Neural Networks and Deep Learning

CS5242

Wei WANG

National University of Singapore

cs5242@comp.nus.edu.sg

Recap

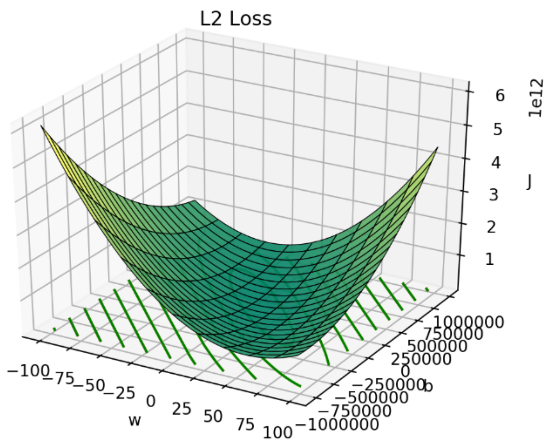
- AI, ML, and DL
- Topics: MLP, CNN, RNN
- Pre-requisite and workload
  - Run assignment 0
  - Understand house price prediction

Recap

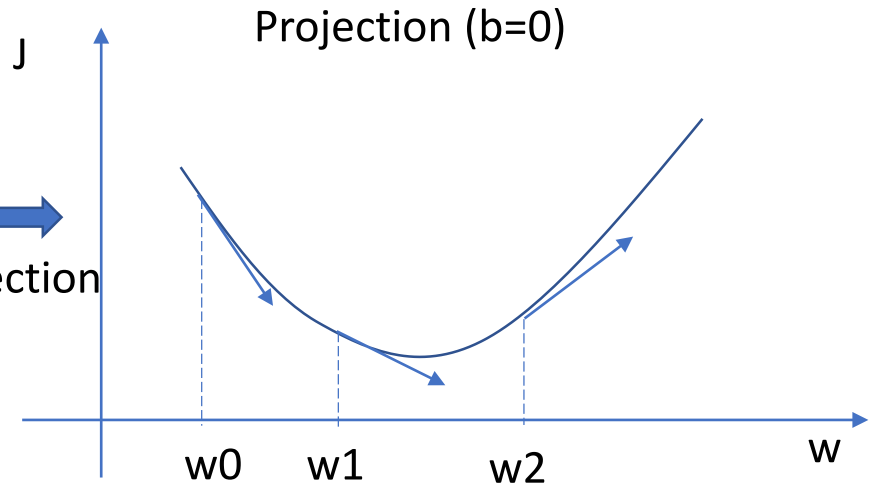
- House price prediction
  - **Sample** (example): (one house, price) is a sample
  - **Feature** $x \in R^m$: attributes used to represent the house, e.g. size, #floors
  - Ground-truth **label** y: the real price
  - Model: linear regression, $\tilde{y} = w^T x + b, w \in R^m, b \in R$
  - **Parameters**: w, b
  - **Prediction** $\tilde{y}$: the price predicted by our ML model
  - **Loss function**: objective for training the model.
    - $J(w,b) = \frac{\sum_{<x,y>\in S_{train}} L(x,y|w,b)}{|S_{train}|} = \frac{\sum_{i=1}^{n} L(x^{(i)},y^{(i)}|w,b)}{n}$
    - $L(x,y|w,b) = |\tilde{y} - y|^2$
  - Back-propagation and gradient descent

Training by gradient descent

- Gradient descent (GD) algorithm for optimization



projection



α is called the learning rate, which controls the moving step length. It important for convergence. If it is large, w would oscillate around the optimal position. If it is small, it would take many iterations to reach the optimal position.

Initialize w as w0

Compute $\frac{\partial J}{\partial w0}$, negative;

Move w from w0 to the right by

$$\mathrm{w1} = \mathrm{w0} - \alpha \frac{\partial J}{\partial w0}$$

Compute $\frac{\partial J}{\partial w1}$, negative;

Move w from w1 to the right by

$$\mathrm{w2} = \mathrm{w1} - \alpha \frac{\partial J}{\partial w1}$$

Compute $\frac{\partial J}{\partial w2}$, positive

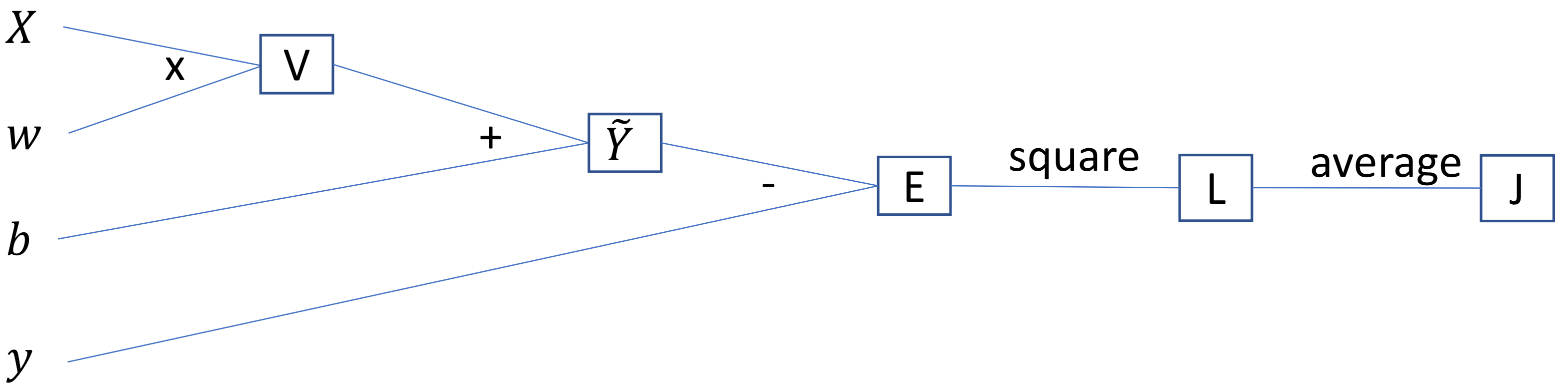
Move w from w2 to the left by

$$\mathrm{w3} = \mathrm{w2} - \alpha \frac{\partial J}{\partial w2}$$

Gradually decrease J and move w to the optimal position

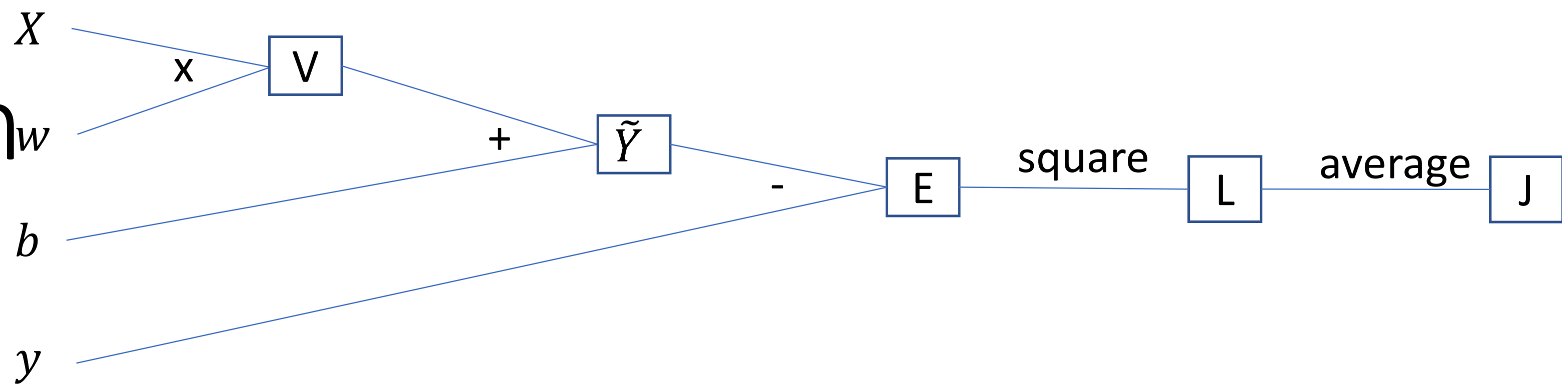
Back-propagation

- Forward
- $X \in R^{m*n}, Y \in R^{1*n}$



- $V = w^T X, \in R^{1*n}$
- $\tilde{Y} = \mathrm{V} + \mathrm{b}, \in R^{1*n}$
- $E = \tilde{Y} - Y, \in R^{1*n}$
- $L = E^2, \in R^{1*n}$
- $J = numpy.average(L) \in R^+$

Back-propagation



- Backward
  - $\frac{\partial J}{\partial L} = \left[\frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \ldots\right] \quad , \in R^{1*n}$
  - $\frac{\partial J}{\partial E} = \frac{\partial J}{\partial L} \times \frac{\partial L}{\partial E} = \frac{\partial J}{\partial L} \times 2E = 2E/n, \in R^{1*n}$
  - $\frac{\partial J}{\partial \tilde{Y}} = \frac{\partial J}{\partial E} \times \frac{\partial E}{\partial \tilde{Y}} = \frac{\partial L}{\partial E} \times [1,1,1,\ldots] = 2E/n\ , \in R^{1*n}$
  - $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \tilde{Y}} \cdot \frac{\partial \tilde{Y}}{\partial b} = \frac{\partial L}{\partial \tilde{Y}} \cdot [1,1,1,\ldots] \quad , \in R$ (dot product)
  - $\frac{\partial J}{\partial V} = \frac{\partial J}{\partial \tilde{Y}} \times \frac{\partial \tilde{Y}}{\partial V} = \frac{\partial L}{\partial \tilde{Y}} \times [1,1,1,\ldots] = 2E/n, \in R^{1*n}$
  - $\frac{\partial J}{\partial w} = \left(\frac{\partial L}{\partial V}\frac{\partial V}{\partial w}\right)^T = X\left(\frac{\partial L}{\partial V}\right)^T, \in R^m$ (matrix-matrix product)

× : element-wise multiplication

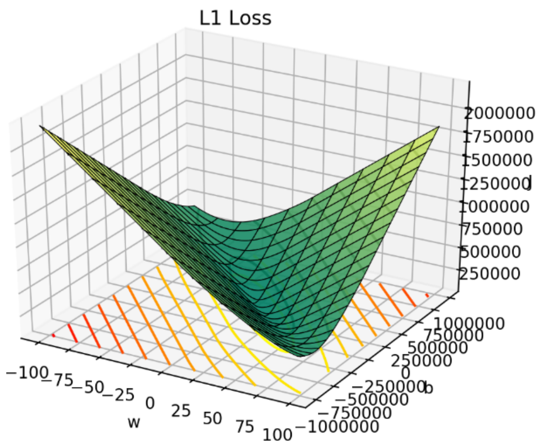
Element-wise multiplication?
dot product?
matrix product?
transpose ?

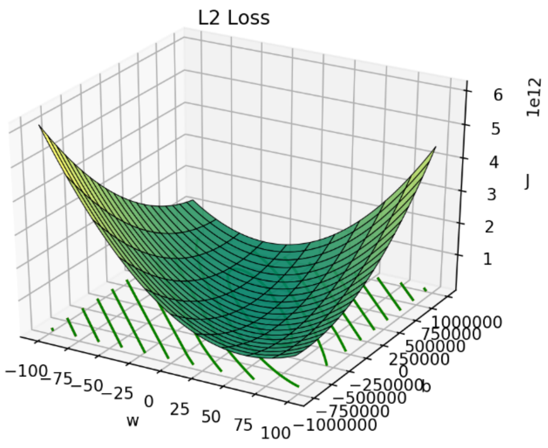
Shape check: for every node in the graph, its shape should be the same during forward and backward.

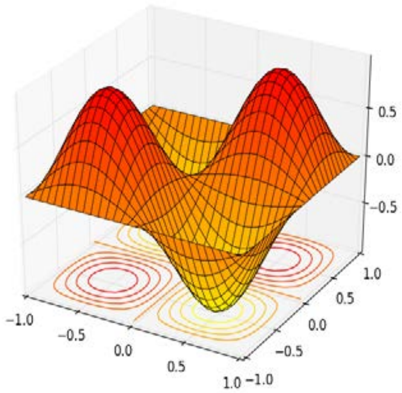
Gradient descent

http://ruder.io/optimizing-gradient-descent/

Loss contour

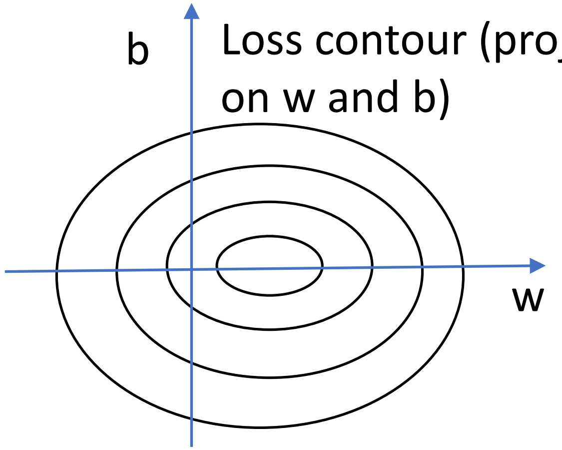




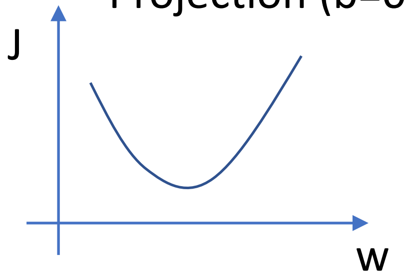


Source from
https://goo.gl/ULkt2Y

Loss contour (projection on w and b)

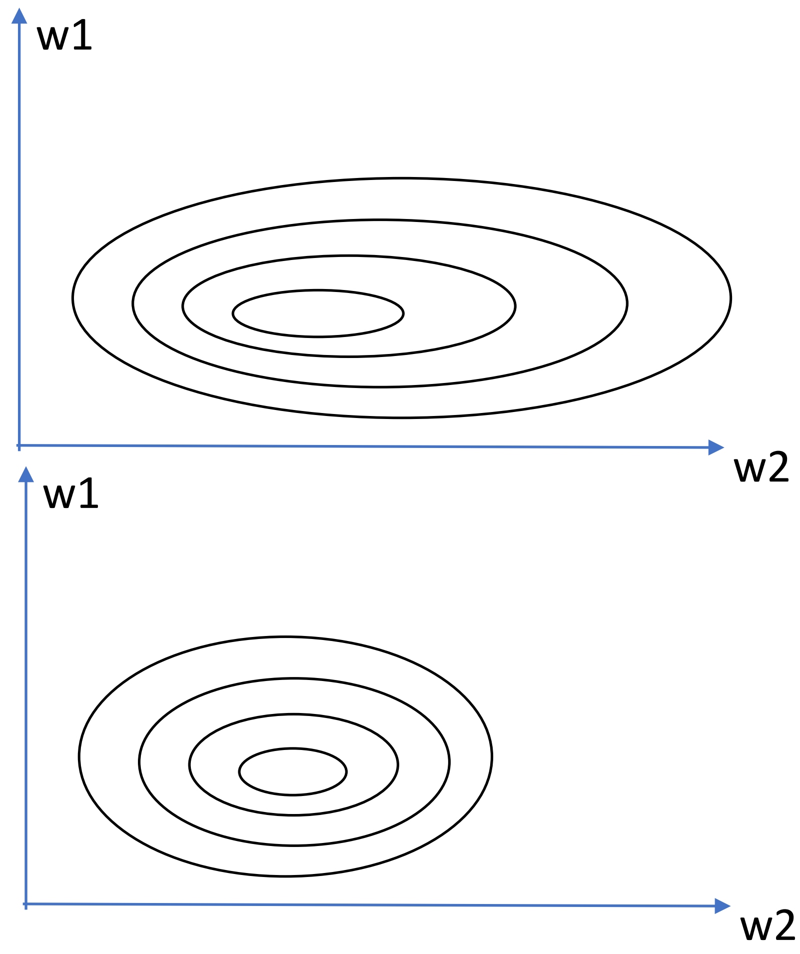


Projection (b=0)



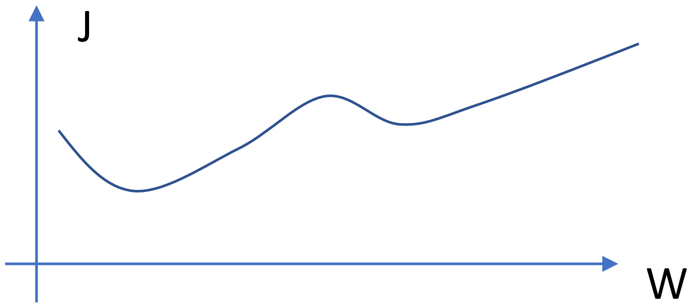
Data normalization

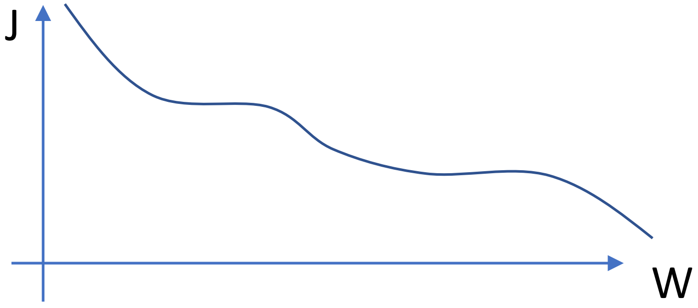
- The attributes of a house varies in scale
- Normalize the attributes into similar scale

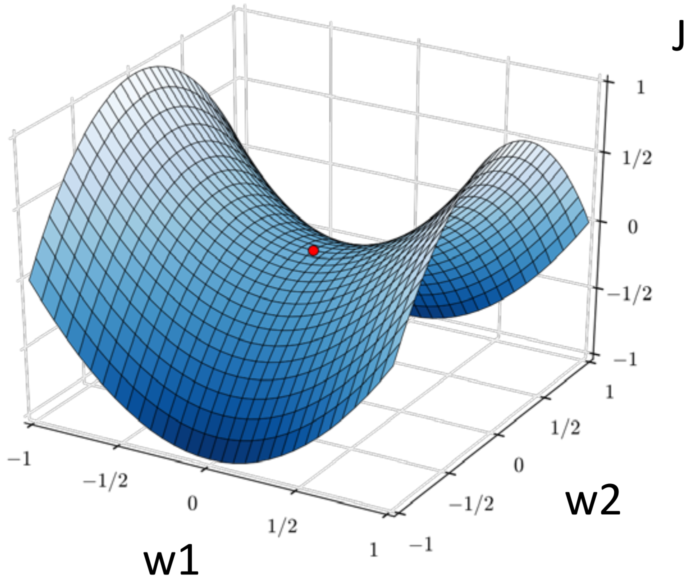


Challenges of gradient descent

- Local optimum
- Saddle points







from: https://en.wikipedia.org/wiki/Saddle_point

Gradient descent (GD)

Stochastic gradient descent (SGD)

Mini-batch SGD

Mini-batch SGD with momentum

RMSProp

Adam

The Evolution of Gradient Descent

- https://www.youtube.com/watch?v=nhqo0u1a6fw

Bias and variance

Training and testing

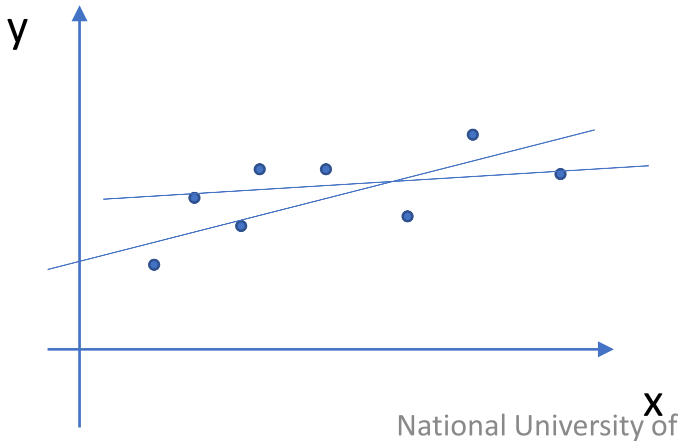
- Train a model over experience data (training samples); and then
- Deploy the model to do prediction for **new** data;

Training and testing performance

- Training
  - minimize the error over training data
- Testing
  - Fix the model parameters
  - Make predictions on unseen data samples (i.e. new data)
- Ideal case
  - Small training error and small testing error.

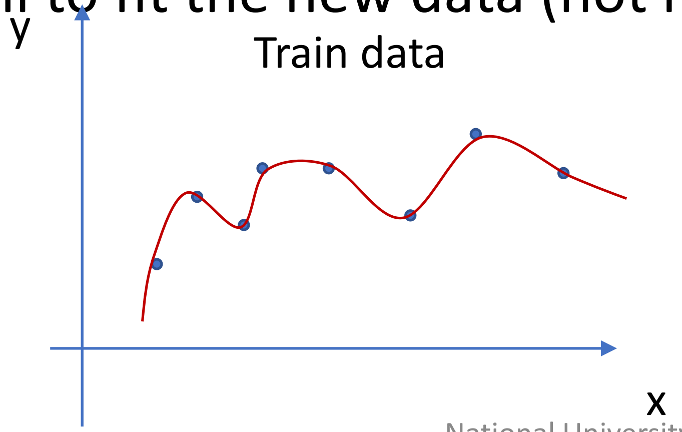
Underfitting

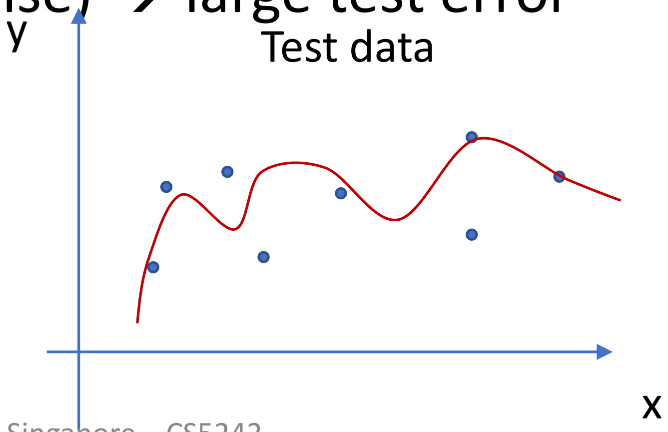
- For example,
  - $\tilde{y} = xw + b$
- The model is too simple to model the data
  - Low capacity/complexity



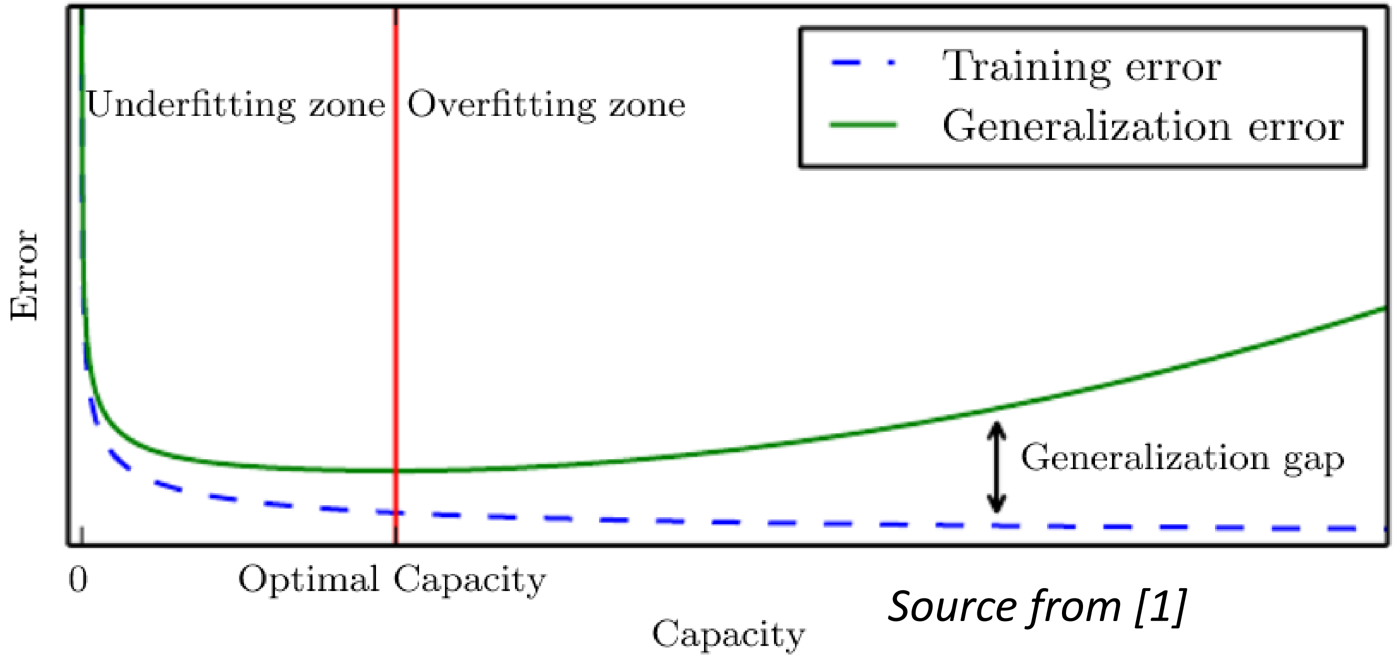
Overfitting

- Good performance on seen data, i.e. training data
- Bad performance on unseen data, i.e. test data
- $\hat{f}(x) = f(x) + \varepsilon$
- We learn the model $\hat{f}(x)$ over noisy data $f(x) + \varepsilon = y$
  - Small training error → the model fits the noise very well
  - It may fail to fit the new data (not noise) → large test error





Underfitting and overfitting



Source from [1]

Bias and variance

$$\mathrm{E}\left[\left(y-\hat{f}(x)\right)^2\right] = \mathrm{Bias}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

where

$$\mathrm{Bias}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x) - f(x)\right]$$

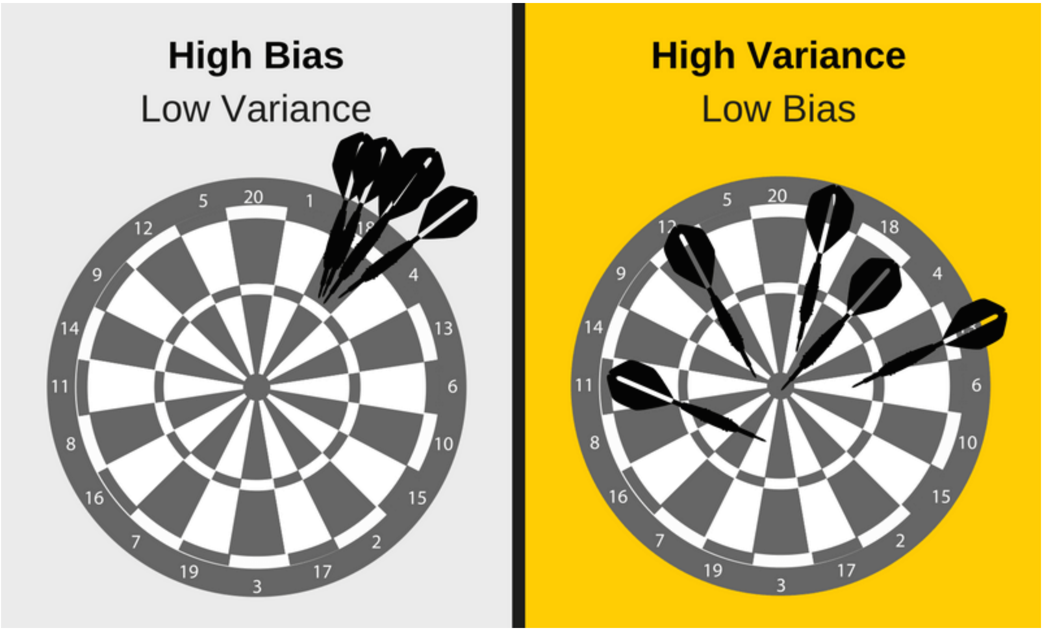
and

$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x)^2\right] - \mathrm{E}\left[\hat{f}(x)\right]^2$$

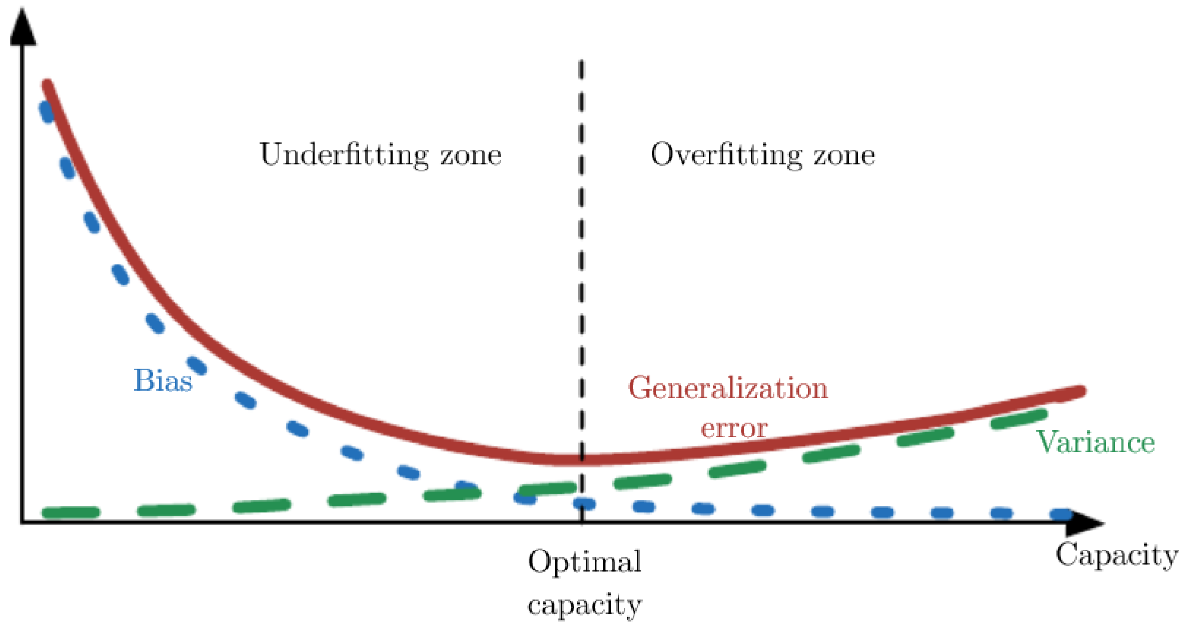
- https://ml.berkeley.edu/blog/2017/07/13/tutorial-4/

Bias and variance

- https://elitedatascience.com/bias-variance-tradeoff



All in one picture



Source from [1]

Diagnosis of underfitting and overfitting

- $\tilde{y} = w_1 x + b$
- $\tilde{y} = w_1 x + w_2 x^2 + b$
- $\tilde{y} = w_1 x + w_2 x^2 + w_3 x^3 + b$
- $\tilde{y} = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$
- Solution
  - Compare training error and testing error
  - Select the optimal capacity
- This process is like training a model over the test data

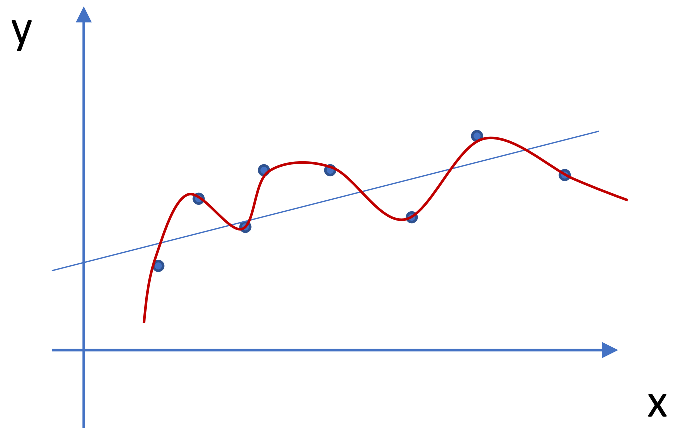
Diagnosis of underfitting and overfitting

- Validation data

- Benchmark overfitting
  - ImageNet

Solutions for underfitting

- Increase model capacity
  - Capacity?
    - Use more features
    - Use complex models
      - $\tilde{y} = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + b$



Solutions for overfitting

- Decrease the gap |training error – test error|
  - Training data
    - See almost all cases although not every sample in the test data
  - Capacity
    - Regularization of the hypothesis space
      - Add prior or constraint to the functions.
        - L2
      - Dropout
      - Parameter sharing
  - Ensemble
    - Dropout
  - Early stopping

L2 regularization

- $min_\theta J(\theta)$
  - $min_{w,b} J(w,b) = \frac{\sum_{<x,y>\in S_{train}} L(x,y|w,b)}{|S_{train}|} = \frac{\sum_{i=1}^{n} L(x^{(i)},y^{(i)}|w,b)}{n}$
- $min_\theta J(\theta) + \lambda|\theta|^2$
  - Partial derivative
    - derivate from both $J(\theta)$ and the regularization term
    - $\frac{\partial J}{\partial \theta} + 2\lambda\theta$
    - Plot $J(\theta) + \lambda|\theta|^2$ instead of $J(\theta)$
  - Real experience
    - https://github.com/BVLC/caffe/blob/master/examples/cifar10/cifar10_full.prototxt#L140

- Bias and variance
- https://www.youtube.com/watch?v=SjQyLhQIXSM

Reference

- [1] Goodfellow Ian, Bengio Yoshua, Courville Aaron. Deep learning. MIT Press. http://www.deeplearningbook.org. Chapter 5.
- https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/
- https://elitedatascience.com/bias-variance-tradeoff
- Curse of dimensionality https://goo.gl/4UT253