Deep Learning Homework 1

Björn Pettersson, bjpe900 2025-81022

27-09-2025

Problem 1 : Bias-Variance Trade-off and Overfitting (10 Points)

1.1

Bias is error due to overly simplstic assumptions and varaicne is the error due to sensitivity to fluctations in the traning data, our assumptions is essentially here the degree polynomial we try to fit, if we fit a very low degree polynomical such as in model1, getting high errors this suggests high bias. Ie our assumption that the data can be fit to a linear model is likely wrong, but we see a small train thest gap for model1, meaning the number of ways we can fit this linear equation to the data is quite small, in the set of possible fits we have low variance. This means the model generalises well, we approximate the true error by the test score. The difference in error we see on the specific model we trained on, vs the true error is a good approximation for weather the lower test error is due to our spefic fit, ie if we have a lot of possible specific fits this difference is likely higher, ie we have high variance. This is the case for model2. The low train error however suggests that our model fits the data very well, ie we have low bias, ie our assumption that the data can be fit by a 12 dgree polynomial is proabably right.

1.2

Model 1 (underfitting)

Both traning and test error are quite large but the difference is small. The large size of the error suggests that our model might not be complex enough to fit the data well, we may be underfitting, however the small difference to the test error suggests that the little we do pick up from the data gneeralises quite well, ie we are not overfitting.

Model2 (overfitting)

Train error is very small, test error is very large. Model complexity is high, this suggests that our very complex model fits not only the signal in the data

but also the noise, this makes it bad at generalising, which is why we see much higher test error. This suggests the model might be overfitting.

1.3

For model 2 we hav low bias high variance, and overfitting. We want to apply regualisation to reduce the overfitting.

Reduce the polynomial degree

This is the most obious way to tackle the overfitting, because it does not require more data, ie if we had more data adding it would also be ideal to deal with overfitting, but usually we use the data we have so this may not be the most simple way. Our results suggest we would prefer something between model 1 underfitting and model 2 overfitting in terms of polynomical complexity since niether of these are optimal. Reducing the model complexity to something in between model 1 and 2 may be more ideal.

Early stopping

Another simple regualisation tecnique may be to simply stop traning earler before the model starts to overfit, this way we can keep our initial architecutre and initial data while mittigating the problem.

Cross validation

This is good if we have very little data, and want maximum feedback for hyper parameter tunring that may reduce overfitting. Instead of one train test split we do k train test splits. Then we can use the feedback from k tranings sessions for experimenting with hyper parameters and observing train test error. This gives us more feedback given our limited data. Then we can apply traning on the whole dataset if we want using these parameters.

Problem 2 : MLE/MAP (20 Points)

2.1

We find the log likelihood as:

$\ell(\lambda) = \sum_{i=1}^{\mathrm{n}}(-\lambda + x_i ln\lambda)$

We maximise the log likelihood by differentiating adn setting to 0.

$$\frac{d\ell}{d\lambda} = -n + \frac{\sum_u x_i}{\lambda} = 0$$

We get the estimator:

$$\hat{\lambda}_{MLE} = \frac{1}{n}\sum_{i=1}^{\ddot{\mathrm{n}}} x_i = \bar{x}$$

2.1

We have some possion distribution, and we want to fit it to our data using the paramter λ.

$$P(X = x_i \mid \lambda) = f(x_i)$$

we ahve iid samples $x_1, ...x_n$, we multiply the probability of seeing each outcome xi based on our $f(x_i)$ function, that depeneds on our lambda, a better λ fit gives a higher join probability of observing our outcomes.

$$L(\lambda) = \prod_{i=1}^{\mathrm{n}} f(x_i) = \prod_{i=1}^{\mathrm{n}} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$$

We find the log likelihood (the max of this funciton is the same as for the likelihood)

$$\begin{aligned}
\ell(\lambda) &= \sum_{i=1}^{\mathrm{n}} ln(\frac{e^{-\lambda}\lambda^{x_i}}{x_i!}) \\
&= \sum_{i=1}^{\mathrm{n}} ln(e^{-\lambda}) + ln(\lambda^{x_i}) - ln(x_i!) \\
&= \sum_{i=1}^{\mathrm{n}} -\lambda + x_i ln(\lambda) - ln(x_i!)
\end{aligned}$$

We maximise the likelihood that our data fits the distirbution given lambda:

$$\begin{aligned}
\frac{\partial \ell}{\partial \lambda} &= \sum_{i=1}^{\mathrm{n}} -1 + \lambda^{-1} x_i \\
&= -n + \lambda^{-1} n n^{-1} \sum_{i=1}^{\mathrm{n}} x_i \\
&= -n + \lambda^{-1} n \bar{x}
\end{aligned}$$

Set to 0:

$$\begin{aligned}
-n + \lambda^{-1} n \bar{x} &= 0 \\
\frac{1}{\lambda} &= \frac{n}{n\bar{x}} \\
\frac{1}{\lambda} &= \frac{1}{\bar{x}} \\
\hat{\lambda} &= \bar{x}
\end{aligned}$$

2.2

For our function $f(\lambda) = \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$

Given from our previous solution our estimator is $\hat{\lambda}_{MLE} = \bar{x}$

a)

We calculate it for each of the three classes given our data:

- $\hat{\lambda_1} = (3+7)/2 = 5$
- $\hat{\lambda_2} = (8+12)/2 = 10$
- $\hat{\lambda_3} = (6+14)/2 = 10$

b)

Test points

1: $pt = (10, 1)$, $\lambda = 5$:

$$f(\lambda) = \frac{e^{-5}5^{10}}{10!} = 0.01813279$$

2: $pt = (10, 2)$, $\lambda = 10$:

$$f(\lambda) = \frac{e^{-10}10^{10}}{10!} = 0.12511004$$

3: $pt = (10, 3)$, $\lambda = 10$:

$$f(\lambda) = \frac{e^{-10}10^{10}}{10!} = 0.12511004$$

Probabilities

$$\begin{aligned}
P(y = i \mid x) &= \frac{f_i(x)P(y=i)}{\sum_j f_j(x)P(y=j)} \\
&= \frac{f_i(x)P(y=i)}{\sum_j f_j(x)P(y=j)} \\
&= \frac{f_i(x)}{f_1(x) + f_2(x) + f_3(x)} \\
&= \frac{f_i(10)}{f_1(10) + f_2(10) + f_3(10)} \\
&= \frac{f_i(10)}{0.01813279 + 0.12511004 + 0.12511004} \\
&= \frac{f_i(10)}{0.26835287}
\end{aligned}$$

For the y values

1.

$$P(y = 1 \mid x = 10) = \frac{0.01813279}{0.26835287} = 0.0676 \approx 6.76\%$$

2.

$$P(y = 2 \mid x = 10) = \frac{0.12511004}{0.26835287} = 0.4662 \approx 46.62\%$$

3.

$$P(y = 3 \mid x = 10) = \frac{0.12511004}{0.26835287} = 0.4662 = 46.62\%$$

c)

We have a tie in probability for class 2 and 3, but we see that the test point is more likely to belong there than to the first class.

2.3

a)

$$P(x = 10 \mid y = i) = f_i(10)$$

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

We know:

- $P(y = 1) = \frac{1}{4}$
- $P(y = 2) = \frac{1}{4}$
- $P(y = 3) = \frac{1}{2}$
- $P(x = 10) = \sum_{j=1}^{3} P(x = 10 \mid y = j)P(y = j) =$

$$P(y = i \mid x = 10) = \frac{P(x = 10 \mid y = i)P(y = i)}{P(x = 10)}$$

For y=1 with prior:

$$\begin{aligned} P(y = 1 \mid x = 10) &= \frac{f_1(10) \cdot 0.25}{0.25f_1(10) + 0.25f_2(10) + 0.5f_3(10)} \\ &= \frac{0.01813279 \cdot 0.25}{0.25 \cdot 0.01813279 + 0.25 \cdot 0.12511004 + 0.5 \cdot 0.12511004} \\ &= 0.04608513163 \end{aligned}$$

For y=2 with prior:

$$\begin{aligned} P(y = 2 \mid x = 10) &= \\ &= \frac{0.12511004 \cdot 0.25}{0.25 \cdot 0.01813279 + 0.25 \cdot 0.12511004 + 0.5 \cdot 0.12511004} \\ &= 0.3179716228 \end{aligned}$$

For y=3 with prior:

$$\begin{aligned} P(y=3 \mid x=10) = \\ =&\frac{0.12511004 \cdot 0.25}{0.25 \cdot 0.01813279 + 0.25 \cdot 0.12511004 + 0.5 \cdot 0.12511004} \\ =&0.3179716228 \end{aligned}$$

b)

It is again a tie between class 2 and 3.

Problem 3 Logistic Regression

3.1

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

we have loss:

$$\begin{aligned} l(w) &= \sum_{x,y \in D} logP(y \mid x, w) \\ &= \sum_{x \in D} \sum_{y \in D} logP(y \mid x, w) \end{aligned}$$

For binary y:

$$P(y=1 \mid x, w) = \sigma(w^T x)$$

$$P(y=0 \mid x, w) = 1 - \sigma(w^T x)$$

$$l(w) = \sum_{x \in D} log\left(y[\sigma(w^T x)] + (1-y)[1-\sigma(w^T x)]\right)$$

sicne x also is binar for a single z we have:

$$l(w) = log\left(y[\sigma(w^T x)] + (1-y)[1-\sigma(w^T x)]\right)$$

For each data point: $z_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3}$

- Pt1:$(1,0,0;0) : z_1 = w_0 + w_1 * 1 + w_2 * 0 + w_3 * 0 = w_0 + w_1$
- Pt2:$(0,1,0;0) : z_2 = w_0 + w_2$
- Pt3:$(0,0,1;1) : z_3 = w_0 + w_3$

- Pt4:$(1,1,1;1) : z_4 = w_0 + w_1 + w_2 + w_3$

Loss values

pt1

$$l(w) = log\left(0[\sigma(w_0 + w_1)] + (1-0)[1 - \sigma(w_0 + w_1)]\right) = log[1 - \sigma(w_0 + w_1)]$$

pt2

$$l(w) = log[1 - \sigma(w_0 + w_2)]$$

pt3

$$l(w) = log\left(1[\sigma(w_0 + w_1)] + (1-1)[1 - \sigma(w_0 + w_1)]\right) = log[\sigma(w_0 + w_3)]$$

pt2

$$l(w) = log[1 - \sigma(w_0 + w_1 + w_2 + w_3)]$$

For our whole dataset:

$$\begin{aligned} L(w) = log[1 - \sigma(w_0 + w_1)] + log[1 - \sigma(w_0 + w_2)] + \\ log[\sigma(w_0 + w_3)] + log[1 - \sigma(w_0 + w_1 + w_2 + w_3)] \end{aligned}$$

3.2

For a single data point (x, y), the derivative of the log-likelihood w.r.t. w_j is

$$\frac{\partial \ell}{\partial w_j} = (y - \sigma(z))x_j,$$

where $x_0 = 1$ for the bias term w_0. For the data points:

$$\begin{aligned}
&\text{Point 1: } (x_1, y_1) = (1,0,0;0) \quad \Longrightarrow \frac{\partial \ell_1}{\partial w_0} = -\sigma(w_0 + w_1), \quad \frac{\partial \ell_1}{\partial w_1} = -\sigma(w_0 + w_1), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \frac{\partial \ell_1}{\partial w_2} = 0, \quad \frac{\partial \ell_1}{\partial w_3} = 0 \\
&\text{Point 2: } (0,1,0;0) \quad \Longrightarrow \frac{\partial \ell_2}{\partial w_0} = -\sigma(w_0 + w_2), \quad \frac{\partial \ell_2}{\partial w_2} = -\sigma(w_0 + w_2) \\
&\text{Point 3: } (0,0,1;1) \quad \Longrightarrow \frac{\partial \ell_3}{\partial w_0} = 1 - \sigma(w_0 + w_3), \quad \frac{\partial \ell_3}{\partial w_3} = 1 - \sigma(w_0 + w_3) \\
&\text{Point 4: } (1,1,1;1) \quad \Longrightarrow \frac{\partial \ell_4}{\partial w_0} = 1 - \sigma(z_4), \quad \frac{\partial \ell_4}{\partial w_1} = 1 - \sigma(z_4), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \frac{\partial \ell_4}{\partial w_2} = 1 - \sigma(z_4), \quad \frac{\partial \ell_4}{\partial w_3} = 1 - \sigma(z_4)
\end{aligned}$$

Total:

$$\frac{\partial \ell}{\partial w_0} = -\sigma(w_0 + w_1) - \sigma(w_0 + w_2) + (1 - \sigma(w_0 + w_3)) + (1 - \sigma(z_4))$$
$$\frac{\partial \ell}{\partial w_1} = -\sigma(w_0 + w_1) + (1 - \sigma(z_4))$$
$$\frac{\partial \ell}{\partial w_2} = -\sigma(w_0 + w_2) + (1 - \sigma(z_4))$$
$$\frac{\partial \ell}{\partial w_3} = (1 - \sigma(w_0 + w_3)) + (1 - \sigma(z_4))$$

where $z_4 = w_0 + w_1 + w_2 + w_3$.

3.3

(1)Predicted probabilities

For each training example, $z = w_0 + w_1x_1 + w_2x_2 + w_3x_3$. At $w^{(0)} = (0, 0, 0, 0)$, all $z_i = 0$. Thus:

$$\hat{y}_1 = \sigma(w_0 + w_1) = \sigma(0) = 0.5$$
$$\hat{y}_2 = \sigma(w_0 + w_2) = \sigma(0) = 0.5$$
$$\hat{y}_3 = \sigma(w_0 + w_3) = \sigma(0) = 0.5$$
$$\hat{y}_4 = \sigma(w_0 + w_1 + w_2 + w_3) = \sigma(0) = 0.5$$

So the predicted probabilities are:

$$\hat{y} = (0.5, 0.5, 0.5, 0.5).$$

(2)numerical gradient

The gradient formula for logistic regression is

$$\frac{\partial \ell}{\partial w_j} = \sum_{i=1}^{4} (y_i - \hat{y}_i) x_{ij},$$

with $x_{i0} = 1$ for the bias term w_0. Compute each component:

$$\frac{\partial \ell}{\partial w_0} = \sum_i (y_i - \hat{y}_i) \cdot 1 = (0 - 0.5) + (0 - 0.5) + (1 - 0.5) + (1 - 0.5) = 0$$
$$\frac{\partial \ell}{\partial w_1} = \sum_i (y_i - \hat{y}_i) x_{i1} = (0 - 0.5) \cdot 1 + (0 - 0.5) \cdot 0 + (1 - 0.5) \cdot 0 + (1 - 0.5) \cdot 1 = 0$$
$$\frac{\partial \ell}{\partial w_2} = \sum_i (y_i - \hat{y}_i) x_{i2} = (0 - 0.5) \cdot 0 + (0 - 0.5) \cdot 1 + (1 - 0.5) \cdot 0 + (1 - 0.5) \cdot 1 = 0$$
$$\frac{\partial \ell}{\partial w_3} = \sum_i (y_i - \hat{y}_i) x_{i3} = (0 - 0.5) \cdot 0 + (0 - 0.5) \cdot 0 + (1 - 0.5) \cdot 1 + (1 - 0.5) \cdot 1 = 1$$

(3)numerical gradient at $w^{(0)}$

$$\nabla \ell(w^{(0)}) = \left(\frac{\partial \ell}{\partial w_0}, \frac{\partial \ell}{\partial w_1}, \frac{\partial \ell}{\partial w_2}, \frac{\partial \ell}{\partial w_3} \right) = (0, 0, 0, 1)$$

Predicted probabilities for all points: $\hat{y} = (0.5, 0.5, 0.5, 0.5)$ Gradient at $w^{(0)} = (0, 0, 0, 0)$: $\nabla \ell(w^{(0)}) = (0, 0, 0, 1)$

Problem 4 Multi layer perceptrons

4.1

AND

$$y = z(W^2(W^1 x + b^1) + b^2)$$

for function:

$$z(x) = \begin{cases} 1 & x \geq 0 \\ 0 & else \end{cases}$$

We can see if b1, b2 are just -1 we will have 0 for 1,1 and
For AND we expect:

- $(0, 0) \to 0$
- $(0, 1) \to 0$
- $(1, 0) \to 0$
- $(1, 1) \to 1$

We have linear network

With and we could consider a super simple transformation, [1,1]^T, this gives 0 or 1 for all inputs except [1,1] for which it gives 2. So all we need to do is shift it downward so that max is 0.

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We subtract so that max is 0:

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - [2]$$

Now can we write our network as this transformation?
We essenatilly have:

$$y = z(\begin{bmatrix} 1 & 1 \end{bmatrix} (\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}) + [-2])$$

$$y = z(W^2(W^1x + b^1) + b^2)$$

This should produce the same AND output.

OR:

For AND we expect:

- $(0, 0) \to 0$
- $(0, 1) \to 1$
- $(1, 0) \to 1$
- $(1, 1) \to 1$

Lets try to find the simples way to solve this, with a simple sum we see that OR will create a sum larger than 1 where we expect output 1, and 0 when we expect output 0, so again we can just subtract enough to make the threshold be 1, creating output 0:

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We simply subtract 1

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [-1]$$

Now we should have for [1,1]->1, and [1,0]->0 which both output 1 via z. And for [0,0]->-1 which ouput 0.

Now can we write our network as this transformation?

We essenatilly have:

$$y = z(\begin{bmatrix} 1 & 1 \end{bmatrix} (\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}) + [-1])$$

$$y = z(W^2(W^1x + b^1) + b^2)$$

XOR

We can not implement XOR wth a linear network XOR is not linearly separable.

4.2

We use a two-layer network with a 2-unit hidden layer and a single output unit. The hidden activations use ReLU and the final output is hard-thresholded:

$$h = \mathrm{ReLU}\big(W^{(1)}x + b^{(1)}\big),$$
$$y = z\big(W^{(2)}h + b^{(2)}\big), \qquad z(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}.$$

Construction idea. Let $s = x_1 + x_2 \in \{0, 1, 2\}$. Define hidden units

$$h_1 = \mathrm{ReLU}(s) = s, \qquad h_2 = \mathrm{ReLU}(s-1) = \max(0, s-1),$$

so the pair (h_1, h_2) equals

s	0	1	2
h_1	0	1	2
h_2	0	0	1

We seek an affine map $o = a\,h_1 + b\,h_2 + c$ such that the threshold $z(o)$ equals XOR, i.e.

$$z(o) = \begin{cases} 0 & s = 0 \\ 1 & s = 1 \\ 0 & s = 2 \end{cases}.$$

One feasible choice is

$$a = 0.5, \quad b = -1, \quad c = -0.5,$$

since

$$\begin{aligned} &s = 0: \; o = 0.5 \cdot 0 + (-1) \cdot 0 - 0.5 = -0.5 < 0 \\ &s = 1: \; o = 0.5 \cdot 1 + (-1) \cdot 0 - 0.5 = 0 \quad \text{(threshold gives 1)} \\ &s = 2: \; o = 0.5 \cdot 2 + (-1) \cdot 1 - 0.5 = -0.5 < 0. \end{aligned}$$

Explicit weights and biases. Choose hidden weights and biases so that both hidden units compute the stated linear functions of x:

$$W^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad b^{(1)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}.$$

Thus for input $x = (x_1, x_2)^\top$,

$$\begin{aligned} h_1 &= \mathrm{ReLU}(1 \cdot x_1 + 1 \cdot x_2 + 0) = \mathrm{ReLU}(s) = s, \\ h_2 &= \mathrm{ReLU}(1 \cdot x_1 + 1 \cdot x_2 - 1) = \mathrm{ReLU}(s-1). \end{aligned}$$

Set the output weights and bias as row vector and scalar:

$$W^{(2)} = \begin{bmatrix} 0.5 & -1 \end{bmatrix}, \qquad b^{(2)} = -0.5.$$

Then the pre-threshold output is $o = W^{(2)}h + b^{(2)} = 0.5h_1 - 1 \cdot h_2 - 0.5$, and final prediction $y = z(o)$.

Verification on all four inputs. Compute s, $h = (h_1, h_2)$, o and y.

x	$s = x_1 + x_2$	$h = (h_1, h_2)$	$o = 0.5h_1 - 1 \cdot h_2 - 0.5$	$y = z(o)$
$(0,0)$	0	$(0,0)$	$0.5 \cdot 0 - 1 \cdot 0 - 0.5 = -0.5$	0
$(1,0)$	1	$(1,0)$	$0.5 \cdot 1 - 1 \cdot 0 - 0.5 = 0$	1
$(0,1)$	1	$(1,0)$	$0.5 \cdot 1 - 1 \cdot 0 - 0.5 = 0$	1
$(1,1)$	2	$(2,1)$	$0.5 \cdot 2 - 1 \cdot 1 - 0.5 = -0.5$	0

The outputs y are $[0, 1, 1, 0]$, which equals $\text{XOR}(x_1, x_2)$. Thus the network implements XOR.