University of Toronto
Department of Electrical and Computer Engineering
Winter Semester 2025

Midterm Exam

for the Course

ECE 1508: Applied Deep Learning

Date	**February 27, 2025**
Duration	**100 Minutes**
Number of Questions	**4**
Total Points	**100**

Permissions The exam is open book. You can use all **printed materials** as well as a **calculator**. Using electronic devices is **not permitted**.

Student Information Please write your information and sign the designated field.

Name .....

Student Number .....

Signature .....

Question	Question 1	Question 2	Question 3	Question 4	Total
Mark	/17	/33	/30	/20	/100

This page is left empty.

The microscopic behavior of a thermodynamic system is described by its *micro-state* vector. With a given micro-state, the system can be in either an *equilibrium* or *non-equilibrium* condition.

We aim to develop a machine learning model that determines the *condition* of a system (*equilibrium* or *non-equilibrium*) based on *its micro-state vector*. For data acquisition, we use a computer simulator that can perform the following simulation: starting from a *random seed*, it generates a *micro-state* vector and *evaluates the system's condition*, i.e., it determines whether the system with the generated micro-state is in equilibrium or non-equilibrium. The simulator is fast, and we can run it as much as we like.

Answer the following items:

1. *6 Points* Formulate this problem as a *supervised learning* task. Explain how you can make a *dataset* for this problem. Specify a *deep model* that you can use for this learning task. You do *not* need to specify hyperparameters. It is enough to specify the input and output layers and explain how the output layer is activated.

This is a supervised problem → data point = micro-state, label = condition

Input (features) → micro-state vector

Label (targets) → condition (equallibrium or non-equilibrium)

Fast simulator:

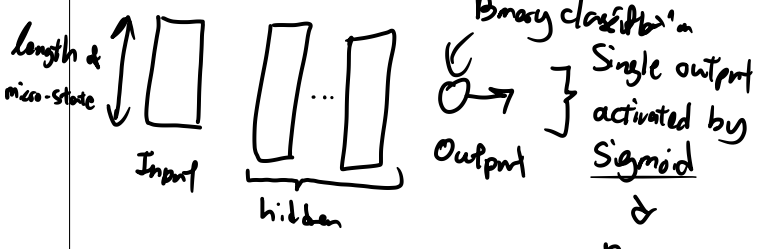
Generate as much labels as we want

Dataset = $\{(x_1, y_1), \ldots (x_N, y_N)\}$

micro-state ← ↳ condition { 1 equilibrium, 0 non-equilibrium

we run the simulator N times to collect samples

Use a NN - MLP



This MLP:

$\hat{y} = \sigma(z)$, between 0 & 1

If $\hat{y} > 0.5$: predict equilibrium

If $\hat{y} < 0.5$: predict non_equil

All deep model:

MLP - Vector data - General Purpose. simple deep model

CNN - Images, spatial pattern - Use local receptive fields and shared weights

RNN - Sequence (text, time series) - Handle variable-length temporal data

Transformer - Text, speech, vision - Need big data

Autoencoder - Dimension reduction / feature learning - Un supervised

---

CNN - Receptive field → A small part of image

- Weight-Sharing

⇓

Like a cookie, same star-filter, easy to know what a star looks like.

RNN - Order matters

Sentence "The cat sat on a mat"

RNN Update hidden state with each new word.

Once reach mat, it remember cat, predict next word as "purr"

For next word in sentence,

Stock price forecast.

Speech recognition

Video frame generation

Transformer:

Attention - Focus on most relevant parts of input

"Animal didn't cross street cuz it is tired"

- Attention helps model look "it" as animal, not Street.

a.k.a paying attention

- Need parameters

↓

weights + biases

↓

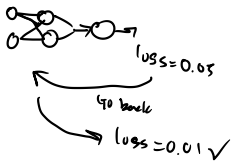
How much salt, sauce, cooking time for good burger?

- Adjustable things = parameters.

- Then have loss

↓

Concept of backpropagation.



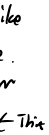
- GPT - billions of parameters
  - Learn grammar
    If only see 5 sentences
    - You get weird rules
    If have 10 mil sentences
    - You understand shit
  - GPU
    - 1000 chef cooking burgers
  - CPU
    - 1 chef cooking 1.

Autoencoder:

Compressor for images

- Imagine cat painting with spills of paint, = noise = random extra junk that isn't needed.

For - image compression
- Noise removal
- Anomaly detection ← Kinda like heartrate.



← This is faulty, it learns that

- Dimension reduction for unsupervised learning

↓

No teacher:

Supervised: Given microstate, yes or no, equilibrium or non-equil

Unsupervised: Group similar micro-states together, no labels given

---

Supervised: I give you data with yes or no, next time you post something, you give me yes or no

Unsupervised: You figure yourself

Parameter - Model learns itself, during training ⇒ weight, biases

Hyperparameter: You set, before training → learning rate, # of layers...

Stochastic = random, small random batch.

GD = Mistake loss

SGD = Faster, but more noise

CE - Classification

90% it is a cat = good - low loss
90% it is a dog = bad - high loss
CE punish you for being confidently wrong

MSE - Regression
- Measure distance of numbers
- How far off are your predictions

Fine-Tuning:

Base model: Train on general English text

Fine-Tuned model: Train on legal documents → become good at law questions

2. *6 Points* To train the model, we use the following *wrongly-written* cross-entropy as the loss function

$$\mathcal{L}(y, v) = \begin{cases} -\log y & v = 0 \\ -\log(1-y) & v = 1 \end{cases},$$

where v refers to the true label (either 0 or 1), and y is the activated output of the model.

Explain how you should modify your model during *inference* after training it with this loss.

- Testing, deployment phase

The loss makes y to get close to "1" when $v=0$ & close to "0" when $v=1$

We thus infer as $\rightarrow \begin{cases} \hat{v}=0 & y \geq 0.5 \\ \hat{v}=1 & y<0 \end{cases}$

(Basically to flip the label)

3. *5 Points* Assume that after training you observe *overfitting*. Propose a solution that can potentially resolve this issue *without changing the loss function or training loop*.

Data generation

We generate more samples via simulator using other random seeds

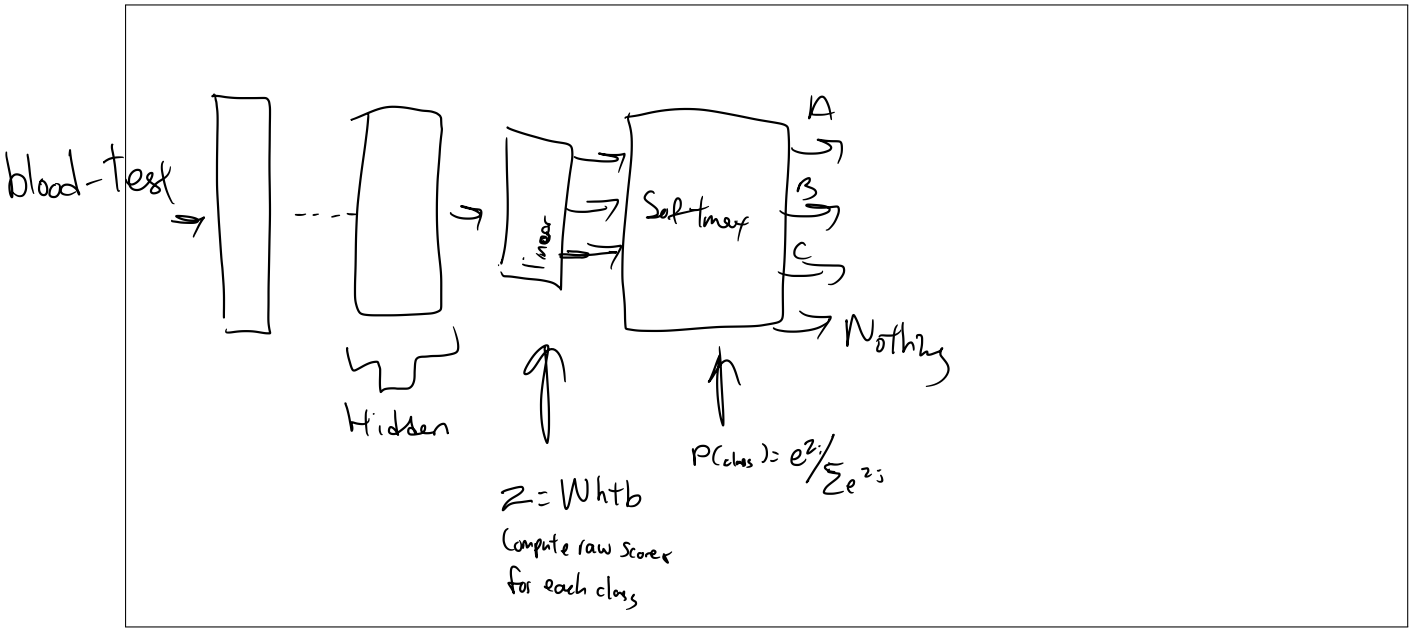
Question 2: Classification 33 Points

We aim to train a model for the following task: *given a blood-test, we want to make a diagnosis*. There are three possible diagnoses: *A, B, or C*. We have also the option to report *no diagnosis* for a blood-test.

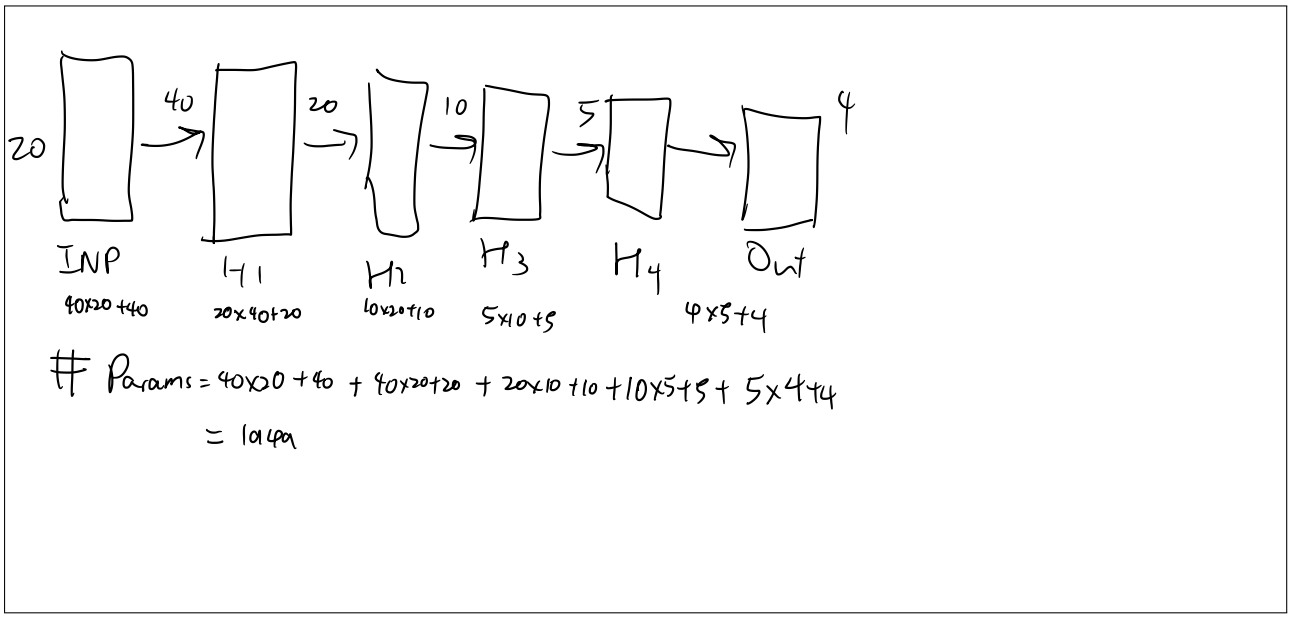
You are given a dataset of blood-tests, each being labeled by a *single* corresponding diagnosis. The information on each blood-test is represented as a vector of 20 entries, each showing a blood parameter. For the model, we use a fully-connected feedforward neural network (FNN) with *four hidden layers*. The width of the *first* hidden layer is *double* the width of the input layer. The next hidden layers shrink in width with factor 2, i.e., the second hidden layer is half wide the first one, the third is half the second, and so on.

At this point, assume that each blood-test is diagnosed either with *a single diagnosis (A, B, or C)* or *noting*. Answer the following items:

1. *6 Points* Design the output layer of the FNN. Explain why you have chosen this design.



2. *6 Points* For your design in Part 1, determine the *number of learnable parameters* in the model.



FNN:

Binary Classification - Sigmoid - Yes or no

Multi-Class-Classification - Softmax

Multi-label Classification - Sigmoid per Class

3. *5 Points* You are given the following function:

$$a(x) = \begin{cases} \dfrac{\exp\{x\}}{1+\exp\{x\}} & x \geq 0 \\ \tanh(x) & x < 0 \end{cases},$$

and asked to set your activation function to

$$f(x) = \alpha a(x) + \beta s(x)$$

for some *real scalars* α *and* β, where $s(x)$ is the step function. Suggest a good choice for the pair (α, β). Explain why you believe that this is a good choice.

$\frac{e^0}{1+e^0} = \frac{1}{1+1} = \frac{1}{2}$

$\tanh(0) = 1$

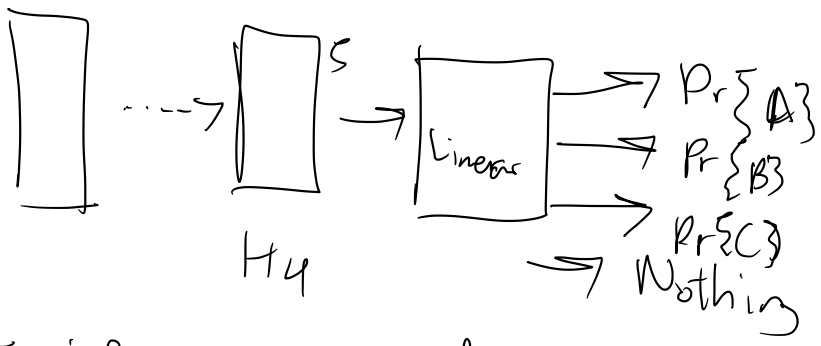
$\alpha = 1, \beta = -\frac{1}{2}$

$x = 0$ $f(x)$ is continuous

We now consider another variant of the problem, in which each blood-test is diagnosed with *multiple diagnoses*. For example a blood-test can be diagnosed with both A and B, another test with A, B, and C, another with only C, another with *nothing*, and etc. For this case, answer the following items:

4. *6 Points* Modify the output layer that you suggested in Part 1, such that the FNN becomes a valid model for this problem. Explain how you can use this model for *inference*.

Use Sigmoid



For inference, we report any diagnosis that have Pr more than 0.5

5. [5 Points] Propose a loss function for this modified model.

True label = $\begin{bmatrix} A \\ B \end{bmatrix} \sim v = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$$\text{Loss} = CE(v_1, y_1) + CE(v_2, y_2) + CE(v_3, y_3)$$

(A & B positive, C negative)

$v = [1, 1, 0]$, $y = [0.9, 0.8, 0.1]$ ← Prob

$$L = -\sum_{i=1}^{3} [v_i \log(y_i) + (1 - v_i) \log(1 - y_i)]$$

$$\text{Loss} = BCE(1, 0.9) + BCE(1, 0.8) + BCE(0, 0.1)$$

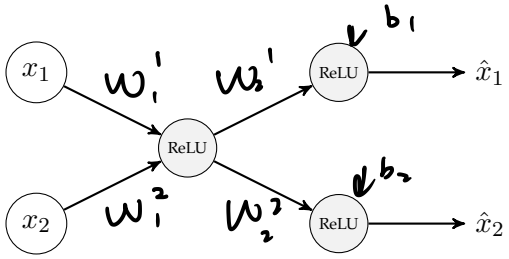
6. [5 Points] Can we use the *our dataset (in which each blood-test is labeled with a single diagnosis)* to train this modified model? Give a short justification to your answer.

Yes. Our model is learning features for our diagnosis perfectly

To compress and decompress a data-point $\boldsymbol{x} \in \mathbb{R}^2$, we use the following model:

- We pass $\boldsymbol{x}$ through a neuron with *no bias* to get a compressed version y. This neuron is activated by the *ReLU* function.
- To decompress y, we pass it through two neurons (*with bias*) activated by *ReLU*.

The end-to-end model from an input $\boldsymbol{x}$ to the decompressed version $\hat{\boldsymbol{x}}$ is shown in the following diagram:



In this diagram, we consider

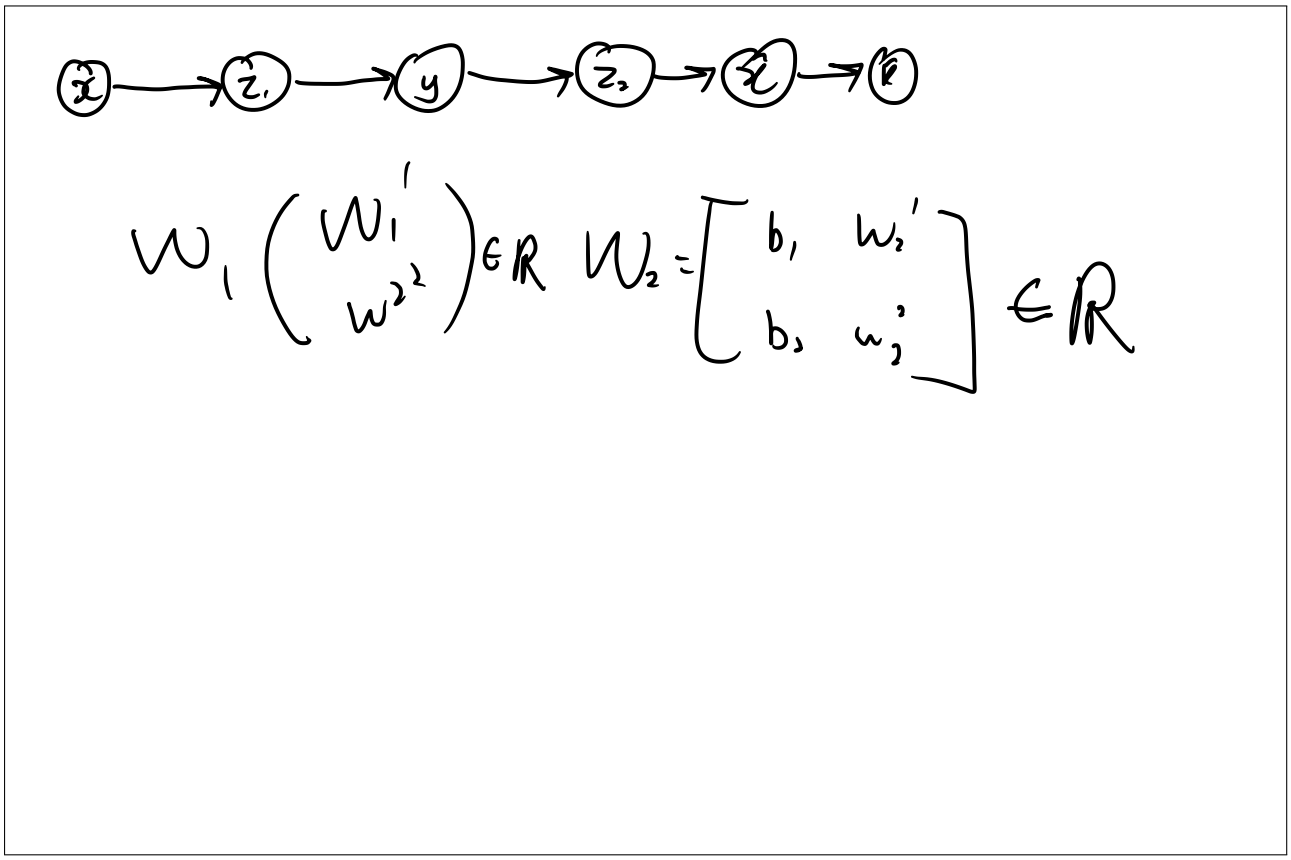
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \text{and} \qquad \hat{\boldsymbol{x}} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}.$$

We aim to get a good reconstruction. Therefore, we define the loss to be

$$\mathcal{L}\left(\boldsymbol{x}, \hat{\boldsymbol{x}}\right) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2,$$

with $\|\cdot\|$ denoting the Euclidean norm (ℓ_2 norm). In this model, we *initiate all weights and biases with* 0.5. Answer the following items:

1. *6 Points* Sketch a computation graph from input $\boldsymbol{x}$ to the sample loss.



2. *7 Points* Considering the initial weights and biases, pass the sample

$$\boldsymbol{x} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

forward, and compute the value of all nodes in the computation graph.

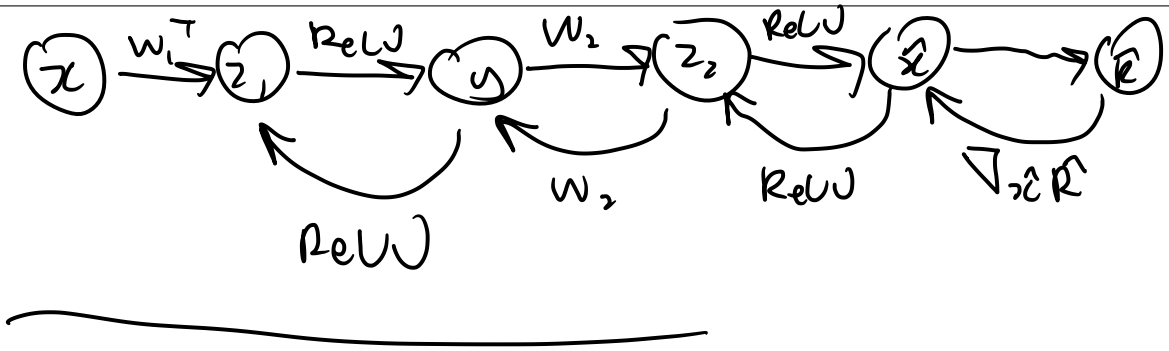
$z_1 = (0.5, 0.5)\begin{pmatrix} 4 \\ 2 \end{pmatrix} = 2 + 1 = 3$

$y = \mathrm{ReLU}(3) = 3$

$z_2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{bmatrix} 0.5(1) + 0.5(3) \\ 0.5(1) + 0.5(3) \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

$\hat{x} = \mathrm{ReLU}(z_2) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$

3. *12 Points* Use the *backpropagation* algorithm and compute the gradient of the sample loss for the given sample $\boldsymbol{x}$ in Part 2 *with respect to the weights of the compressing neuron*, i.e., the one in the first hidden layer.
Attention: *Only solutions that use backpropagation are marked.*



$$\frac{\partial R}{\partial w_1}, \frac{\partial R}{\partial w_2}$$

$$L = \|x - \hat{x}\|^2$$

$$= 2(\hat{x} - x)$$

$$\nabla_{\hat{x}} \hat{R} = 2\begin{pmatrix} \hat{x}_1 - x_1 \\ \hat{x}_2 - x_2 \end{pmatrix} = 2\left(\begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 4 \\ 2 \end{bmatrix}\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$ReLU = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightsquigarrow \nabla_{z_2} \hat{R} = \begin{pmatrix} 4 \\ 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\nabla_y \hat{R} = w_2^T \nabla_{z_2} \hat{R} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \begin{pmatrix} 4 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$\rightsquigarrow \frac{\partial \hat{R}}{\partial y} = 2$$

$$ReLU(3) = 1 \times \frac{\partial \hat{R}}{\partial y} = 1 \cdot 2 = 2$$

$$2\begin{pmatrix} 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 8 \\ 4 \end{pmatrix}$$

4. *5 Points* A friend suggests that you change the ReLU activation to *Soft ReLU*, defined as

$$f(x) = \log(1 + \exp\{x\}).$$

Do you think that your training can benefit from this change? *If yes*, explain how substantial the gain is. *If no*, give a reason.

Yes, it can.

SoftReLU has gradient everywhere and can help in GD.

Question 4: Optimizer 20 Points

The *California Housing Dataset* has 20,000 samples. We are to train a model with this dataset. For the optimizer, we use the *vanilla* mini-batch stochastic gradient descent (SGD) with batch-size $\Omega = 100$.

For training, we want to perform 4000 *steps of gradient descent* (**not** *epochs*) in total with the optimizer. Answer the following items:

1. *5 Points* Compute the number of epochs that completes 4000 *steps of gradient descent*.

Batch size
$\Omega = 100 \Rightarrow$ we have $\frac{20000}{100} = 200$ mini. batches

$\#\text{epoch} = \frac{4000}{200} = 20$

2. *5 Points* We increase the batch-size to $\Omega = 400$. What do you expect to observe?

$\Omega = 400 \Rightarrow \#\text{epoch} = 80$, $\#$ mini-batches $= 50$

we spend more time training

4 times less variance.

→ more noisy or inconsistent

3. *5 Points* Suggest a modification to the *vanilla mini-batch SGD*, which can reduce the variance of updated weights in each iteration *without changing the batch-size*. Explain your answer.
Attention: *You **cannot** name an alternative optimizer. You should modify the vanilla SGD.*

We can use momentum instead of sample gradient

$m^{(t)} = \beta m^{(t-1)} + (1-\beta) g^{(t)}$ for some β close to 1.

↑ current gradient

4. *5 Points* Is Adam a better optimizer than your modified SGD in Part 3? Give a reason.

???

RMSprop scale learning rate individually

therefore Adam better