Regression and Classification

Deep Learning Lecture 2

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2017

Table of Contents

Logistics

- If you are auditing, please drop me an email again. So that I can put you in the mailing list for any info

Logistics

- If you are auditing, please drop me an email again. So that I can put you in the mailing list for any info
- I will put up the programming assignment like today. It is basically from the assignment of Stanford CS231n last year. You don't need to finish the complete assignment. Just need to finish the softmax (Q.3) and the neural network (Q.4) parts

Logistics

- If you are auditing, please drop me an email again. So that I can put you in the mailing list for any info
- I will put up the programming assignment like today. It is basically from the assignment of Stanford CS231n last year. You don't need to finish the complete assignment. Just need to finish the softmax (Q.3) and the neural network (Q.4) parts
  - The assignment will be due 2 weeks from now
  - There is a 3% (of everything) bonus for the first one to complete and *work perfectly*

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M
  - If $B = A^T$, $b_{ij} = a_{ji}$
- Quiz: for a n-dim vector $\mathbf{v}$,
  - What is the dimension of $\mathbf{v}^T\mathbf{v}$?

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M
  - If $B = A^T$, $b_{ij} = a_{ji}$
- Quiz: for a n-dim vector $\mathbf{v}$,
  - What is the dimension of $\mathbf{v}^T\mathbf{v}$?
    - 1×1 (inner product)
  - What is the dimension of $\mathbf{v}\mathbf{v}^T$?

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M
  - If $B = A^T$, $b_{ij} = a_{ji}$
- Quiz: for a n-dim vector $\mathbf{v}$,
  - What is the dimension of $\mathbf{v}^T\mathbf{v}$?
    - 1×1 (inner product)
  - What is the dimension of $\mathbf{v}\mathbf{v}^T$?
    - $n \times n$ (outer product)

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

E.g., $f((x_1, x_2, x_3)) = (x_1 + 2)x_2^2 x_3$

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

E.g., $f((x_1, x_2, x_3)) = (x_1 + 2)x_2^2 x_3$

$$\nabla f(\mathbf{x}) = \begin{pmatrix} x_2^2 x_3 \\ 2(x_1 + 2)x_2 x_3 \\ (x_1 + 2)x_2^2 \end{pmatrix}$$

and $\nabla f(\mathbf{x})|_{(0,1,0)}$?

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

E.g., $f((x_1, x_2, x_3)) = (x_1 + 2)x_2^2 x_3$

$$\nabla f(\mathbf{x}) = \begin{pmatrix} x_2^2 x_3 \\ 2(x_1 + 2)x_2 x_3 \\ (x_1 + 2)x_2^2 \end{pmatrix}$$

and $\nabla f(\mathbf{x})|_{(0,1,0)}$?

$$\nabla f(\mathbf{x})|_{(0,1,0)} = (0, 0, 2)^T$$

Loss function for regression

Let' start with the regression problem. Recall from last lecture that

- We are trying to learn a function $f(x; W)$ such that for training input x_i and desired output y_i, $f(x_i; W) \sim y_i$

We can define a loss (aka cost, objective) function $L(\cdot, \cdot)$ to measure the discrepancy between the desired output and the actual output

Loss function for regression

Let' start with the regression problem. Recall from last lecture that

- We are trying to learn a function $f(x; W)$ such that for training input x_i and desired output y_i, $f(x_i; W) \sim y_i$

We can define a loss (aka cost, objective) function $L(\cdot, \cdot)$ to measure the discrepancy between the desired output and the actual output

- During training, a reasonable goal will simply be to

$$\min_W \sum_i L(f(x_i; W), y_i),$$

where in the objective function, we are summing the corresponding loss over all pair of training data

Loss function for regression

Let' start with the regression problem. Recall from last lecture that

- We are trying to learn a function $f(x; W)$ such that for training input x_i and desired output y_i, $f(x_i; W) \sim y_i$

We can define a loss (aka cost, objective) function $L(\cdot, \cdot)$ to measure the discrepancy between the desired output and the actual output

- During training, a reasonable goal will simply be to

$$\min_W \sum_i L(f(x_i; W), y_i),$$

  where in the objective function, we are summing the corresponding loss over all pair of training data

- For regression, it is common to use mean square error for loss function, i.e., $l(f(x_i; W), y_i) = (f(x_i; W) - y_i)^2$

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age =29, what is his mass?

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age =29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T \mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age =29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T \mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight
- Given training data, we need to find $\mathbf{w}$
  - $\mathbf{x}_1 = (1.68, 31.80, 43.34, 1)^T$, $y_1 = 87.50$
  - $\mathbf{x}_2 = (1.80, 33.11, 16.69, 1)^T$, $y_2 = 110.06$
  - $\cdots$
  - $\mathbf{x}_N = (1.83, 33.79, 43.30, 1)^T$, $y_N = 112.33$

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age =29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T \mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight
- Given training data, we need to find $\mathbf{w}$
  - $\mathbf{x}_1 = (1.68, 31.80, 43.34, 1)^T$, $y_1 = 87.50$
  - $\mathbf{x}_2 = (1.80, 33.11, 16.69, 1)^T$, $y_2 = 110.06$
  - $\cdots$
  - $\mathbf{x}_N = (1.83, 33.79, 43.30, 1)^T$, $y_N = 112.33$
- Write $X_{train} = \left(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\right)$ and $\mathbf{y}_{train} = (y_1, y_2, \cdots, y_N)^T$, we want

$$\mathbf{y}_{train} \sim X_{train}^T \mathbf{w}$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$L(\mathbf{w})$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$\begin{aligned}
&L(\mathbf{w})\\
=&\frac{1}{2}(\mathbf{y}_{train} - X_{train}^T\mathbf{w})^T(\mathbf{y}_{train} - X_{train}^T\mathbf{w})
\end{aligned}$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$
\begin{aligned}
& L(\mathbf{w}) \\
= & \frac{1}{2}(\mathbf{y}_{train} - X_{train}^T \mathbf{w})^T (\mathbf{y}_{train} - X_{train}^T \mathbf{w}) \\
= & \frac{1}{2}\left(\mathbf{y}_{train}^T \mathbf{y}_{train} - \mathbf{w}^T X_{train} \mathbf{y}_{train} - \mathbf{y}_{train}^T X_{train}^T \mathbf{w} + \mathbf{w} X_{train} X_{train}^T \mathbf{w}\right)
\end{aligned}
$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$
\begin{aligned}
&L(\mathbf{w}) \\
=&\frac{1}{2}(\mathbf{y}_{train} - X_{train}^T\mathbf{w})^T(\mathbf{y}_{train} - X_{train}^T\mathbf{w}) \\
=&\frac{1}{2}\left(\mathbf{y}_{train}^T\mathbf{y}_{train} - \mathbf{w}^T X_{train}\mathbf{y}_{train} - \mathbf{y}_{train}^T X_{train}^T\mathbf{w} + \mathbf{w}X_{train}X_{train}^T\mathbf{w}\right)
\end{aligned}
$$

- Let's compute the gradient (try it out at home),

$$
\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train}\mathbf{y}_{train} + X_{train}X_{train}^T\mathbf{w}
$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$
\begin{aligned}
&L(\mathbf{w})\\
=&\frac{1}{2}(\mathbf{y}_{train} - X_{train}^T\mathbf{w})^T(\mathbf{y}_{train} - X_{train}^T\mathbf{w})\\
=&\frac{1}{2}\left(\mathbf{y}_{train}^T\mathbf{y}_{train} - \mathbf{w}^T X_{train}\mathbf{y}_{train} - \mathbf{y}_{train}^T X_{train}^T\mathbf{w} + \mathbf{w}X_{train}X_{train}^T\mathbf{w}\right)
\end{aligned}
$$

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train}\mathbf{y}_{train} + X_{train}X_{train}^T\mathbf{w}$$

- At the optimum, we expect $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$. Thus

$$\mathbf{w} = \underbrace{(X_{train}X_{train}^T)^{-1}X_{train}}\mathbf{y}_{train}$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$
\begin{aligned}
&L(\mathbf{w})\\
=&\frac{1}{2}(\mathbf{y}_{train} - X_{train}^T\mathbf{w})^T(\mathbf{y}_{train} - X_{train}^T\mathbf{w})\\
=&\frac{1}{2}\left(\mathbf{y}_{train}^T\mathbf{y}_{train} - \mathbf{w}^T X_{train}\mathbf{y}_{train} - \mathbf{y}_{train}^T X_{train}^T\mathbf{w} + \mathbf{w}X_{train}X_{train}^T\mathbf{w}\right)
\end{aligned}
$$

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train}\mathbf{y}_{train} + X_{train}X_{train}^T\mathbf{w}$$

- At the optimum, we expect $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$. Thus

$$\mathbf{w} = \underbrace{(X_{train}X_{train}^T)^{-1}X_{train}}_{(X_{train}^T)^\dagger}\mathbf{y}_{train}$$

Experiment

- mass $=$ bmi $\times$ height^2

Experiment

- mass $=$ bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg

Experiment

- mass $=$ bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg
- Trained weights: (1.17e+02, 3.11, 8.97e-03, -2.05e+02) # (height,bmi,age,1)

Experiment

- mass $=$ bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg
- Trained weights: (1.17e+02, 3.11, 8.97e-03, -2.05e+02) # (height,bmi,age,1)
- The weights are quite reasonable
  - mass should not really depend on age
  - height should have a stronger effect to mass than bmi

Experiment

- mass = bmi × height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg
- Trained weights: (1.17e+02, 3.11, 8.97e-03, -2.05e+02) # (height,bmi,age,1)
- The weights are quite reasonable
  - mass should not really depend on age
  - height should have a stronger effect to mass than bmi
- MSE: 6.63. It is a bit high, let's try to reduce it

Expanding features...

- Let's include some higher "order" features. For the raw feature x_1, x_2, x_3, we can also include products of them as a feature. So a new feature vector becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3),$$

where the function of mapping from the raw feature to the new feature vector is sometimes known as the basis or kernel function

Expanding features...

- Let's include some higher "order" features. For the raw feature x_1, x_2, x_3, we can also include products of them as a feature. So a new feature vector becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3),$$

  where the function of mapping from the raw feature to the new feature vector is sometimes known as the basis or kernel function
- We can do linear regression just as before, just the number of weights increases from 4 to 10

Expanding features...

- Let's include some higher "order" features. For the raw feature x_1, x_2, x_3, we can also include products of them as a feature. So a new feature vector becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3),$$

  where the function of mapping from the raw feature to the new feature vector is sometimes known as the basis or kernel function
- We can do linear regression just as before, just the number of weights increases from 4 to 10
- MSE: 1.01. Nice!

Expanding features (con't)...

- Let's go even higher order and also include products like $x_1 x_2 x_3$ and $x_1^2 x_2$. So the new feature vector now becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1 x_2, x_1 x_3, x_2 x_3, x_1^3, x_2^3, x_3^3, x_1^2 x_2, \cdots)$$

Expanding features (con't)...

- Let's go even higher order and also include products like $x_1x_2x_3$ and $x_1^2x_2$. So the new feature vector now becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3, x_1^3, x_2^3, x_3^3, x_1^2x_2, \cdots)$$

- Again we will do linear regression as before, the number of weights now increases from to 25

Expanding features (con't)...

- Let's go even higher order and also include products like $x_1x_2x_3$ and $x_1^2x_2$. So the new feature vector now becomes

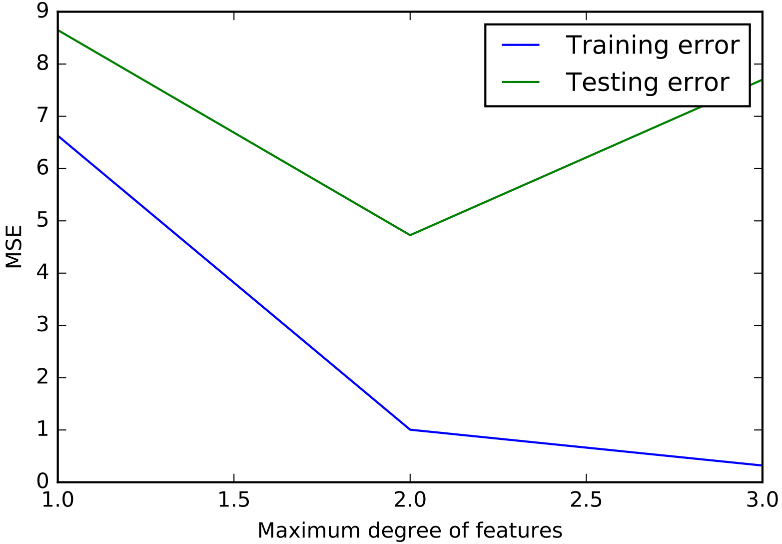
$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3, x_1^3, x_2^3, x_3^3, x_1^2x_2, \cdots)$$

- Again we will do linear regression as before, the number of weights now increases from to 25
- MSE: 0.32...

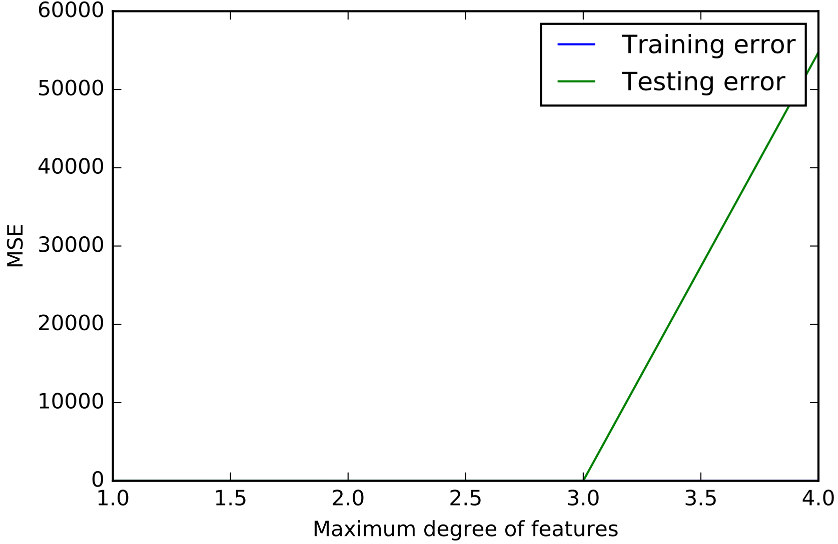
Expanding features (con't)...

- We can go further to the 4-th order and the number of weights now increases to 70
- MSE: 1.13e-12. Wow!

Wait, how about testing error?



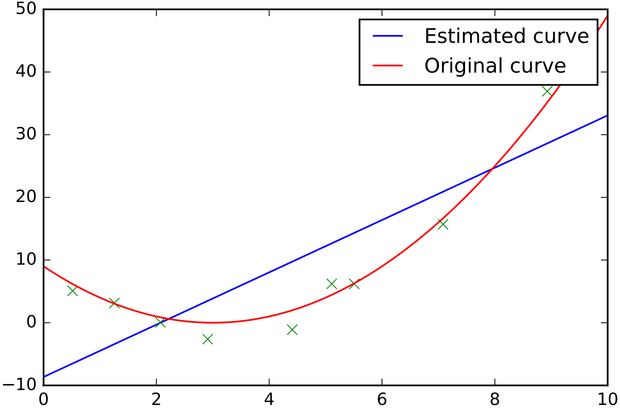
Wait, how about testing error...? Oops



Curve fitting

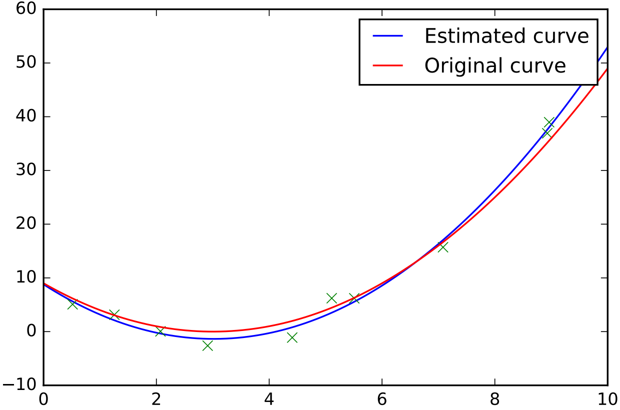
Why is it so bad for testing? Let's visit another even simpler example

- Let's try to fit a quadratic curve $y = (x-3)^2$ with linear regression. And again our training data will be wiggled a little bit by a Gaussian noise



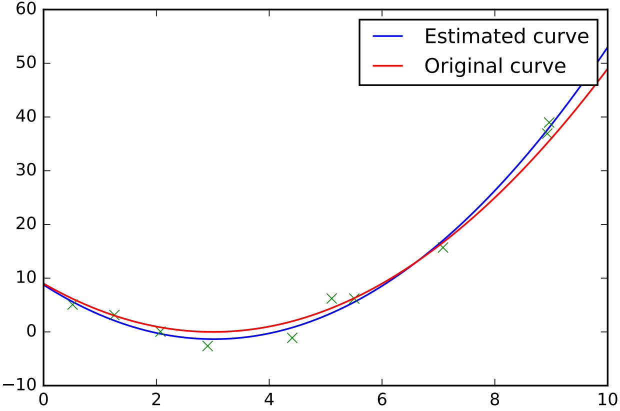
Curve fitting (2nd order)

Let's include higher order feature just as before. Take $(1, x, x^2)$ as feature by including x^2



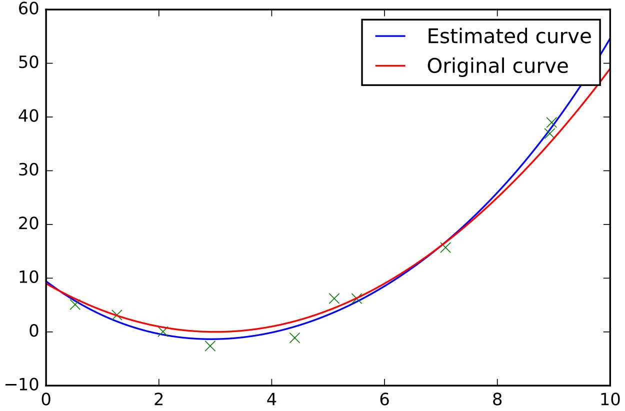
Curve fitting (3rd order)

$(1, x, x^2, x^3)$



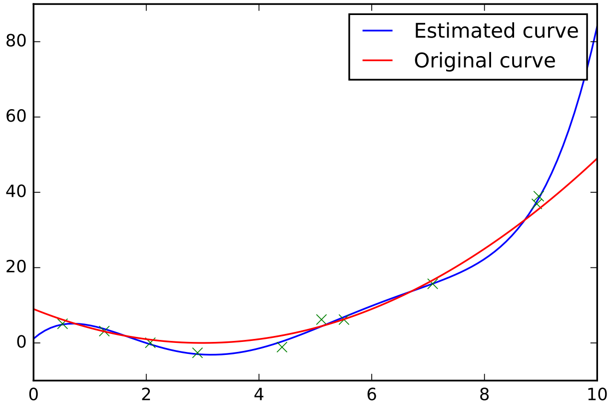
Curve fitting (4th order)

$(1, x, x^2, x^3, x^4)$



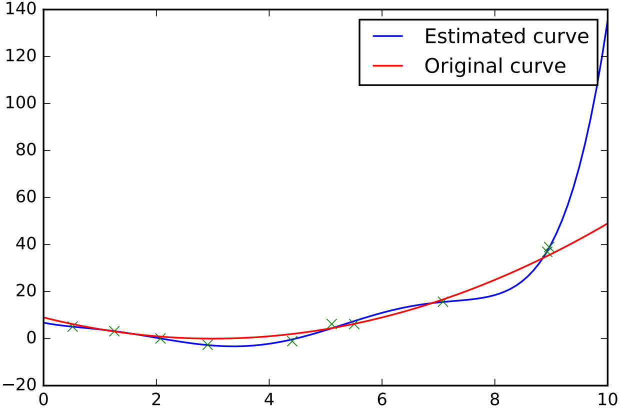
Curve fitting (5th order)

$(1, x, x^2, x^3, x^4, x^5)$



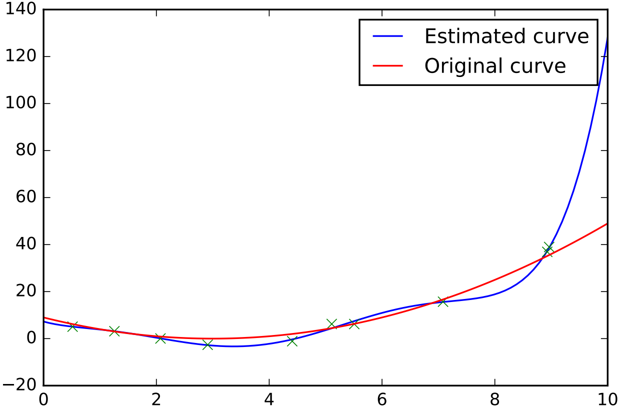
Curve fitting (6rd order)

$(1, x, x^2, x^3, x^4, x^5, x^6)$



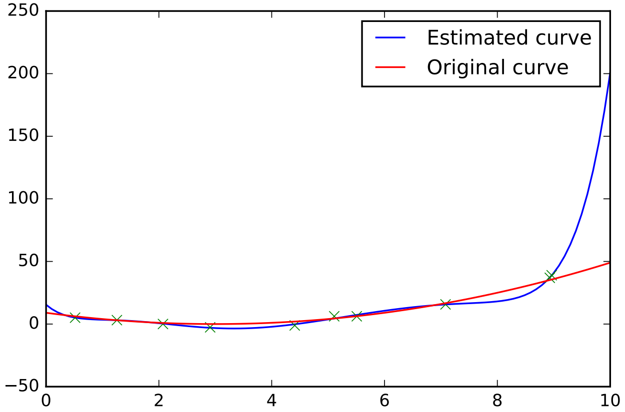
Curve fitting (7rd order)

$(1, x, x^2, x^3, x^4, x^5, x^6, x^7)$



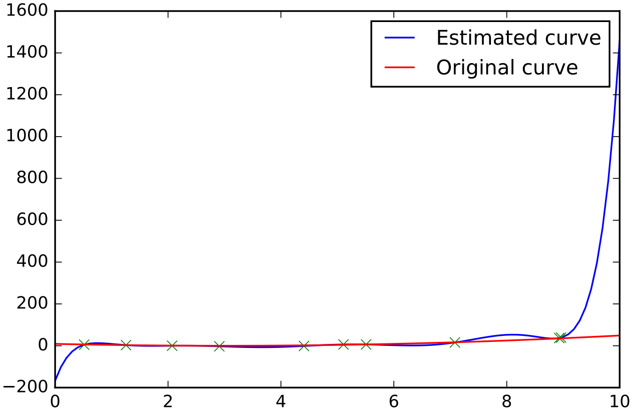
Curve fitting (8th order)

$(1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8)$

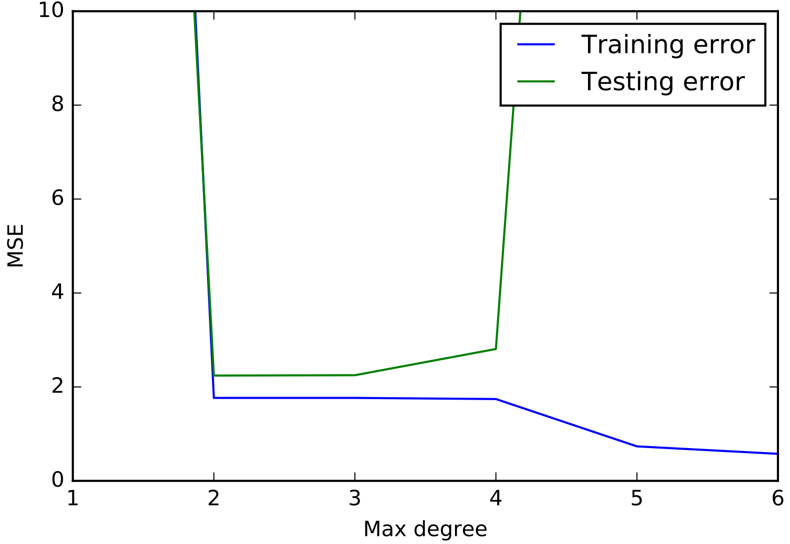


Curve fitting (9th order)

$$(1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9)$$



Overfitting vs underfitting



Lesson learned

- Given sufficiently complex model, we can learn “anything”, but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts

Lesson learned

- Given sufficiently complex model, we can learn "anything", but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...

Lesson learned

- Given sufficiently complex model, we can learn "anything", but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...
  - Unlike optimization, we don't actually know the true objective function
  - Loss function is just an approximated goal

Lesson learned

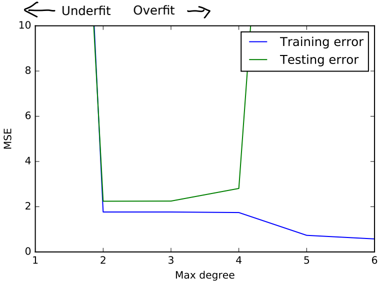
- Given sufficiently complex model, we can learn "anything", but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...
  - Unlike optimization, we don't actually know the true objective function
  - Loss function is just an approximated goal
- Should try to avoid neither overfitting nor underfitting

Lesson learned

- Given sufficiently complex model, we can learn "anything", but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...
  - Unlike optimization, we don't actually know the true objective function
  - Loss function is just an approximated goal
- Should try to avoid neither overfitting nor underfitting
  - Everything should be made as simple as possible, but not simpler – Albert Einstein
  - Occam's razor: overly complex model is not a good thing (if you don't have sufficient data to fit the model). Should shave out all redundancy (from the model)

High-bias vs high-variance

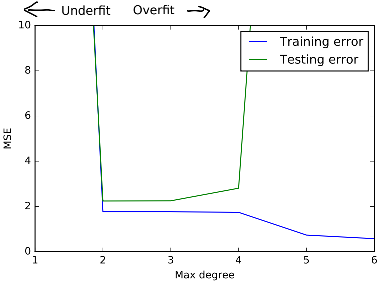
Sometimes we also refer to overfitting and underfitting roughly as high-variance and high-bias



High-bias vs high-variance

Sometimes we also refer to overfitting and underfitting roughly as high-variance and high-bias

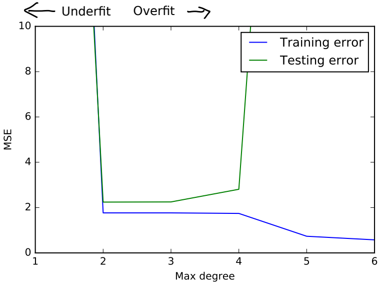
- High-bias: model is too rigid to learn (thus biased) and it cannot adapt to the data



High-bias vs high-variance

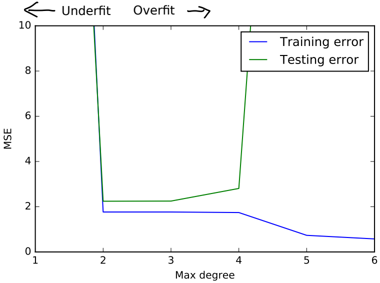
Sometimes we also refer to overfitting and underfitting roughly as high-variance and high-bias

- High-bias: model is too rigid to learn (thus biased) and it cannot adapt to the data
- High-variance: model is too elastic and can fit any arbitrary data. When fitted with different training data, the weights just converge to totally different values (thus high variance)



More on overfitting (high-variance)

- In the high-variance domain, the model is essentially learning the training data noise. That's why weights converge to different values for different training data
- Model complexity is relative. If more training data are available, the model used to be overfitted may not be overfitted anymore. So should we change a model every time we added new data?!



Regularization

Rather than using a simple model, we could restrain a more complex model from running wild with additional constraints. This process is commonly known as regularization

- As regularization can mitigate the overfitting problem, we can use a more expressive model even when we have only few data. And the same model can be used as data size increases
- A regularized complex model typically outperforms an unregularized simple model

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

- For example, in ridge regression, we try to achieve this by simply including $\frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$ in the loss objective function. Thus

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$$

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

- For example, in ridge regression, we try to achieve this by simply including $\frac{1}{2}\lambda \mathbf{w}^T \mathbf{w}$ in the loss objective function. Thus

$$
\begin{aligned}
L(\mathbf{w}) &= \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \frac{1}{2}\lambda \mathbf{w}^T\mathbf{w} \\
&= \frac{1}{2}\left(\mathbf{y}^T\mathbf{y} - \mathbf{w}^T X\mathbf{y} - \mathbf{y}^T X^T\mathbf{w} + \mathbf{w}[XX^T + \lambda I]\mathbf{w}\right)
\end{aligned}
$$

- And the gradient is

$$
\nabla_{\mathbf{w}} L(\mathbf{w}) = -X\mathbf{y} + [XX^T + \lambda I]\ \mathbf{w}
$$

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

- For example, in ridge regression, we try to achieve this by simply including $\frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$ in the loss objective function. Thus

$$\begin{aligned} L(\mathbf{w}) &= \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \frac{1}{2}\lambda \mathbf{w}^T\mathbf{w} \\ &= \frac{1}{2}\left(\mathbf{y}^T\mathbf{y} - \mathbf{w}^T X\mathbf{y} - \mathbf{y}^T X^T\mathbf{w} + \mathbf{w}[XX^T + \lambda I]\mathbf{w}\right) \end{aligned}$$

- And the gradient is

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X\mathbf{y} + [XX^T + \lambda I]\ \mathbf{w}$$

- As before, if we set $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$, we have

$$\mathbf{w} = [XX^T + \lambda I]^{-1} X\mathbf{y}$$

Lasso

- Another common regularization is lasso. Instead of $\lambda \mathbf{w}^T \mathbf{w}$, the scaled l_1-norm of $\mathbf{w}$, $\lambda \|\mathbf{w}\|_\mathbf{1}$ is added to the loss objective function Thus, we want to

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \lambda \|\mathbf{w}\|_1,$$

where $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_D|$

[1] The ridge regression function in the 0.18.1 version of sciki-learn appears to have bug. Both ridge regression and lasso function are implemented as lasso.

Lasso

- Another common regularization is lasso. Instead of $\lambda \mathbf{w}^T \mathbf{w}$, the scaled l_1-norm of $\mathbf{w}$, $\lambda \|\mathbf{w}\|_1$ is added to the loss objective function Thus, we want to

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \lambda \|\mathbf{w}\|_1,$$

  where $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_D|$
- Unlike ridge regression, one cannot write the close form solution directly though
  - We will discuss how the optimization is typically performed later
  - For the next several slides, I just used sciki-learn library[1] in Python

[1] The ridge regression function in the 0.18.1 version of sciki-learn appears to have bug. Both ridge regression and lasso function are implemented as lasso.

Lasso

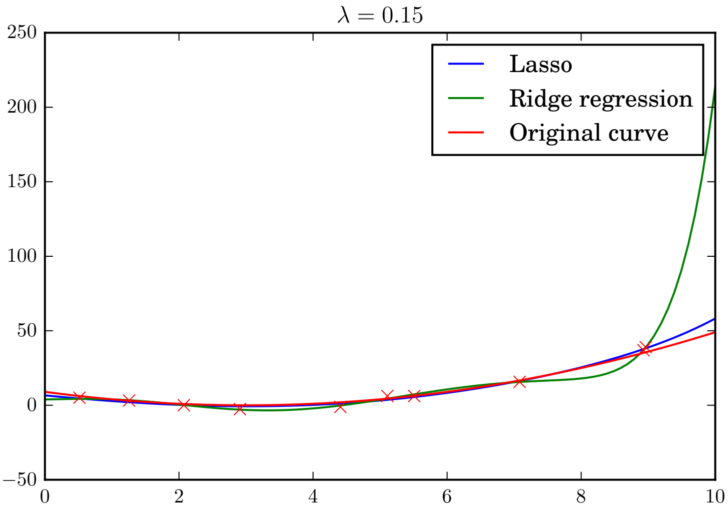
- Another common regularization is lasso. Instead of $\lambda \mathbf{w}^T \mathbf{w}$, the scaled l_1-norm of $\mathbf{w}$, $\lambda \|\mathbf{w}\|_1$ is added to the loss objective function Thus, we want to

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \lambda \|\mathbf{w}\|_1,$$

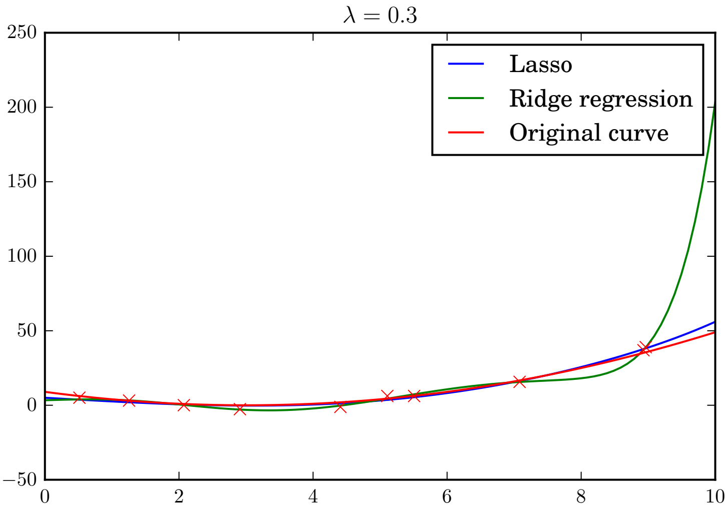
  where $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_D|$
- Unlike ridge regression, one cannot write the close form solution directly though
  - We will discuss how the optimization is typically performed later
  - For the next several slides, I just used sciki-learn library[1] in Python
- Lasso tends to enforce a sparse weight solution. It was popular several years ago because of compressed sensing

[1] The ridge regression function in the 0.18.1 version of sciki-learn appears to have bug. Both ridge regression and lasso function are implemented as lasso.

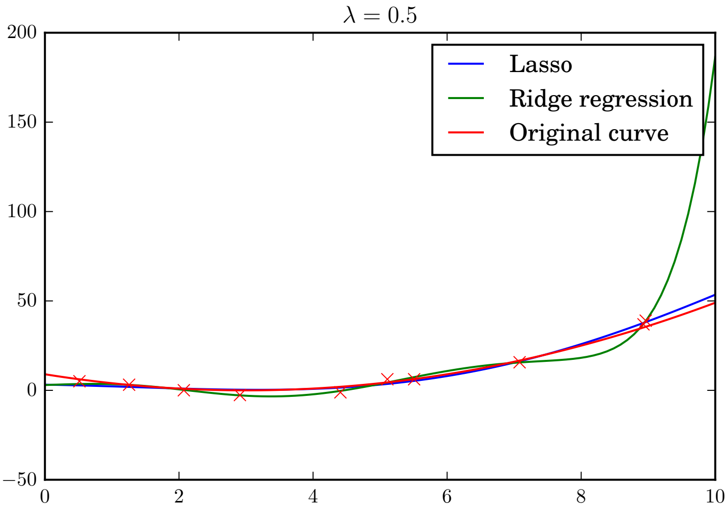
Curve fitting with Lasso and ridge regression (degree=9)



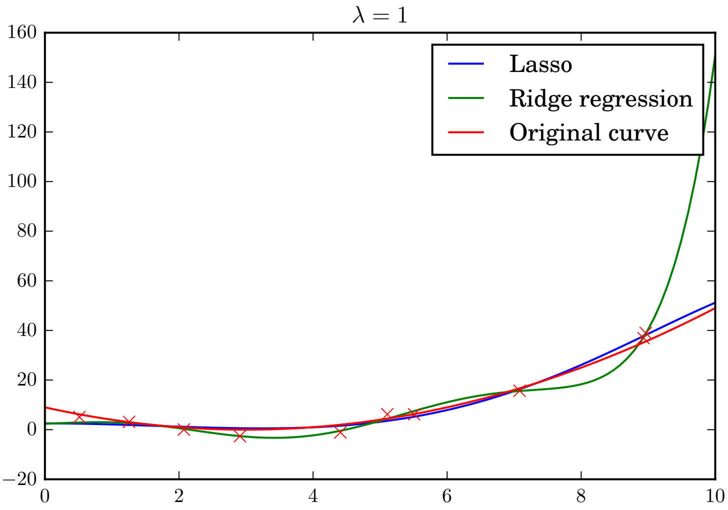
Curve fitting with Lasso and ridge regression (degree=9)



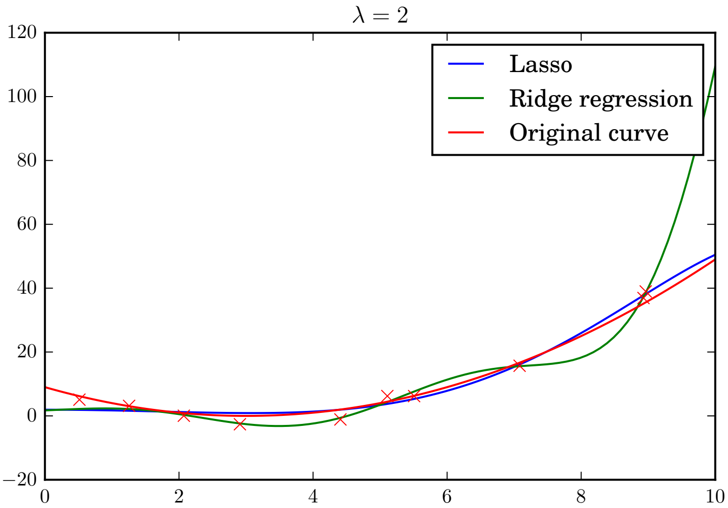
Curve fitting with Lasso and ridge regression (degree=9)



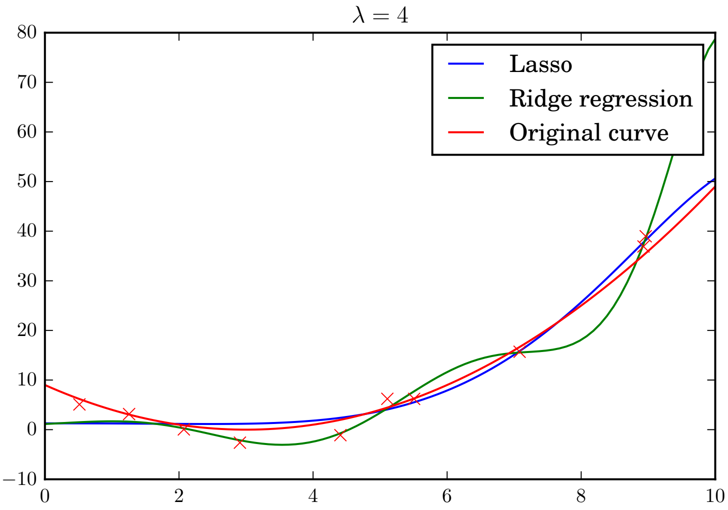
Curve fitting with Lasso and ridge regression (degree=9)



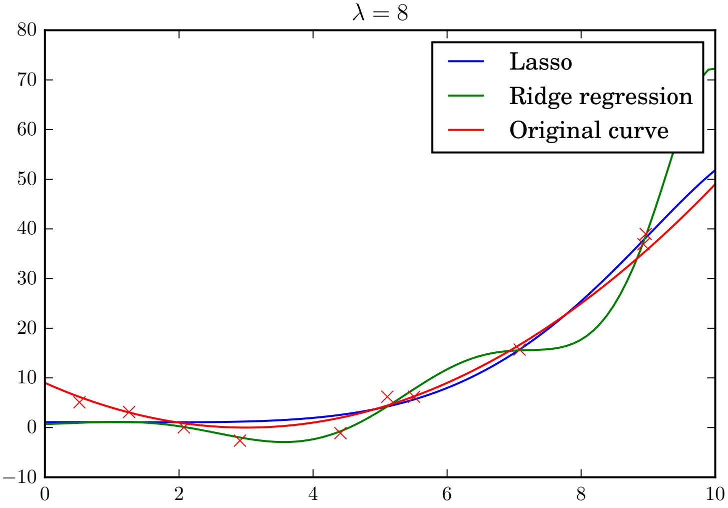
Curve fitting with Lasso and ridge regression (degree=9)



Curve fitting with Lasso and ridge regression (degree=9)



Curve fitting with Lasso and ridge regression (degree=9)



Conclusion

- Machine learning is all about generalization (from data)
- One can decrease the training error to arbitrarily small (by increasing model complexity)
- On the other hand, we really only care about test error, which is composed of
  - Bias: High bias when model is too rigid (model complexity is too low) to adapt to the training data
  - Variance: High variance when model is too flexible (model complexity is too high) that different sets of training data will converge to completely different weight parameters
- Occam's razor: a good explanation should be minimal

Conclusion

- For supervised learning systems (both classification and regression), we can typically reduce it to an optimization problem of minimizing a loss function (instead of training error) w.r.t. some weights
- Regularization terms can typically be incorporated in the loss function to keep the weights from running wild
- It is almost always better to use a more complex but regularized model than a simple model when one has sufficient training data
  - Provided that one regularized wisely
  - That is why deep neural networks typically work better

Linear classification

The same linear regression idea can be transferred to classification problems

- Consider binary classification whether an image contains a cat or not
  - We can first vectorize the input image into a column vector $\mathbf{x}$ (with an extra 1 appended to account for bias)

Linear classification

The same linear regression idea can be transferred to classification problems

- Consider binary classification whether an image contains a cat or not
  - We can first vectorize the input image into a column vector $\mathbf{x}$ (with an extra 1 appended to account for bias)
  - E.g., for a very small 2×2 image patch $\begin{pmatrix} 10 & 25 \\ 36 & 90 \end{pmatrix}$, it will be converted to

$$\mathbf{x} = (10, 25, 36, 90, 1)^T$$

Linear classification

The same linear regression idea can be transferred to classification problems

- Consider binary classification whether an image contains a cat or not
  - We can first vectorize the input image into a column vector $\mathbf{x}$ (with an extra 1 appended to account for bias)
  - E.g., for a very small 2×2 image patch $\begin{pmatrix} 10 & 25 \\ 36 & 90 \end{pmatrix}$, it will be converted to

$$\mathbf{x} = (10, 25, 36, 90, 1)^T$$

  - We will decide if the image contains a cat of not by verifying if

$$\mathbf{x}^T \mathbf{w} \lesseqgtr 0,$$

where we will need to obtain the weight $\mathbf{w}$ through training (more later)

Logistic regression

- We can introduce a scoring function

$$f(\mathbf{x};\mathbf{w}) = H(\mathbf{x}^T\mathbf{w}),$$

where $H(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$ is a step function and we have a cat if $f(\mathbf{x};\mathbf{w}) = 1$ and no cat if $f(\mathbf{x};\mathbf{w}) = 0$

Logistic regression

- We can introduce a scoring function

$$f(\mathbf{x};\mathbf{w}) = H(\mathbf{x}^T\mathbf{w}),$$

where $H(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$ is a step function and we have a cat if $f(\mathbf{x};\mathbf{w}) = 1$ and no cat if $f(\mathbf{x};\mathbf{w}) = 0$

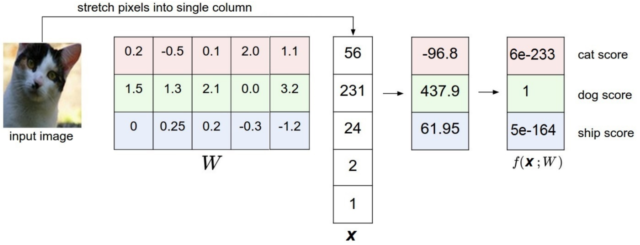
- Note that $f(\mathbf{x};\mathbf{w})$ essentially is a perceptron model and is difficult to train because of the discontinuity of $H(\cdot)$. Instead, we could replace $H(\cdot)$ by the sigmoid (or logistic) function $S(t) = \frac{1}{1+e^{-t}}$
  - Hence, known as logistic regression

Loss function of logistic regression

Another advantage of using $S(\cdot)$ is that we can interpret the output as probability and then the loss function can be specified by a "cross-entropy loss" as follows (will explain next)

$$L(\mathbf{w};\mathbf{x}) = \begin{cases} -\log f(\mathbf{x};\mathbf{w}), & \text{if the image is a cat} \\ -\log(1-f(\mathbf{x};\mathbf{w})), & \text{otherwise} \end{cases}$$

Softmax classifier

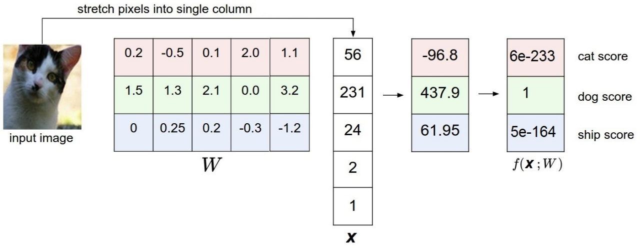


- For multiclass problem, we can extend the logistic scoring function to

$$f_i(\mathbf{x}; W) = \sigma_i(W\mathbf{x}),$$

where $\sigma_i(\mathbf{y}) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$ is known as a softmax function and is really just a normalized exponential function

Softmax classifier



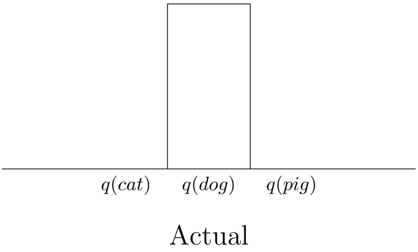
- For multiclass problem, we can extend the logistic scoring function to

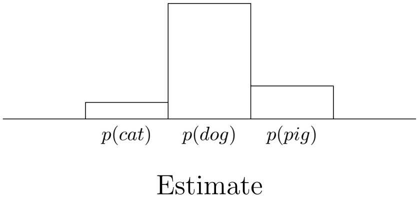
$$f_i(\mathbf{x}; W) = \sigma_i(W\mathbf{x}),$$

where $\sigma_i(\mathbf{y}) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$ is known as a softmax function and is really just a normalized exponential function

- Again, we can interpret $f_i(\mathbf{x}; W)$ as the estimated probability of $\mathbf{x}$ belong to class i
  - E.g., $p(cat; \mathbf{x}, W) = f_{cat}(\mathbf{x}; W)$

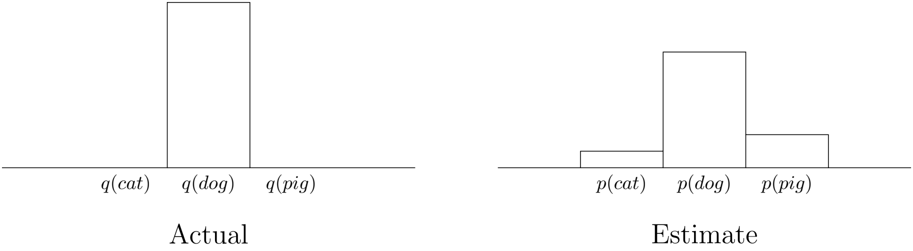
Cross entropy loss function





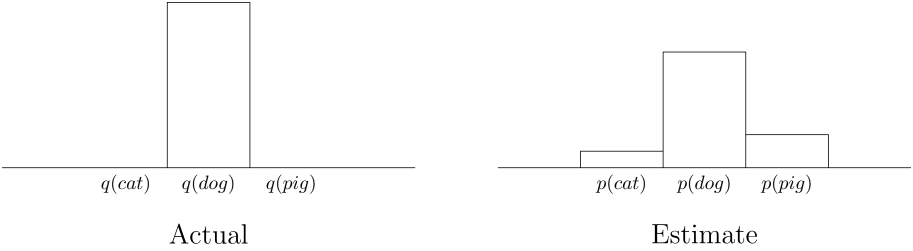
- Let say the image is actually a dog. We can express this as a distribution as shown on the left

Cross entropy loss function



- Let say the image is actually a dog. We can express this as a distribution as shown on the left
- Ideally we would like the estimated probability distribution matches the actual one

Cross entropy loss function



- Let say the image is actually a dog. We can express this as a distribution as shown on the left
- Ideally we would like the estimated probability distribution matches the actual one
- We can measure the difference between two distributions with KL-divergence given by

$$KL(q\|p) = \sum_i q_i \log \frac{q_i}{p_i}$$

Cross entropy loss function (con't)

- For more about KL-divergence, please check out the information theory textbook by Cover and Thomas or take my class next semester. For our discussion here, you only need to know that
  - $KL(q\|p)$ is more positive if p and q are more different
  - $KL(q\|p) = 0$ if p and q are identical

Cross entropy loss function (con't)

- For more about KL-divergence, please check out the information theory textbook by Cover and Thomas or take my class next semester. For our discussion here, you only need to know that
  - $KL(q\|p)$ is more positive if p and q are more different
  - $KL(q\|p) = 0$ if p and q are identical
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = -\underbrace{\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

Cross entropy loss function (con't)

- For more about KL-divergence, please check out the information theory textbook by Cover and Thomas or take my class next semester. For our discussion here, you only need to know that
  - $KL(q\|p)$ is more positive if p and q are more different
  - $KL(q\|p) = 0$ if p and q are identical
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = -\underbrace{\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i$$

Cross entropy loss function (con't)

- For more about KL-divergence, please check out the information theory textbook by Cover and Thomas or take my class next semester. For our discussion here, you only need to know that
  - $KL(q\|p)$ is more positive if p and q are more different
  - $KL(q\|p) = 0$ if p and q are identical
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = -\underbrace{\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i = -\log p_{j(\mathbf{x})}$$

Cross entropy loss function (con't)

- For more about KL-divergence, please check out the information theory textbook by Cover and Thomas or take my class next semester. For our discussion here, you only need to know that
  - $KL(q\|p)$ is more positive if p and q are more different
  - $KL(q\|p) = 0$ if p and q are identical
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = -\underbrace{\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i = -\log p_{j(\mathbf{x})} = -\log f_{j(\mathbf{x})}(\mathbf{x};W) = -\log \sigma_{j(\mathbf{x})}(W\mathbf{x}),$$

where $j(\mathbf{x})$ is the actual class index of $\mathbf{x}$

Cross entropy loss function (con't)

- For more about KL-divergence, please check out the information theory textbook by Cover and Thomas or take my class next semester. For our discussion here, you only need to know that
  - $KL(q\|p)$ is more positive if p and q are more different
  - $KL(q\|p) = 0$ if p and q are identical
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = -\underbrace{\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i = -\log p_{j(\mathbf{x})} = -\log f_{j(\mathbf{x})}(\mathbf{x}; W) = -\log \sigma_{j(\mathbf{x})}(W\mathbf{x}),$$

where $j(\mathbf{x})$ is the actual class index of $\mathbf{x}$

- The total loss is just sum over all training $\mathbf{x}$: $L(W) = \sum_{\mathbf{x}} L(W;\mathbf{x})$

Derivative of softmax loss

- Denote $\mathbf{o} \triangleq W\mathbf{x}$, and it is easy to verify that $\frac{\partial}{\partial o_i}\sigma_j(\mathbf{o}) = -\sigma_i(\mathbf{o})\sigma_j(\mathbf{o})$ and $\frac{\partial}{\partial o_i}\sigma_i(\mathbf{o}) = \sigma_i(\mathbf{o})(1-\sigma_i(\mathbf{o}))$.

[1] It is okay if you don't get follow everything for this sides. You probably won't need it when you implement using packages

Derivative of softmax loss

- Denote $\mathbf{o} \triangleq W\mathbf{x}$, and it is easy to verify that $\frac{\partial}{\partial o_i}\sigma_j(\mathbf{o}) = -\sigma_i(\mathbf{o})\sigma_j(\mathbf{o})$ and $\frac{\partial}{\partial o_i}\sigma_i(\mathbf{o}) = \sigma_i(\mathbf{o})(1-\sigma_i(\mathbf{o}))$. Thus,

$$\frac{\partial}{\partial o_i}L(W;\mathbf{x}) = -\frac{\partial}{\partial o_i}\sum_j q_j \log \sigma_j(\mathbf{o}) = q_i - \sum_j q_j \frac{\sigma_i \sigma_j}{\sigma_j} = q_i - \sigma_i(\mathbf{o})$$

[1] It is okay if you don't get follow everything for this sides. You probably won't need it when you implement using packages

Derivative of softmax loss

- Denote $\mathbf{o} \triangleq W\mathbf{x}$, and it is easy to verify that $\frac{\partial}{\partial o_i}\sigma_j(\mathbf{o}) = -\sigma_i(\mathbf{o})\sigma_j(\mathbf{o})$ and $\frac{\partial}{\partial o_i}\sigma_i(\mathbf{o}) = \sigma_i(\mathbf{o})(1-\sigma_i(\mathbf{o}))$. Thus,

$$\frac{\partial}{\partial o_i}L(W;\mathbf{x}) = -\frac{\partial}{\partial o_i}\sum_j q_j \log \sigma_j(\mathbf{o}) = q_i - \sum_j q_j \frac{\sigma_i \sigma_j}{\sigma_j} = q_i - \sigma_i(\mathbf{o})$$

- Using chain rule

$$\frac{\partial}{\partial w_{i,j}}L(W;\mathbf{x}) = \sum_k \frac{\partial}{\partial o_k}L(W;\mathbf{x})\frac{\partial o_k}{\partial w_{i,j}} = \frac{\partial}{\partial o_i}L(W;\mathbf{x})x_j$$
$$=(q_i - \sigma_i(\mathbf{o}))x_j$$

[1] It is okay if you don't get follow everything for this sides. You probably won't need it when you implement using packages

Optimization

- For linear regression and ridge regression, we have a close form solution for minimizing the loss function but in most other models, we do not

Optimization

- For linear regression and ridge regression, we have a close form solution for minimizing the loss function but in most other models, we do not
- In practice, to minimize the loss function w.r.t. the weight W, we can use simple steepest descent. That is,

$$W = W - \Delta W \quad \text{with} \quad \Delta W = \epsilon \nabla_W L(W),$$

where ϵ is the learning rate and suppose to be small. It is often just set heuristically. We may talk more about it later in this course

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow
  - Recall that $L(W)$ supposes to a sum over individual loss of all training data $L(W; \mathbf{x})$

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow
  - Recall that $L(W)$ supposes to a sum over individual loss of all training data $L(W; \mathbf{x})$
  - But $L(W)$ is really just an approximate as any training set is stochastic in natural in any case. Why not just approximate $L(W)$ not as refined with few data? That is, just pick a subset $\mathcal{X}_i$ from the training set and use

$$L_i(W) = \sum_{\mathbf{x} \in \mathcal{X}_i} L(W; \mathbf{x})$$

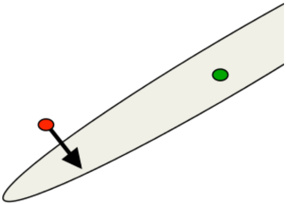
instead. And this is known as the mini-batch gradient descent

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow
    - Recall that $L(W)$ supposes to a sum over individual loss of all training data $L(W; \mathbf{x})$
    - But $L(W)$ is really just an approximate as any training set is stochastic in natural in any case. Why not just approximate $L(W)$ not as refined with few data? That is, just pick a subset $\mathcal{X}_i$ from the training set and use
    $$L_i(W) = \sum_{\mathbf{x} \in \mathcal{X}_i} L(W; \mathbf{x})$$
    instead. And this is known as the mini-batch gradient descent
- One may go to the extreme and only pick one $\mathbf{x}$ to estimate the gradient. This formally is known as the stochastic gradient descent. But in practice, no one uses it. But people often say stochastic gradient descent when they actually mean mini-batch gradient descent

Gradient descent with moment

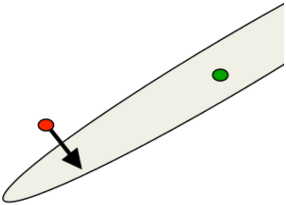
- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle



[1]Slide borrowed from Hinton's coursera course

Gradient descent with moment

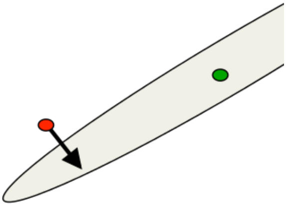
- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle
  - The gradient is big in the direction in which we only want to travel a small distance
  - The gradient is small in the direction in which we want to travel a large distance



[1] Slide borrowed from Hinton's coursera course

Gradient descent with moment

- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle
  - The gradient is big in the direction in which we only want to travel a small distance
  - The gradient is small in the direction in which we want to travel a large distance
- A simple solution is to introduce "momentum" to the change of W. That is, $\Delta W = \lambda(\epsilon \nabla_W L(W)) + (1-\lambda)\Delta W^{(old)}$
- Will talk more about optimization methods later. So much for today



[1] Slide borrowed from Hinton's coursera course

Remark on computing gradient

- For the previous discussion, we always assume that the gradient can be found analytically. In practice, it may not be true also

Remark on computing gradient

- For the previous discussion, we always assume that the gradient can be found analytically. In practice, it may not be true also
- But gradient of $L(W)$ can easily be computed numerically. For example, say $W = \begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}$,

$$\frac{\partial}{\partial W_{1,1}} L(W) \approx \frac{1}{h}\left[L\left(\begin{pmatrix} 4.1 + h & 3.3 \\ -1.2 & 2.1 \end{pmatrix} \right) - L\left(\begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix} \right) \right]$$

Remark on computing gradient

- For the previous discussion, we always assume that the gradient can be found analytically. In practice, it may not be true also
- But gradient of $L(W)$ can easily be computed numerically. For example, say $W = \begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}$,

$$\frac{\partial}{\partial W_{1,1}} L(W) \approx \frac{1}{h}\left[L\left(\begin{pmatrix} 4.1+h & 3.3 \\ -1.2 & 2.1 \end{pmatrix}\right) - L\left(\begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}\right)\right]$$

- Actually, the numerical gradient is useful even if an analytical gradient exists. It at least provides a mean to debug your system
  - And luckily, for some packages such as Theano, they automatically find the analytical gradient for you

Conclusion

- For classification, we can feed the output of a linear regressor to a logistic function or softmax function to form a linear classifier
  - For only two classes, we have the logistic “regression” classifier
  - For multi-class cases, we have the softmax classifiers
- For finding the optimal weights, we may not be able to get the solution right away analytically (possible though for linear regression and ridge regression)
  - Can optimize iteratively with gradient descent
  - Can speed up gradient descent by using mini-batch instead of full batch
  - Momentum is a common trick to improve optimization efficiency also

Presentation logistics

- Tentative start dates: late-Feb
- Pick your packages, give me your preference by next class. Include your highest three preferred packages with *sorted order*

Presentation logistics

- Tentative start dates: late-Feb
- Pick your packages, give me your preference by next class. Include your highest three preferred packages with *sorted order*
- We may go through a lottery if too many of you pick the same packages. Aim to cover the whole list if possible
- The order of presentation is based on popularity of your topic. The most popular topic will be presented first and so on

Presentation logistics

- Tentative start dates: late-Feb
- Pick your packages, give me your preference by next class. Include your highest three preferred packages with *sorted order*
- We may go through a lottery if too many of you pick the same packages. Aim to cover the whole list if possible
- The order of presentation is based on popularity of your topic. The most popular topic will be presented first and so on
- Within a topic, the order will be sorted by the order of your last name
  - Sorry, Amal and Mohamed. :)

Presentation logistics (con't)

- I will let you know who are presenting the same package as you. I will let you all to coordinate yourself. Try to present different part of the package from your classmate. A little overlapping is acceptable though

Presentation logistics (con't)

- I will let you know who are presenting the same package as you. I will let you all to coordinate yourself. Try to present different part of the package from your classmate. A little overlapping is acceptable though
- $>$ 20 minutes and $<$ 40 minutes presentation, don't drag too long. Try to get quick to the point. We have a relatively big class

Presentation logistics (con't)

- I will let you know who are presenting the same package as you. I will let you all to coordinate yourself. Try to present different part of the package from your classmate. A little overlapping is acceptable though
- $>$ 20 minutes and $<$ 40 minutes presentation, don't drag too long. Try to get quick to the point. We have a relatively big class
- I'll try to record the video and share with you all (maybe put up on YouTube)

Presentation logistics (con't)

- I will let you know who are presenting the same package as you. I will let you all to coordinate yourself. Try to present different part of the package from your classmate. A little overlapping is acceptable though
- $>$ 20 minutes and $<$ 40 minutes presentation, don't drag too long. Try to get quick to the point. We have a relatively big class
- I'll try to record the video and share with you all (maybe put up on YouTube)
- Incentives: two bonus awards (5% each) for the best presentation according to my taste and popular votes from you all, respectively

	Pros	Cons
Caffe	• Don't need to write code	• Adding module is harder (need C++) • RNN is not support
Torch	• Easy to create own module • Can do RNN	• Lua
Theano	Flexible and powerful	Kind of low-level
Tensorflow	Industry loves it	Slow
Keras / Lasagne	Less verbose than Theano	Less flexible
MXnet	Rumored to be fast	Unpopular
Matconvnet	MATLAB	CNN only

Final reminder

- Assignment 1 will be due **2 weeks** later
- First successful submission has 3% (of everything) bonus