Introduction to Machine Learning feat. TensorFlow

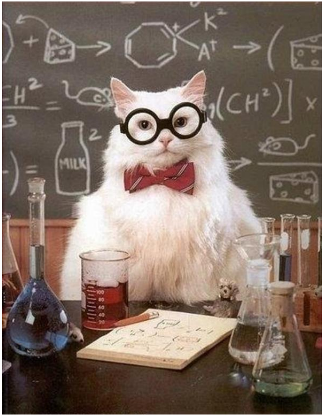
Peter Goldsborough

July 9, 2016

July 9, 2016

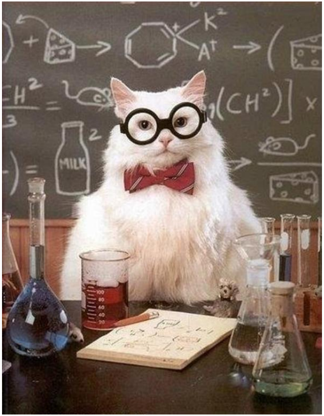
Table of Catents

Table of Catents

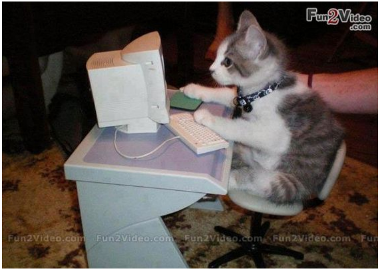


Theory

Table of Catents



Theory



Practice

Background

Background

- CS Student @ TUM

Background

- CS Student @ TUM
- Google & Bloomberg Intern

Background

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

Background

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

Seminar Topic: *Deep Learning With TensorFlow*

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

What is Machine Learning?

What is Machine Learning?

(and can I eat it?)

Definition I

Definition I

Machine Learning is cool.

Definition II

Machine Learning is *really* cool.

Definition III

Definition III

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

Definition III

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

What it does is get more from less.

Definition III

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

What it does is get more from less.

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

[**?**]

Definition IV

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

[**?**]

Definition III

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

Definition III

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

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

The Task, T

The Task, T

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

The Task, T

- Discriminate by the way an algorithm processes an example $\mathbf{x} \in \mathbb{R}^n$
- $\mathbf{x}$ contains features, such as (lunch, dinner)

The Task, T

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

The Task, T

Classification

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

The Task, T

Classification

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

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

The Task, T

Classification

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

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

The Task, T

Classification

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

- Categorize an input **x** into one of k classes
- Image classification
- Recognizing handwritten digits

The Task, T

Classification

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

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

The Task, T

Regression

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

The Task, T

Regression

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

- Predict a scalar value given input features

The Task, T

Regression

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

- Predict a scalar value given input features
- Algorithmic Trading

The Task, T

Regression

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

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

The Task, T

Regression

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

- 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 : Language $\rightarrow$ Text

The Task, T

Transcription

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

- Transform language from an unstructured representation into text

The Task, T

Transcription

f : Language $\rightarrow$ Text

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

The Task, T

Transcription

f : Language $\rightarrow$ 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 : Language $\rightarrow$ 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 : Stuff $\rightarrow$ 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 : Stuff $\rightarrow$ More Stuff

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

The Task, T

Synthesis

f : Stuff $\rightarrow$ 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 to train our algorithm to generate more data to train our algorithm to generate more data to train our algorithm to generate more data ...

How do we make our algorithm learn?

How do we make our algorithm learn?

Unsupervised Learning

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

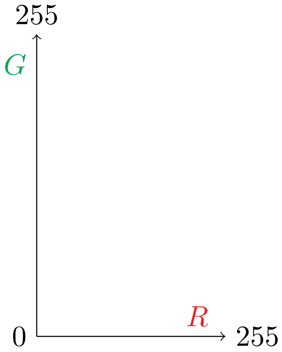
Methods: Features

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

$$0 \xrightarrow{R} 255$$

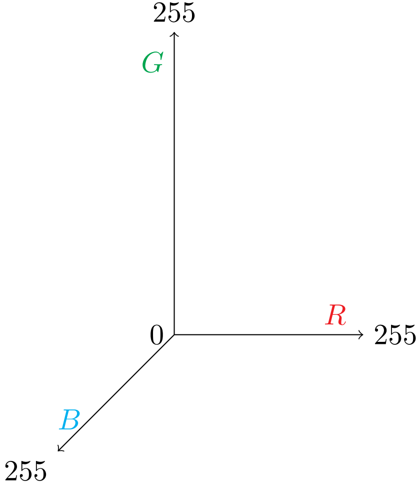
Methods: Features

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



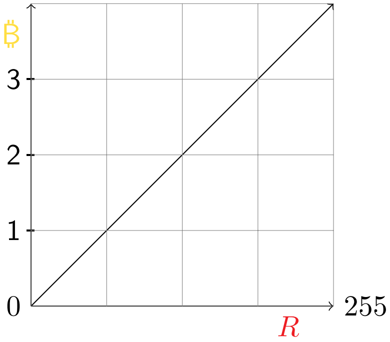
Methods: Features

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



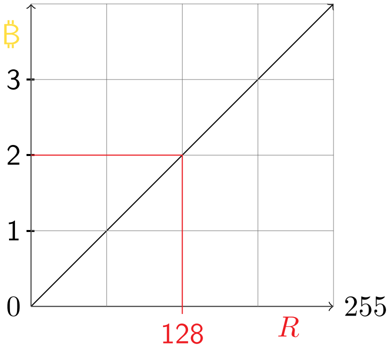
Methods: Features

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



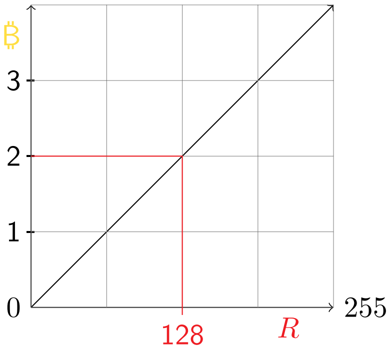
Methods: Features

- 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 $\mathbf{D} \in \mathbb{R}^{n \times k}$

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

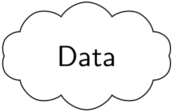
Design Matrix

Methods: Data

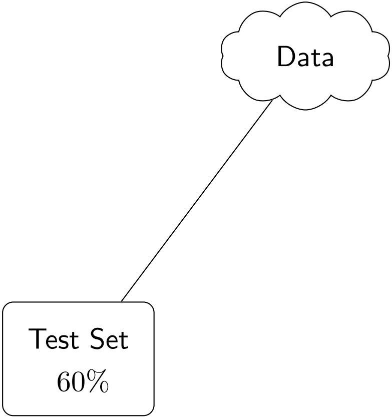
- To make our algorithm learn, we need to feed it (lots of) data
- In practice, we organize our data into a matrix $\mathbf{D} \in \mathbb{R}^{n \times k}$

```
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
```

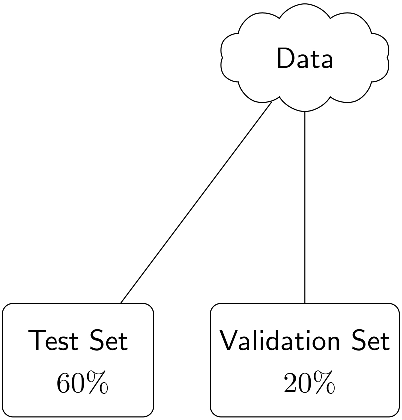
Methods: Data



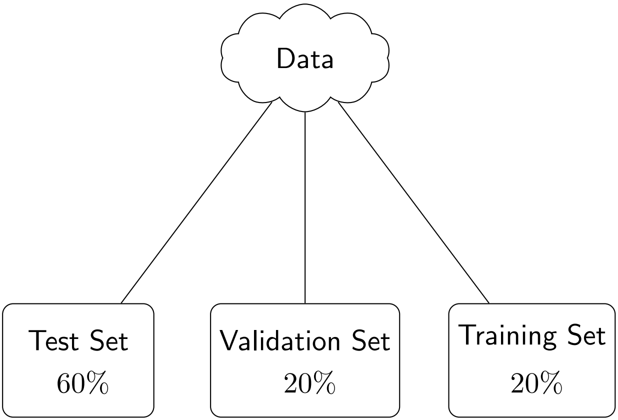
Methods: Data



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

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

Methods: Measuring Performance

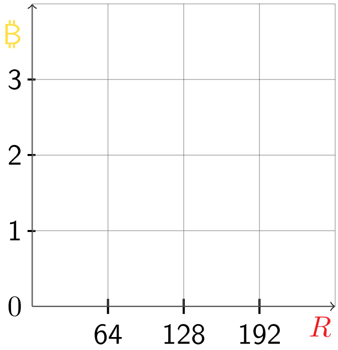
Regression

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

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

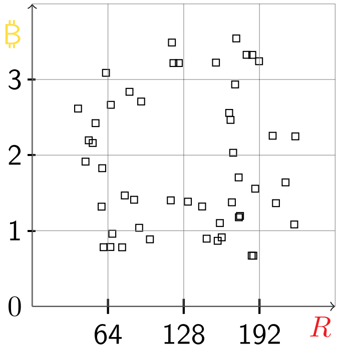
Methods: Measuring Performance

Regression



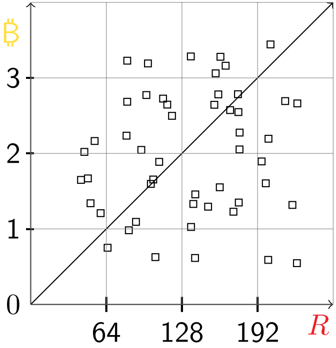
Methods: Measuring Performance

Regression



Methods: Measuring Performance

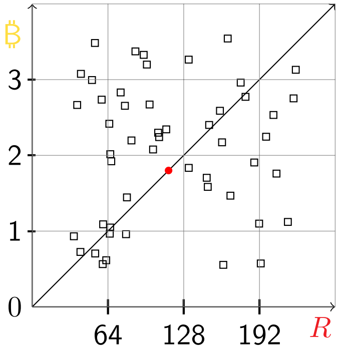
Regression



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

Methods: Measuring Performance

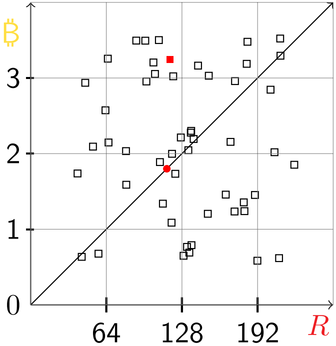
Regression



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

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

Methods: Weight Update

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

Methods: Weight Update

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

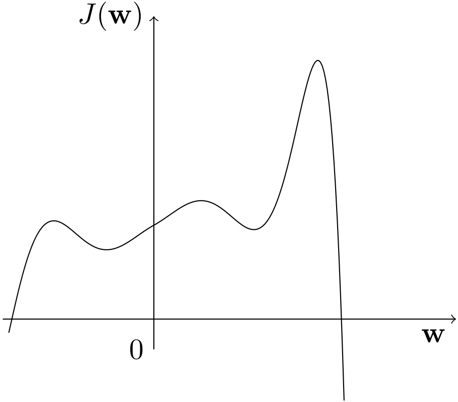
Methods: Weight Update

- $L(\mathbf{y}, \hat{\mathbf{y}})$ 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:

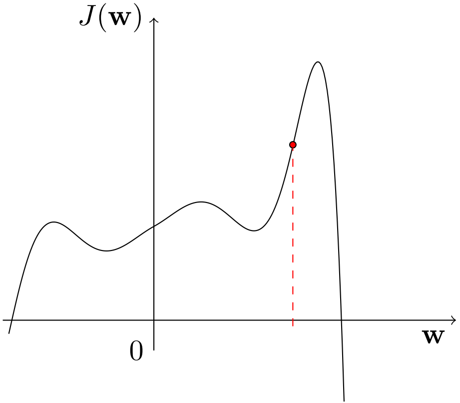
We can think of $\mathbf{Y}, \hat{\mathbf{Y}}$ as being *constant* and write:

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

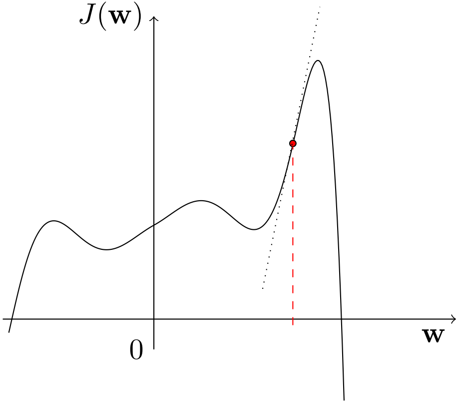
Methods: Weight Update



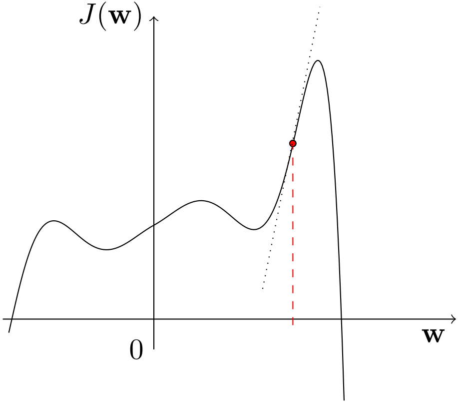
Methods: Weight Update



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,

$$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x}$$

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

Digression: Calculus Recap

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

  $$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x}$$

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

Digression: Calculus Recap

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

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

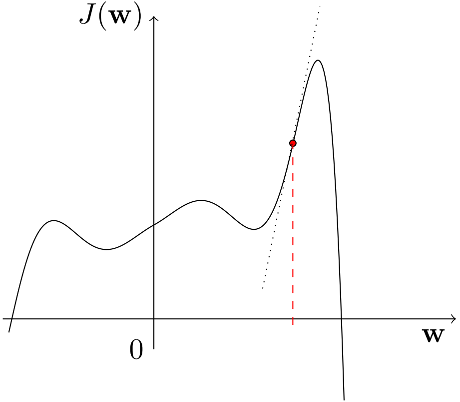
  is a partial derivative, and
- ∇f the vector containing the partial derivatives of w:

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

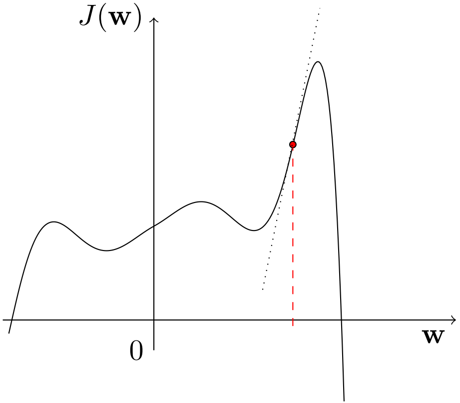
Digression: Calculus Recap

- Let $f : \mathbb{R}^n \to \mathbb{R}$ 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:
$$\nabla f = \left(\frac{\partial f}{\partial v_1}, \ldots, \frac{\partial f}{\partial v_n} \right)^\top$$
- ∇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

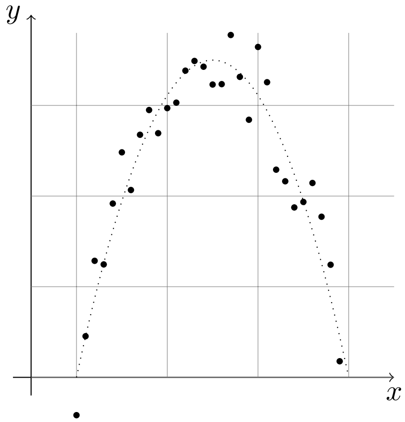
Methods: Regularization

- 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

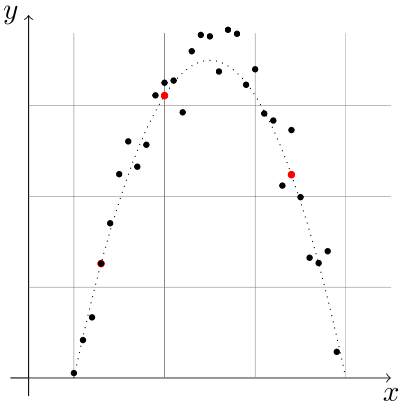
Methods: Regularization

- 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*

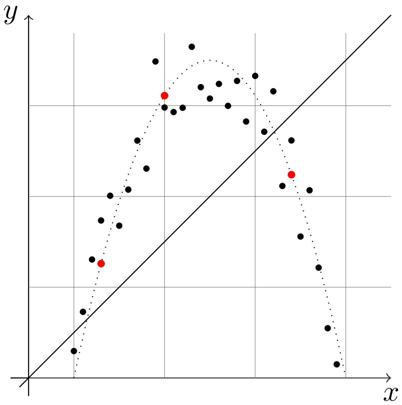
Concepts: Model Capacity



Concepts: Model Capacity

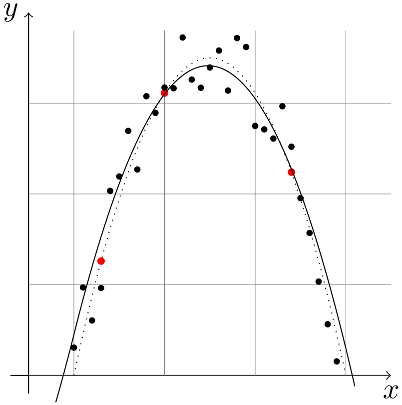


Concepts: Model Capacity



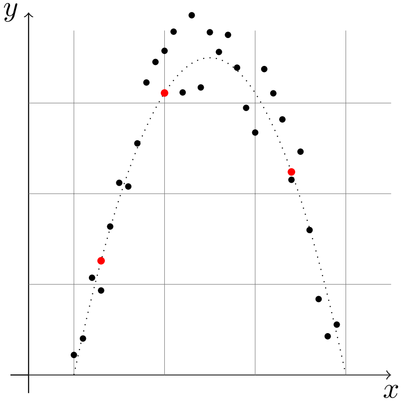
Underfitting (low capacity)

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

References