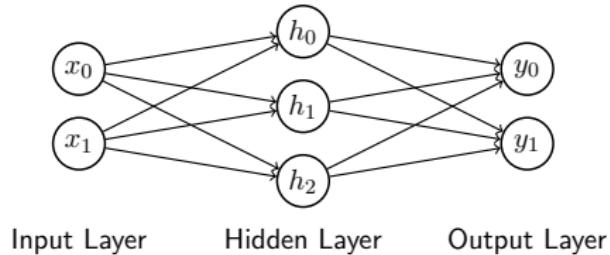
$$\begin{bmatrix} 10 & 365 \\ 3 & 120 \\ 0 & 1000 \end{bmatrix} \times \begin{bmatrix} -0.04 & -0.43 & 0.57 \\ 0.04 & 0.52 & -0.6 \end{bmatrix} +_b \begin{bmatrix} -2.03 & -0.26 & 2.16 \end{bmatrix} \mathbf{b}_1$$

D **W₁**

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

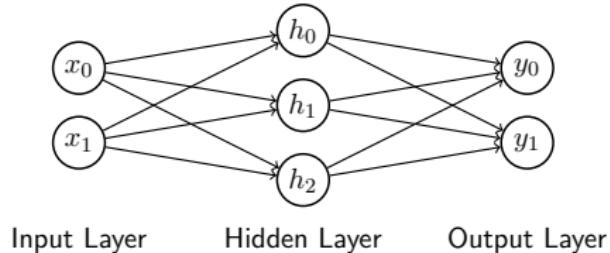
H

Neural Networks by Foot



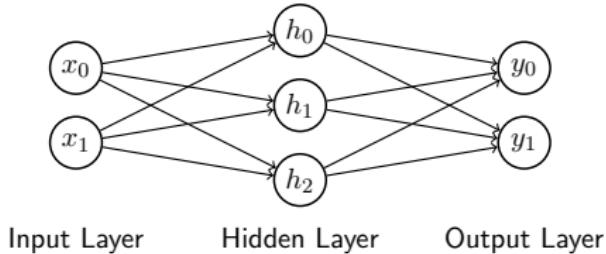
$\text{relu}(\mathbf{H})$

Neural Networks by Foot



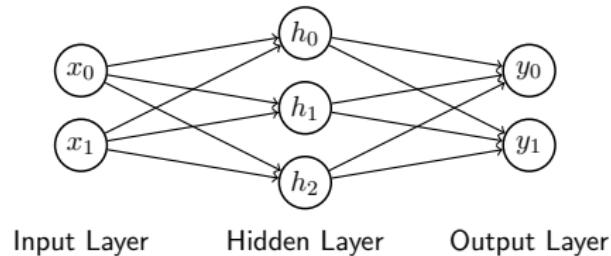
$$\text{relu}(\mathbf{H}) = \max\{0, \mathbf{H}\}$$

Neural Networks by Foot



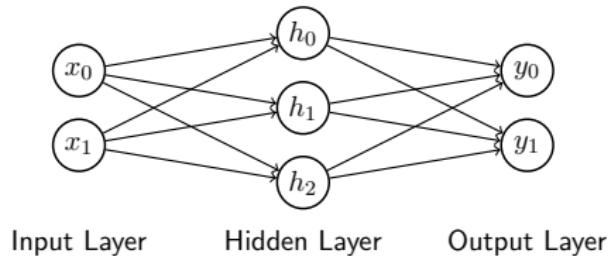
$$\text{relu}(\mathbf{H}) = \max\{0, \mathbf{H}\} = \begin{bmatrix} 13.37 & 183.66 & 0.0 \\ 3.05 & 60.33 & 0.0 \\ 41.13 & 515.49 & 0.0 \end{bmatrix} \mathbf{H}'$$

Neural Networks by Foot



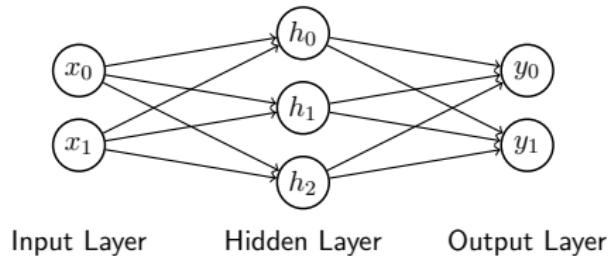
\mathbf{H}'

Neural Networks by Foot



$$\mathbf{H}' \times \begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix} \mathbf{W}_2$$

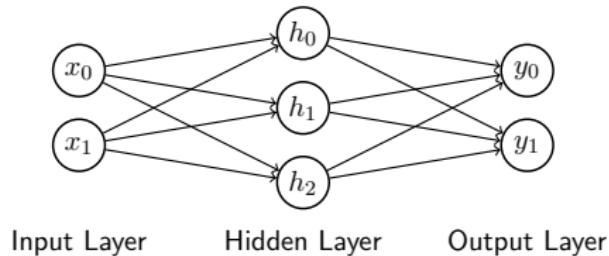
Neural Networks by Foot



$$\mathbf{H}' \times \begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix} +_b \begin{bmatrix} -0.19 & -0.49 \end{bmatrix} \mathbf{b}_2$$

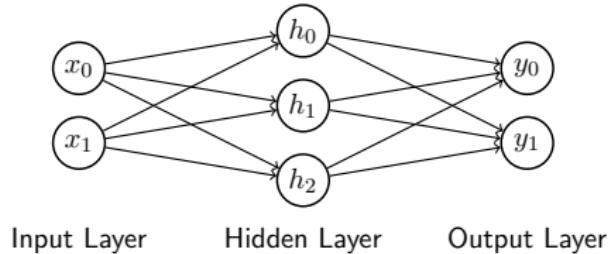
\mathbf{W}_2

Neural Networks by Foot



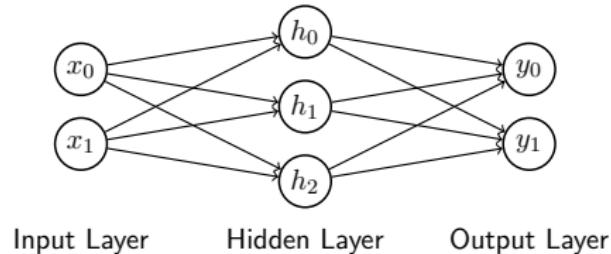
$$\begin{aligned} \mathbf{H}' \times & \begin{bmatrix} 0.44 & 0.19 \\ -0.04 & 1.14 \\ 0.68 & 0.85 \end{bmatrix} +_b \begin{bmatrix} -0.19 & -0.49 \end{bmatrix} \\ & \mathbf{W}_2 \\ = & \begin{bmatrix} -2.53 & 211.85 \\ -1.55 & 69.01 \\ -5.16 & 596.17 \end{bmatrix} \\ & \mathbf{Y} \end{aligned}$$

Neural Networks by Foot



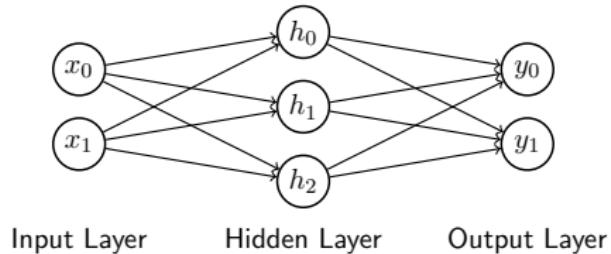
$$\text{softmax}(\mathbf{Y}) =$$

Neural Networks by Foot



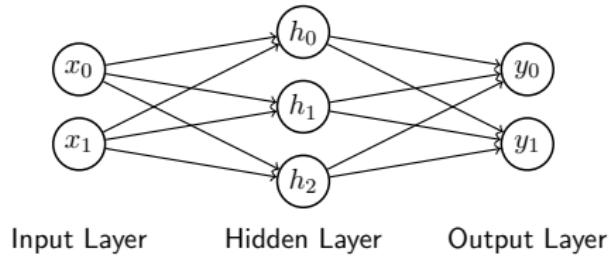
$$\text{softmax}(\mathbf{Y}) = \frac{\begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \\ 0.0 & 1.0 \end{bmatrix}}{\mathbf{Y}}$$

Neural Networks By Foot



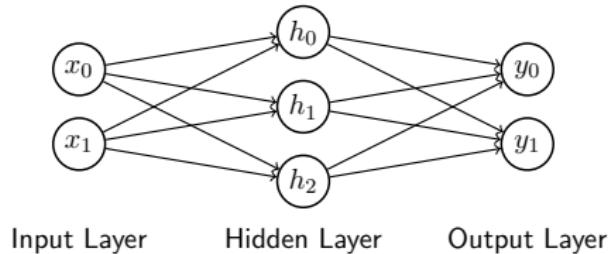
- ▶ Let \mathcal{W} be $[\mathbf{W}_1, \mathbf{W}_2]$

Neural Networks By Foot



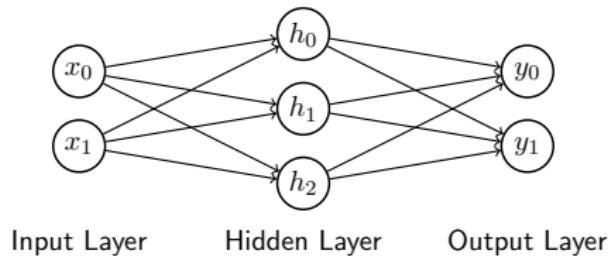
- ▶ Let \mathcal{W} be $[\mathbf{W}_1, \mathbf{W}_2]$
- ▶ Then $J(\mathcal{W})$ is the network's loss

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

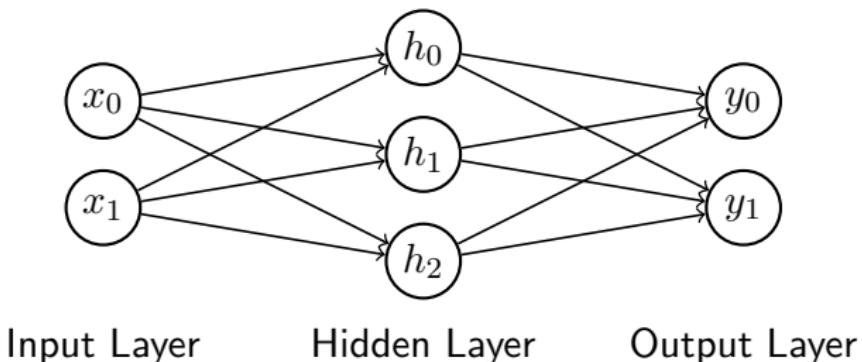
- ▶ One recent regularization technique is *Dropout*
- ▶ It is used especially for deep neural networks
- ▶ Basic idea:
 - ▶ We deactivate neurons on each activation with probability p
 - ▶ Can be seen as sampling a new neural network each time
 - ▶ This prevents neurons from depending too much on each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

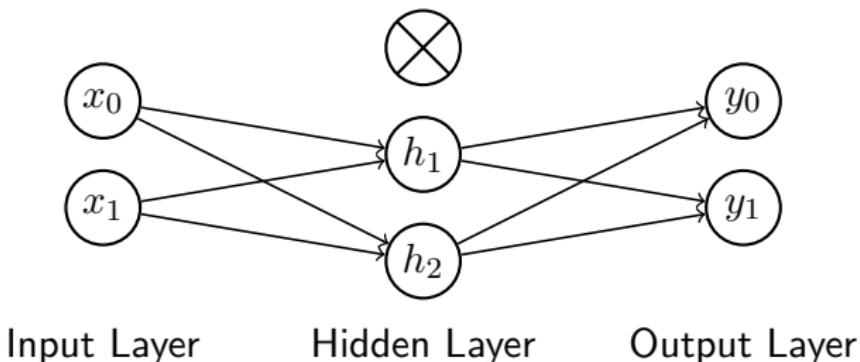
Dropout

- ▶ 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



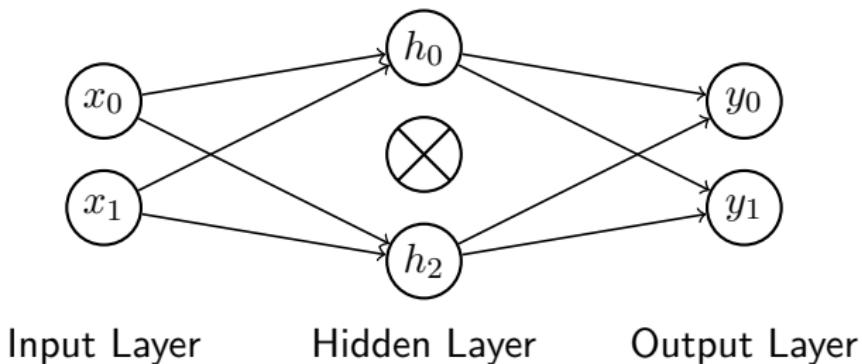
Dropout

- ▶ 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



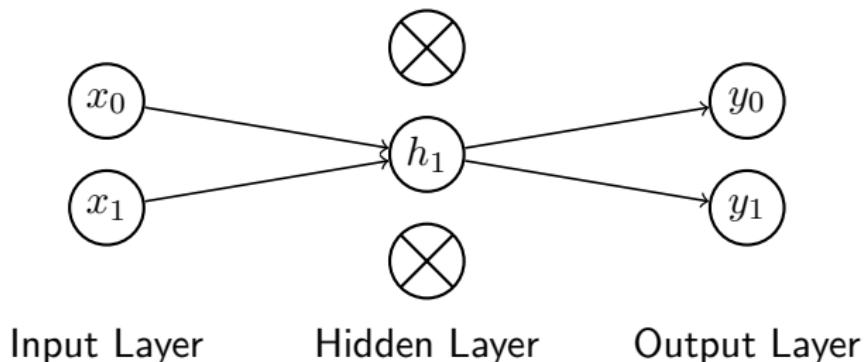
Dropout

- ▶ 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



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

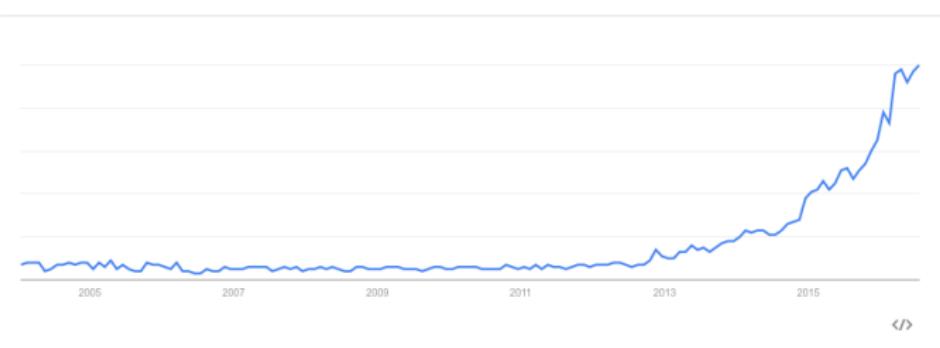
Deep Learning

The why, the what

Deep Learning

The why, the what and the ugly.

Deep Learning



Basic Definition

Basic Definition

- ▶ Let \mathcal{L} denote the number of hidden layers in a neural network

Basic Definition

- ▶ Let \mathcal{L} denote the number of hidden layers in a neural network
- ▶ Then we call neural network *deep*, if

Basic Definition

- ▶ Let \mathcal{L} denote the number of hidden layers in a neural network
- ▶ Then we call neural network *deep*, if
 1. It is trained at more than 10,000m below sea level, or

Basic Definition

- ▶ Let \mathcal{L} denote the number of hidden layers in a neural network
- ▶ Then we call neural network *deep*, if
 1. It is trained at more than 10,000m below sea level, or
 2. $\mathcal{L} > 1$

Basic Definition

- ▶ Let \mathcal{L} denote the number of hidden layers in a neural network
- ▶ Then we call neural network *deep*, if
 1. It is trained at more than 10,000m below sea level, or
 2. $\mathcal{L} > 1$
- ▶ To really understand why many layers are a good idea, we must understand why few layers might be a bad idea

Universal Approximation Theorem

Universal Approximation Theorem

- Theoretically, a single layer is just enough

Universal Approximation Theorem

- ▶ Theoretically, a single layer is just enough
- ▶ However, we would need exponentially many units for discrete functions

Universal Approximation Theorem

- ▶ Theoretically, a single layer is just enough
- ▶ However, we would need exponentially many units for discrete functions
- ▶ And infinitely many for continuous functions

Universal Approximation Theorem

- ▶ Theoretically, a single layer is just enough
- ▶ However, we would need exponentially many units for discrete functions
- ▶ And infinitely many for continuous functions
- ▶ Just use an infinitely sized hash table

The Curse of Dimensionality

The Curse of Dimensionality

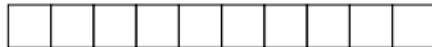
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



1 feature = 10 outputs

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



1 feature = 10 outputs

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



1 feature = 10 outputs

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



1 feature = 10 outputs

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



1 feature = 10 outputs

The Curse of Dimensionality

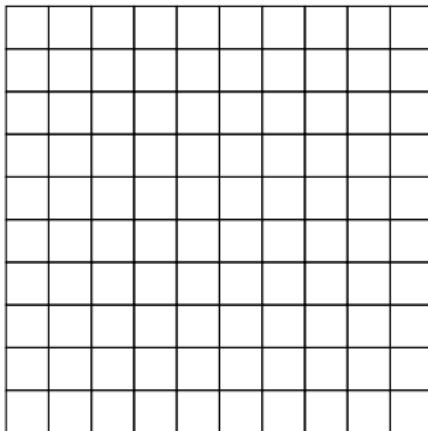
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



1 feature = 10 outputs

The Curse of Dimensionality

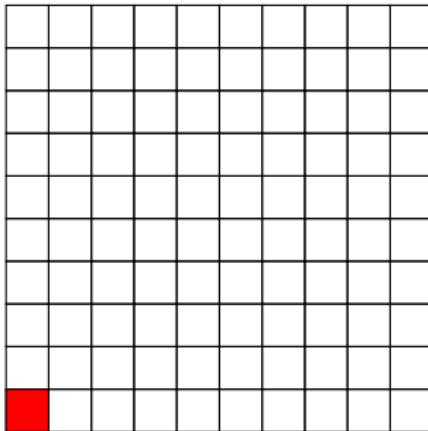
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



2 features = 100 outputs

The Curse of Dimensionality

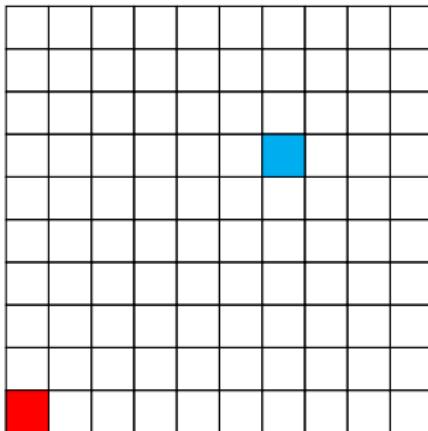
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



2 features = 100 outputs

The Curse of Dimensionality

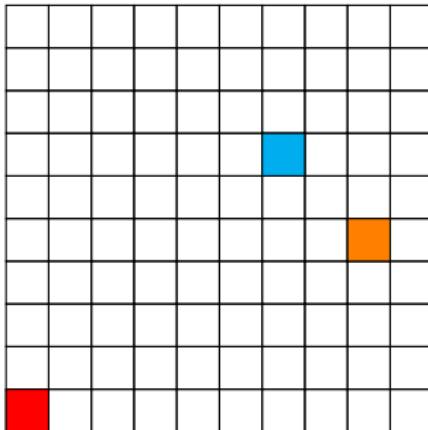
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



2 features = 100 outputs

The Curse of Dimensionality

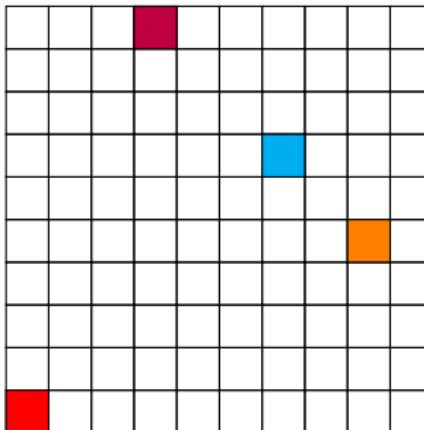
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



2 features = 100 outputs

The Curse of Dimensionality

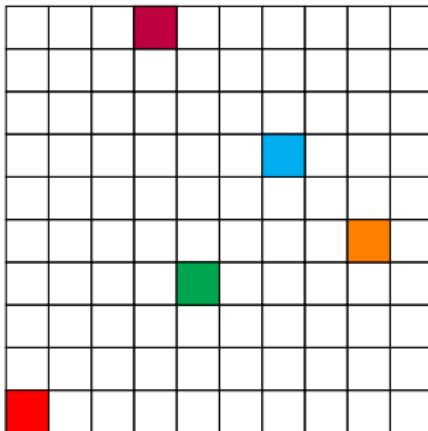
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



2 features = 100 outputs

The Curse of Dimensionality

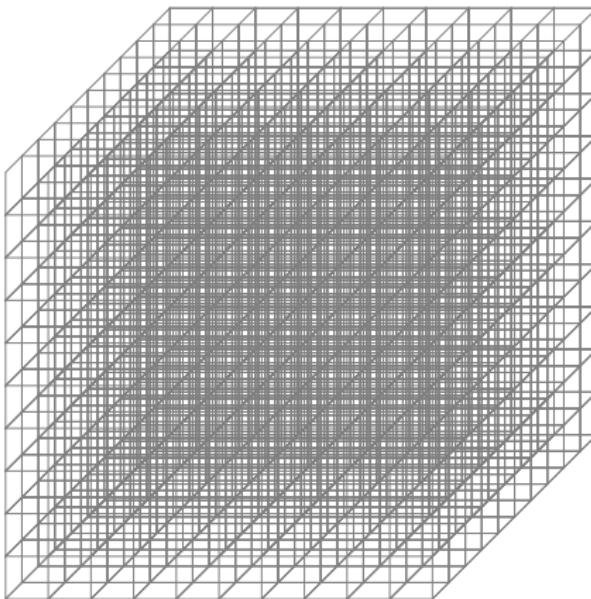
- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



2 features = 100 outputs

The Curse of Dimensionality

- ▶ The number of possible outputs scales exponentially with the dimensionality of our dataset
- ▶ Assume our features have 10 possible values



3 features = 100 outputs

The Curse of Dimensionality

- ▶ Why is this a problem?

The Curse of Dimensionality

- ▶ Why is this a problem?
- ▶ To predict an output, we must have seen at least one example

The Curse of Dimensionality

- ▶ Why is this a problem?
- ▶ To predict an output, we must have seen at least one example
- ▶ In high dimensions, an algorithm cannot possibly be trained on all possible output

The Curse of Dimensionality

- ▶ Why is this a problem?
- ▶ To predict an output, we must have seen at least one example
- ▶ In high dimensions, an algorithm cannot possibly be trained on all possible output, unless

The Curse of Dimensionality

- ▶ Why is this a problem?
- ▶ To predict an output, we must have seen at least one example
- ▶ In high dimensions, an algorithm cannot possibly be trained on all possible output, unless

We make assumptions about our data

The Curse of Dimensionality

- ▶ Assumptions either

The Curse of Dimensionality

- ▶ Assumptions either
 - ▶ Explicitly reduce the output space, or

The Curse of Dimensionality

- ▶ Assumptions either
 - ▶ Explicitly reduce the output space, or
 - ▶ Create dependencies between outputs

The Curse of Dimensionality

- ▶ Assumptions either
 - ▶ Explicitly reduce the output space, or
 - ▶ Create dependencies between outputs
- ▶ *Local constancy* assumption (prior)

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \varepsilon)$$

The Curse of Dimensionality

- ▶ Assumptions either
 - ▶ Explicitly reduce the output space, or
 - ▶ Create dependencies between outputs
- ▶ *Local constancy assumption (prior)*

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \varepsilon)$$



The Curse of Dimensionality

- ▶ Assumptions either
 - ▶ Explicitly reduce the output space, or
 - ▶ Create dependencies between outputs
- ▶ *Local constancy assumption (prior)*

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \varepsilon)$$



The Curse of Dimensionality

- ▶ Assumptions either
 - ▶ Explicitly reduce the output space, or
 - ▶ Create dependencies between outputs
- ▶ *Local constancy* assumption (prior)

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \varepsilon)$$

- ▶ Don't need as much data any more!



The Curse of Dimensionality

Deep Learning assumes that data is structured

The Curse of Dimensionality

Deep Learning assumes that data is structured
hierarchically

The Curse of Dimensionality

Deep Learning assumes that data is structured
hierarchically

Note:

According to the *No Free Lunch Theorem* it is just as bad as flipping a coin.

Convolutional Neural Networks

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

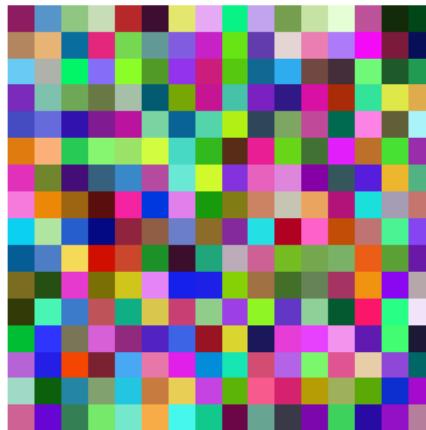
Convolutional Neural Networks

- ▶ 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



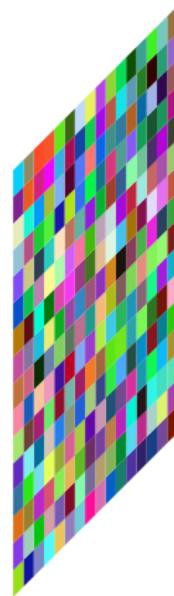
Convolutional Neural Networks

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

24	25	50	44	0	77	57	85	81	76	3	93	74	31	92	43
94	50	73	28	72	65	8	59	23	38	14	17	63	44	84	75
43	71	90	31	49	1	28	38	10	16	55	8	42	41	62	39
45	50	67	34	12	3	58	20	67	90	54	99	0	45	88	26
42	19	7	3	98	88	94	69	3	4	98	8	54	0	94	27
57	98	93	9	25	4	24	2	74	83	77	25	7	50	65	67
28	59	31	40	78	4	25	38	70	5	29	1	24	73	52	12
52	44	5	85	5	20	54	58	73	12	80	17	72	11	31	51
46	16	16	52	28	8	43	27	28	21	74	92	55	80	23	49
54	98	47	44	59	17	44	21	62	69	39	22	13	11	28	30
46	8	57	56	92	22	49	19	56	21	7	17	66	54	75	99
58	22	35	2	41	12	96	65	59	73	56	39	64	57	29	77
4	87	62	46	57	55	24	31	9	72	9	65	67	51	27	59
12	28	8	12	39	95	62	29	15	55	82	62	54	84	31	95
90	79	33	82	53	73	51	39	20	1	26	78	0	14	65	6
18	6	98	22	8	5	74	67	66	95	0	66	59	3	78	48

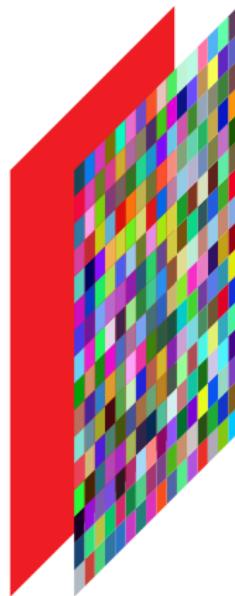
Convolutional Neural Networks

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



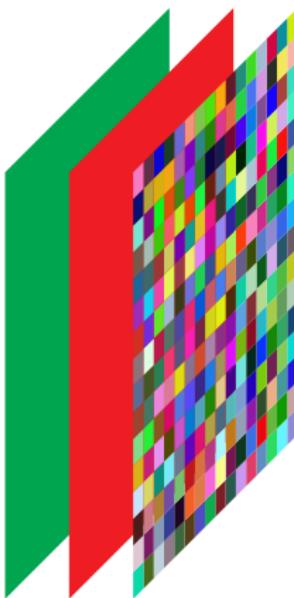
Convolutional Neural Networks

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



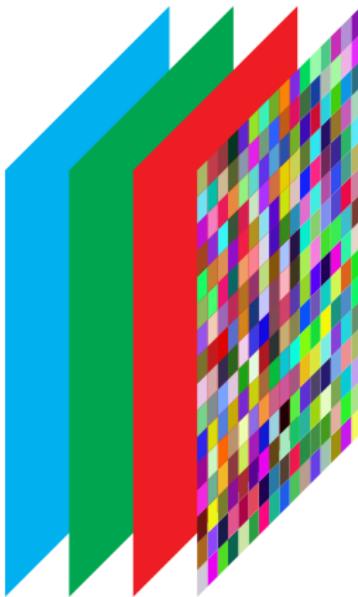
Convolutional Neural Networks

- ▶ 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

Convolutional Neural Network

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

Convolutional Neural Network

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

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

Convolutional Neural Network

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

R *G* *B*

R.flatten = [116 80 170 194]

Convolutional Neural Network

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

`R.flatten = [116 80 170 194]`

`G.flatten = [82 78 5 236]`

Convolutional Neural Network

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

R *G* *B*

$$R.\text{flatten} = [116 \ 80 \ 170 \ 194]$$

$$G.\text{flatten} = [82 \ 78 \ 5 \ 236]$$

$$B.\text{flatten} = [76 \ 139 \ 245 \ 236]$$

Convolutional Neural Network

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

$$R.\text{flatten} = [116 \ 80 \ 170 \ 194]$$

$$G.\text{flatten} = [82 \ 78 \ 5 \ 236]$$

$$B.\text{flatten} = [76 \ 139 \ 245 \ 236]$$

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

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: $100 \times 120,000 = 12,000,000$
 - ▶ With 5 layers: $12,000,000 \times 5 = 60,000,000$ weights

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: $100 \times 120,000 = 12,000,000$
 - ▶ With 5 layers: $12,000,000 \times 5 = 60,000,000$ weights
 - ▶ $1000 \times 1000 \rightarrow 1,500,000,000$ weights

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: $100 \times 120,000 = 12,000,000$
 - ▶ With 5 layers: $12,000,000 \times 5 = 60,000,000$ weights
 - ▶ $1000 \times 1000 \rightarrow 1,500,000,000$ weights
 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

Convolutional Neural Network: Mechanics

Recipe for a Convolutional Neural Network

Convolutional Neural Network: Mechanics

Recipe for a Convolutional Neural Network

- ▶ Ingredients

Convolutional Neural Network: Mechanics

Recipe for a Convolutional Neural Network

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

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 $k \times k \times d$

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

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

- ▶ 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

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

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
- ▶ Compute the “dot product” for each configuration

Convolutional Neural Network: Mechanics

Recipe for a Convolutional Neural Network

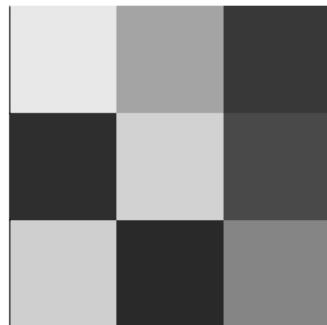
- ▶ Ingredients

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
- ▶ 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

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

Image

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

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

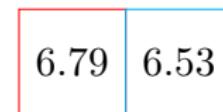
6.79	6.53
------	------

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

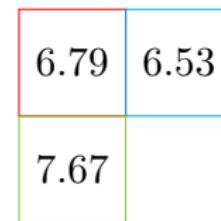


Output

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

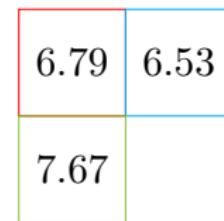


Output

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



Output

Convolutional Neural Network: Mechanics

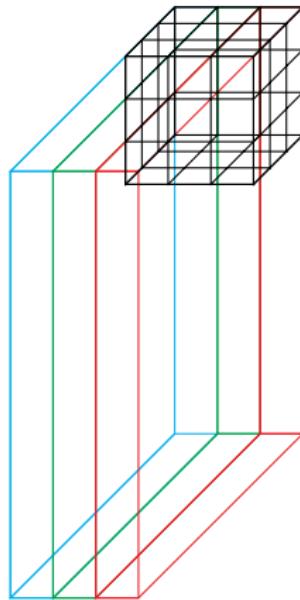
0.4	0.9	0.1
0.7	$5.7 \cdot 0.2$	$2.4 \cdot 0.6$
0.8	$3.1 \cdot 0.3$	$0.9 \cdot 0.5$

Image

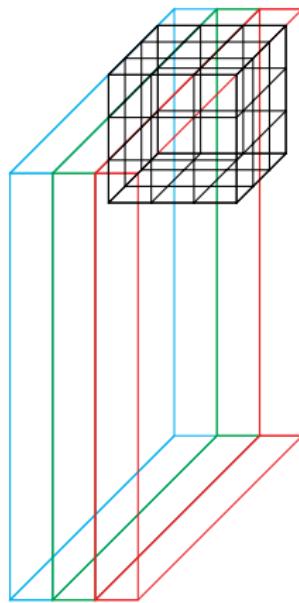
6.79	6.53
7.67	3.96

Output

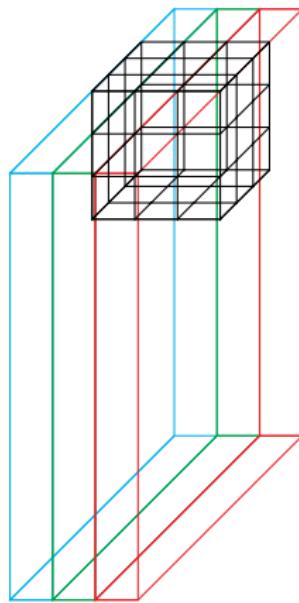
Convolutional Neural Networks: Mechanics



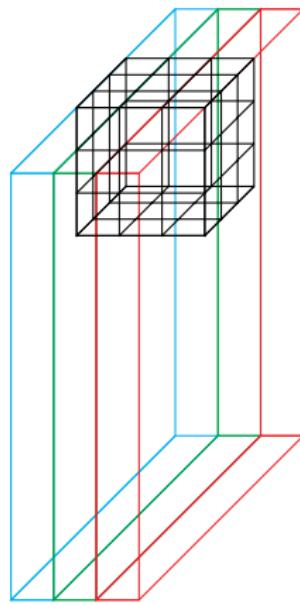
Convolutional Neural Networks: Mechanics



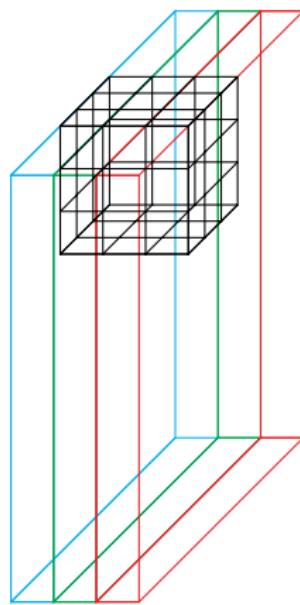
Convolutional Neural Networks: Mechanics



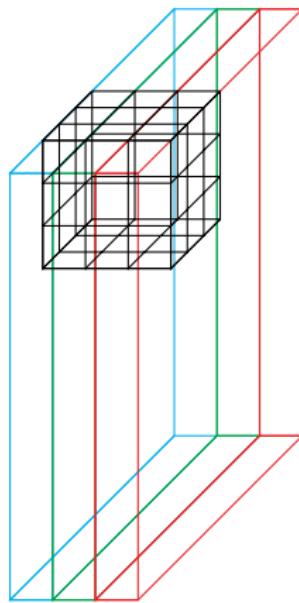
Convolutional Neural Networks: Mechanics



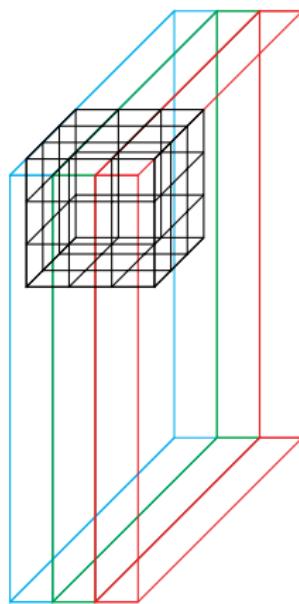
Convolutional Neural Networks: Mechanics



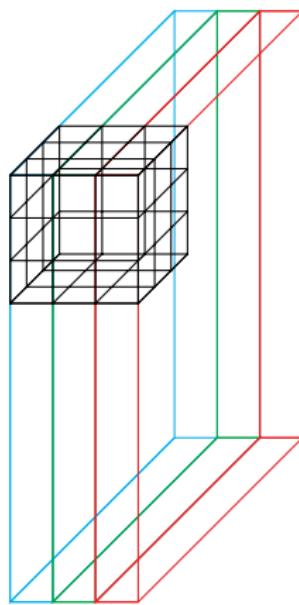
Convolutional Neural Networks: Mechanics



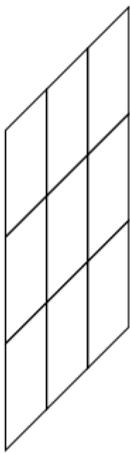
Convolutional Neural Networks: Mechanics



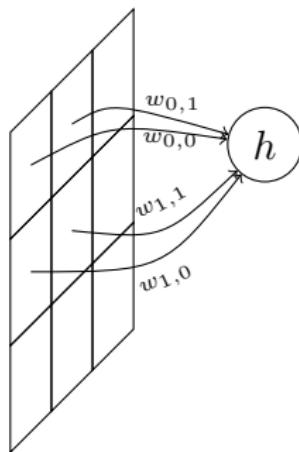
Convolutional Neural Networks: Mechanics



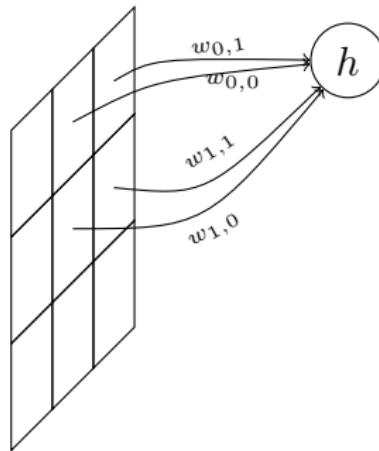
Convolutional Neural Network: Mechanics



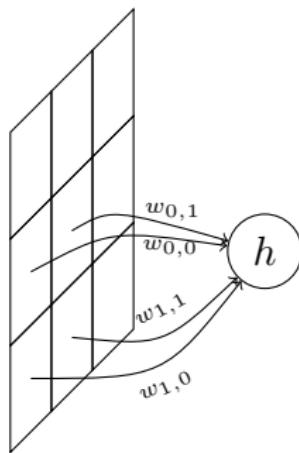
Convolutional Neural Network: Mechanics



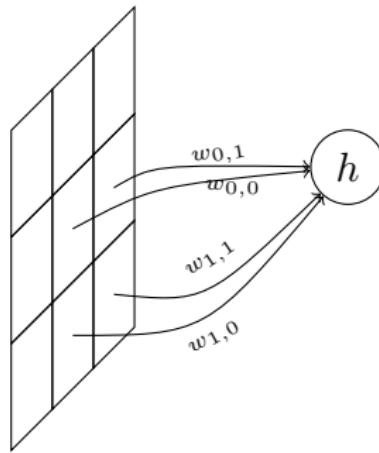
Convolutional Neural Network: Mechanics



Convolutional Neural Network: Mechanics



Convolutional Neural Network: Mechanics



Convolutional Neural Network: Hyperparameters

- ▶ Convolutional Neural Networks have three hyperparameters

Convolutional Neural Network: Hyperparameters

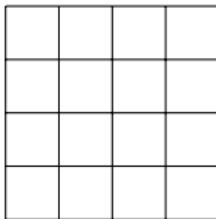
- ▶ Convolutional Neural Networks have three hyperparameters
 1. Kernel size

Convolutional Neural Network: Hyperparameters

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

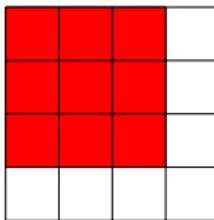
Convolutional Neural Network: Hyperparameters

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



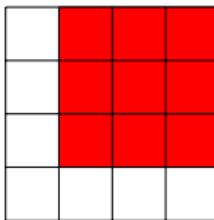
Convolutional Neural Network: Hyperparameters

- ▶ 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)

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)

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)

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)

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

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
```

[Kar16b]

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

Convolutional Neural Networks: Intuition

What's in a kernel?

0	1	0
0	1	0
0	1	0

Patterns

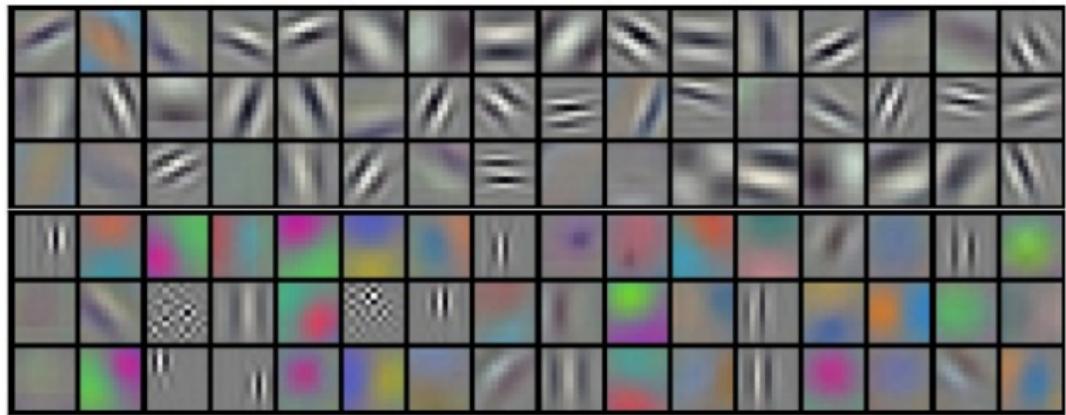
Convolutional Neural Networks: Intuition

What's in a kernel?

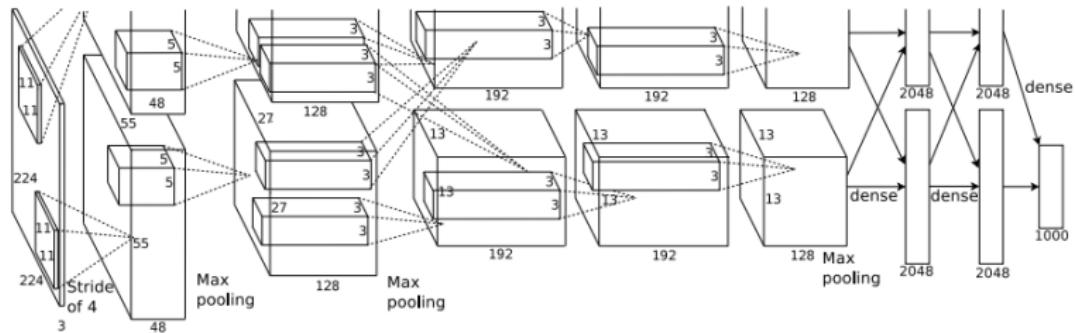
0	0	0
-1	1	0
0	0	0

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

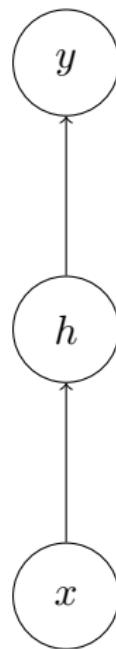
I enjoy eating dinner with people

Sequences

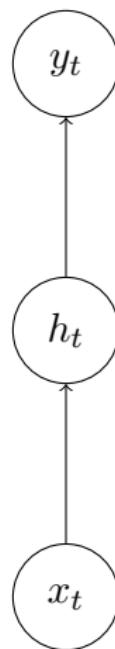
- ▶ Humans think in sequences
- ▶ Simple neural networks don't
- ▶ Sequences give single entities context
- ▶ The key to understanding sequences is memory

I enjoy eating dinner with people

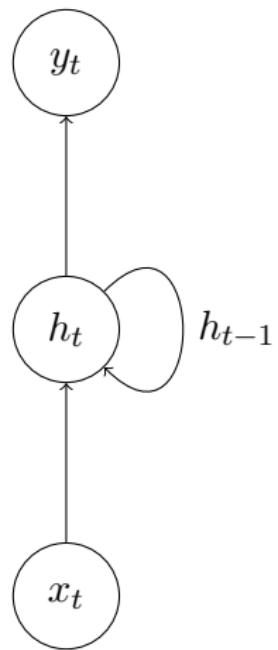
Machine Memory



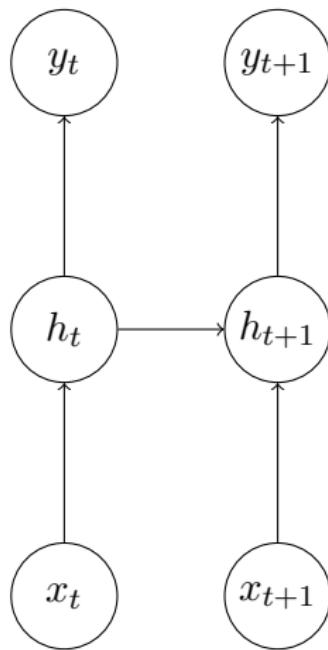
Machine Memory



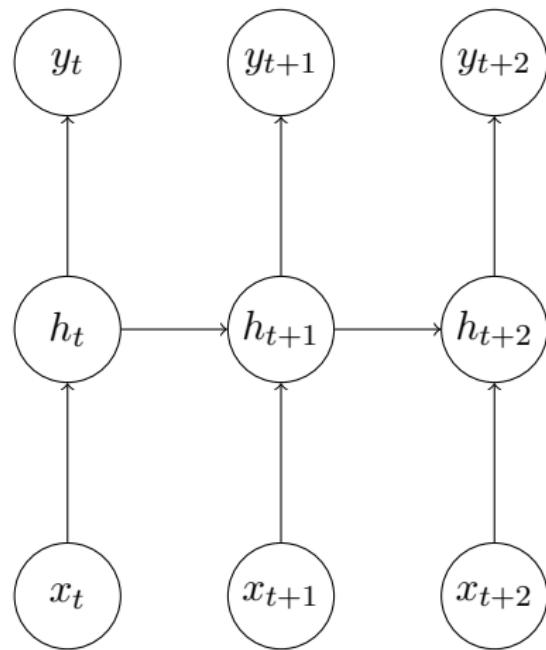
Machine Memory



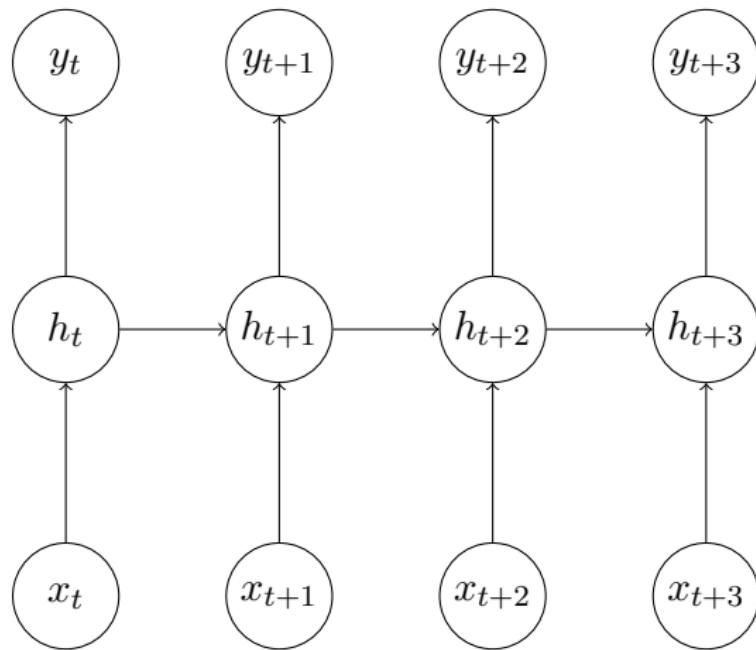
Machine Memory



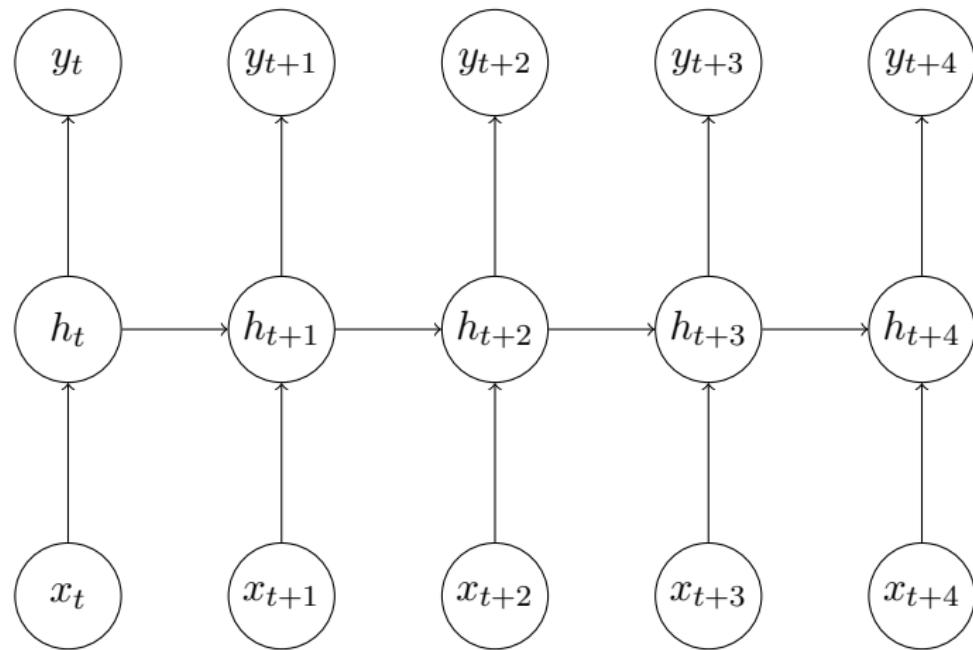
Machine Memory



Machine Memory



Machine Memory



Machine Memory

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

Machine Memory

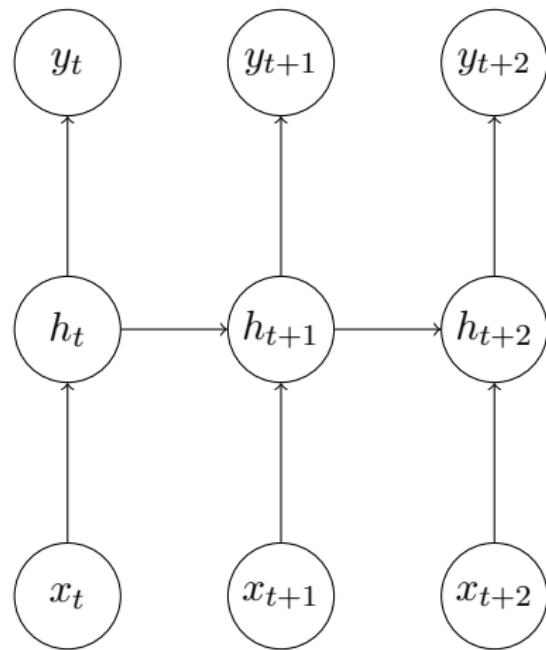
$$h_t = f(\mathbf{w}^\top [h_{t-1}, x] + b)$$

Recurrent Neural Networks

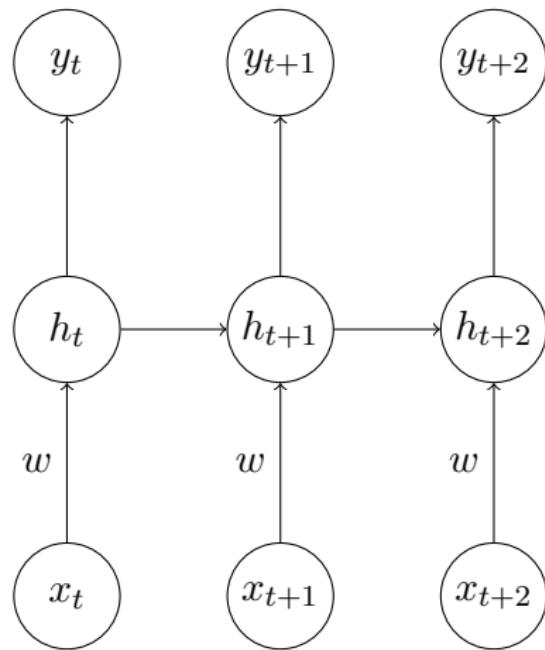
- ▶ Recurrent Neural Networks (RNNs) share weights through time
- ▶ They have *memory*
- ▶ And they have a problem:

The Vanishing Gradient Problem

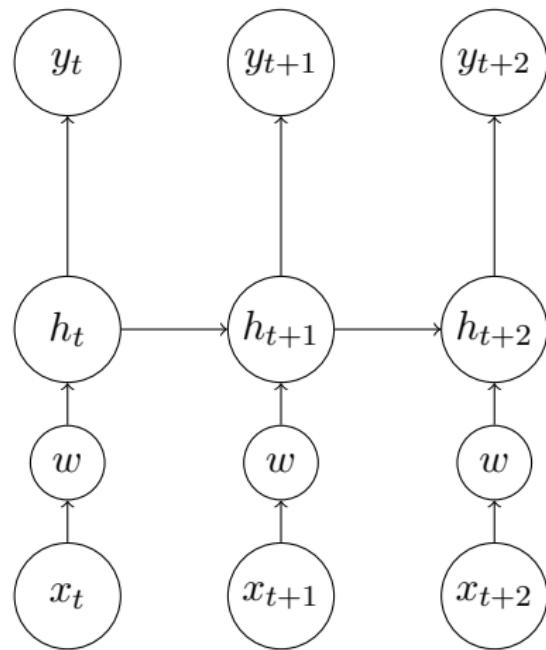
Machine Memory



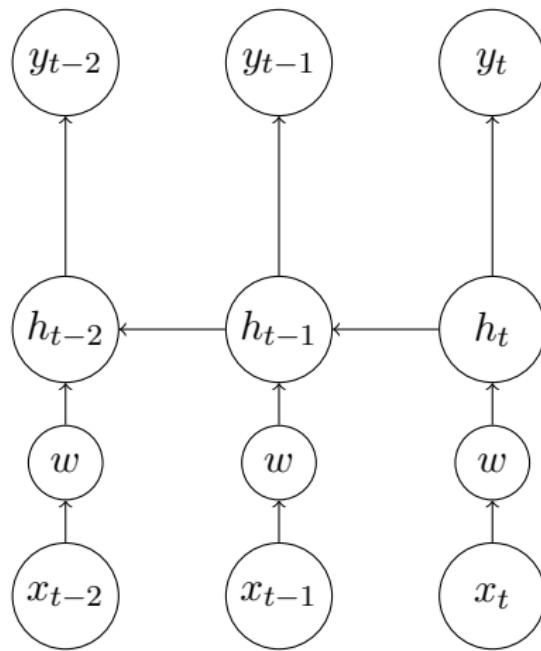
Machine Memory



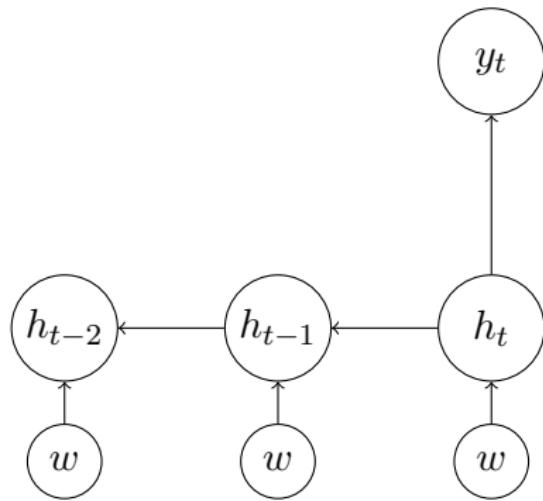
Machine Memory



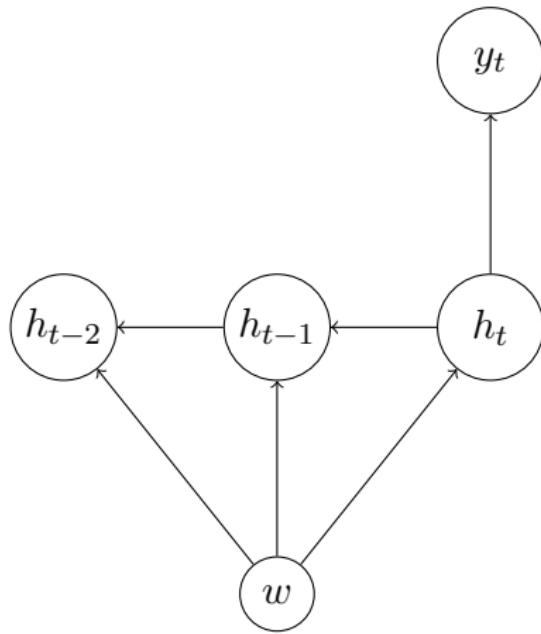
Machine Memory



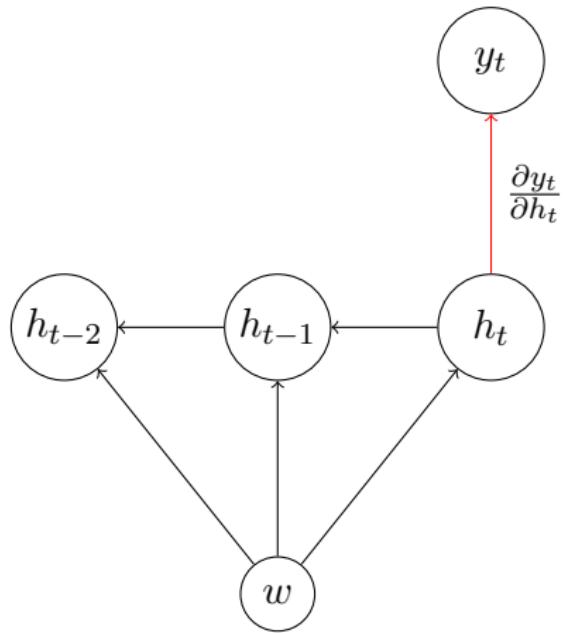
Machine Memory



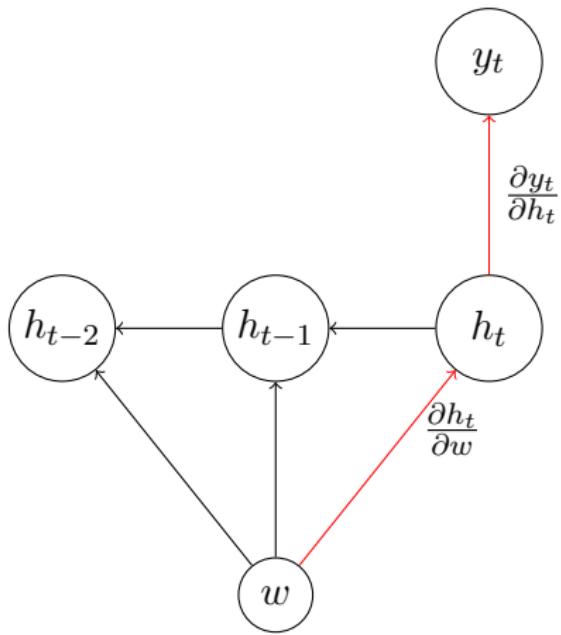
Machine Memory



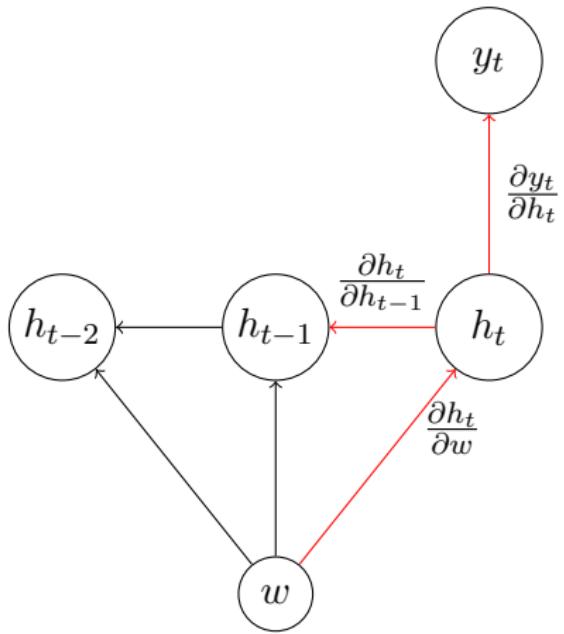
Machine Memory



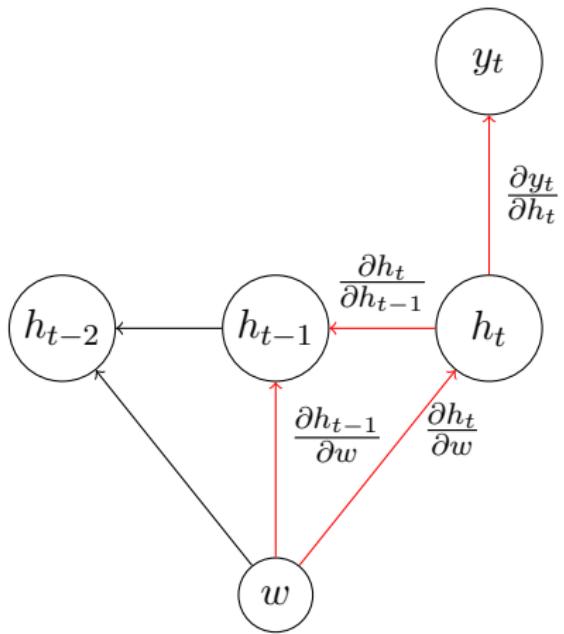
Machine Memory



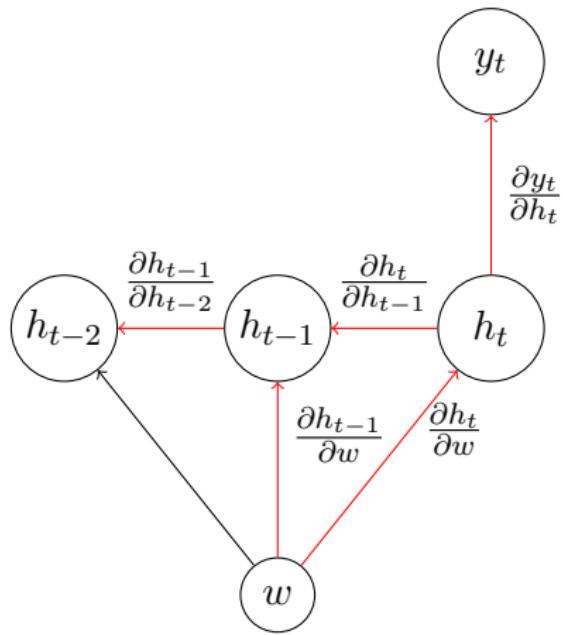
Machine Memory



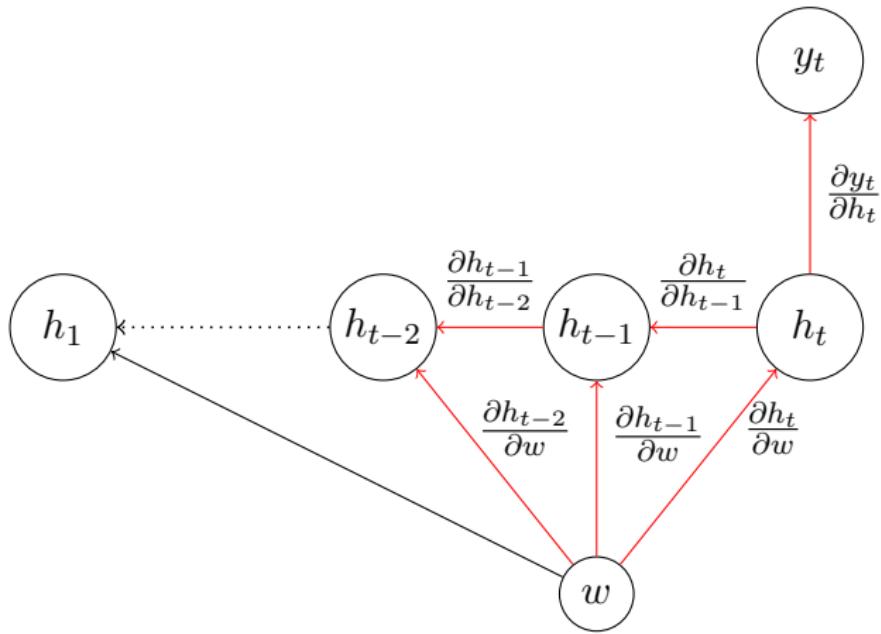
Machine Memory



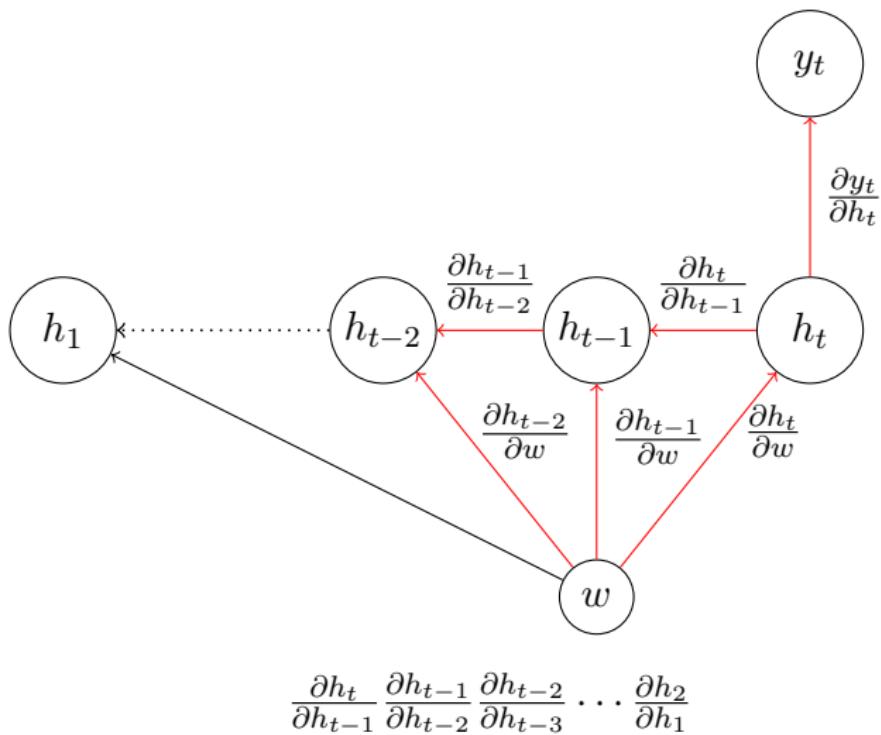
Machine Memory



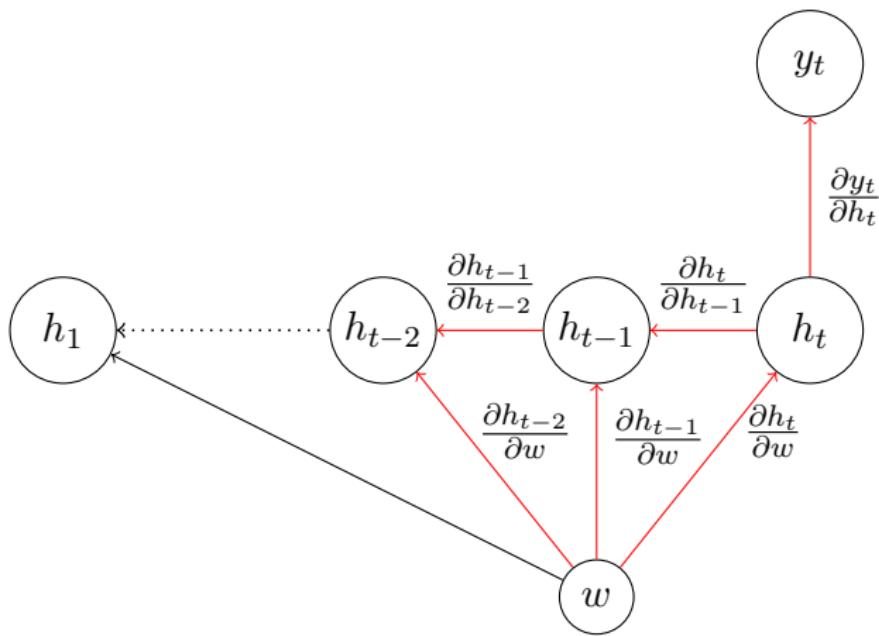
Machine Memory



Machine Memory

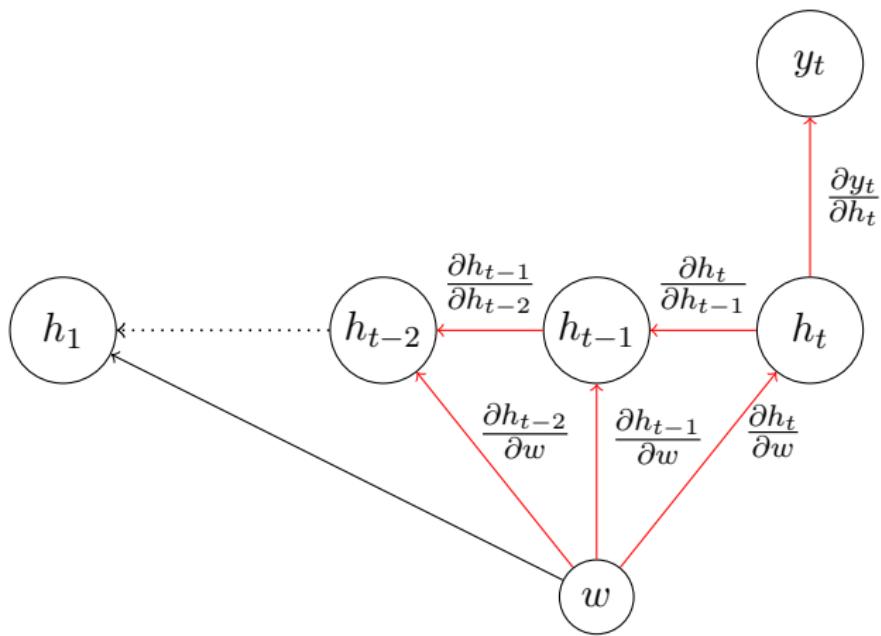


Machine Memory



$$0.1 \cdot 0.1 \cdot 0.1 \cdots 0.1$$

Machine Memory



$$0.1 \cdot 0.1 \cdot 0.1 \cdots 0.1 \approx 0$$

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 $G_i(x, h_{t-1}) = \sigma(\mathbf{w}_i^\top [x, h_{t-1}] + b_i)$

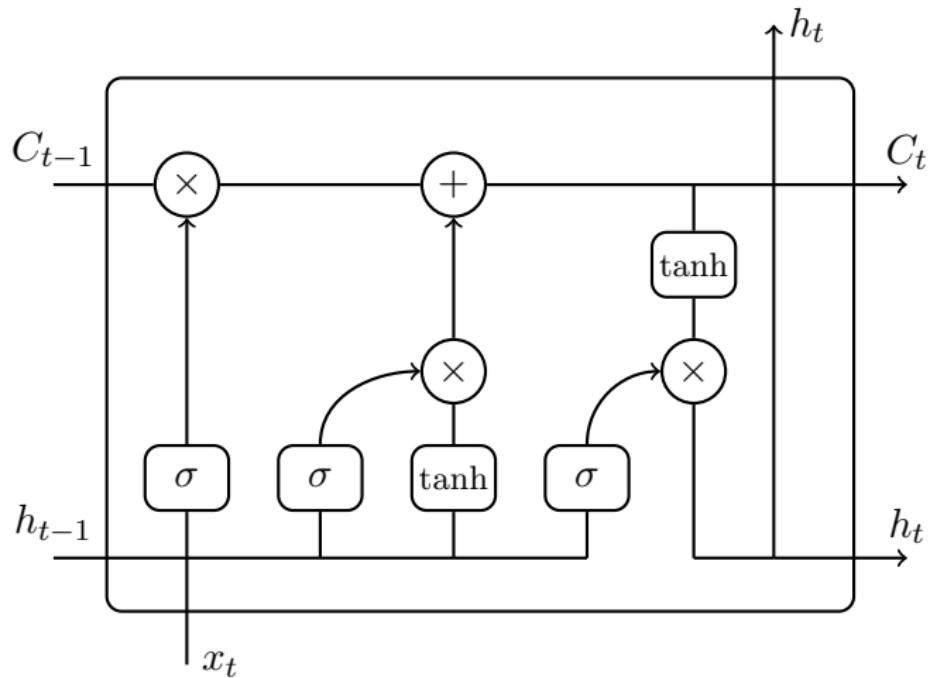
LSTMs

- ▶ 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 $G_f(x, h_{t-1}) = \sigma(\mathbf{w}_f^\top [x, h_{t-1}] + b_f)$

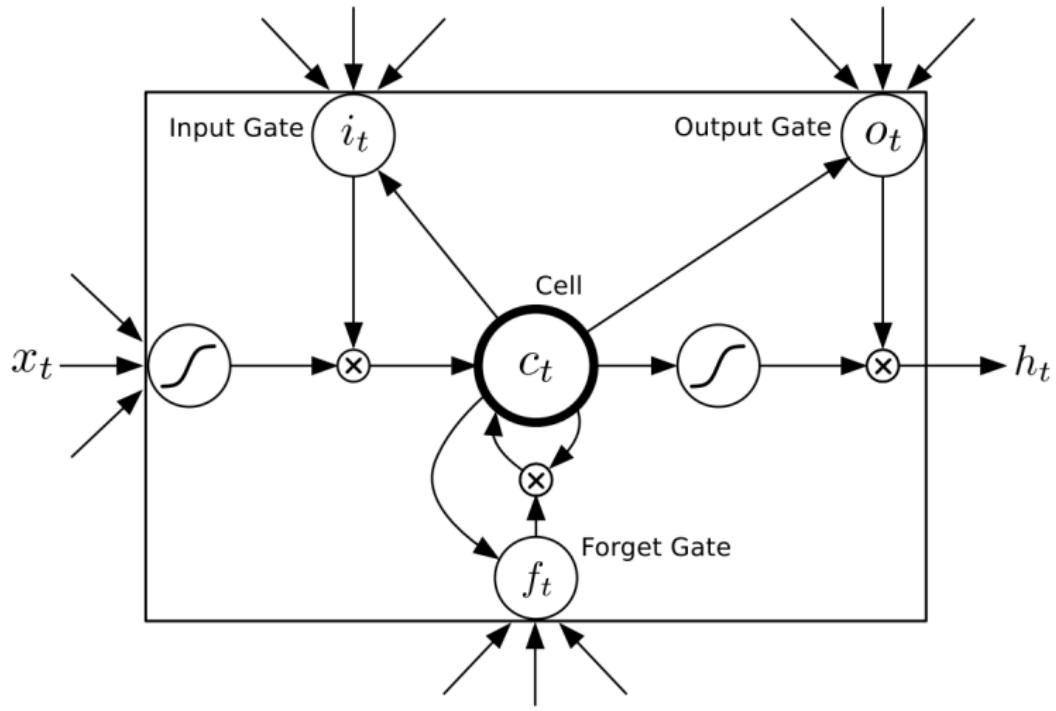
LSTMs

- ▶ 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-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh
eoase rrranbyne 'nhthnee e plia tkIrgd t o idoe ns,smtt h
ne etie h,hregtrs nigtike,aoaenns Ing*

Iteration: 100

[Kar16a]

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

[Kar16a]

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 oft paitt thin wall. Which das stimm*

Iteration: 500

[Kar16a]

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

[Kar16a]

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

[Kar16a]

LSTMs: What can they do?

They can write Linux kernel code!

Deep Learning

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

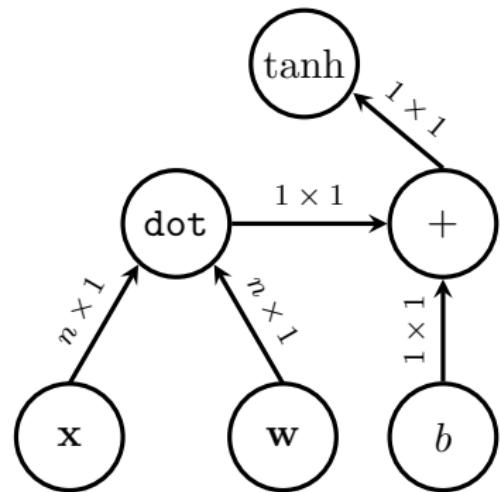
The Ugly

TensorFlow

Feel the TensorFlow

Computational Paradigms

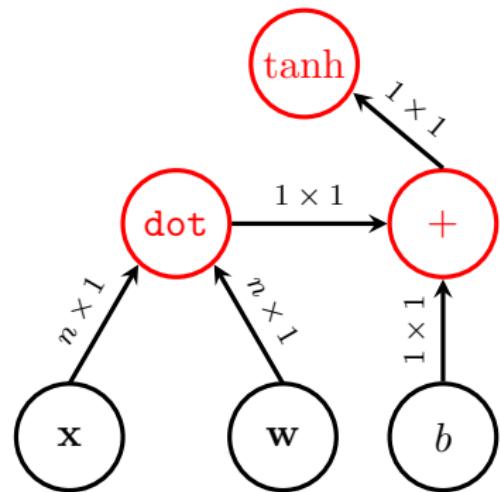
Computational Paradigms



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

Computational Graphs

Computational Paradigms

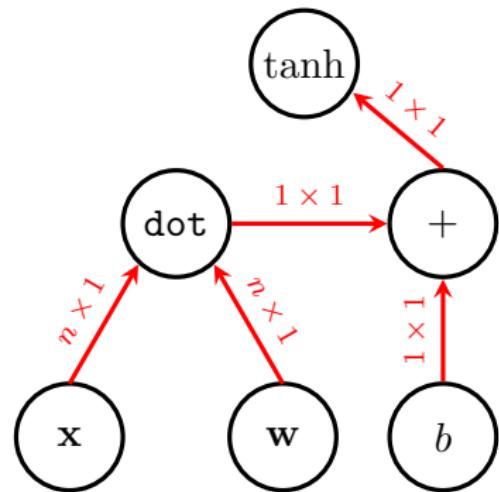


$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

1. Operations

Computational Paradigms

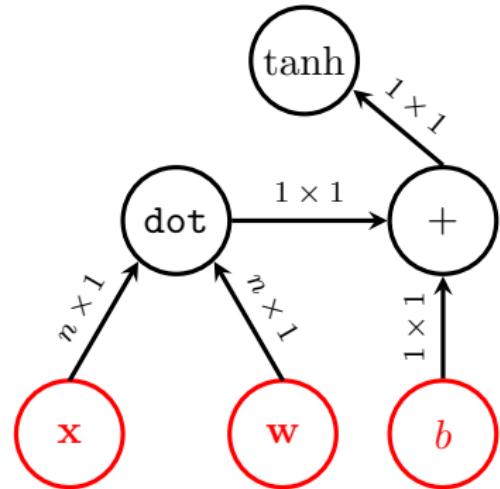


$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

1. Operations
2. Tensors

Computational Paradigms

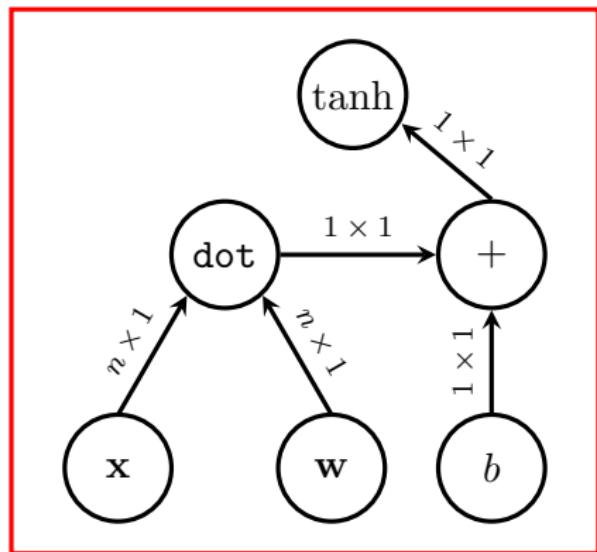


$$\hat{y} = \tanh(\mathbf{x}^\top \mathbf{w} + b)$$

Computational Graphs

1. Operations
2. Tensors
3. Variables

Computational Paradigms



Computational Graphs

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

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

Execution Model

Execution Model

Actors

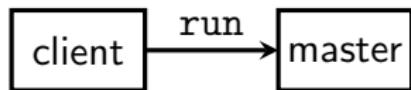
Execution Model



Actors

1. Client

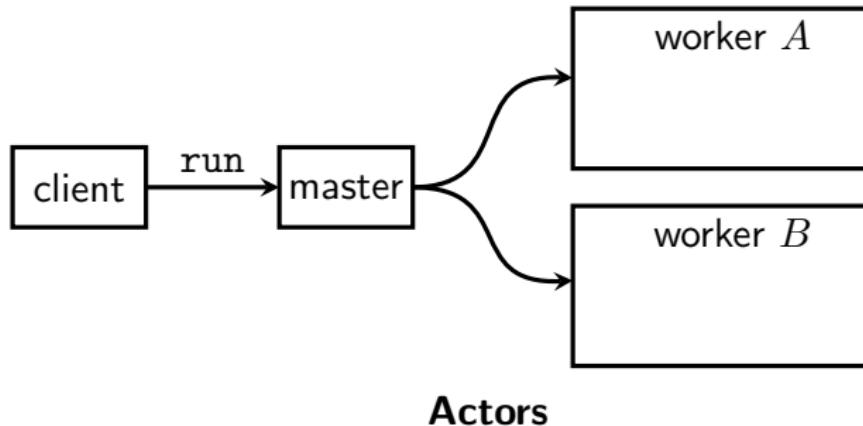
Execution Model



Actors

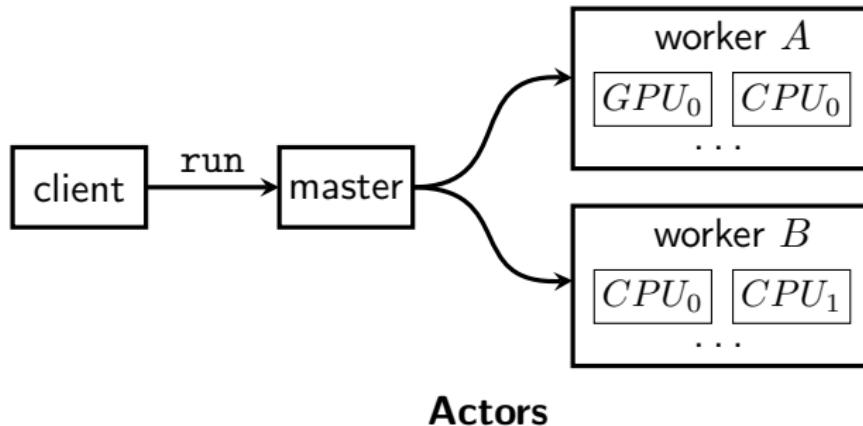
1. Client
2. Master

Execution Model



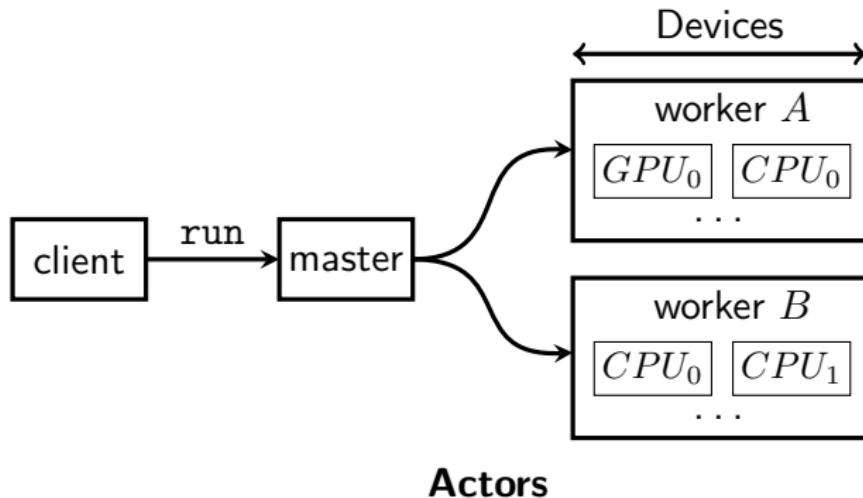
1. Client
2. Master
3. Workers

Execution Model



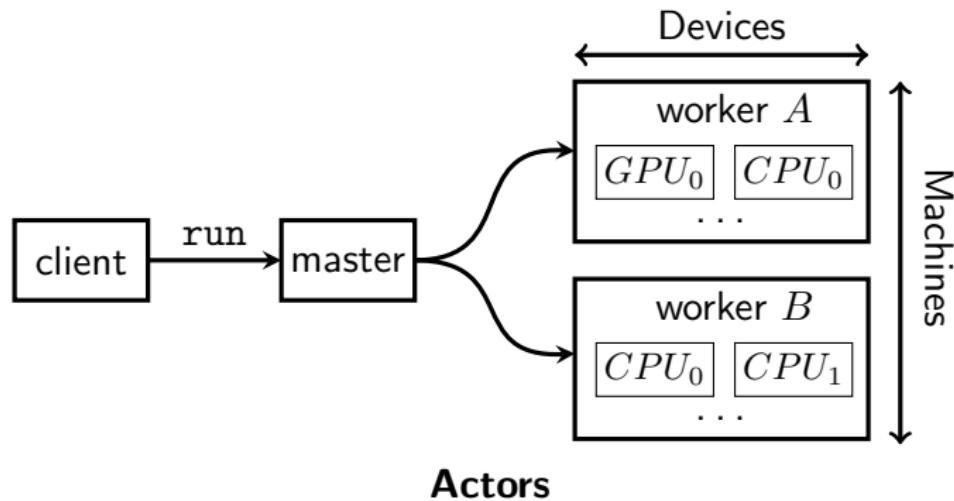
1. Client
2. Master
3. Workers
4. Devices

Execution Model



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

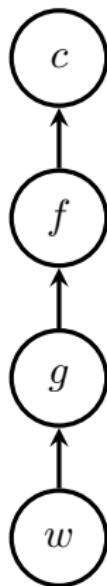
Execution Model



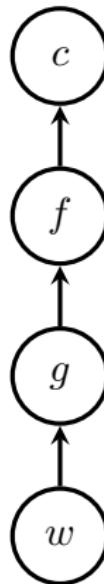
1. Client
2. Master
3. Workers
4. Devices

Back Propagation

Back Propagation



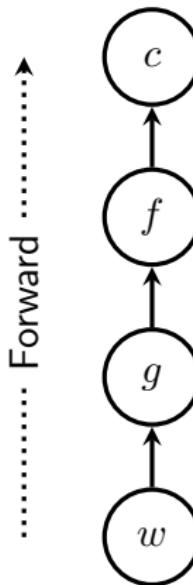
Back Propagation



Symbol to Number Differentiation

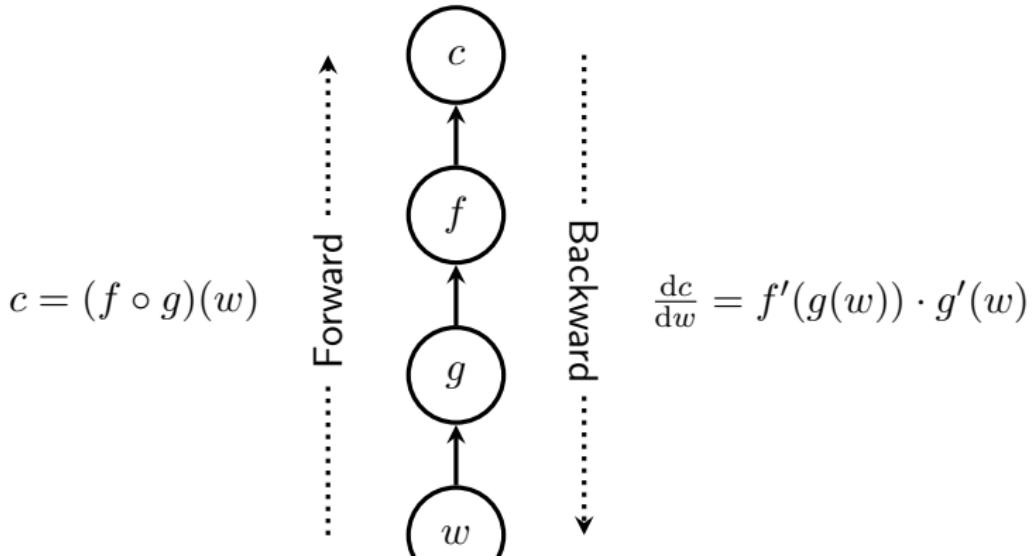
Back Propagation

$$c = (f \circ g)(w)$$



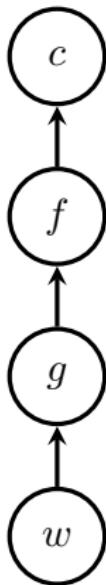
Symbol to Number Differentiation

Back Propagation



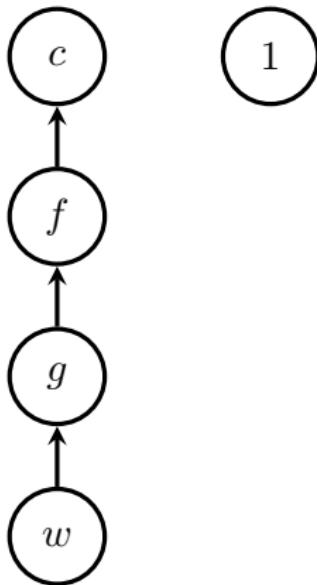
Symbol to Number Differentiation

Back Propagation



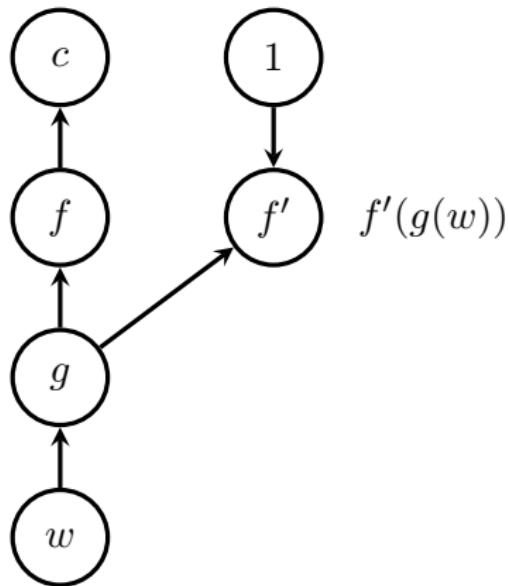
Symbol to Symbol Differentiation

Back Propagation



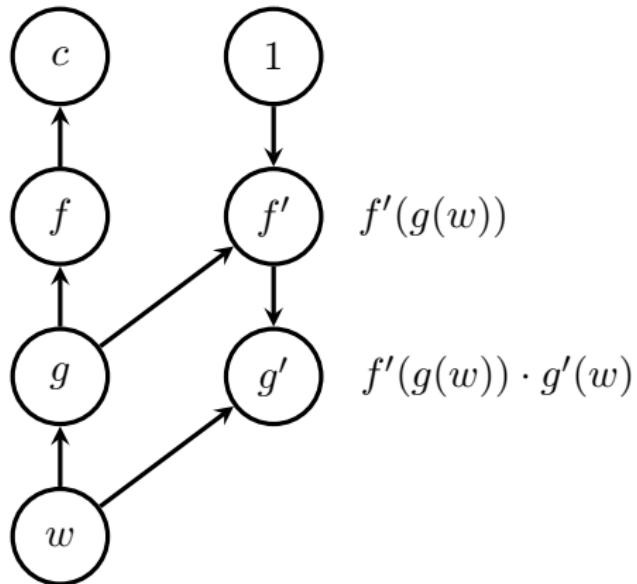
Symbol to Symbol Differentiation

Back Propagation



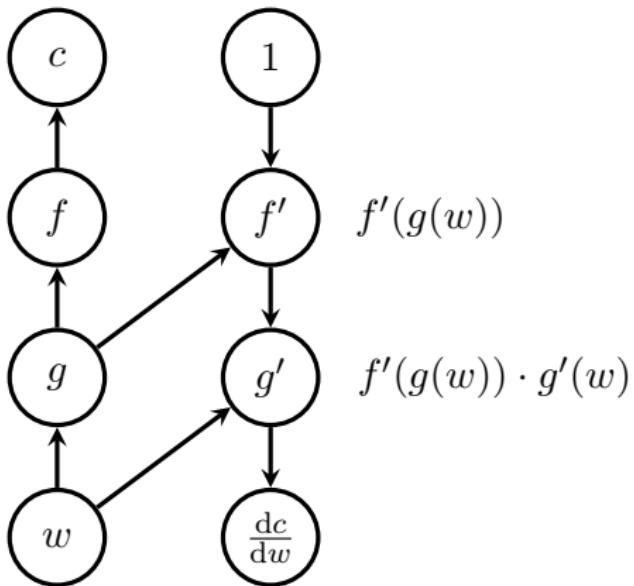
Symbol to Symbol Differentiation

Back Propagation



Symbol to Symbol Differentiation

Back Propagation



Symbol to Symbol Differentiation

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

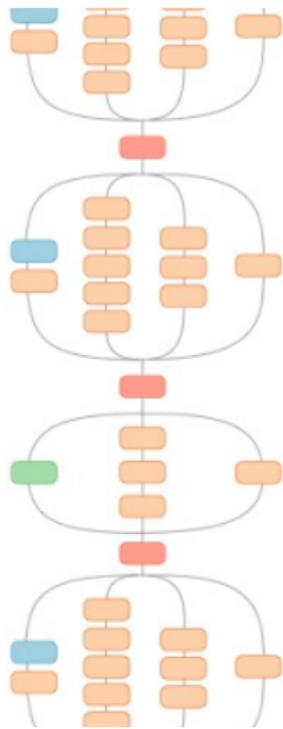
Visualization Tools

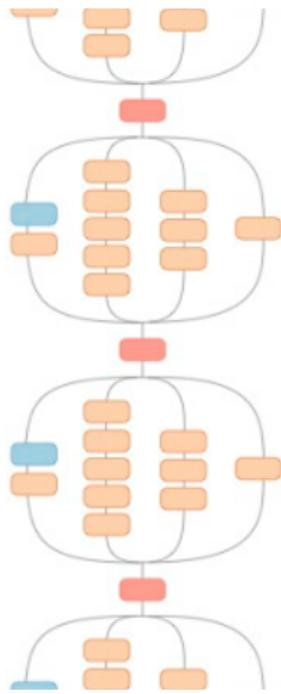
- ▶ Deep Neural Networks have the tendency of being . . . deep
- ▶ Easy to drown in the complexity of an architecture
- ▶ > 36,000 nodes for Google's *Inception* model

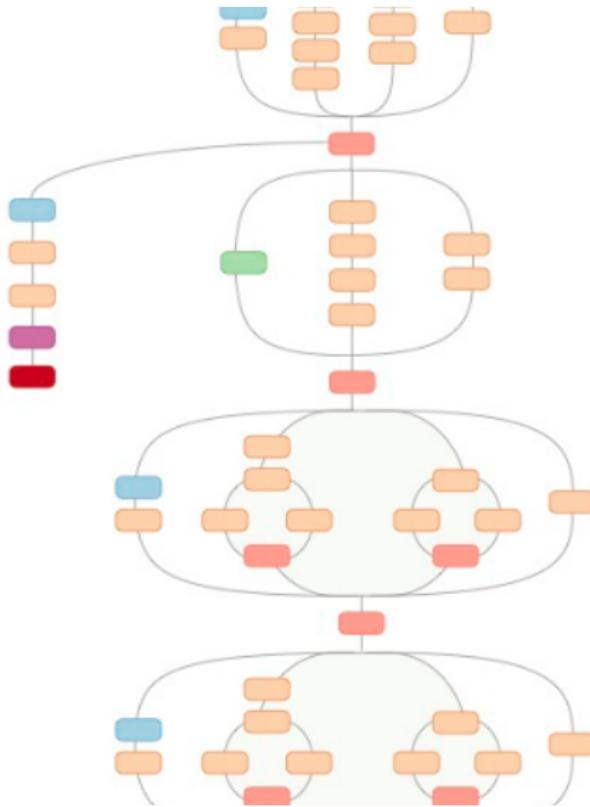
Visualization Tools

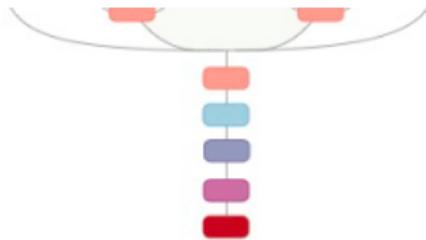
- Deep Neural Networks have the tendency of being . . . deep
- Easy to drown in the complexity of an architecture
- > 36,000 nodes for Google's *Inception* model



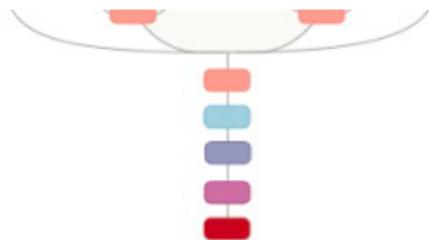








Source: <http://googleresearch.blogspot.de/2016/03/train-your-own-image-classifier-with.html>



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

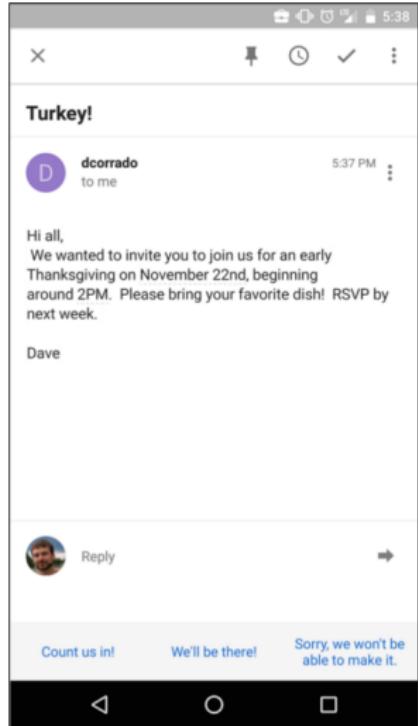
TensorBoard to the Rescue

Use Cases

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

Use Cases

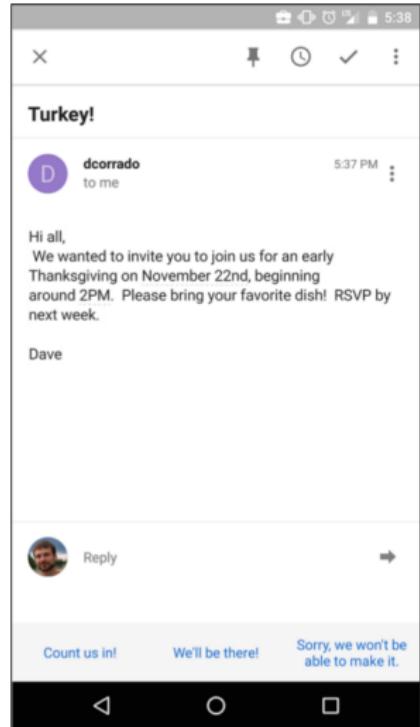
- ▶ Smart email replies in Google *Inbox*



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

Use Cases

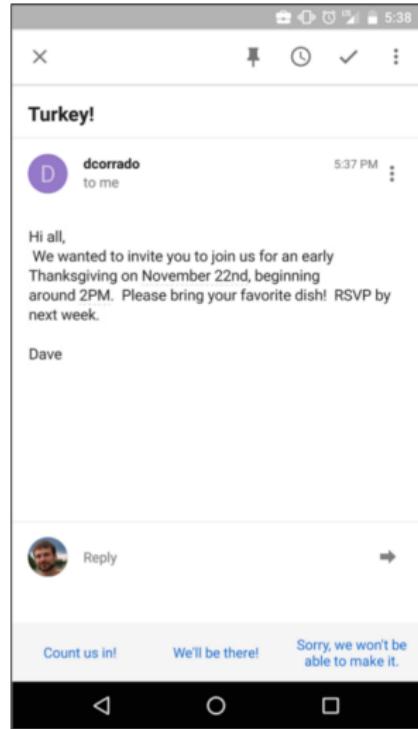
- ▶ Smart email replies in Google *Inbox*
- ▶ Emails mapped to “thought vectors”



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>

Deep Learning in Action #4

ACM Munich Student Chapter

Deep Learning in Action #4

ACM Munich Student Chapter

- Monday, August 1, 2016

Deep Learning in Action #4

ACM Munich Student Chapter

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

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](https://twitter.com/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>