CS230: Deep Learning

Fall Quarter 2020
Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice	16	
2	Short Answers	16	
3	Convolutional Architectures	16	
4	Movie Posters	21 + 3 (bonus)	
5	Backpropagation	28	
6	Numpy Coding	14	
Total		111 + 3 (bonus)	

The exam contains 24 pages including this cover page.

- If you wish to complete the midterm in LaTeX, please download the project source's ZIP file here. (The Stanford Box link, just in case you face issues with the hyperlink: https://stanford.box.com/s/gm5h2ovq5om637uwm0skov7p00ocwijp)
- This exam is open book, but collaboration with anyone else, either in person or online, is strictly forbidden pursuant to The Stanford Honor Code.
- In all cases, and especially if you're stuck or unsure of your answers, **explain your work, including showing your calculations and derivations!** We'll give partial credit for good explanations of what you were trying to do.

Name: ____

SUNETID: ________________________________@stanford.edu

The Stanford University Honor Code:
I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signature: ________________________________

Question 1 (Multiple Choice Questions, 16 points)

For each of the following questions, circle the letter of your choice. Each question has AT LEAST one correct option unless explicitly mentioned. No explanation is required.

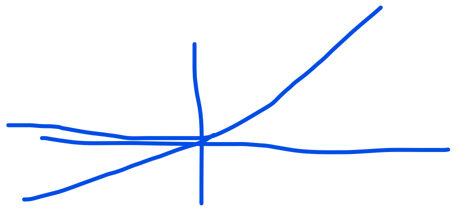
(a) **(2 points)** You are training a large feedforward neural network (100 layers) on a binary classification task, using a sigmoid activation in the final layer, and a mixture of *tanh* and *ReLU* activations for all other layers. You notice your weights to your a *subset* of your layers stop updating after the first epoch of training, even though your network has not yet converged. Deeper analysis reveals the *gradients* to these layers completely, or almost completely, go to zero very early on in training. Which of the following fixes could help? (You also note that your loss is still within a reasonable order of magnitude).

- (i) Increase the size of your training set
- (ii) Switch the ReLU activations with leaky ReLUs everywhere
- (iii) Add Batch Normalization before every activation
- (iv) Increase the learning rate

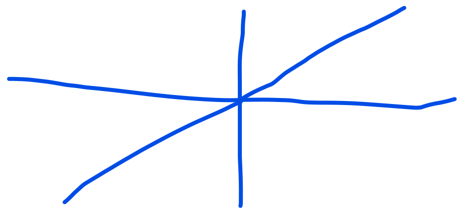
Solution: (ii), (iii).
Classic *vanishing gradient* problem. Increasing size of the training set (i) doesn't help as the issue lies with the learning dynamics of the network. Varying the learning rate (iv) might help the network learn faster, but as the problem states the gradients to specific layers almost completely go to zero, so the issue seems to be localized to specific layers.
(ii) Solves the problem of *dying relus* by passing some gradient signal back through all relu layers.
(iii) Adding BatchNorm prior to every activation ensures the *tanh* layers have inputs distributed *closer* to the linear region of the activation, so the *elementwise* derivative across the layer evaluates closer to 1.

(b) **(2 points)**Which of the following would you consider to be valid activation functions (elementwise non-linearities) to train a neural net in practice?

- ✓ [✓ (i)] $f(x) = -\min(2, x)$
- ✗ (ii) $f(x) = 0.9x + 1$
- ✓ [✓ (iii)] $f(x) = \begin{cases} \min(x, .1x) & \mid x >= 0 \\ \min(x, .1x) & \mid x < 0 \end{cases}$ $\rightarrow$



- ✗ (iv) $f(x) = \begin{cases} \max(x, .1x) & \mid x >= 0 \\ \min(x, .1x) & \mid x < 0 \end{cases}$



Solution: **(i), (iii)**.
(ii) and (iv) are linear functions, therefore quite useless as activations. (i) and (iii) are both concave, but nonlinear.

(c) **(2 points)** Which of the following techniques can be used to reduce model overfitting?

(i) Data augmentation
(ii) Dropout
(iii) Batch Normalization $\rightarrow$ Adding noise
(iv) Using Adam instead of SGD

Solution: i, ii, iii

(d) **(2 points)** You are training a convolutional neural network on the ImageNet dataset, and you are thinking of using gradient descent as your optimization function. Which of the following is true? (Note: faster here is defined in terms of wall clock time)

(i) It is possible for Stochastic Gradient Descent to converge faster than Batch Gradient Descent
(ii) It is possible for Mini Batch Gradient Descent to converge faster than Stochastic Gradient Descent
(iii) It is possible for Mini Batch Gradient Descent to converge faster than Batch Gradient Descent
(iv) It is possible for Batch Gradient Descent to converge faster than Stochastic Gradient Descent

Solution: i, ii, iii

(e) **(2 points)** Which of the following is true about dropout?

(i) Dropout leads to sparsity in the trained weights
(ii) At test time, dropout is applied with inverted keep probability
(iii) The larger the keep probability of a layer, the stronger the regularization of the weights in that layer
(iv) None of the above

Solution: iv

(f) **(2 points)** During backpropagation, as the gradient flows backward through a *sigmoid* non-linearity, the gradient will always:

(i) Increase in magnitude, maintain polarity

(ii) Increase in magnitude, reverse polarity

(iii) Decrease in magnitude, maintain polarity

(iv) Decrease in magnitude, reverse polarity

Solution: (iii). $\frac{\partial\sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$. This evaluates to a value inside $(0, .25]\forall x$

(g) **(2 points)** You are training a Generative Adversarial Network to generate images of reptiles. But, you think your Generator might be showing mode collapse. Which of the options below could be indicators of this problem?

(i) The generator is only producing images of komodo dragons

(ii) The generator loss is oscillating.

(iii) The generator loss remains low whereas the discriminator loss is high

(iv) The discriminator has high accuracy on real images but low accuracy on fake ones

Solution: (i), (ii).
Mode collapse can be found by inspecting the Generator, and doesn't have much to do with the discriminator and it's loss. (iii) only indicates that the Discriminator is bad. (iv) just shows that the Discriminator is not able to distinguish the fake images from the Generator, the Generator here can be very good but still may/may not have mode collapse.
(i) Only producing images of a single class/example indicates mode collapse.
(ii) The generator loss oscillation can indicate the Generator is shifting from one mode to another, but is not able to generate all the modes. Thus, can be an indicator for mode collapse.

(h) **(2 points)** You are benchmarking runtimes for layers commonly encountered in CNNs. Which of the following would you expect to be the fastest (in terms of floating point operations)?

(i) Conv layer (convolution operation + bias addition)

(ii) Max pooling

(iii) Average pooling

(iv) Batch Normalization

Solution: (ii).
Only n comparison operations required to find max, all other layers require *at least* n*+1* operations (n denotes the size of activation).

Question 2 (Short Answers, 16 points)

The questions in this section can be answered in 2-4 sentences. Please be concise in your responses.

(a) **(2 points)** You come across a nonlinear function that passes 1 if its input is nonnegative, else evaluates to 0, i.e.

$$f(x) = \begin{cases} 1 & \mid x >= 0 \\ 0 & \mid x < 0 \end{cases}$$

A friend recommends you use this non-linearity in your convolutional neural network with the Adam optimizer. Would you follow their advice? Why or why not?

Solution: Don't listen to them! Although this is *technically* a non-linearity, specifically a *discontinuous nonlinear step function*, the gradient is 0 everywhere but the origin. Thus it would pass almost no gradient back during backprop, which is crucial when optimizing with ADAM or any other descent-based optimizer.

(b) **(2 points)** Give a method to fight vanishing gradient in fully-connected neural networks. Assume we are using a network with Sigmoid activations trained using SGD.

Solution: Intentionally vague question, accept if any of these responses are present

- Any neural network modification such as using ResNet blocks, switching Sigmoid activations with ReLU, using better initializations.

(c) **(2 points)** You are designing a deep learning system to diagnose chest cancer through X-ray images. What do you think might be the most appropriate evaluation metric and why: Accuracy, Precision, Recall, F1 score.

$$\frac{TP}{TP+FP}$$

$$\frac{TP}{TP+FN}$$

Solution: We should use Recall here, as any false-negatives from the system are much more harmful.

(d) **(2 points)** You are training on a GAN to generate cool octopus images. But you are only able to curate a small number of images of actual octopus. You decide that image augmentation might be a good idea to improve GAN training. You try using three common augmentation techniques: adding blur to the image, changing the color of pixels and flipping the left-right axis of the image. Which of these do you think might be a good idea to help the Generator output better octopus images and why?

Solution: We should only use left-right flip augmentation, as the other ones will create artifacts in the generated images like having unreal colors or generating blurred images.

(e) **(2 points)** Suppose you have built a model to predict a car's fuel performance (e.g. how many miles per gallon) based on engine size, car weight, etc... (e.g. many attributes about the car). Your boss now has the great idea of using your trained model to build a car that has the best possible fuel performance. The way this is done will be by varying the parameters of the car, e.g. weight and engine size and then using your model to predict fuel performance. The parameters will then be chosen such that the predicted fuel performance is the best. Is this a good idea? Why? Why not?

Solution: Intentionally vague question, accept if any of these responses are present

- Might ask to you use parameters not feasible in the real world
- Correlation does not imply causation. Your set of features/parameters may not be representative of all causal factors

(f) **(2 points)** You are working as a Machine Learning engineer at Hooli inc and you are trying to build a model that predicts whether a person will click on a given advertisement. You remember from CS230 that creating an ensemble of models (averaging the results of the models), can lead to better predictions. You try 2 approaches:

- You train a very large model, which takes many hours to train.
- You train 10 slightly smaller models and average the results of each. The entire process takes a smaller amount of time compared to the first model.

Both these models give you an accuracy of 90% on your training set. You need to pick an approach to deploy on a single-GPU machine that needs to provide predictions in real-time. Quick turn-around is of the utmost importance here. Which approach is more suitable for production, and why?

Solution: 1st one, since at test time the cost of the 2nd model is significantly higher (need to average 10 predictions)

(g) **(2 points)** You are training a single-layer, feedforward neural network with a softmax activation function in the final layer to classify among 99 classes, with a *cross-entropy loss* training objective. Recall, the cross-entropy loss function:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y} \cdot \log(\hat{\mathbf{y}})$$

where $\mathbf{y}$ is the *one-hot encoded* label, and $\hat{\mathbf{y}}$ is the *predicted* probability distribution over labels.

You decide to independently sample your initial weights from a Gaussian of mean 0, standard deviation 0.0001. You can assume perfect class balance in dataset.

You *accidentally* set a **0 learning rate**. What would you expect your average loss after the first training epoch to be? Provide a brief explanation (1-2 sentences) as to why this is so. You don't have to calculate the exact numerical value - feel free to leave your answer as a fraction.

Solution:

The network isn't learning anything, and the weights are randomly distributed *close to zero*. The cross-entropy loss evaluates the *logarithm* of probability for the *correct* class. The softmax final layer has 99 output nodes, and with a perfectly balanced training set, the 'correct' predictions will be randomly, *evenly* distributed across all these nodes. Thus, the *expected* loss value would be

$$L(\hat{y}, y) = -\log(\frac{1}{99})$$

(h) **(2 points)** You are building the next state-of-the-art CNN for a vision task, using a stack of modules that each look similar to the following:

(Layer Input) → (Conv Layer) → (Batch Norm) → (Activation) → (Next Layer Input)

You remember from your CS230 knowledge that each Conv layer has a set of learnable weights and biases. A colleague advises you to not learn biases (set them all to zero, forever) for these layers. Would performance be affected if you chose to follow their advice? Briefly explain (1-2 sentences) your answer.

Solution: **Performance wouldn't be affected**. The positioning of the BatchNorm *immediately after* the Conv ensures that the BN layer can learn any bias previously learnt by the Conv layer. Indeed, original implementations of the https://arxiv.org/abs/1512.03385 had their convolutional biases switched off.

Question 3 Convolutional Architectures (16 points)

Consider the convolutional neural network defined by the layers in the left column below. Fill in the shape of the output volume and the number of parameters at each layer. You can write the activation shapes in the format (H, W, C), where H, W, C are the *height, width* and *channel* dimensions, respectively. Unless specified, *assume padding 1, stride 1 where appropriate.*
Notation:

- CONVx-N denotes a convolutional layer with N filters with height and width equal to x.
- POOL-n denotes a n×n max-pooling layer with stride of n and 0 padding.
- FLATTEN flattens its inputs, identical to `torch.nn.flatten` / `tf.layers.flatten`
- FC-N denotes a fully-connected layer with N neurons

Layer	Activation Volume Dimensions	Number of parameters
Input	$32 \times 32 \times 3$	0
CONV3-8		
Leaky ReLU		
POOL-2		
BATCHNORM		
CONV3-16		
Leaky ReLU		
POOL-2		
FLATTEN		
FC-10		

Solution:

Layer	Activation Volume Dimensions	Number of parameters
Input	$32 \times 32 \times 3$	0
CONV3-8	32 x 32 x 8	8*(3x3x3 + 1) = 224
Leaky ReLU	32 x 32 x 8	0
POOL-2	16 x 16 x 8	0
BATCHNORM	16 x 16 x 8	2*8
CONV3-16	16 x 16 x 16	16*(3x3x8 + 1) = 1168
Leaky ReLU	16 x 16 x 16	0
POOL-2	8 x 8 x 16	0
FLATTEN	16*8*8	0
FC-10	10	(8x8x16 + 1) * 10 = 10250

4

Question 4 (Movie Posters, 21 + 3 bonus points)

You have been tasked by the Supreme Leader of the Republic of Wadiya to build a deep learning model to help him decide what Wadiyan movie to watch. Specifically you're asked to build a classifier that takes in an image of a movie poster and classifies it into one of four genres: comedy, horror, action, and romance. You have been provided with a large dataset of movie posters where each movie poster corresponds to a move with exactly one of these genres.

(i) **(2 points)** You want to get a sense of what the human level performance is for the task. How would you go about getting an estimate of human level performance?

Solution: Conduct a survey on some humans with your dataset. Other answers acceptable here

(ii) **(2 points)** You now estimate that the human level performance on this task is 95%. What does that tell you about the Bayes Error Rate of the task?

Solution: Bayes Error Rate must be $\leq 5\%$

You decide to use cross entropy loss to train your network. Recall that the cross-entropy loss for a single example is defined as follows: $L_{CE}(\hat{y}, y) = -\sum_{i=1}^{n_y} y_i log(\hat{y}_i)$. where $\hat{y} = (\hat{y_1}, \hat{y_2}...\hat{y_{n_y}})^T$ represents the predicted probability distribution over the classes and $y = (y_1, y_2...y_{n_y})^T$ represents the ground truth vector, which is zero everywhere except for the correct class (eg $y = (1, 0, 0, 0)^T$ for comedy, and $y = (0, 0, 1, 0)^T$ for action).

(iii) **(2 points)** Suppose you're given an example poster of a horror movie. If the model correctly predicts the resulting probability distribution as $\hat{y} = (0.1, 0.4, 0.3, 0.2)$, what is the value of the cross-entropy loss? You can give an answer in terms of logarithms.

Solution: $-log0.4$

(iv) **(2 points)** After some training, the model now incorrectly predicts romance with distribution $\langle 0, 0.4, 0, 0.6 \rangle$ for the same poster. What is the new value of the cross-entropy loss for this example?

Solution: $-log0.4$

(v) **(2 points)** You train an initial model that achieves a 90% accuracy on the training dataset. What kind of problems is your model experiencing, and suggest a possible solution.

Solution: Model has high bias. Can try more powerful model

(vi) **(3 points)** After tuning the model architecture, you find that the softmax classifier works well. Specifically, the last layer of your network computes logits $z = (z_1, ..., z_{ny})$, which are then fed into the softmax activation.the model achieves 100% accuracy on the training data. However, you observe that the training loss doesn't quite reach zero. Show why the cross-entropy loss can never be zero if you are using a softmax activation.

$\boxed{e^{x} > 0}$ $-\log\left(\frac{e^{z_c}}{e^{z_1}+\ldots+e^{z_{n_y}}}\right)$

always greater than

Solution: Given correct class c, the loss reduces to the term $\underbrace{log \sum_{i=1}^{n_y} e_i^z}_{} - \underbrace{loge^{z_c}}_{}$; as all the exponentials are non-zero, this loss cannot be zero (the sum on the left will always be greater than the right). Solutions can be shown intuitively by breaking down exponential terms and their relationship to the loss function.

(vii) **(2 points)** While the model does well on the training set, it only achieves an accuracy of 85% on the dev set. You conclude that the model is overfitting, and plan to use L1 or L2 regularization to fix the issue. However, before you can do so, you learn from your lab mate that some of the examples in the data may be incorrectly labeled. Which form of regularisation would you prefer to use and why?

Solution: L1, because for an example that is incorrectly labeled L2 will afford a higher penalty

(viii) **(2 points)** Your model now has 100% accuracy on the training set, and 96% accuracy on the dev set! You now decide to expand the model to posters of movies belonging to multiple genres. Now, each poster can have multiple genres associated with it; for example, the poster of a movie like "The Wadiyan Avengers" falls under both "comedy" and "action". Propose a way to label new posters, where each example can simultaneously belong to multiple classes.

Solution: Use multi-hot encoding, e.g. $(1, 0, 0, 1)$ would be comedy and romance

(ix) **(2 points)** To avoid extra work, you decide to retrain a new model with the same architecture (softmax output activation with cross-entropy loss). Explain why this is problematic.

Solution: This is problematic because softmax activation with CE-loss unfairly penalizes examples with many labels. In the extreme case, if the example belongs to all classes and the model correctly predicts $\left\langle \frac{1}{n_y}, \ldots, \frac{1}{n_y} \right\rangle$, then the CE-loss becomes $-\sum_{i=1}^{n_y} \log \hat{y}_i = n_y \log n_y$, which will be far bigger than the loss for most single-class examples.

(x) **(2 points)** Propose a different activation function for the last layer and a loss function that are better suited for this multi-class labeling task.

Solution: We can formulate this as *ny* independent logistic regression tasks, each trying to predict whether the example belongs to the corresponding class or not. Then the loss can simply be the average of *ny* logistic losses over all classes

(xi) **Bonus (3 points)** Prove the following lower bound on the cross-entropy loss for an example with K correct classes:

$$L_{CE}(\hat{y}, y) \geq KlogK$$

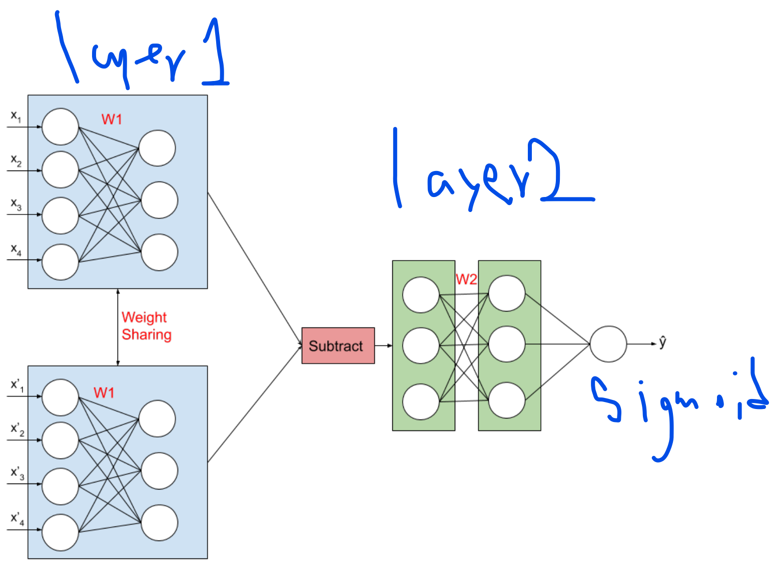
Assume you are still using softmax activation with cross-entropy loss with ground truth vector $y \in \{0, 1\}^{n_y}$ with K nonzero components.

Solution:

Let S denote the set of classes the given example belongs to (note $|S| = L$). Then,

$$\begin{aligned}
\mathcal{L}_{CE}(\hat{y}, y) &= -\sum_{i \in S}^{n_y} \log \hat{y}_i \\
&= (-L) \sum_{i \in S}^{n_y} \frac{1}{L} \log \hat{y}_i \\
&\geq (-L) \log \left(\sum_{i \in S}^{n_y} \frac{1}{L} \hat{y}_i \right) \qquad \text{(by Jensen's Inequality)} \\
&= (-L) \log \frac{1}{L} \qquad \text{(softmax sums to 1)} \\
&= L \log L
\end{aligned}$$

Question 5 (Backpropagation , 28 points)



A siamese neural network consists of twin networks which accept distinct inputs but share the same weights. The outputs of the twin networks are usually joined later on by more layers.
Let's assume we have a two layer siamese neural network, as defined below:

$$
\begin{aligned}
z_1 &= W_1 x^{(i)} + b_1 \\
a_1 &= ReLU(z_1) \\
z_2 &= W_1 x'^{(i)} + b_1 \\
a_2 &= ReLU(z_2) \\
a &= a_1 - a_2 \\
z_3 &= W_2 a + b_2 \\
\hat{y}^{(i)} &= \sigma(z_3) \\
L^{(i)} &= y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \\
J &= -\frac{1}{m} \sum_{i=1}^{m} L^{(i)}
\end{aligned}
$$

Note that $x^{(i)}$, $\cancel{x'^{(i)}}$ represents a pair of single input examples, and are each of shape $D_x \times 1$. Further $y^{(i)}$ is a single output label and is a scalar. There are m examples in our dataset. We use D_{a_1} nodes in our first hidden layers; that is, z_1's ~~and z_2~~'s shape is $D_{a_1} \times 1$. Note that the first two layers share the same weights.

(i) **(2 points)** What are the shapes of $\underline{W_1}, \underline{b_1}, \underline{W_2}, \overset{1}{\underline{b_2}}$? If we were vectorizing across multiple examples, what would the shapes of X and Y be instead?

$a_1 \times d_1$ $a_1 \times 1$ $1 \times a_1$

Solution: $W_1 \in \mathcal{R}^{D_{a1} \times D_x}, b_1 \in \mathcal{R}^{D_{a1} \times 1}, W_2 \in \mathcal{R}^{1 \times D_{a1}}, b_2 \in \mathcal{R}^{1 \times 1}$.
$X \in \mathcal{R}^{D_x \times m}, y \in \mathcal{R}^{m \times 1}$ after vectorizing.

(ii) **(2 points)** What is $\partial J / \partial z_3$? Refer to this result as $\delta_1^{(i)}$. Hint: You can simplify the equation in terms of $\hat{y}^{(i)}$.

Solution:

$$-\frac{1}{m}(y^{(i)}/\hat{y}^{(i)} - (1 - y^{(i)})/(1 - \hat{y}^{(i)})) * \sigma(z_3)(1 - \sigma(z_3))$$

or

$$\frac{1}{m}(\hat{y}^{(i)} - y^{(i)})$$

(iii) **(2 points)** What is $\partial z_3 / \partial a$? Refer to this result as $\delta_2^{(i)}$.

Solution: W_2

(iv) **(2 points)** What is $\partial a / \partial z_2$? Refer to this result as $\delta_3^{(i)}$.

Solution: $-H(z_2)$, where H is the Heaviside step function.

(v) **(2 points)** What is $\partial a/\partial z_1$? Refer to this result as $\delta_4^{(i)}$.

Solution: $H(z_1)$, where H is the Heaviside step function.

(vi) **(2 points)** What is $\partial z_1/\partial W_1$?

Solution: ${x^{(i)}}^T$

(vii) **(6 points)** What is $\partial J/\partial W_1$? It may help to reuse work from the previous parts. Hint: Be careful with the shapes!

Solution: Using chain rule for partial derivatives, we get:
(Note the element-wise product)

$$\delta_1^{(i)} * \delta_2^{(i)} \circ (\delta_3^{(i)} * {x'^{(i)}}^T + \delta_4^{(i)} * {x^{(i)}}^T)$$

(viii) **(10 points)** Write down the update rules for W_1, b_1 and W_2, b_2 using simple gradient descent with learning rate α. All gradients should be expanded reusing work from previous parts.

Solution:

$$\begin{aligned}
\partial J/\partial W_1 &= \delta_1^{(i)} * \delta_2^{(i)} \circ (\delta_3^{(i)} * x'^{(i)^T} + \delta_4^{(i)} * x^{(i)^T}) \\
\partial J/\partial b_1 &= \delta_1^{(i)} * \delta_2^{(i)} \circ (\delta_3^{(i)} + \delta_4^{(i)}) \\
\partial J/\partial W_2 &= \delta_1^{(i)} * a^T \\
\partial J/\partial b_2 &= \delta_1^{(i)}
\end{aligned}$$

$$W_i^{(t+1)} = W_i^t - \alpha \sum_{i=1}^{m} \partial J/\partial W_i$$

$$b_i^{(t+1)} = b_i^t - \alpha \sum_{i=1}^{m} \partial J/\partial b_i$$

Question 6 (Numpy Coding , 14 points)

You are hired by Dr. Doofenshmirtz of Evil, Inc to create his latest weapon: the `video-flashinator`. Dr. Doofenshmirtz's mother suffers from Epilepsy and your task is to find all the flashes in a given video using Numpy. However, you don't have too long because Perry the Platypus is on his way to stop your plans. You are given a pseudocode that you must convert into numpy. Fast!
Hint: Vectorize your code to thwart Perry's plans.

- Solution with no loops: 14 points
- Solution with one loop: 12 points
- Solution with 2+ loops: 10 points

Pseudocode:

You are given a numpy array of shape $(F, 1280, 720, 3)$ where F is the number of frames (a frame is a still image in a video), 1280x720 is the dimension of each frame, and each image has 3 channels.

1. Convert it to a grayscale image with dimension (F, 1280, 720) and apply the following formula to each pixel to get the luminance values for each pixel.

$$f(x) = 4 \cdot (0.2x + 1)^{2.2}$$

2. For each frame (apart from the first one) subtract the previous one from it to get the delta in luminance for each pixel in this frame. These changes can be positive or negative. You will have $(F - 1)$ of these with 1280x720 luminance delta arrays. We call these delta-frames.

3. Create a table T of shape $(F - 1, 1)$, with one value corresponding to each delta-frame. To calculate this: find the average of the 5 largest positive changes for each of these, and the average of the 5 largest negative changes. Find the one with the largest magnitude and that is the value for this diff frame.

   Example 1 Let's say the 5 most positive values are $[10, 10, 10, 10, 20]$. Average $= 12$

   And the 5 most negative values are $[-10, -10, -10, -10, -10]$. Average $= -10$

$$|12| > |-10| \rightarrow T[i] = 12$$

   Example 2 Let's say the 5 most positive values are $[10, 10, 10, 10, 10]$. Average $= 10$

   And the 5 most negative values are $[-10, -10, -10, -10, -20]$. Average $= -12$

$$|10| < |-12| \rightarrow T[i] = -12$$

4. Return the table T.

```
import numpy as np

## params: frames: np.array (F, 1280, 720, 3)
## returns: T: np.array (F - 1, 1)

def video_flashinator(frames):
```

Solution:

```python
import numpy as np

## params: frames: np.array (F, 1280, 720, 3)
## returns: T: np.array (F - 1, 1)

def video_flashinator(frames):
    avg_lum_frames = 4 * (0.2 * frames.mean(axis=3) + 1)**2.2
    change_lum = np.delete(avg_lum_frames, 0, 0) - np.delete(avg_lum_frames, -1, 0)

    pos_lum = change_lum.copy().reshape(change_lum.shape[0], -1)
    pos_lum[pos_lum < 0] = 0
    pos_lum.sort(axis=1)
    pos_lum = np.flip(pos_lum, axis=1)

    neg_lum = -change_lum.copy().reshape(change_lum.shape[0], -1)
    neg_lum[neg_lum < 0] = 0
    neg_lum.sort(axis=1)
    neg_lum = np.flip(neg_lum, axis=1)

    p_avgL = pos_lum[:,:10].mean(axis=1)
    n_avgL = neg_lum[:,:10].mean(axis=1)

    T = p_avgL - n_avgL
    T[T > 0] = p_avgL[T > 0]
    T[T <= 0] = -n_avgL[T <= 0]
    return T
```

END OF PAPER