Assignment Lecture 1

Image classification with gradient descent

Håkon Hukkelås | hakon.hukkelas@ntnu.no

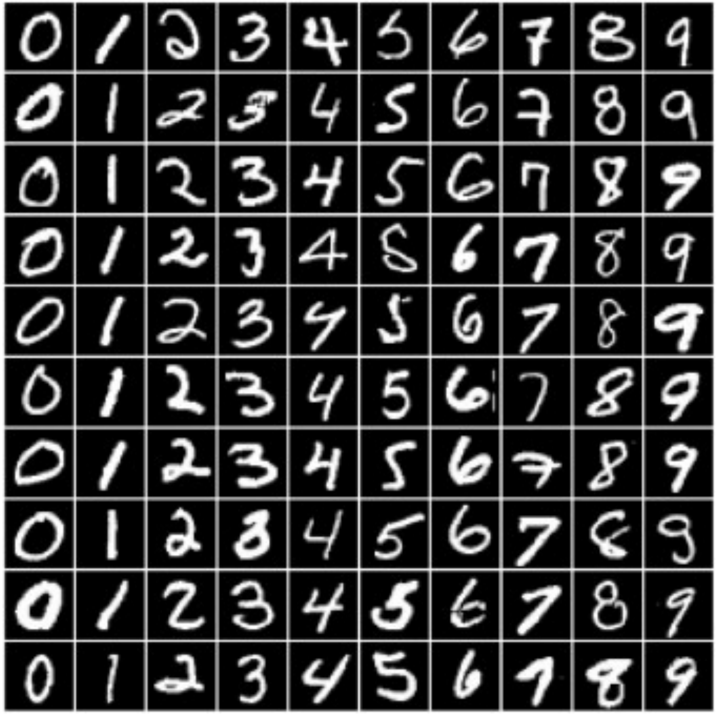
TDT4265 Course work

- Assignment 1: Single-layer neural networks
- Assignment 2: Backpropagation + “tricks of the trade”
- Assignment 3: Convolutional NN’s; Pytorch/Tensorflow
- Assignment 4: Object detection + “traditional” computer vision
- Final project: Open-ended project of your choosing

- Total % of grade:
    - Assignments: 24% (6% per assignment)
    - Project: 16%

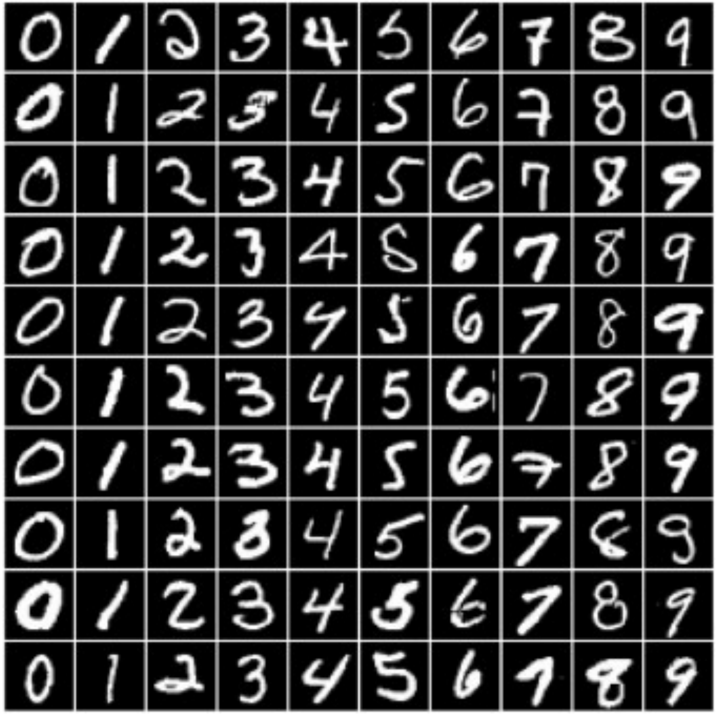
The task at hand

- Digit recognition (MNIST)
- Logistic regression (binary classification)
- Softmax regression (multi-class classification)
- All supervised learning!



MNIST

- 70,000 handwritten digits
- 28x28 grayscale images
- Correct label for each image
- Simple “toy” dataset
- State-of-the-art: 99.75% accuracy



NTNU

Task 1

- Derivation of logistic regression & softmax regression update rules
- Chain rule, chain rule, chain rule!

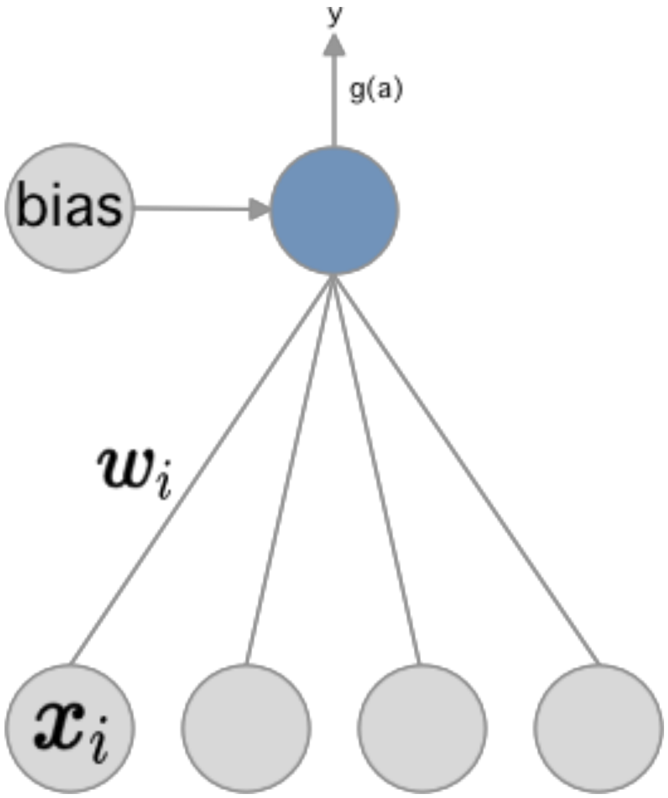
Logistic Regression (Task 2)

- Binary classification
- Single layer network (Linear regression)
- Classify the numbers 2 vs 3
    - Remove all numbers that are not 2 or 3

The neural network

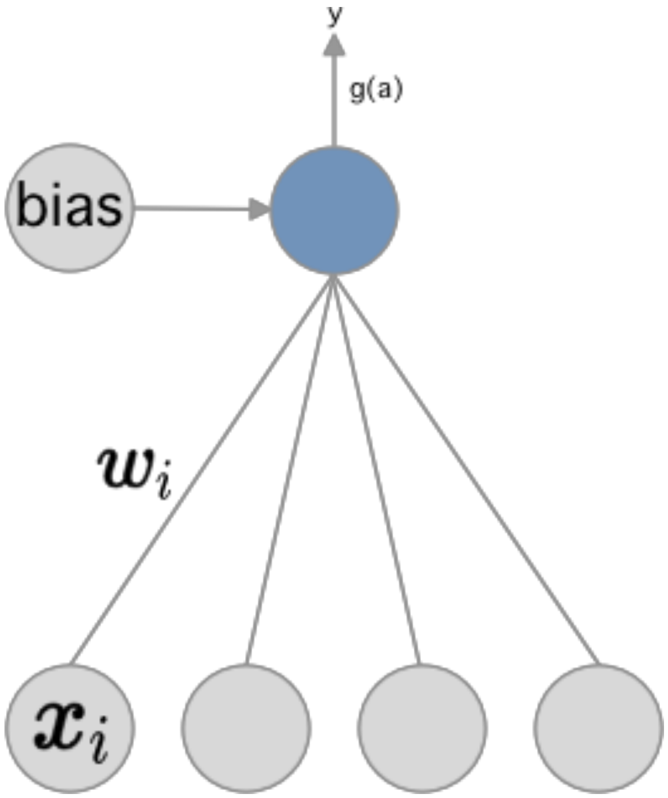
- Input nodes: 28x28 = 784
- Output nodes: 1
- Predict 1 or 0

- y: output probability
- xi: input from pixel i
- wi: weight for input i
- g: activation function



The neural network

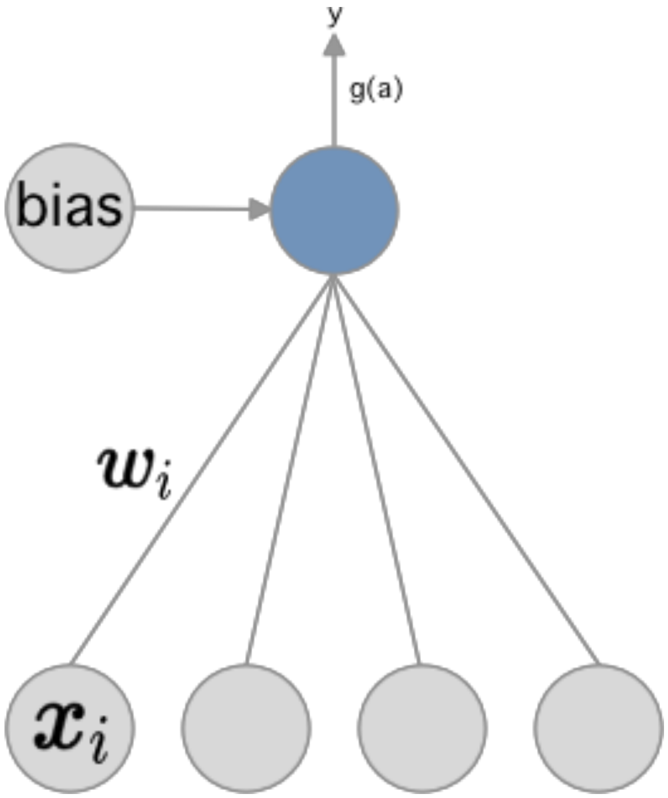
- Node activation: $$a = \sum_i w_i x_i + b$$
- Output: $y = g(a)$



- y: output probability
- xi: input from pixel i
- wi: weight for input i
- g: activation function

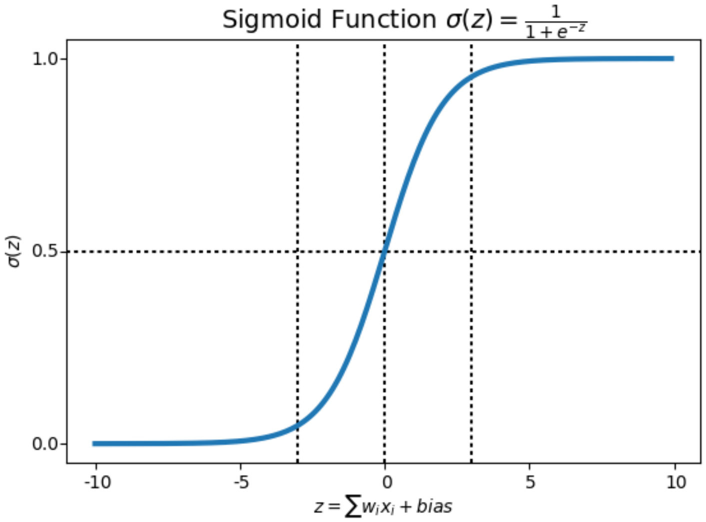
The neural network

- We will use: $$g(a) = \frac{1}{1+e^{-a}}$$
- In total: $$y = \frac{1}{1+e^{-w\cdot x-b}}$$
- Final decision: $$\text{output} = \begin{cases} 1 & \text{if} y \geq 0.5 \\ 0 & \text{else} \end{cases}$$



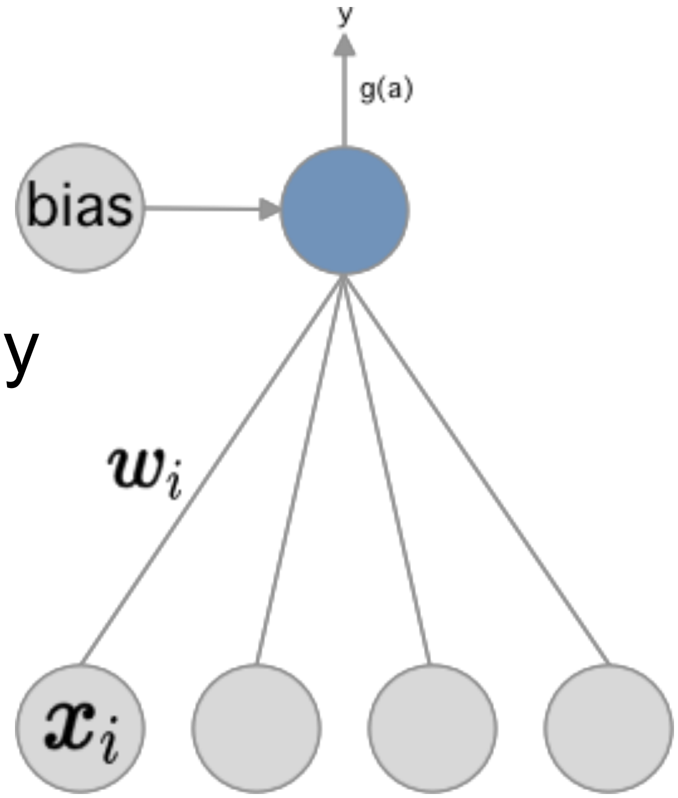
Sigmoid function

- Activation function
- Squash output to [0,1]



The neural network

- This is binary classification!
- How to train?
  - Minimize binary cross-entropy



Objective function

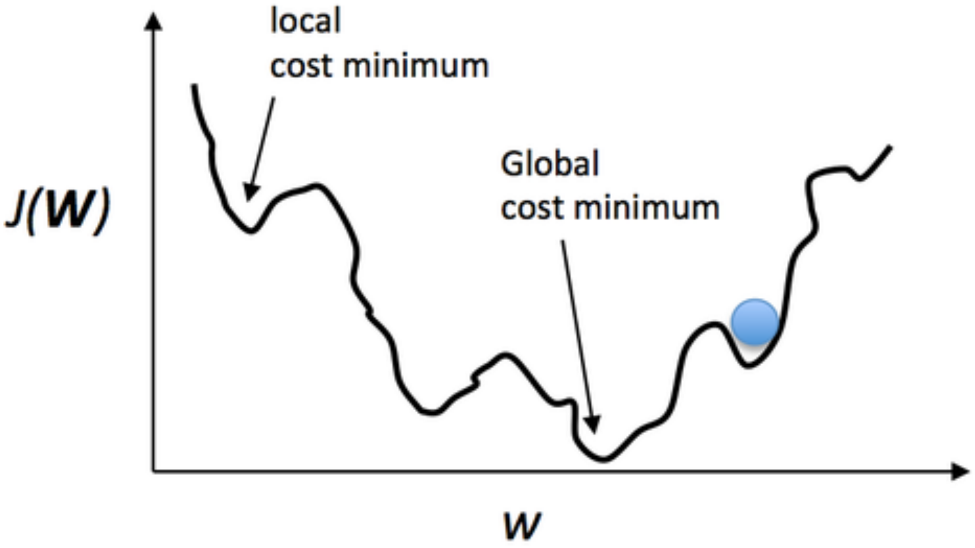
- Often referred to as cost function or loss function
- Binary cross entropy loss:

$$E(w) = -\frac{1}{N}\sum_{n=1}^{N} t^n \ln(y^n) + (1 - t^n)\ln(1 - y^n)$$

- t: target / label
- y: predicted probability
- N: number of training samples

Minimizing the objective function

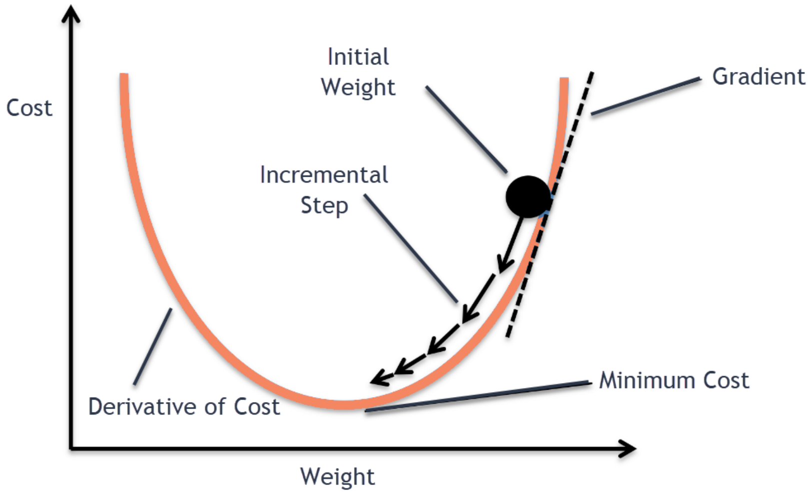
- Objective function is “always” non-convex



Gradient descent

- The building block of all neural networks
- Minimize the objective function

$$w_{t+1} = w_t - \alpha \frac{\partial E^n(w)}{\partial w}$$



- alpha: learning rate

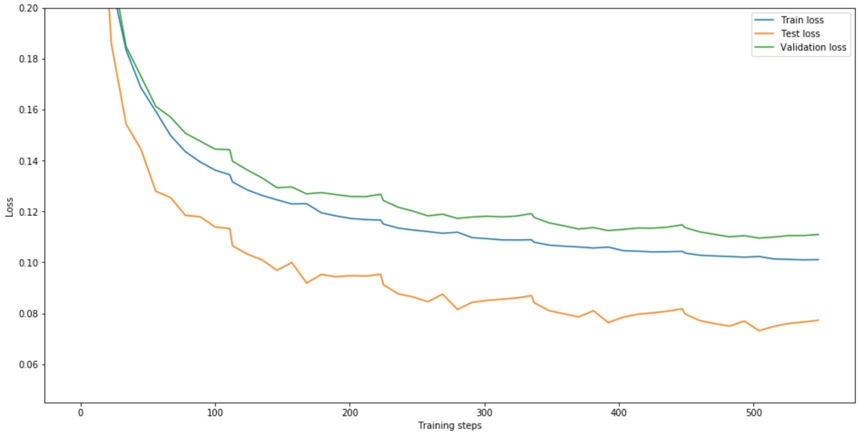
Mini-Batch gradient descent

- Find average weight change for n samples
- Why?
  - Robust and stable weight change
  - Can avoid local minima
  - Computational efficient

$$w_{t+1} = w_t - \alpha \sum_{n=1}^{N} \frac{\partial E^n(w)}{\partial w}$$

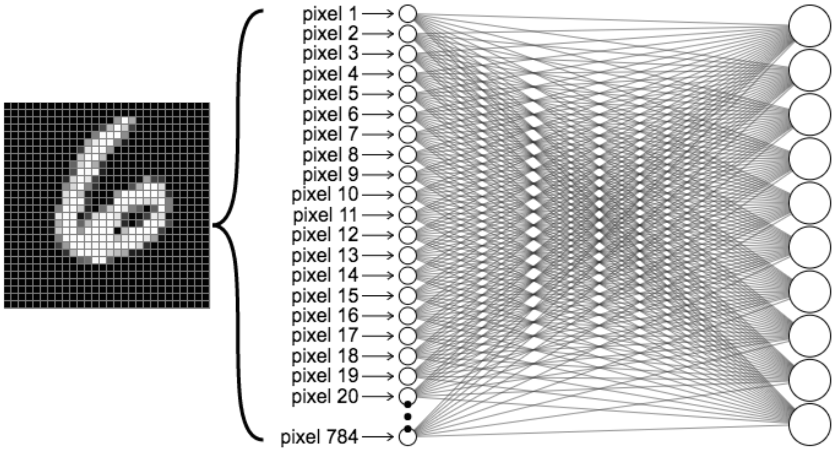
Measure performance

- Loss plots are simple to read
- Accuracy: Number of samples correctly classified
- Expect ~95% accuracy for task 2



Softmax Regression (Task 3)

- For multi-class classification
- Your task: classify all digits (10 classes)
- The network:
  - Input nodes: 28x28 = 784
  - Output nodes: 10

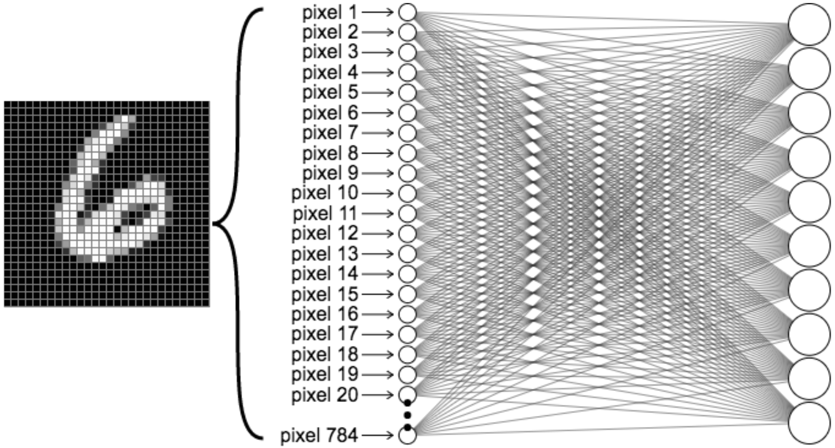


Softmax Regression

- Different activation function! Called softmax:

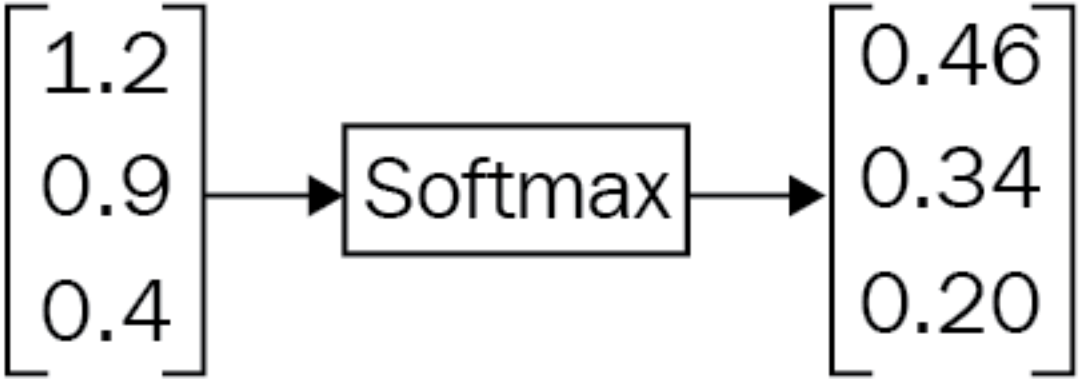
$$y_k = \frac{e^{a_k}}{\sum_{k'} e^{a_k}}, \text{ where } a_k = w_k^T x + b_k$$

- y_k represents the probability that x is in class k



Softmax activation function

- Squash output to [0,1]
- Sum of all outputs = 1
- Ensures competition between classes



One-hot encoding

- Required for multi-class classification
- Perform “binarization” of the categories

- 1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
- 3 -> [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
- 8 -> [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Objective function

- Cross-entropy loss (Same as for logistic regression)

$$E = -\sum_{n}^{N}\sum_{k=1}^{C} t_k^n \ln(y_k^n)$$

- Minimize with batch-gradient descent

- t_k: k’th index of the one-hot encoded vector
- y_k: probability that x is in class k

Final concepts

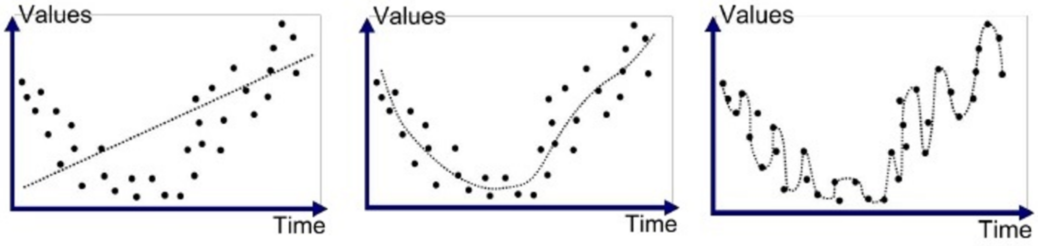
- All required to implement
  - Dataset splitting
  - Early stopping
  - L2 Regularization
  - Annealing learning rate

Dataset, validation set

- Validation set: A percentage of the training set (10% is good)
- Why validation set?
  - Prevent data-snooping on the test set
  - Use for hyperparameter tuning
  - Evaluate the model
  - No test set in the real world!
- When to use test set?
  - When you have decided a final trained model

Overfitting

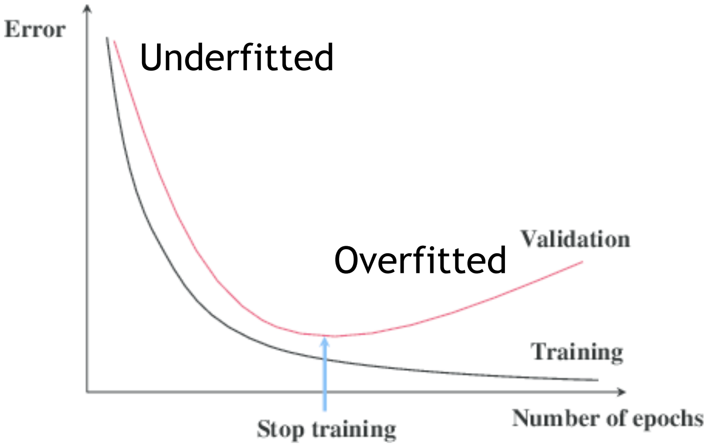
- Model memorise training points
- Does not generalise to the underlying function



Underfitted Good Fit/Robust Overfitted

Early stopping

- Stop training when validation loss reaches a minimum
- How?
  - Stop training if validation loss increases for n steps
- Use the weights at validation loss minimum as the final weights



L2 Regularization

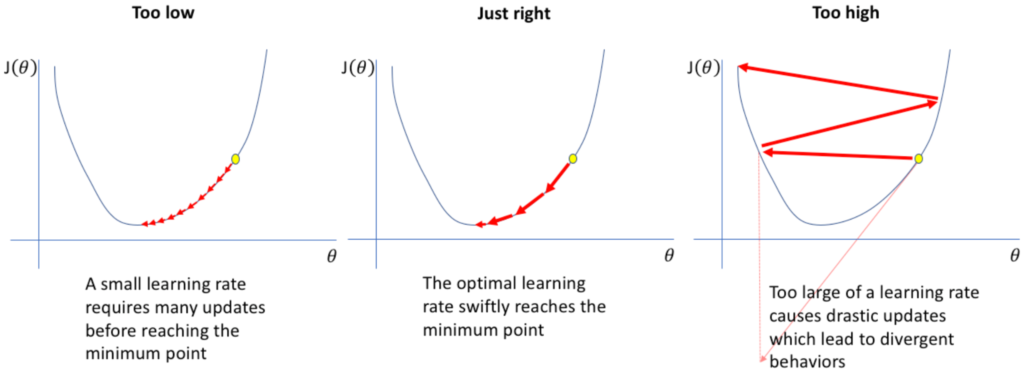
- Prevents overfitting on training data
- New objective function:

$$J(w) = E(w) + \lambda ||w||^2$$

- Shrinks the magnitude of the weights
  => Reduces model complexity

Annealing learning rate

- Reduce the learning rate over time
- Slow decay of learning rate: Computationally inefficient
- High decay of learning rate: Unable to reach a good minimum



Practical example

- In jupyter notebook
- Presentation + notebook is on blackboard