Generalisation and Optimisation

ATML track 1: Optimization and Neural Networks

Rik Sarkar

Recap

- We use $\mathcal{H}$ for hypothesis or model class (e.g. NN architecture)
  - And each $h \in \mathcal{H}$ is a possible model (e.g. an assignment of weights to edges)

- On data point x
  - True label is y and label computed by model h is $\hat{y} = h(x)$
  - Error or risk or loss of model h:
    - $\ell(\hat{y}, y) = \begin{cases} 1 \text{ if } \hat{y} \neq y \\ 0 \text{ if } \hat{y} = y \end{cases}$
- Empirical (training) loss over training set S for a model h:
  - $L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell\left(h(x_i), y_i\right)$

- The best possible model in $\mathcal{H}$ is one that has lowest empirical loss:
  - $h^* = \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$

Today

- Generalisation
- Sample complexity
- Model complexity
- Linear classification and logistic regression
- Gradient descent

- Sample exam question online
- Exercises for week 1& 2 will be put online (and posted on piazza)
  - For tutorials next week

Discussion

- What is a case for a finite $\mathcal{H}$?
  - Can you think of a situation where we are trying to choose from a few (e.g. 5) possible models?

True loss

- When we use the model in a real situation it get inputs from the data generating distribution $\mathcal{D}$

- What we really want, is minimize the “True loss”:
  - Expected loss over the distribution $\mathcal{D}$
  - Written as $L_{\mathcal{D}}(h)$

- Problem: We do not know $\mathcal{D}$

Test sample T

- We do not have access to $\mathcal{D}$
- Thus we use a test set $T \sim \mathcal{D}^t$ with t samples
- Compute the average loss $L_T(h)$ as an estimate of $L_{\mathcal{D}}(h)$

Generalisation and generalisation gap

- We want the empirical loss of the model to be close to the true loss
- This is measured by the generalisation gap (sometimes confusingly also called generalisation loss)
  - $|L_S(h) - L_{\mathcal{D}}(h)|$

- Which term among $L_S(h)$ and $L_{\mathcal{D}}(h)$ do you think will be larger?

- If true loss is much larger than empirical (training) loss, what do you think has happened?

Decomposition of true loss

- True loss: $L_{\mathcal{D}}(h_S) = \epsilon_{app} + \epsilon_{est}$

- Approximation error $\epsilon_{app} = L_{\mathcal{D}}(h^*)$
  - Min true error in the hypothesis class
  - Limitation of the choice of hypothesis class
- Estimation error $\epsilon_{est} = L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*)$:
  - Difference between approximation error and true error
  - Error due to sampling and choosing suboptimal h_S (overfitting, poor training etc)

Minimising only Empirical risk

- Suppose our sole objective was empirical risk
- That is, we want $L_S(h)$ to be as small

- You are given the training set S
- And you can choose any kind of model to get best performance on S
- How would you make your model?

Additional reason for choice of a specific $\mathcal{H}$

- Choice of $\mathcal{H}$ represents our knowledge of what is generally good for the application and the real world
  - Not just what is good for the data

- E.g. specific architectures for specific applications in vision, NLP, audio, medical diagnostics etc.

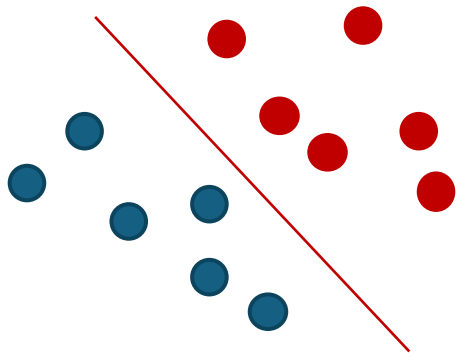
What is a good enough model?

- We are always training with a random sample of data
- We hope the training data is good – similar to the real data
  - But there is always some chance that it is not.
  - Our definition of a good model based on the training data has to be probabilistic

- We use an (ϵ, δ) guarantee: model $h_S = A(S)$ is good if
  - $\mathbb{P}[L_{\mathcal{D}}(h_S) \leq \epsilon] \geq 1 - \delta$
    - The true loss of h is smaller than ϵ
    - With probability at least $1 - \delta$
  - For small positive (ϵ, δ)

Sample complexity: how much data does it take to find a good model?

- Sample S of size m
  - We will see how sample complexity changes with $\mathcal{H}$
- Assume
  - We have a finite $\mathcal{H}$, with $|\mathcal{H}|$ models
  - And $h^* \in \mathcal{H}$ has $L_{\mathcal{D}}(h^*) = 0$ (there is a perfect model)
    - Not realistic, but helps with the analysis



- Algorithm
  - Compute the training loss $L_S(h)$ of each $h \in \mathcal{H}$
  - Find a model h_S with loss = 0

Useful relations

- For $0 < p < 1$ (e.g. p is a probability)
- $(1-p)^{\frac{1}{p}} \leq 1/e$

- Union bound:
  - If A and B are event, then: $P(A\ or\ B) \leq P(A) + P(B)$
  - Writing A and B as sets: $P(A \cup B) \leq P(A) + P(B)$

- Theorem: If $m \geq \frac{\ln(|\mathcal{H}|/\delta)}{\epsilon}$ Then $\mathbb{P}[L_{\mathcal{D}}(h_S) \leq \epsilon] \geq 1 - \delta$

Proof

- Suppose $H_B \subset \mathcal{H}$ are the bad models (i.e with $L_{\mathcal{D}}(h) > \epsilon$)
  - They are incorrect on ϵ fraction of data
- We will show that the probability of selecting a model in H_B is small
- Suppose h_B is a bad model
- Probability that h_B gets a label wrong is $> \epsilon$
- Probability that h_B gets a label right is $\leq 1 - \epsilon$
- Probability that h_B gets all m labels right is $\leq (1 - \epsilon)^m \leq e^{-\epsilon m}$

- Probability that one bad model in H_B gets everything right is $\leq e^{-\epsilon m}$
- Probability one or more models in H_B get everything right is
  - $\leq |H_B| e^{-\epsilon m} \leq |\boldsymbol{\mathcal{H}}| e^{-\epsilon m}$
  - Probability of selecting a bad model
- We want $|\boldsymbol{\mathcal{H}}| e^{-\epsilon m} \leq \delta$
  - Probability of selecting a good model is $\geq 1 - \delta$
- Exercise: Solve for m to get $m \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}$

Discussion of sample complexity

- Sample complexity $m \geq \frac{\ln\left(\frac{|\mathcal{H}|}{\delta}\right)}{\epsilon}$ or, $m \geq \frac{\ln|\mathcal{H}|+\ln\frac{1}{\delta}}{\epsilon}$
- Suffices to get high probability $(1-\delta)$ of high accuracy (small error ϵ)
  - Learning is possible from small amounts of data

- $\ln|\mathcal{H}|$: complexity to represent model class
  - what is $\log|\mathcal{H}|$?
  - Number of bits to identify a model in $\mathcal{H}$

- The proof is for finite $\mathcal{H}$ and realizability assumption (there is a zero loss model)
  - But the form of the result holds in more general scenarios

Infinite hypothesis classes and model representation

- Suppose a model has one real valued parameter
  - Then $\mathcal{H}$ is equivalent to $\mathbb{R}$

- If a model has n real valued parameters (e.g. NN with n edges)
- Then the model class is equivalent to $\mathbb{R}^n$
- That is, think of it as an n dimensional space
  - Where each point is a model
  - Each dimension is a parameter weight
- We write $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots \theta_n\}$ to represent a model
  - Assuming that the class/architecture is known
  - Sometimes $\boldsymbol{w}$ is used in place of $\boldsymbol{\theta}$

Model complexity

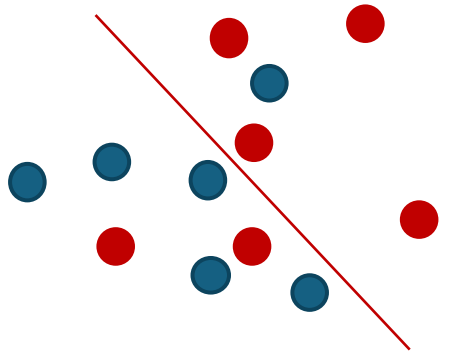
- A simple way of looking at model complexity is the number of parameters.
  - The number of values needed to identify a particular model within the model class
- More parameters: more expressivity/capacity of model
  - Capable of doing more things (complex classifications)
- Sample complexity increases with model complexity

- But model complexity or expressivity is a bit more complex than just number of parameters (we will discuss more later)

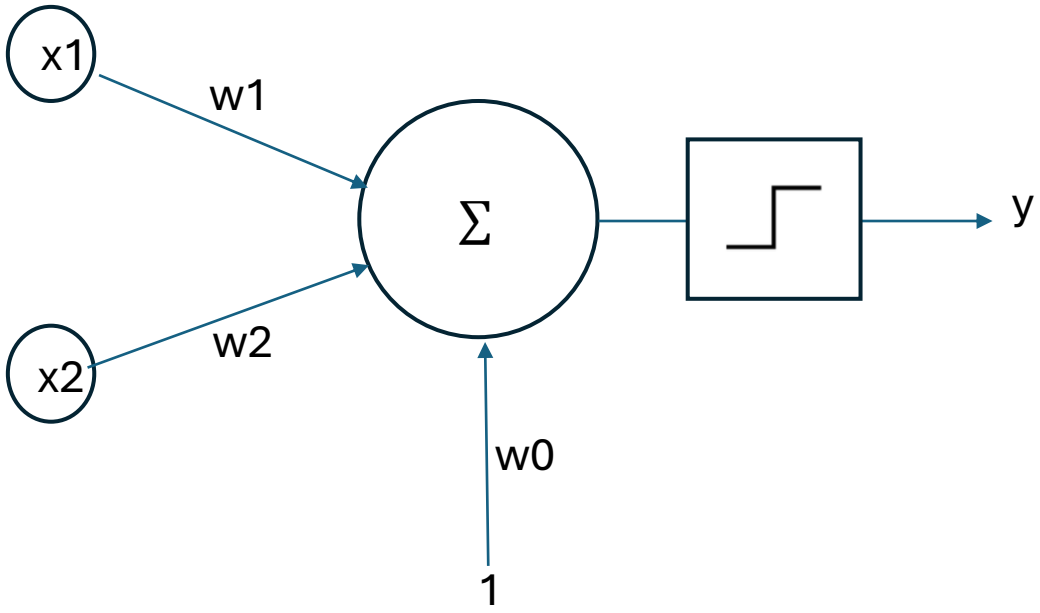
- Model space $\mathcal{H} \approx \mathbb{R}^n$

- Data space $\mathcal{X} \approx \mathbb{R}^F$
  - F is the number o features

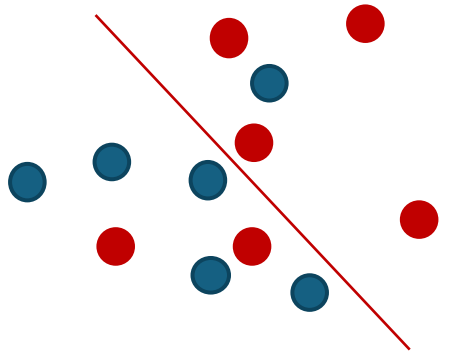
Linear classifiers: a single neuron



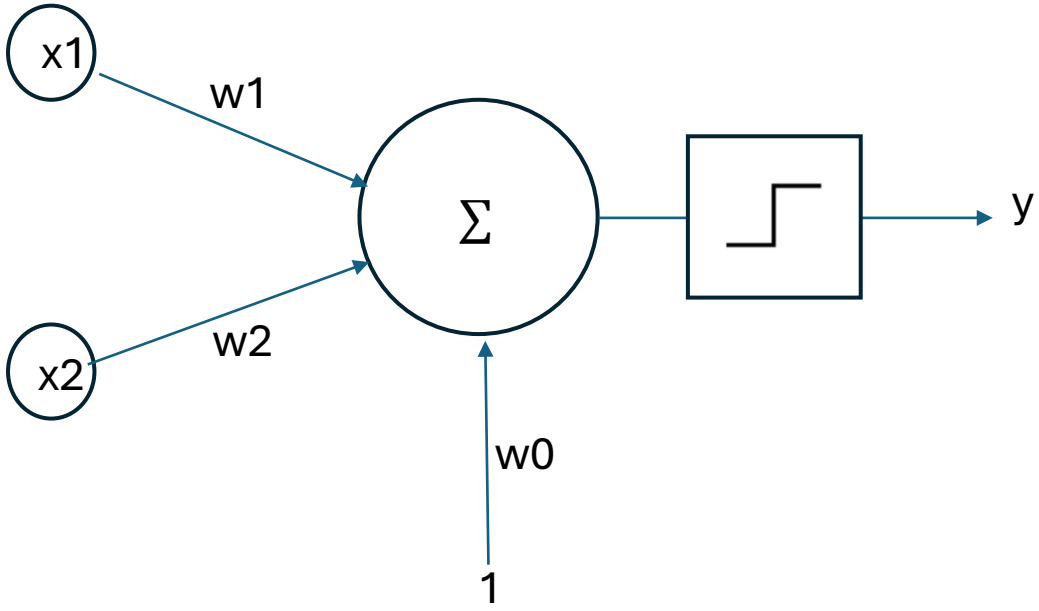
- Suppose we want to get the best possible classification by a linear space
  - Straight line in $\mathbb{R}^2$
  - Flat plane in $\mathbb{R}^3$
  - In general, a hyperplane $\mathbb{R}^{n-1}$ in $\mathbb{R}^n$
- Neuron (perceptron) with threshold activation
- $y = (w_1 x_1 + w_2 x_2 + w_0 . 1 \geq 0)$
  - Truth value 0/1 (0r, -1/+1)



Single neuron



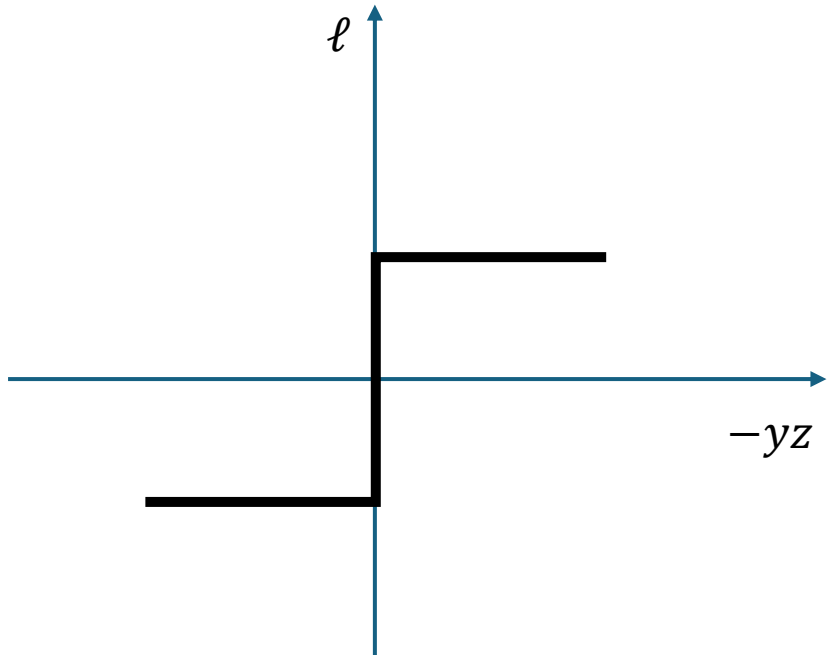
- Perceptron with threshold activation
  - $w_1, w_2, w_0 \in \mathbb{R}$
- $\hat{y} = (w_1 x_1 + w_2 x_2 + w_0 . 1 \geq 0)$
  - Truth value 0/1 (0r, -1/+1)
- We often write
  - $z = w \cdot x$



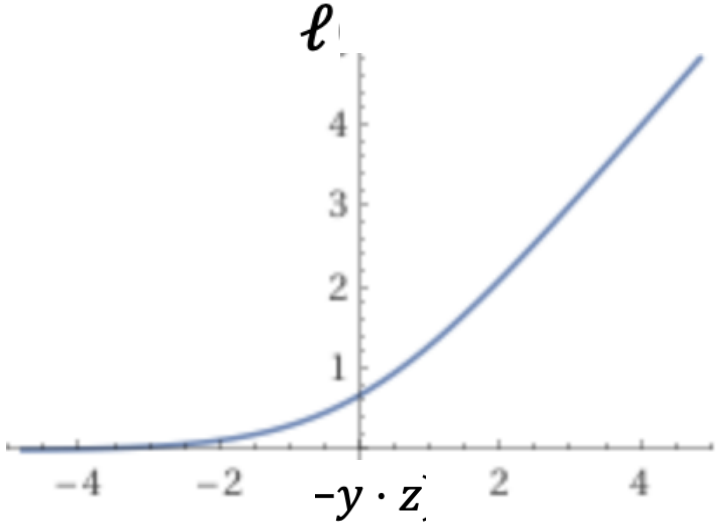
Binary loss

- $\ell =$ (1 for $\hat{y} \neq y$, and -1 for $\hat{y} = y$)

- Or, $\ell = \quad [\operatorname{sgn} z \neq \operatorname{sgn} y]$
- Or $\ell = \operatorname{sgn}(-yz\ )$



Logistic regression (used for classification!)



- The logistic loss function is:
  - $\ell\big(h_w, (x, y)\big) = \log(1 + \exp(-y \cdot z))$
  - If y, z are same sign, ℓ gets smaller with z
  - y, z are different signs, ℓ is larger with z
- Question: if x-axis was "z", what would the plot look like?

- The logistic loss is based on the logistic function $f(x) = \frac{1}{1+e^{-x}}$
  - Exercise: what does look like? (look up on wikipedia)

Logistic loss of S

- For a training dataset S
- We use the average logistic loss
  - $L_S(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\log(1+e^{-y_i z_i})$
- So, the best model w is the one with min logistic loss:
  - $argmin_{\boldsymbol{w}}\frac{1}{m}\sum_{i=1}^{m}\log(1+e^{-y_i z_i})$

- We need an algorithm to find the model with the best loss
- Question: why do we not use the binary loss?

Gradient descent

- Idea:
- Start with some random values of parameters $\boldsymbol{w}$
- Measure loss $L_S(\boldsymbol{w})$ on S
- Make small change in $\boldsymbol{w}$ so that loss $L_S(\boldsymbol{w})$ becomes smaller
  - Repeat until $L_S(\boldsymbol{w})$ no longer becomes smaller

- Remember that $\boldsymbol{w}$ is a point in $\mathbb{R}^n$
- So small change in $\boldsymbol{w}$ is moving to a nearby point in $\mathbb{R}^n$

Gradient

- Gradient (a vector derivative in multiple dimensions)
  - The direction and speed of fastest increase

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$$

  - (each w_i is a parameter or dimension of the model)

- Partial derivatives
  - Compute the derivative along each dimension, put them in a vector

Gradient descent

- Gradient is $\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$

- Gradient represents the direction in which f increases fastest
- Gradient Descent: At every step t :
  - $w^{t+1} = w^t - \eta \nabla L_S(\boldsymbol{w}^t)$
  - (Move in the direction that f decreases fastest With a step scale of η)

- After T steps, output the average vector $\bar{\boldsymbol{w}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}^t$
- Other version: output final vector $\boldsymbol{w_T}$