CS230: Deep Learning

Winter Quarter 2021

Stanford University

Midterm Examination

180 minutes

	Problem	Full Points	Your Score
1	Multiple Choice	16	
2	Short Answers	16	
3	Convolutional Architectures	20	
4	Augmenting with Noise	13	
5	Binary Classification	10	
6	Backpropagation	20	
7	Numpy Coding	20 (+10)	
Total		115 (+10)	

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: https://stanford.box.com/s/9dhx0l4jaqmk3o1fk3egfuisbvbx69k1 (Access to this Stanford Box link requires you to be signed in with your Stanford account)
- 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)** Your model for classifying different dog species is getting a high training set error. Which of the followings are promising things to try to improve your classifier?

(i) Use a bigger neural network

(ii) Get more training data

(iii) Increase the regularization parameter lambda

(iv) Increase the parameter keep_prob in dropout layer (assume the classifier has dropout layers)

(b) **(2 points)** Which of the followings are true about Batch Normalization?

(i) Batch Norm layers are skipped at test time because a single test example cannot be normalized.

(ii) Its learnable parameters can only be learned using gradient descent or mini-batch gradient descent, but not other optimization algorithms.

(iii) It helps speed up learning in the network.

(iv) It introduces noise to a hidden layer's activation, because the mean and the standard deviation are estimated with a mini-batch of data.

(c) **(2 points)** If your input image is 64x64x16, how many parameters are there in a single 1x1 convolution filter, including bias?

(i) 2

(ii) 17

(iii) 4097

(iv) 1

(d) **(2 points)** Which *one* of the following statements on initialization is *false*?

(i) The variances of layer outputs remain unchanged during training when Xavier initialization is used.

(ii) Initializing all weights to a positive constant value isn't sufficient to break learning symmetry during training.

(iii) Different activation functions may benefit from different types of initializations.

(iv) It's possible to break symmetry by initializing weights to be sampled uniformly from the set $\{-1, 1\}$.

(e) **(2 points)** The shape of your input image is (n_h, n_w, n_c); the convolution layer uses a 1-by-1 filter with stride = 1 and padding = 0. Which of the following statements are correct?

(i) You can reduce n_c by using 1x1 convolution. However, you cannot change n_h, n_w.

(ii) You can use a standard maxpooling to reduce n_h, n_w, but not n_c.

(iii) You can use a 1x1 convolution to reduce n_h, n_w, n_c.

(iv) You can use maxpooling to reduce n_h, n_w, n_c.

(f) **(2 points)** Which of the following statements on regularization are true?

(i) Using L-2 regularization enforces a Laplacian prior on your network weights.

(ii) Batch normalization can have an implicit regularizing effect, especially with smaller minibatches.

(iii) Layernorm's regularization effect increases with larger batch sizes.

(iv) Your choice of regularization factor can cause your model to underfit.

(g) **(2 points)** What is the benefit of using Momentum optimization?

(i) Simple update rule with minimal hyperparameters

(ii) Helps get weights out of local minima

(iii) Effectively scales the learning rate to act the same amount across all dimensions

(iv) Combines the benefits of multiple optimization methods

(h) **(2 points)** Which of the below can you implement to solve the exploding gradient problem?

(i) Use SGD optimization

(ii) Oversample minority classes

(iii) Increase the batch size

(iv) Impose gradient clipping

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)** Why is scaling (γ) and shifting (β) often applied after the standard normalization in the batch normalization layer?

(b) **(2 points)** You are solving a biometric authentication task (modeled as binary classification) that uses fingerprint data to help users log into their devices. You train a classification model for user A until it achieves $> 95\%$ classification accuracy on a development ("dev") set for user A. However, upon deployment the model fails to correctly authenticate user A about half the time (50% misclassification rate).
List **one** factor you think could have contributed to the *mismatch* in classification rates between the dev set and deployment, and how you'd go about fixing this issue.

(c) **(2 points)** A convolutional neural network has 4 consecutive layers as follows:

3x3 conv (stride 2) - 2x2 Pool - 3x3 conv (stride 2) - 2x2 Pool

How large is the support (the set of image pixels which activate) of a neuron in the 4^{th} non-image layer of this network?

(d) **(2 points)** Is it always a good strategy to train with large batch size? Why or why not?

(e) **(2 points)** What is the purpose of using 1x1 convolution?

(f) **(2 points)** Why is the sigmoid activation function susceptible to the vanishing gradient problem?

(g) **(2 points)** Say you are trying to solve a binary classification problem where the positive class is very underrepresented (e.g. 9 negatives for every positive). Describe precisely a technique which you can use during training which helps alleviate the class imbalance problem. Would you apply this technique at test time? Why or why not?

(h) **(2 points)** Let **p** be the **probability of keeping** neurons in a dropout layer. We have seen that in forward passes, we often scale activations by dividing them by p during training time.

You accidentally train a model with dropout layers *without* dividing the activations by p at train time. How would you resolve this issue at test time? Please justify your answer mathematically.

Question 3 (Convolutional Architectures, 20 points)

Consider a convolutional neural network block whose input size is $64 \times 64 \times 8$. The block consists of the following layers:

- A convolutional layer 32 filters with height and width 3 and 0 padding which has both a weight and a bias (i.e. CONV3-32)
- A 2×2 max-pooling layer with stride 2 and 0 padding (i.e. POOL-2)
- A batch normalization layer (i.e. BATCHNORM)

Compute the output activation volume dimensions and number of parameters of the layers. 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)** What is the output activation volume dimensions and number of parameters for CONV3-32?

ii. **(2 points)** What is the output activation volume dimensions and number of parameters for POOL2?

iii. **(2 points)** What is the output activation volume dimensions and number of parameters for BATCHNORM?

Now you will design a very small convolutional neural network for the task of digit predict: given a 16×16 image (with 3 channels i.e. RGB), you want to predict the digit shown in the image. Therefore, this is a 10-class classification problem.

Your network will have 4 layers, given in order: a convolutional layer, a max-pooling layer, a flatten layer, and a fully-connected layer.

Design the neural network to solve this problem. Of course, there are many, many solutions to this question. To narrow the solution space, here are some restrictions:

- Your convolutional layer must have a stride of 1 and have 4 filters. Because of memory limits, this layer's **activation volume** should not have no more than 576 total elements in the tensor per input image (e.g. a $2 \times 2 \times 8$ tensor has $2 * 2 * 8 = 32$ total elements in it).
- Your max-pooling layer must have the same stride as pool size e.g. a 2×2 pooling layer must have a stride of 2.
- Again because of memory limits, your final fully-connected layer will not have a bias term, and its total number of **parameters** cannot exceed 1440.

The rest is up to you. Answer the following questions to incrementally build out your small CNN architecture.

iv. **(4 points)** What are the hyperparameters of the convolutional layer you propose (i.e. filter size, padding)? Also, what are the activation volume dimensions for this layer?

v. **(4 points)** What are the hyperparameters of the max-pooling layer you propose (i.e. pool size)? Also, what are the activation volume dimensions for this layer?

vi. **(2 points)** What are the activation volume dimensions of the final fully-connected layer?

vii. **(4 points)** You start training your model and notice underfitting, so you decide to add data augmentation as part of your preprocessing pipeline. Given that you are working with images of handwritten digits, for each data augmentation technique, state whether or not the technique is appropriate for the task. If not, explain why not.

(a) Scaling slightly

(b) Flipping vertically or horizontally

(c) Rotating by 90 or 180 degrees

(d) Shearing slightly

Question 4 (Augmenting with Noise, 13 points)

You are tasked with solving a fitting a linear regression model on a set of m datapoints where each feature has some dimensionality d. Your dataset can be described as the set:

$$\{x^{(i)}, y^{(i)}\}_{i=1}^{m}, \text{ where } x^{(i)} \in \mathbb{R}^d$$

For all parts of this problem, assume m is very large (you can consider the limit $m \to \infty$). You initially decide to optimize the loss objective:

$$J = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - x^{(i)T}\theta)^2$$

using Batch Gradient Descent. Here, $\theta \in \mathbb{R}^d$ is your weight vector. Assume you are ignoring a bias term for this problem.

i. **(3 points)** Write each update of the *batch* gradient descent, $\dfrac{\partial J}{\partial \theta}$ in **vectorized** form. Your solution should be a **single** vector (no summation terms) in terms of the matrix $\mathbf{X}$ and vectors $\mathbf{Y}$ and θ, where

$$\mathbf{X} = \begin{bmatrix} x^{(1)^T} \\ \vdots \\ x^{(m)^T} \end{bmatrix} \text{ and } Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

A coworker suggests you augment your dataset by adding Gaussian noise to your features. Specifically, you would be adding *zero-mean, Gaussian* noise of *known vairance* σ^2 from the distribution

$$\mathcal{N}(0, \sigma^2 I)$$

where $I \in \mathbb{R}^{dxd}, \sigma \in \mathbb{R}$

This modifies your original objective to:

$$J_* = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - (x^{(i)} + \delta^{(i)})^T\theta)^2$$

where $\delta^{(i)}$ are **i.i.d.** noise vectors, $\delta^{(i)} \in \mathbb{R}^d$ and $\delta^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$.

ii. **(6 points)** Express the **expectation** of the modified objective J_* over the gaussian noise, $\mathbb{E}_{\delta\sim\mathcal{N}}[J_*]$, as a function of the original objective J added to a term *independent* of your data. Your answer should be in the form

$$\mathbb{E}_{\delta\sim\mathcal{N}}[J_*] = J + C$$

where C is independent of points in $\{x^{(i)}, y^{(i)}\}_{i=1}^m$.

Hint:

- For a Gaussian random vector δ with mean zero, covariance matrix $\sigma^2 I$,

  $$\mathbb{E}_{\delta\sim\mathcal{N}}[\delta\delta^T] = \sigma^2 I$$

  and

  $$\mathbb{E}_{\delta\sim\mathcal{N}}[\delta] = \mathbf{0}$$

  As a consequence, $\mathbb{E}_{\delta\sim\mathcal{N}}[(\delta^T v)^2] = \sigma^2 v^T v$ for a constant vector v

iii. **(2 points)** In expectation, what effect would the addition of the noise have on model overfitting/underfitting? Explain why.

iv. **(2 points)** Consider the limits $\sigma \to 0$ and $\sigma \to \infty$. What impact would these extremes in the value of σ have on model training (relative to no noise added)? Explain why.

Question 5 (Binary Classification, 10 points)

You are building a classification model to distinguish between labels from a *synthetically* generated dataset. More specifically, you are given a dataset,

$$\{x^{(i)}, y^{(i)}\}_{i=1}^{m}, \text{ where } x^{(i)} \in \mathbb{R}^2, y^{(i)} \in \{0, 1\}$$

The data is generated with the scheme:

$$X|Y = 0 \sim \mathcal{N}(2, 2)$$

$$X|Y = 1 \sim \mathcal{N}(0, 3)$$

You can assume the dataset is perfectly balanced between the two classes.

1. **(2 points)** As a baseline, you decide to use a logistic regression model to fit the data. Since the data is synthesized easily, you can assume you have infinitely many samples (i.e. $m \to \infty$). Can your logistic regression model achieve 100% training accuracy? Explain your answer.

2. **(8 points)** After training on a large training set of size M, your logistic regressor achieves a training accuracy of T. Can the following techniques, *applied individually*, improve over this *training accuracy*? Please justify your answer in a single sentence.

   (a) Adding a regularizing term to the binary cross entropy loss function for the logistic regressor

   (b) Standardizing all training samples to have *mean zero* and *unit variance*

   (c) Using a 5-hidden layer feedforward network *without* non-linearities in place of logistic regression

   (d) Using a 2-hidden layer feedforward network *with* ReLU in place of logistic regression

Question 6 (Backpropagation, 20 points)

The softmax function has the desirable property that it outputs a probability distribution, and is often used as activation function in many classification neural networks.
Consider a 2-layer neural network for K-class classification using softmax activation and cross-entropy loss, as defined below:

$$\begin{aligned}
\mathbf{z}^{[1]} &= W^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \\
\mathbf{a}^{[1]} &= \text{LeakyReLU}(\mathbf{z}^{[1]}, \alpha = 0.01) \\
\mathbf{z}^{[2]} &= W^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z}^{[2]}) \\
L &= -\sum_{i=1}^{K} \mathbf{y}_i \log(\hat{\mathbf{y}}_i)
\end{aligned}$$

where the model is given input $\mathbf{x}$ of shape $D_x \times 1$, and one-hot encoded label $\mathbf{y} \in \{0,1\}^K$. Assume that the hidden layer has D_a nodes, i.e. $\mathbf{z}^{[1]}$ is a vector of size $D_a \times 1$. Recall the softmax function is computed as follows:

$$\hat{\mathbf{y}} = [\frac{\exp(\mathbf{z}_1^{[2]})}{Z}, \ldots, \frac{\exp(\mathbf{z}_K^{[2]})}{Z}]$$

where $Z = \sum_{j=1}^{K} \exp(\mathbf{z}_j^{[2]})$

(i) **(2 points)** What are the shapes of $W^{[2]}, b^{[2]}$? If we were vectorizing across m examples, i.e. using a batch of samples $X \in \mathcal{R}^{D_x \times m}$ as input, what would be the shape of the output of the hidden layer?

(ii) **(2 points)** What is $\partial \hat{\mathbf{y}}_k / \partial z_k^{[2]}$? Simplify your answer in terms of element(s) of $\hat{\mathbf{y}}$.

(iii) **(2 points)** What is $\partial \hat{\mathbf{y}}_k / \partial z_i^{[2]}$, for $i \neq k$? Simplify your answer in terms of element(s) of $\hat{\mathbf{y}}$.

(iv) **(3 points)** Assume that the label $\mathbf{y}$ has 1 at its k^{th} entry, and 0 elsewhere. What is $\partial L / \partial z_i^{[2]}$? Simplify your answer in terms of $\hat{\mathbf{y}}_i$. Hint: Consider both cases where $i = k$ and $i \neq k$.

(v) **(1 points)** What is $\partial z^{[2]} / \partial a^{[1]}$? Refer to this result as δ_1.

(vi) **(2 points)** What is $\partial a^{[1]} / \partial z^{[1]}$? Refer to this result as δ_2. Feel free to use curly brace notation.

(vii) **(6 points)** Denote $\partial L / \partial z^{[2]}$ with δ_0. What is $\partial L / \partial W^{[1]}$ and $\partial L / \partial b^{[1]}$? You can reuse notations from previous parts. Hint: Be careful with the shapes.

(viii) **(2 points)** To avoid running into issues with numerical stability, one trick may be used when implementing the softmax function. Let $m = \max_{i=1}^{K} z_i$ be the max of z_i, then

$$\hat{\mathbf{y}}_i = \frac{\exp(\mathbf{z}_i^{[2]} - m)}{\sum_{j=1}^{K} \exp(\mathbf{z}_j^{[2]} - m)}$$

What is the numerical problem with the initial softmax computation? Why the modified formula would help resolving that problem?

Question 7 (Numpy Coding , 20 points (+10 bonus points))

Stanford Medical School reaches out to you for your help to analyze an MRI dataset. Since the dataset contains 3-D images, you decide to build a model with 3-D convolution (Conv3D). However, you soon find that your most powerful weapon, NumPy, does not have a good Conv3D implementation. Instead of waiting for NumPy's developers to respond to your feature request, you figure that you would just implement one yourself.

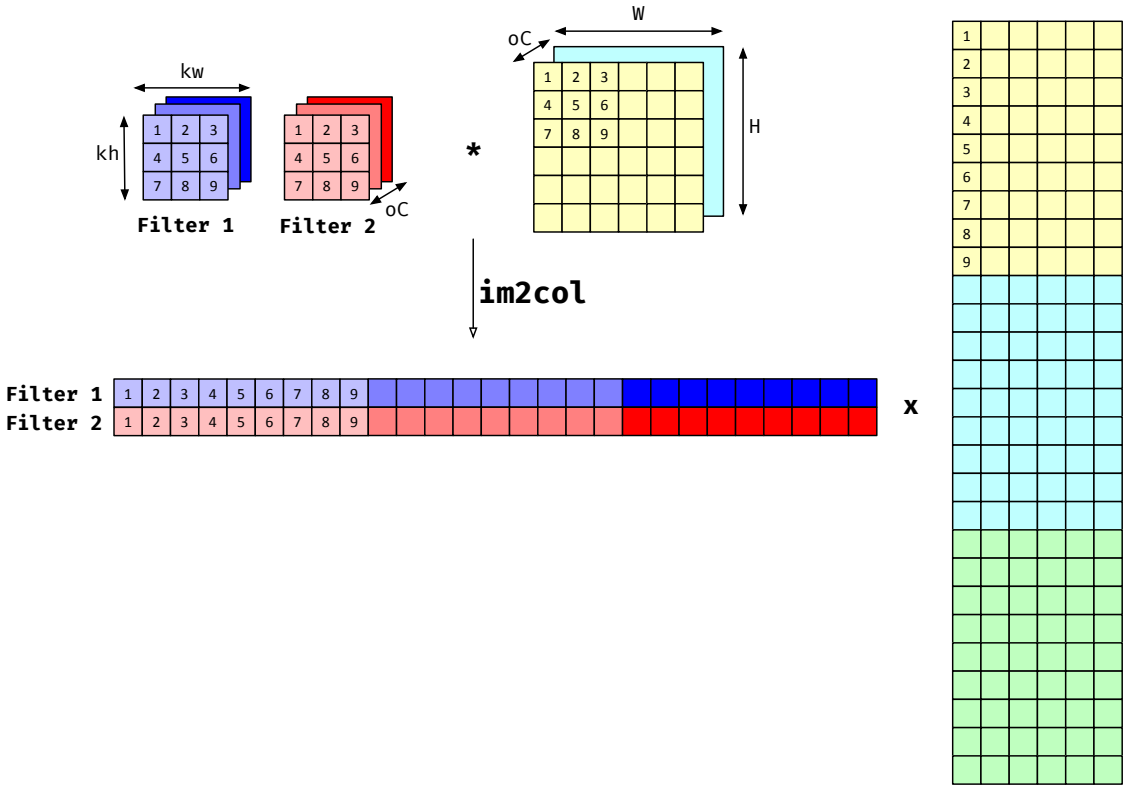
You are given two NumPy ndarrays: an image array $\in \mathbb{R}^{N,iC,H,W,D}$ and a filter tensor $\in \mathbb{R}^{oC,iC,kw,kh,kd}$, with the following specifications:

- N: Number of images within a batch.
- iC, oC: Numbers of input and output channels, respectively.
- H, W, D: Height, width, and depth of an MRI frame.
- kw, kh, kd: Sizes of a filter's height, width, and depth dimension.

A Conv3D is very similar to a Conv2D, with the only difference being that Conv3D also performs convolution (also known as correlation before the era of deep learning) on a third dimension, D. In your case, you choose to start with the simplest Conv3D settings: there's no padding or dilation, and the stride size is set to 1. To speed up the kernel execution, you should first transform the two ndarrays into two matrices. Then, you would do a matrix multiplication, a highly optimized NumPy subroutine, to generate the final result.

Pseudocode:

1. **Transform N-D data into 2-D**. The idea here is that NumPy is not great with pure Python loops. However, we can reformat data into matrices to utilize NumPy's fast matrix multiplication support. To do so, we need to extend a trick called image-to-column (im2col) so that it can handle a 3-D image. What is im2col? Roughly speaking, it divides an image into kw-by-kh sub-images, Then, im2col flattens the sub-images into columns and stacks them together. At the same time, the filters are also flattened into a matrix. By multiplying the filter matrix with the stacked columns of an image, we effectively perform a convolution:



Obviously, this approach only works for 2-D images for now. Hence, at this step, your goal is to extend im2col to work on 3-D images.

2. **Data reformatting**. After doing im2col followed by a matrix multiplication, you need to reformat the data so that it is still in the shape of (N, oC, oW, oH, oD), where oW, oH, oD are the width, height, and depth of the output tensor.

Implementing Conv3D

We ask that you write down a Conv3D implemented using Python for-loops. Obviously, this approach will not run efficiently, but it gives you a baseline you can improve on. We have extra bonus points for you if you could figure out how to replace the loops with NumPy subroutines! For all of the subproblems in this section, you can assume that you have access to the stride parameter.

7.1 Implement get_tiled_dim (5 points) First, you need to calculate the output dimensions of Conv3D. Please implement get_tiled_dim. The input arguments are **img_dim**, the size of an image dimension, and **kern_dim**, the size of a dimension of the kernel.

```
import numpy as np
def get_tiled_dim(img_dim: int, kern_dim: int) -> int:
    """

    :param img_dim: The size of an image dimension.
    :param kern_dim: The size of a kernel dimension.
    :return: The size of the output dimension.
    """
```

7.2 Conv3D in loop-form (15 points) Then, you need to implement the actual Conv3D using NumPy matrix multiplication and Python loops. Hint: use the get_tiled_dim function to calculate the output dimensions.

```
import numpy as np

def conv3d_naive(image: np.ndarray, weight: np.ndarray) -> np.ndarray:
    """

    :param image: Images with depth information of size
                  N by iC by D by H by W.
    :param weight: Filters of size oC by iC by kw by kh by kd.
    :return: The result of running Conv3D
    """
```

7.3 Conv3D using fast NumPy subroutines (Bonus, 10 points) Since Python is an interpreted language, Python loops can be very slow. Luckily, NumPy provides subroutines that allow us to manipulate large tensor data efficiently. Can you implement a Conv3D without any loops involved? Hint: you can take a look at np.lib.stride_tricks.sliding_window_view.

```
import numpy as np

def conv3d_no_loop(image: np.ndarray, weight: np.ndarray) -> np.ndarray:
    """

    :param image: Images with depth information of size
                  N by iC by D by H by W.
    :param weight: Filters of size oC by iC by kw by kh by kd.
    :return: The result of running Conv3D
    """
```

END OF PAPER