UNIVERSITY OF BRISTOL

January 2024 Examination Period

School of Engineering, Mathematics and Technology

Third Year Examination for the Degree of
Bachelor of Science and Master of Engineering

EXAM PAPER CODE: EMAT-31530J
UNIT CODE: EMAT-31530

Introduction to AI

TIME ALLOWED:
2 Hours

Answers to EMAT-31530: Introduction to AI

Answer all 15 questions.
All questions should be answered on the MCQ sheet.
Each question has exactly *one* correct answer.
All answers will be used for assessment.
The maximum for this paper is *100 marks*.
The exam is closed-book (so no additional materials are allowed).
You may write workings out on the exam paper, and blank pages are provided at the end for this purpose. These workings out will not be collected or marked. You must enter your answers on the provided answer sheet only.

Other Instructions:

You may use a calculator.
Only non-programmable calculators may be used.

Using the computer marked sheets:

All questions in this examination will be computer marked. It is crucial that you follow the instructions below and fill in the red coloured answer sheets carefully. Use a PENCIL (not pen). Use a pencil eraser to correct any errors you make.

Insert your candidate name, title of the examination, unit code and date onto the answer sheets in the relevant boxes. Fill in your student number:

TURN OVER ONLY WHEN TOLD TO START WRITING

Help Formulas:

2D Convolution:

$$a_{x,y} = \sum_{\delta_x=-1}^{1} \sum_{\delta_y=-1}^{1} h_{x+\delta_x, y+\delta_y} W_{\delta_x, \delta_y}$$

Optimal weights for linear regression:

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Matrix inversion:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Matrix Determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Multi-class cross-entropy loss:

$$\text{cross entropy}(\boldsymbol{\ell}, y) = -\log \ell_y + \log \sum_c \exp(\ell_c).$$

Binary cross-entropy loss:

$$\text{cross entropy}(\ell, y) = \begin{cases} \log(1 + e^{-\ell}) & \text{if } y = 1 \\ \log(1 + e^{\ell}) & \text{if } y = 0 \end{cases}$$

ReLU:

$$\text{relu}(x) = \begin{cases} x & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Q1. Consider the following code:

```
import torch as t

A = t.randn(1,3)
B = t.randn(2,1)
C = A+B
```

What is the shape of `C`?

A. `t.Size([2,3])`

B. `t.Size([2,1])`

C. `t.Size([1,3])`

D. `t.Size([2,1,3])`

E. We can't compute `C` as there's a shape error

[6 marks]

Q2. I want to generate and compute averages of a *huge* tensor of standard Gaussian noise. The random tensor is of size `t.Size([100000, 1000])`, and I want to end up with 100,000 values, each of which is an average of 1,000 standard Gaussian values (i.e. I want to end up with a tensor of averages of size `t.Size([100000])`). I have a GPU. What is the most efficient way to compute these averages? (Note that all the options compute the right quantity, the question is which is fastest?)

A.
```
avg = t.randn(100000, 1000).mean(1).to(device="cuda")
```

B.
```
avg = t.randn(100000, 1000).to(device="cuda").mean(1)
```

C.
```
avg = t.randn(100000, 1000, device="cuda").mean(1)
```

D.
```
noise = t.randn(100000, 1000)
avg = t.zeros(100000)
for i in range(100000):
    avg[i] = noise[i, :].mean()
```

E.
```
noise = t.randn(100000, 1000, device="cuda")
avg = t.zeros(100000, device="cuda")
for i in range(100000):
    avg[i] = noise[i, :].mean()
```

[6 marks]

Solution: `avg = t.randn(100000, 1000, device="cuda").mean(1)`

As this generates random numbers on the GPU, then sums them on the GPU as a block. Other solutions either generate random numbers on CPU (which is slower), or do the averages within a for loop (which is also slower).

Q3. For the data in the table, fit a model of the form $\hat{y} = w_1 + w_2 x$

x	y
-1.1	-4.2
-0.3	-2.3
0.2	-0.1
0.6	2.1
1.4	3.9

A. $w_1 = -0.63, w_2 = 3.39$

B. $w_1 = -0.71, w_2 = 3.37$

C. $w_1 = -0.67, w_2 = 3.43$

D. $w_1 = -0.73, w_2 = 3.32$

E. $w_1 = -0.78, w_2 = 3.29$

[6 marks]

Q4. Consider the following choices of loss function. Which one **CANNOT** be optimized using gradient descent as it does not have meaningful gradients?

A. $\mathcal{L} = \sum_i (\hat{y}(x_i) - y_i)^2$

B. $\mathcal{L} = \sum_i (\hat{y}(x_i) - y_i)^4$

C. $\mathcal{L} = \sum_i |\hat{y}(x_i) - y_i|$

D. $\mathcal{L} = -\sum_i \frac{1}{1+(\hat{y}(x_i)-y_i)^2}$

E. $\mathcal{L} = \sum_i \Theta(|\hat{y}(x_i) - y_i| - 0.1)$

Remember that Θ is the Heaviside step,

$$\Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

[6 marks]

Solution: $\mathcal{L} = \sum_i \Theta(|\hat{y}(x_i) - y_i| - 0.1)$ As the Heaviside step is not continuous.

Q5. Compute the sum of cross-entropy losses for binary classification, where the logits are given by,

$$\text{logits}(x) = -2 + x + 3x^2$$

with data,

x	y
-1.1	0
-0.3	0
0.2	0
0.6	1
1.4	1

A. 2.05

B. 2.10

C. 2.15

D. 2.20

E. None of the above.

[6 marks]

Q6. There are some key requirements for a function that maps multi-class logits to a probability distribution:

- Must always form a valid probability distribution (all probabilities are positive, and sum to 1).
- Must have useful gradients.
- Probability must not decrease as logits increase.

Which function satisfies all these requirements?

A. $\frac{\ell_y^2}{\sum_{c=1}^{C} \ell_c^2}$

B. $\frac{\ell_y^3}{\sum_{c=1}^{C} \ell_c^3}$

C. $\frac{\textbf{relu}(\ell_y)+\epsilon}{\sum_{c=1}^{C}(\textbf{relu}(\ell_c)+\epsilon)}$

D. relu$(\ell_y)/C$, where C is the number of classes

E. $1 - \delta_{y,\text{class with highest logits}}$

Here, C is the number of classes, y is the true target class and ϵ is a small constant (e.g. $\epsilon = 10^{-5}$) used here to avoid divide-by-zero.

Hint: to exclude a couple of possibilities, consider a setting where one of the logits is negative. *[7 marks]*

Solution:

- $\frac{\ell_y^2}{\sum_c \ell_c^2}$ doesn't work, as probability gets bigger if a negative ℓ_y gets even more negative.
- $\frac{\ell_y^3}{\sum_c \ell_c^2}$ doesn't work, as probability is negative if logits is negative.
- relu$(\ell_y)/C$ doesn't normalize to 1.
- $1 - \delta_{y,\text{class with highest logits}}$ doesn't give sensible gradients, where δ_{ij} is the Kronecker delta.

Q7. Which statement about neural networks is **FALSE**?

A. The nonlinearity is necessary, to ensure that the whole network doesn't become equivalent to just a linear function.

B. Modern networks typically use a sigmoid nonlinearity.

C. We don't usually have a nonlinearity at the output, as the nonlinearity usually has bounded outputs (e.g. the sigmoid gives outputs between 0 and 1), whereas we usually want our network's output to be unbounded (which is most obvious in regression).

D. In PyTorch, we typically represent a neural network layer as an object/class.

E. But in PyTorch you don't *have* to represent a neural network layer as an object/class.

[7 marks]

Solution: We don't typically apply a nonlinearity at the output layer, precisely due to these constraints. e.g. a relu nonlinearity will also impose constraints (that the output is non-negative).

Q8. Compute the backward pass for the following nonlinearity (assume s is fixed, so we're only looking for $\frac{d\mathcal{L}}{da}$).

$$h = \sigma(sa)$$

i.e. compute $\frac{d\mathcal{L}}{da}$ in terms of $\frac{d\mathcal{L}}{dh}$. Remember that,

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\sigma(-x)$$

A. $\frac{d\mathcal{L}}{da} = \sigma(sa)\sigma(-sa)\frac{d\mathcal{L}}{dh}$
B. $\frac{d\mathcal{L}}{da} = a\sigma(sa)\sigma(-sa)\frac{d\mathcal{L}}{dh}$
C. $\frac{d\mathcal{L}}{da} = s\sigma(sa)\sigma(-sa)\frac{d\mathcal{L}}{dh}$
D. $\frac{d\mathcal{L}}{da} = sa\sigma(sa)\sigma(-sa)\frac{d\mathcal{L}}{dh}$
E. $\frac{d\mathcal{L}}{da} = s\sigma(a)\sigma(-a)\frac{d\mathcal{L}}{dh}$

[7 marks]

Solution:

$$\frac{d\mathcal{L}}{da} = \frac{\partial h}{\partial a}\frac{d\mathcal{L}}{dh}$$

Use the chain-rule,

$$\begin{aligned} u &= sa \\ \frac{\partial h}{\partial a} &= \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial a} \\ \frac{\partial h}{\partial a} &= s\sigma(u)\sigma(-u) \\ \frac{\partial h}{\partial a} &= s\sigma(sa)\sigma(-sa) \end{aligned}$$

Q9. Compute the backward pass for the following multiplication operation,

$$y = ax.$$

i.e. compute $\frac{d\mathcal{L}}{da}$ and $\frac{d\mathcal{L}}{dx}$ in terms of $\frac{d\mathcal{L}}{dy}$.

A. $\frac{d\mathcal{L}}{da} = a\frac{d\mathcal{L}}{dy}$ and $\frac{d\mathcal{L}}{dx} = a\frac{d\mathcal{L}}{dy}$
B. $\frac{d\mathcal{L}}{da} = a\frac{d\mathcal{L}}{dy}$ and $\frac{d\mathcal{L}}{dx} = x\frac{d\mathcal{L}}{dy}$
C. $\frac{d\mathcal{L}}{da} = x\frac{d\mathcal{L}}{dy}$ **and** $\frac{d\mathcal{L}}{dx} = a\frac{d\mathcal{L}}{dy}$

D. $\frac{d\mathcal{L}}{da} = x\frac{d\mathcal{L}}{dy}$ and $\frac{d\mathcal{L}}{dx} = x\frac{d\mathcal{L}}{dy}$

E. None of the above

[7 marks]

Solution:

$$\frac{d\mathcal{L}}{da} = \frac{\partial y}{\partial a}\frac{d\mathcal{L}}{dy} = x\frac{d\mathcal{L}}{dy} \quad (1)$$

$$\frac{d\mathcal{L}}{dx} = \frac{\partial y}{\partial x}\frac{d\mathcal{L}}{dy} = a\frac{d\mathcal{L}}{dy}. \quad (2)$$

Q10. Which of the following statements about the implementation of backprop is **FALSE**:

A. In a standard backprop implementation, you would retain all the computed intermediate values from the forward pass.

B. These values are retained in a so-called "compute graph".

C. The compute graph can be implemented using objects, where each object represents the output of a single function call. This object remembers what function was called, what arguments were used and the resulting value.

D. To compute the gradients of the loss wrt the parameters, you iterate through each operation in the compute graph, computing $\frac{d\mathcal{L}}{d\text{outputs}}$ as a function of $\frac{d\mathcal{L}}{d\text{inputs}}$.

E. None of the above.

[7 marks]

Solution: Backprop computes $\frac{d\mathcal{L}}{d\text{inputs}}$ as a function of $\frac{d\mathcal{L}}{d\text{outputs}}$ (i.e. the other way around from in the question).

Q11. Consider a debiased and a non-debiased version of RMSProp, with $\beta_2 = 0.999$, Non-debiased RMSProp:

$$\begin{aligned}\langle g^2 \rangle_{t+1} &= \beta_2 \langle g^2 \rangle_t + (1-\beta_2) g^2_{\text{mb};t} \\ w_{t+1} &= w_t + \frac{\eta}{\sqrt{\langle g^2 \rangle_{t+1}} + \epsilon} \hat{m}_{t+1}.\end{aligned}$$

Debiased RMSProp:

$$\begin{aligned}\langle g^2 \rangle_{t+1} &= \beta_2 \langle g^2 \rangle_t + (1-\beta_2) g^2_{\text{mb};t} \\ \hat{v}_{t+1} &= \frac{\langle g \rangle_{t+1}}{1-(\beta_2)^t} \\ w_{t+1} &= w_t + \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon} \hat{m}_{t+1}.\end{aligned}$$

Non-debiased RMSProp in effect modifies the learning rate, relative to debiased RMSProp. Specifically,

A. Non-debiased RMSProp starts at a higher learning rate, and converges back to the usual RMSProp learning rate after a few thousand timesteps.

B. Non-debiased RMSProp starts at a **higher** learning rate, and converges back to the usual RMSProp learning rate after a few **tens** of timesteps.

C. Non-debiased RMSProp starts at a **lower** learning rate, and converges back to the usual RMSProp learning rate after a few **thousand** timesteps.

D. Non-debiased RMSProp starts at a **lower** learning rate, and converges back to the usual RMSProp learning rate after a few **tens** of timesteps.

E. None of the above.

[7 marks]

Solution: $\langle g^2 \rangle_{t+1}$ is initially very small, which implies a higher effective learning rate at the start. As $\beta_2 = 0.999$, we know that $\langle g^2 \rangle_{t+1}$ will converge to steady-state after a few thousand iterations.

Q12. Consider doing gradient descent,

$$x_{\text{next}} = x - \eta \frac{\partial \mathcal{L}(x)}{\partial x},$$

on the the following objective,

$$\mathcal{L}(x) = x^2.$$

Consider the following 4 learning rates. Select the largest stable learning rate (i.e. the largest learning rate that eventually converges to the right answer):

A. $\eta = 0.4$

B. $\eta = 0.6$

C. $\eta = 0.9$

D. $\eta = 1.1$

E. None of the above are stable.

[7 marks]

Solution: Try doing a couple of steps of gradient descent for each learning rate. If the magnitude of x increases, the learning rate is unstable. Alternatively, the threshold between stability and instability, is when $x \to -x$,

$$-x = x - \eta \frac{\partial x^2}{\partial x} \tag{3}$$
$$-x = x - 2\eta x \tag{4}$$
$$-1 = 1 - 2\eta \tag{5}$$
$$2 = 2\eta \tag{6}$$
$$\eta = 1 \tag{7}$$

Q13. Which of these is the key, defining feature of overfitting?

A. Far better train than test performance.

B. Predictions vary rapidly as a function of x.

C. Complex function class.

D. No regularisation.

E. Poor train and test performance.

[7 marks]

Q14. Which of these statements about standard vs decoupled weight decay is **FALSE**?

A. Weight decay in Adam can be interpreted as a modified loss.

B. Decoupled weight decay in AdamW can be interpreted as a modified loss.

C. Both Adam and AdamW use $\sqrt{\langle g^2 \rangle}$ to normalize the gradient update.

D. Adam also normalizes the weight decay term using $\sqrt{\langle g^2 \rangle}$.

E. Decoupled weight decay in AdamW does not normalize the decay term.

[7 marks]

Q15. Consider a 2D Convolution, with an input width=5, height=7. If we have kernel size=3, padding=0, stride=2, what is the output size?

A. output width=1 and output height=3

B. output width=1 and output height=4

C. output width=2 and output height=3

D. output width=2 and output height=4

E. None of the above.

[7 marks]

END OF PAPER

The following pages are left blank for your rough workings. They will not be collected or marked. You must enter your answers on the provided answer sheet only.