CS230: Deep Learning

Spring Quarter 2021
Stanford University

Midterm Examination

Suggested duration: 180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice	16	
2	Short Answers	16	
3	Convolutional Architectures	20	
4	L^1 regularization	13	
5	Backpropagation with GANs	25	
6	Numpy Coding	15	
Total		105	

The exam contains 21 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/9dhx0l4jaqmk3o1fk3egfuisbvbx69k1)
- 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)** Suppose you have a CNN model for image classification, with multiple Conv, Max pooling, ReLU activation layers, and a final Softmax output. Ignoring the bias and numerical precision issues, which of the statements below are true?

(i) Multiplying the weights by a factor of 10 during inference does not affect the prediction accuracy.

(ii) Multiplying the weights by a factor of 10 during training does not affect training convergence.

(iii) Multiplying the input data by a factor of 10 during inference does not affect the prediction accuracy.

(iv) Subtracting the input data by its mean per channel during inference does not affect the prediction accuracy.

Solution: (i) (iii).
For (ii), multiplying the weights will change the gradient/weight ratio. For (iv), subtracting by mean is a data normalization technique and will affect prediction accuracy.

(b) **(2 points)** Select the methods that can mitigate gradient exploding

(i) Using ReLU activation instead of sigmoid.

(ii) Adding Batch Normalization layers.

(iii) Applying gradient clipping.

(iv) Using residual connection.

Solution: (ii) (iii).
(i) and (iv) are proposed to combat gradient diminishing problems. (ii) can mitigate both gradient diminishing and exploding. (iii) can avoid gradient exploding.

(c) **(2 points)** Select the statements that are true

(i) For a linear classifier, initializing all the weights and biases to zero will result in all the elements in the final W matrix to be the same.

(ii) If we have a small dataset consisting of handwritten alphabets A-Z, and we wish to train a handwriting recognition model, it would be a good idea to augment the dataset by randomly flipping each image horizontally and vertically, given that we know that each letter is equally represented in the original dataset.

(iii) If we have a trained softmax classifier that yields 100% accuracy on the dataset and we change the weights matrix W to 3W, the classifier will maintain the same accuracy and will have a smaller cross entropy loss without regularization.

(iv) If we are using the KNN-method with $L1$ distances to classify images, horizontally flipping each training and test image will not lead to a change in the prediction accuracy.

Solution: **(iv)**.

(d) **(2 points)** Select the statements that are true

(i) If the input to a ConvNet is a zero image (all zeros), the class probabilities will be uniform.

(ii) You train a model and you observe that the validation set accuracy is significantly lower than the train accuracy. Your friend suggests you to use batch normalization and you agree it is a good idea. After using batch normalization, the new model is likely to have a smaller gap between the train and validation accuracies.

(iii) For an n-dimensional vector y, the softmax of y will be the same as the softmax of cy, where c is any real number since softmax normalises the predictions to yield a probability distribution.

(iv) For data normalization, we typically compute the mean and standard deviation across the entire dataset, before splitting it into train/val/test splits.

Solution: **(ii)**.

(e) **(2 points)** At test time, a Dropout layer (implemented as Inverse Dropout) with a dropout probability p will:

(i) Multiply the input tensor by p.

(ii) Drop neurons from the input with probability p.

(iii) Divide the input tensor by p.

(iv) None of the above.

Solution: **(iv)**.

(f) **(2 points)** Which of the following is/are true about the momentum update rule?

(i) It has no hyperparameters.

(ii) It better avoids local minima by keeping running gradient statistics.

(iii) If the network has n parameters, momentum requires that we keep track of O(n) extra parameters

(iv) None of the above

Solution: **(ii), (iii)**.

(g) **(2 points)** Which of the following are true for early stopping during training:

(i) It may reduce the necessity to tune the hyperparameter for number of training epochs

(ii) It increases model variance

(iii) When accuracy reaches a trough on the validation set, we invoke early stopping

(iv) It may dramatically speed up training at the cost of learning less optimal parameters

Solution: **(i), (iv)**.

(h) **(2 points)** Which of the following are true for backpropagation:

(i) If the neural network has $O(n)$ parameters, backpropagation is an $O(n^2)$ operation

(ii) Backpropagation works only if a computational graph has no directed cyclic paths

(iii) We update the β parameter in Adam using backpropagation

(iv) We update the γ parameter in Batch Normalization using backpropagation

Solution: **(ii), (iv)**.

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 begin training a Neural Network, but the loss evolves to be completely flat. List two possible reasons for this.

Solution:

- It is possible that the weights are incorrectly initialized.
- It is also possible that the learning rate is too low.
- Some other answers may also be possible (for eg X not correlated with Y at all is a possibility)

Solutions based on the regularization parameter being too high or uneven distribution of class labels in the dataset are not accepted.

(b) **(2 points)** Your CS 230 project is in collaboration with the California PD, and they require you to identify criminals, given their data. Since being imprisoned is a very severe punishment, it is very important for your deep learning system to not incorrectly identify the criminals, and simultaneously ensure that your city is as safe as possible. What evaluation metric would you choose and why?

Solution: For the model to be a good one, you want to be sure that the person you catch is a criminal (Precision) and you also want to capture as many criminals (Recall) as possible. The F1 score manages this tradeoff.
Students who mention both precision and recall can also be given full credit.

(c) **(2 points)** Although Pooling layers certainly cause a loss of information between Convolutional layers, why would we add Pooling layers to our network?

Solution: Pooling layers cause spatial dimensions to shrink and allow us to use fewer parameters to obtain smaller and smaller hidden representations of the input.

(d) **(2 points)** What does it mean for your model to have high variance? Give one possible way reduce variance in your model.

Solution: It means that your model has overfit the train data/does not exhibit generalizability. Accept any answer that helps with generalizability such as adding more data, adding regularization/dropout, creating a smaller model, etc.

(e) **(2 points)** List one advantage and one disadvantage of having a small batch size for training.

Solution: Disadvantage: Slower training speed; Worse hardware utilization; Problem with batch normalization

Advantage: Less memory consumption; Small batches can offer a regularizing effect that provides better generalization

(f) **(2 points)** Why softmax function is often used for classification problems?

Solution: Softmax output fulfills the constraints of a probability density. (Or other questions mentioning that softmax outputs 0-1 values which sum to 1.)

(g) **(2 points)** L1 regularization gives us more explainable models as they are sparse, so we can discard most covariates as having no effect on the outcome variable; hence we should prefer L1 regularization to L2 even at the cost of performance. Do you agree with this statement? If yes, give an example. If not, why not?

Solution: this is not true as the discarded variables might have strong correlation with a non discarded variable and only one of them might have a non zero coefficient with l1 regularization at random

(h) **(2 points)** In a true/false classification problem, if you had class imbalance (many more false class labels) during training and you downsampled the false class, how would you account for this during testing?

Solution: One would have to adjust the threshold at which one predicts true/-false, and move the true threshold 0.5 to a higher value. Or we upweight downsampled class examples. We also accept answers talking about separately measuring performance on both classes depending on specific problem setting. All answers indicating the same idea will be accepted.

Question 3 (Convolutional Architectures, 20 points)

Say you have an input image whose shape is $128 \times 128 \times 3$. You are deciding on the hyperparameters for a Convolutional Neural Network; in particular, you are in the process of determining the settings for the first Convolutional layer. Compute the output activation volume dimensions and number of parameters of each of the possible settings of the first Convolutional layer, given the input has the shape described above. You can write the activation shapes in the format (H, W, C) where H, W, C are the *height*, *width*, and *channel* dimensions, respectively.

i. **(2 points)** The first Convolutional layer has a stride of 1, a filter size of 3, input padding of 0, and 64 filters.

Solution: Activation volume dimensions: $14 \times 14 \times 64$
Number of parameters: $(3 * 3 * 3 + 1) * 64 = 1792$

ii. **(2 points)** The first Convolutional layer has a stride of 1, a filter size of 5, input padding of 2, and 16 filters.

Solution: Activation volume dimensions: $128 \times 128 \times 16$
Number of parameters: $(5 * 5 * 3 + 1) * 16 = 1216$

iii. **(2 points)** The first Convolutional layer has a stride of 2, a filter size of 2, input padding of 0, and 32 filters.

Solution: Activation volume dimensions: $64 \times 64 \times 32$
Number of parameters: $(2 * 2 * 3 + 1) * 32 = 416$

Now that you have determined the output shapes and number of parameters for these configurations, you are going to create a deeper CNN. Say you create a CNN made of three identical modules, each of which consists of: a Convolutional layer, a Max-Pooling layer, and a ReLU layer. All Pooling layers will have a stride of 2 and a width/height of 2. For example, say we define the Convolutional layer to have stride 1, filter size 1, input padding of 0, and 8 filters. Then the module architecture would be:

- $1 \times 1 \times 8$ Conv with stride 1 and 0 padding
- 2×2 Max-Pool with stride 2
- ReLU

Three such modules make up the entire network. Given the following Convolutional hyperparameters, compute the output activation volume dimensions after passing the input through the entire network, as well as the number of parameters in the entire network

iv. **(4 points)** The Conv layers have a stride of 1, a filter size of 3, input padding of 0, and 64 filters.

Solution: Activation volume dimensions: $14 \times 14 \times 64$
Number of parameters: $(3 * 3 * 3 + 1) * 64 + (3 * 3 * 64 + 1) * 64 * 2 = 75,648$

v. **(4 points)** The Conv layers have a stride of 1, a filter size of 5, input padding of 2, and 16 filters.

Solution: Activation volume dimensions: $16 \times 16 \times 16$
Number of parameters: $(5 * 5 * 3 + 1) * 16 + (5 * 5 * 16 + 1) * 16 * 2 = 14,048$

vi. **(4 points)** The Conv layers have a stride of 2, a filter size of 2, input padding of 0, and 32 filters.

Solution: Activation volume dimensions: $2 \times 2 \times 32$
Number of parameters: $(2 * 2 * 3 + 1) * 32 + (2 * 2 * 32 + 1) * 32 * 2 = 8672$

vii. **(2 points)** Say you want to solve a 10-class classification problem given the input (e.g. the input represents a digit, and you want to predict which digit it is). To do so, you decide to add a Fully-Connected layer after your three Conv-Pool-ReLU modules. The input to the FC layer is the output of the last module, completely flattened. Which of the three previous architectures (given in parts iv., v., and vi.) will allow for the FC layer to have the smallest number of parameters? Give the number of parameters for said layer.

Solution: The last of the three i.e. the architecture from part vi. It has the smallest output, and will result in a FC layer with $(2 * 2 * 32 + 1) * 10 = 1290$ parameters.

Question 4 (L^1 regularization (Lasso), 13 points)

L^1 regularization on the model parameter $\mathbf{w}$ is defined as

$$||\mathbf{w}||_1 = \sum_i |w_i|$$

In this question, you are asked to apply L^1 regularization to a neural network model, of which the original objective function is defined as $J(\mathbf{w}; \mathbf{X}, \mathbf{y})$. The regularized objective function after adding the L^1 normalization term becomes:

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \alpha||\mathbf{w}||_1$$

i. **(3 points)** Write down the corresponding gradient of the regularized objective function $\tilde{J}$. Hint: your answer should include an element-wise sign function $sign(x)$.

Solution:

$$\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \alpha sign(\mathbf{w})$$

ii. **(6 points)** It could be difficult to get a clean algebraic solution to the previous gradient expression. To study how this L^1 normalization term could affect the converged weights, we make following simplifications:

- We apply Taylor expansion to $J(\mathbf{w}; \mathbf{X}, \mathbf{y})$ and discard high-order terms. Specifically, we approximate the original objective function J with

  $$\hat{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = J(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

  , where $\mathbf{w}^*$ is the optimal parameter for the objective function without the regularization term and $\mathbf{H}$ is the Hessian matrix.

- We assume the Hessian is diagonal, $\mathbf{H} = diag([H_{1,1}, ..., H_{n,n}])$, where each $H_{i,i} > 0$. This assumes that there is no correlation between the input features.

Now, write down an analytical solution for each element w_i of $\mathbf{w}$ when the gradient equals to zero. Your answer should be expressed with weights w_i^* ($\mathbf{w}^*$ is the the optimal weights for an unregularized object function), Hesian matrix elements $H_{i,i}$, and coefficient α. Hint: the gradient of $|x|$ at $x = 0$ can take any value between $[-1, 1]$.

Solution:

$$w_i = sign(w_i^*)max\{|w_i^* - \frac{\alpha}{H_{i,i}}, 0\}$$

It is okay to write down the expression under different conditions like:

$$w_i = w_i^* + \frac{\alpha}{H_{i,i}}, \quad \text{when} \quad w_i^* \leq -\frac{\alpha}{H_{i,i}}$$

$$w_i = 0, \quad \text{when} \quad \frac{\alpha}{H_{i,i}} > w_i^* > -\frac{\alpha}{H_{i,i}}$$

$$w_i = w_i^* - \frac{\alpha}{H_{i,i}}, \quad \text{when} \quad w_i^* \geq \frac{\alpha}{H_{i,i}}$$

iii. **(4 points)** Given your answer to previous question, explain how L^1 regularization will affect the weights differently from L^2 regularization. You must discuss large weights and small weights separately.

Solution:

- (2 points) For small weights, L^1 regularization will push them to zeros while L^2 will not. Or: L^1 regularization can lead to sparse weights.

- (2 points) For large weights, L^1 shifts them towards zero by a fixed distance, while L^2 will shift larger weights by longer distances.

Question 5 (Backpropagation with GANs)

In this question, we will workout backpropagation with Generative Adversarial Networks (GANs).

Recall that a GAN consists of a Generator and a Discriminator playing a game. The Generator takes as input a random sample from some noise distribution (e.g., Gaussian), and its goal is to produce something from a target distribution (which we observe via samples from this distribution). The Discriminator takes as input a batch consisting of a mix of samples from the true dataset and the Generator's output, and its goal is to correctly classify whether its input comes from the true dataset or the Generator.

Definitions:

- $X^1, ..., X^n$ is a minibatch of n samples from the target data generating distribution For this question, we suppose that each X^i is a k dimensional vector. For example, we might be interested in generating a synthetic dataset of customer feature vectors in a credit scoring application
- $Z^1, ..., Z^n$ is a minibatch of n samples from some predetermined noise distribution Note that in general these minibatch sizes may be different.
- The generator $g(.; \theta_g) : \mathcal{Z} \to \mathcal{X}$ is a neural network
- The discriminator $d(.; \theta_d) : \mathcal{X} \to (0, 1)$ is a neural network

The log likelihood of the output produced by the discriminator is:

$$L(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \log D(X^i) + \log(1 - D(G(Z^i)))$$

The training of such a GAN system proceeds as follows; given the generator's parameters, the discriminator is optimized to maximize the above likelihood. Then, given the discriminator's parameters, the generator is optimized to minimize the above likelihood. This process is iteratively repeated. Once training completes, we only require the generator to generate samples from our distribution of interest; we sample a point from our noise distribution and map it to a sample using our generator.

The Discriminator (The generator architecture is defined analogously)

- Consider the discriminator to be a network with layers indexed by $1, 2, .., L_d$ for a total of L_d layers.
- Let the discriminator's weight matrix for layer l be W_d^l; lets assume there are no biases for simplicity
- Let the activations produced by a layer l be given by A_d^l, and the pre activation values by z_d^l. Write down

- Let $g_d^l(.)$ be the activation function at layer l

i. The goal of the discriminator is to maximize the above likelihood function. Write down $\nabla_{\theta_d} L(X; \theta_d, \theta_g)$ in terms of $\nabla_{\theta_d} D(.)$ (3 points)

Solution: $\nabla_{\theta_d} L(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \frac{\nabla_{\theta_d} D(X^i)}{D(X^i)} - \frac{\nabla_{\theta_d} D(G(Z^i))}{1-D(G(Z^i))}$

ii. Write down

$$\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}}$$

taking help from your answer in the previous subpart. Remember that the activation function in the last layer of the discriminator is a sigmoid function as the output of the discriminator is a probability. (4 points)

Solution: $\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\sigma(z_d^{L_d}(X_i)) * (1-\sigma(z_d^{L_d}(X_i))}{D(X^i)} - \frac{\sigma(z_d^{L_d}(G(Z_i)) * (1-\sigma(z_d^{L_d} G(Z_i)))}{1-D(G(Z^i))}$
OR $\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}} = \frac{1}{n} \sum_{i=1}^{n} (1 - \sigma(z_d^{L_d}(X_i))) - \sigma(z_d^{L_d}(G(Z_i)))$
OR $\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}} = \frac{1}{n} \sum_{i=1}^{n} (1 - D(X_i))) - D(G(Z_i)))$
OR, partial credit (2 points) for

$\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^{L_d}} = \frac{1}{n}\sum_{i=1}^{n} \frac{g_d'^{L_d}(z_d^{L_d})}{D(X^i)} - \frac{g_d'^{L_d}(z_d^{L_d})}{1-D(G(Z^i))}$

iii. Write down recursively

$$\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^l}$$

in terms of $\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^{l+1}}$

Hint: Your answer will contain w_d^{l+1}, $g_d'^l(.)$ and z_d^l (5 points)

Solution:

$$\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^l} = ((w_d^{l+1})^T \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{l+1}}) g_d'^l(z_d^l)$$

iv. Let the output of the generator be $g(z_i; \theta_g)$

Write down

$$\frac{\partial L(\theta_d, \theta_g)}{\partial g(z_i; \theta_g)}$$

in terms of w_d^1, and $\frac{\partial L(\theta_d,\theta_g)}{\partial z_d^1}$(5 points) Hint: The variable over which we calculate gradient here is similar to z_d^0, except it is supplied post activation Note: This is also how we take gradients wrt inputs of a network; this is useful/required in some tasks. (3 points)

Solution:

$$\frac{\partial L(\theta_d, \theta_g)}{\partial g(z_i; \theta_g)} = (w_d^1)^T \frac{\partial L(\theta_d, \theta_g)}{\partial z_d^1}$$

v. Now we move to the generator. The goal of the generator is to minimize the above likelihood function. Write down $\nabla_{\theta_g} L(\theta_d, \theta_g)$ in terms of $\nabla_{\theta_g} g(.)$ and in terms of $\frac{\partial L(\theta_d, \theta_g)}{\partial g(z_i; \theta_g)}$ calculated in the previous part (5 points)

Solution: $\nabla_{\theta_g} L(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta_g} g(Z^i) . \frac{\partial L(\theta_d, \theta_g)}{\partial g(z_i; \theta_g)}$

vi. Write down a simple gradient based update rule (don't use RMSprop or Momentum, and assume no regularization) for θ_d^{t+1} in terms of $\nabla_{\theta_d} L(\theta_d, \theta_g)$, fixed learning rate α and the current parameters θ_d^t (1 point)

Solution: $\theta_d^{t+1} = \theta_d^t + \alpha \nabla_{\theta_d} L(\theta_d, \theta_g)$

vii. Write down a simple gradient based update rule (don't use RMSprop or Momentum, and assume no regularization) for θ_g^{t+1} in terms of $\nabla_{\theta_g} L(\theta_d, \theta_g)$, fixed learning rate α and the current parameters θ_d^t (1 point)

Solution: $\theta_g^{t+1} = \theta_g^t - \alpha \nabla_{\theta_g} L(\theta_d, \theta_g)$

viii. Now assume you decide to (lazily) use a simplified version of the objective, which we call "Likelihood Version", just to test how it works. Define the following alternative likelihood function.

$$L'(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} D(X^i) + D(G(Z^i)))$$

Note that this likelihood is high when the original likelihood is high, and low otherwise, in general, so it is possible that it works. Write down $\nabla_{\theta_d} L'(\theta_d, \theta_g)$ in terms of $\nabla_{\theta_d} D(.)$ (3 points)

Solution: $\nabla_{\theta_d} L'(\theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta_d} D(X^i) - \nabla_{\theta_d} D(G(Z^i))$
You get full credit if you fixed the typo in this likelihood (there is a minus instead of a plus)

It turns out this new Likelihood (along with gradient clipping) ends up minimizing the earth mover or Wasserstein distance between the target distribution and your generated samples, and the GAN optimized as such is called a WGAN. WGANs have been shown to work better than standard GANs in many cases, and are often preferred to GANs

Question 6 (Numpy Coding , 10 points + 5 points extra credit)

Human visual attention allows us to focus on a certain region with "high resolution" while perceiving the surrounding image in "low resolution", and then adjust the focal point or do the inference accordingly. Given a small patch of an image, pixels in the rest provide clues what should be displayed there.

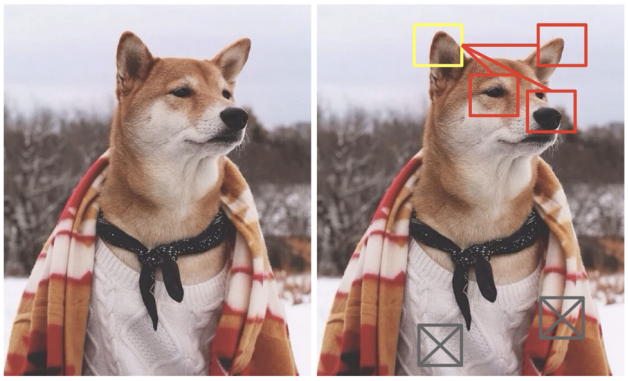

Figure 1: The image on the right is depicting attention computed over particular regions of an image, spotting different features

Attention in deep learning can be broadly interpreted as a kernel of importance weights. Mathematically, it can be described as mapping a *query* and a set of *key-value* pairs to an *output*. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
The Stanford Vision Lab has asked you to implement attention on an image dataset, and they also provide you with the steps required to compute the attention values.
You are given a NumPy ndarray: an image array $\in \mathbb{R}^{H,W,iC}$, and your output is another NumPy ndarray $\in \mathbb{R}^{H,W,oC}$. Here,

- iC: Number of input channels.
- H, W: Height and width of an image frame.
- oC: Number of output channels.

Preprocessing. First, we want our model to be scale invariant; you can achieve it using matrix normalization. In this case, you will convert each image frame to its unit normalized version, using the Frobenius norm of each channel individually. For your convenience, the Frobenius norm for a matrix $A \in \mathbb{R}^{m \times n}$ ($\|A\|_F$) has been defined below.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |\, a_{ij} \,|^2}$$

Computing Attention Scores. Now, we compute attention scores for each pixel in the image following figure 2. At each index $(i, j), i \in H, j \in W$, we consider the query array q_{ij} as pixel array of the image across all channels ($\in \mathbb{R}^{iC}$)and compute the corresponding score, y_{ij} using the following equation. *Hint*: You might want to look at `np.squeeze`

$$y_{ij} = \sum_{a,b=(1,1)}^{(3,3)} \text{softmax}(\mathbf{q_{ij}^T k_{a,b}}) \odot v_{a,b}$$

This process is very similar to convolutions, but instead of convolving with a filter/kernel, we convolve the image with the *key* kernel, compute the softmax output, and compute the final score by multiplying and summing with *value* kernel. For simplicity, assume that the model parameters $\mathbf{k}$ and $\mathbf{v}$ of the given shapes are provided.
In this problem, for each output channel, we take *key* kernel $\mathbf{k}$ of size $3 \times 3 \times iC$ and value kernel $\mathbf{v}$ of size $3 \times 3 \times 1$, and $\mathbf{k_{a,b}} \in \mathbb{R}^{iC}$.

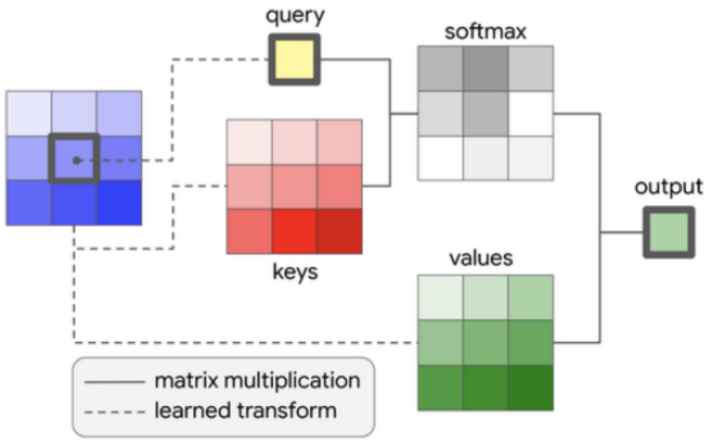

Figure 2: Computation of attention scores for a single pixel across input channels

Given the image, and kernels k and v, you must use NumPy operations to compute the attention scores. Note that comparing attention score calculation to convolution operations, the stride size is 1, and there is no padding.

1. What would be the total number of trainable parameters in your self-attention model, given that the output array $Y \in \mathbb{R}^{H,W,oC}$.

2. Following the above pseudocode and carefully following the dimensional rules, write down an attention model implemented using Python for-loops. We will be awarding extra credit if you could figure out how to replace the loops with NumPy subroutines. We have also provided a snippet of the code as a starter.

Solution:

Number of trainable parameters =

$$3 * 3 * iC * oC + 3 * 3 * 1 * oC = 180$$

END OF PAPER