Generalisation

Tirtharaj Dash

Dept. of CS & IS and APPCAIR
BITS Pilani, Goa Campus

September 11, 2021

Where we are:

Summary of the previous lecture:

- Model capacity and model complexity
- Generalisation error, training error, generalisation gap
- Overfitting and underfitting
- VC dimension
- Model selection

Today, we will know how a set of techniques used to obtain a good model (that generalises well):

- Bias-variance trade-off
- Regularisation: Weight decay, Sparsity regularisation

Deep networks badly overfit.

- Deep networks are extremely flexible; even when there are far more examples than features. A curious case:
    - In 2017, a group of researchers trained deep nets on *randomly-labelled* images.
    - That means, there was no clear pattern linking inputs to outputs.
    - They found that neural net trained with gradient descent could label every image in the training set perfectly. Meaning:
        - If the labels are assigned uniformly at random and there are 10 classes, then no classifier can do better than 10% accuracy on a holdout dataset. The generalisation gap is 90%.
    - This is a curious case showing how expressive models can badly overfit, even when the classes are not clearly mapped from inputs.

Estimators, Bias, Variance I

Function estimation:

- We are interested to predict a variable $\mathbf{y}$ given an input vector $\mathbf{x}$.
- We assume that there is a function $f(\mathbf{x})$ that describes the *approximate* relationship between $\mathbf{y}$ and $\mathbf{x}$.
    - We may assume that $\mathbf{y} = f(\mathbf{x}) + \epsilon$, where ϵ stands for the part of $\mathbf{y}$ that is not predictable from $\mathbf{x}$ (recall our discussion on an ideal model)

Estimators, Bias, Variance II

- In function estimation (or, function approximation) we are interested in approximating f with a model or estimate $\hat{f}$.
  - That is, once we decide on how $\hat{f}$ should look like (recall: hypothsis space, model family), our goal is to finding out the parameters of $\hat{f}$.
  - This is simply estimating parameters $\boldsymbol{\theta}$.
  - The function estimator $\hat{f}$ is a point estimator in the function space (hypothesis space).

Estimators, Bias, Variance III

Bias:

- The bias of an estimator is defined as

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}[\hat{\boldsymbol{\theta}}_m] - \boldsymbol{\theta}$$

  Here,

  - The expectation $\mathbb{E}$ is over the data (seen as samples from a random variable).
  - $\boldsymbol{\theta}$ is the true underlying value of $\boldsymbol{\theta}$ used to define the data-generating distribution.
  - Unbiased estimator: $\text{bias}(\hat{\boldsymbol{\theta}}_m) = 0$ or $\mathbb{E}[\hat{\boldsymbol{\theta}}_m] = \boldsymbol{\theta}$

- We expect an estimator to exhibit low bias.

Estimators, Bias, Variance IV

- Example 1: Gaussian distribution estimator of the mean
  - Consider a set of samples $\{x^{(1)}, \dots, x^{(m)}\}$ that are independently and identically distributed according to a Gaussian distribution

$$p(x^{(i)}) = \mathcal{N}(x^{(i)}; \mu, \sigma^2)$$

    where $i \in \{1, \dots, m\}$.
  - We know the Gaussian p.d.f:

$$p(x^{(i)}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)} - \mu)^2}{\sigma^2}\right)$$

Estimators, Bias, Variance V

- A common estimator of the Gaussian mean parameter is known as the **sample mean** (also called the maximum likelihood estimate for mean):

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

Estimators, Bias, Variance VI

- To determine the bias of the sample mean, we are interested in calculating its expectation. That is:

$$\begin{aligned}
\mathrm{bias}(\hat{\mu}_m) &= \mathbb{E}[\hat{\mu}_m] - \mu \\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \mu \\
&= \frac{1}{m}\sum_{i=1}^{m}\left(\mathbb{E}[x^{(i)}]\right) \\
&= \frac{1}{m}\sum_{i=1}^{m}\mu - \mu \\
&= \mu - \mu = 0
\end{aligned}$$

- Conclusion: The sample mean is an unbiased estimator of Gaussian mean parameter.

Estimators, Bias, Variance VII

- Example 2: Estimator of the variance of a Gaussian distribution
  - Let us first try *sample variance* (also called maximum likelihood estimate for variance), defined as

$$\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2$$

  - We are computing

$$\mathrm{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2$$

  - The first term on the right evaluates to:

$$\begin{aligned}\mathbb{E}[\hat{\sigma}_m^2] &= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] \\ &= \frac{m-1}{m}\sigma^2\end{aligned}$$

  - Therefore, $\mathrm{bias}(\hat{\sigma}_m^2) = -\frac{\sigma^2}{m}$. Conclusion: Biased estimator.

Estimators, Bias, Variance VIII

- Homework: Show that the following is an unbiased estimator for the Gaussian variance parameter

$$\hat{\sigma}_m^2 = \frac{1}{m-1} \sum_{i=1}^{m} \left(x^{(i)} - \hat{\mu}_m \right)^2$$

This is called the unbiased sample variance estimator.

Estimators, Bias, Variance IX

Variance:

- We might want to consider how much we expect the estimator to vary as a function of the data sample.
- This is simply, $\mathrm{Var}(\hat{\theta})$.
- The square root of the variance is called the *standard error*.
- Variance or standard error provides a measure of how we would expect the estimate we compute from data to vary as we independently resample the dataset from the underlying data-generating process.
- We expect an estimator to exhibit low variance.

Estimators, Bias, Variance X

- When we compute any statistic using a finite number of samples, our estimate of the true underlying parameter is uncertain. How?
  - We could have obtained other samples from the same distribution and their statistics would have been different.
- The expected degree of variation in any estimator is a source of error that we want to quantify.
- The standard error of the mean is given by:

$$\mathrm{SE}(\hat{\mu}_m) = \sqrt{\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}$$

- Notice that for large m this approximation is quite reasonable.

Bias-Variance Trade-off I

- We saw that bias and variance measure two different sources of error in an estimator.
- Bias measures the expected deviation from the true value of the function or parameter.
- Variance measures the deviation from the expected estimator value that any particular sampling of the data is likely to cause.

Bias-Variance Trade-off II

- What happens when we are given a choice between two estimators, one with more bias and one with more variance?
- We can compare the mean squared error (MSE) of the estimates:

$$\begin{aligned}\mathrm{MSE} &= \mathbb{E}[(\hat{\theta}_m - \theta)^2] \\ &= \mathrm{bias}(\hat{\theta}_m)^2 + \mathrm{Var}(\hat{\theta}) + \text{irreducible error}\end{aligned}$$

The irreducible error is actually $\mathrm{Var}(\epsilon)$, which is caused due to noise ϵ.

Bias-Variance Trade-off III

- The relationship between bias and variance is tightly linked to the machine learning concepts of capacity, underfitting and overfitting.
- When the generalisation error is measured by MSE (where bias and variance are meaningful components of generalisation error)
  - increasing the capacity tends to increase variance decreases bias
  - decreasing the capacity tends to decrease variance and increase bias

Bias-Variance Trade-off IV

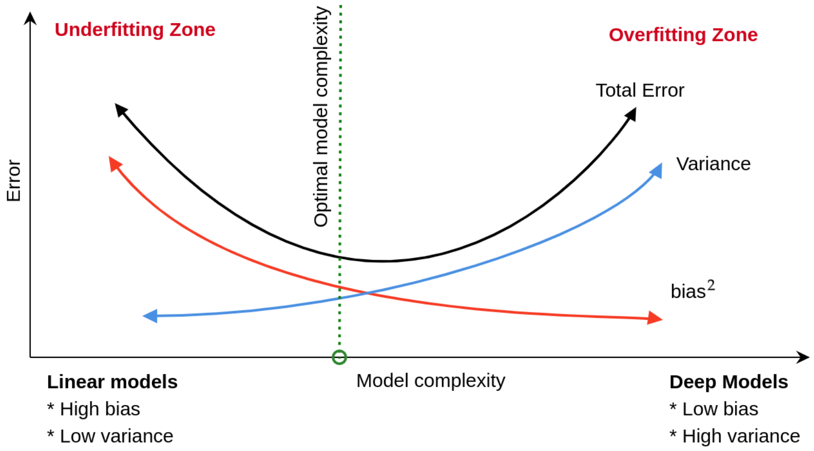
Overfitting (from the prism of bias-variance):

- We now subscribed to a popular notion of *bias-variance tradeoff**.
- We now have a link between model capacity and bias-variance.
- That is:
  - Simpler functions lead to low variance models (but, high bias)
  - Complex functions lead to low bias models (but, high variance)

  (See next for visualising this concept)

Bias-Variance Trade-off V

- The bias-variance trade-off curve:



Bias-Variance Trade-off VI

- We have to remember that: We can reduce the variance but not the bias. (Why?)
  - Bias computation needs the true parameters ($\boldsymbol{\theta}$) or the true function (f).
  - How will we know $\boldsymbol{\theta}$ or f?
  - If we knew either of these two, there is NO machine learning. Is there?

Regularisation I

- How to mitigate overfitting?
  - Collect more training data
  - Issue: costly, time-consuming, maynot be possible
- Alternative way to mitigate overfitting: Reduce number of parameters in a parametric machine learning model
- That is, if we can reduce the number of weights in our neural network, then we are essentially making the model simpler.
- But, how do we "reduce" these weights?
  - Ans: Making some of them 0
  - But, how do we make some of them 0?
    - We force this by modifying the loss function of our model
    - That is, adding a pentaly term to our loss function that penalises models with more parameters.

Regularisation II

- That is, the total loss is becomes:

  $$L(\mathbf{w}) = L_{problem}(\mathbf{w}) + \lambda L_{complexity}(\mathbf{w})$$

  where $L_{problem}$ refers to the problem-specific loss function or performance metric term (e.g. cross-entropy for binary classification, MSE for regression, etc.) and $L_{complexity}$ is the complexity penalty term.
- Further, inclusion of $L_{complexity}$ imposes on the assumption that the prior distribution of $\mathbf{w}$ is $\mathcal{N}(0, \sigma^2)$.

Regularisation III

- λ is a regularization parameter represents relative importance of $L_{complexity}$:
  - $\lambda = 0$: complexity penalty is ignored.
  - $\lambda =$ very large: the constraint imposed by the complexity penalty is by itself sufficient to specify the network, which is another way of saying that the training examples are unreliable.
  - Usually, $\lambda \in [0, 1]$. And, this can be "tuned" using a validation set.

Regularisation IV

- In the simplest form, the term $J_c(\mathbf{w}, \mathbf{b})$ could be written as

$$J_c(\mathbf{w}, \mathbf{b}) = ||f(\mathbf{w}, \mathbf{b})||^2, \tag{1}$$

  where $|| \cdot ||^2$ is a squared norm.
- The function f is considered as a linear function of $\mathbf{w}, \mathbf{b}$ to make the term simple. For example, $f(\mathbf{w}, \mathbf{b}) = \sum (A\mathbf{w} + B\mathbf{b})$

Regularisation V

- Weight-decay operates by forcing some of the synaptic weights in the network to take values approximately to minimal as low as 0, while permitting other weights to retain their relatively large values.
- Accordingly, the weights of the network are grouped roughly into two categories:
  - weights that have a significant influence on the network's performance;
  - weights that have practically little or no influence on the network's performance.
- The weights in the latter category are referred to as *excess* weights.

Regularisation VI

- In the absence of complexity regularization, these weights result in poor generalization by virtue of their high likelihood of taking on completely arbitrary values or causing the network to overfit the data in order to produce a slight reduction in the training error.
- The use of complexity regularization encourages the excess weights to assume values close to zero and thereby improve generalization.
- This way of representing the complexity term in Equation (1) is called as the **weight-decay procedure**.
- Common penalty terms are L_1 and L_2 norm of the weight tensor.

Weight Decay I

Also called L_2 regularisation:

- The model $f = 0$ is the *simplest* function:
  - It always outputs 0 (for a classification problem).
  - This means, in some sense, all the parameters ($\mathbf{w}$) of the model are 0.
- L_2 regularisation enforces the parameters of our model to be pushed to 0.
- Here, $L_{complexity}$ takes the form:

$$L_{complexity} = \frac{1}{2}||\mathbf{w}||^2$$

where $||\cdot||$ is the Euclidean distance of $\mathbf{w}$ to the origin ($\mathbf{0}$) and represents the Frobenius norm if $\mathbf{w}$ is a weight matrix (or, tensor).

(Details on vector, matrix, and tensor operations are already provided as extra reading material in our course page.)

Weight Decay II

- This kind of complexity term ensures that if our weight vector grows too large, our learning algorithm might focus on minimizing the weight norm.
- So, our total loss is now:

$$L = L_{problem} + \frac{\lambda}{2}||\mathbf{w}||^2$$

- Gradient descent update is then:

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}, \text{ for all } i$$

Weight Decay III

- This becomes:

$$w_i = w_i - \eta \frac{\partial L_{problem}}{\partial w_i} - \eta \lambda w_i$$

  or, by rearranging:

$$w_i = (1 - \eta\lambda) w_i - \eta \frac{\partial L_{problem}}{\partial w_i}$$

- Since $\eta \in (0, 1]$ and $\lambda \in [0, 1]$, we have $0 \leq \eta\lambda \leq 1$.
- The term $\eta\lambda w_i$ causes the weights (w_is) to decay in proportion to their size and therefore ensuring that we obtain models (functions) with fewer parameters.

Weight Decay IV

- Implementation of L_2 regularisation in deep networks:
  - Let ℓ denote the layer index, and ℓ_{max} denote the number of layers (depth) in a deep network
  - Let $m^{(\ell)}$ denote the size of layer ℓ (number of neurons at some layer ℓ)
  - Let $\mathbf{W}^{(\ell)} \in \mathbb{R}^{m^{(\ell-1)} \times m^{(\ell)}}$ denote the weight matrix for layer ℓ
  - L_2 regularisation with hyperparameter λ is implemented as:

$$s_2 = \frac{\lambda}{2} \sum_{\ell=1}^{\ell_{max}} ||\mathbf{W}^{(\ell)}||_F^2$$

  - Note that $s_2 \in \mathbb{R}$ (a scalar).
  - Frobenius norm of a matrix is simply the L_2 norm applied after flattening the matrix into a vector.

Weight Decay V

- Some visible properties of L_2:
  - The decay of weights (or, shrinking, reduction) is *multiplicative* and proportional to the value of the weight.
  - It is faster for large weights to shrink than the small weights.
  - That is, the decay de-accelerates as the weights get smaller.

Weight Decay VI

- We state the following without formal proof:
    - In probabilistic terms, if we assumed a prior belief that weights take values from a Gaussian distribution with mean zero then L_2 regularisation is the right approach to regularise the model. (Recall our discussion on deep learning from a Bayesian perspective).

Sparsity regularisation I

Also called L_1 regularisation:

- Similar to L_2 regularisation, L_1 takes also leads to smaller weights.
- The loss here takes the form:

$$L = L_{problem} + \lambda|\mathbf{w}|$$

- Gradient descent update is then:

$$\mathbf{w} = \mathbf{w} - \eta\lambda \ \mathrm{sgn}(\mathbf{w}) - \eta\frac{\partial L_{problem}}{\partial \mathbf{w}}$$

  where, $\mathrm{sgn}(\cdot)$ is the sign function:

$$\mathrm{sgn}(w) = \begin{cases} +1 & w > 1 \\ 0 & w = 0 \\ -1 & w < 1 \end{cases}$$

Sparsity regularisation II

- Implementation of L_1 regularisation in deep networks:
  - Let ℓ, ℓ_{max}, $m^{(\ell)}$, $\mathbf{W}^{(\ell)}$ be as before.
  - L_1 regularisation with hyperparameter λ is implemented as:

$$s_1 = \lambda \sum_{\ell=1}^{\ell_{max}} |\mathbf{W}|$$

  - Note that $s_1 \in \mathbb{R}$ (a scalar).
  - s_1 is computed by summing over the vectors obtained after flattening the matrices.

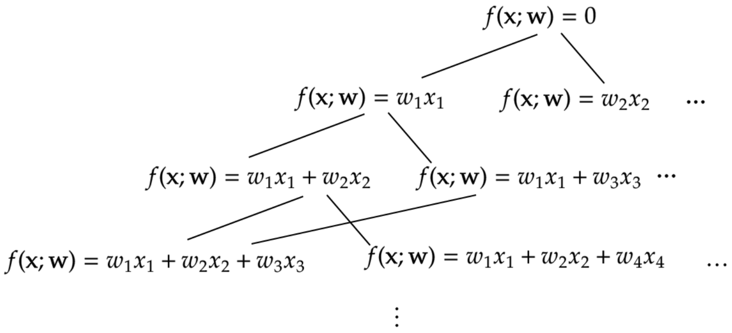
Sparsity regularisation III

- L_1 operates differently as compared to L_2, in that:
  - The weights here are reduced by a fixed amount in every iteration (*substractive*), irrespective of the value of the weights.
  - For larger weights: L_1 is slower than L_2.
  - For smaller weights: L_1 is faster than L_2.
    - This results in models *sparse* model: a model with many 0-ed weights and a few large weights left over.
    - This is the reason L_1 regularisation is also called *sparsity regularisation*.

See also: a visual simulation at Google's ML crash course showing the effects of regularisations.

Sparsity regularisation IV

A generality ordering over functions:



$f(\mathbf{x};\mathbf{w}) = 0$
$f(\mathbf{x};\mathbf{w}) = w_1x_1$
$f(\mathbf{x};\mathbf{w}) = w_2x_2$
...
$f(\mathbf{x};\mathbf{w}) = w_1x_1 + w_2x_2$
$f(\mathbf{x};\mathbf{w}) = w_1x_1 + w_3x_3$
...
$f(\mathbf{x};\mathbf{w}) = w_1x_1 + w_2x_2 + w_3x_3$
$f(\mathbf{x};\mathbf{w}) = w_1x_1 + w_2x_2 + w_4x_4$
...
$\vdots$

This is a rought sketch of the lattice of the models showing how complexity grows with number of parameters in a model (top-to-bottom). The diagram is shown for the linear model family.