CS 231N
Convolutional Neural Networks for Visual Recognition
Spring 2019 Sample Midterm Exam

May 7, 2019

Full Name: ______________________________________

Question	Score
True/False (20 pts)	
Multiple Choice (40 pts)	
Short Answer (40 pts)	
Total (100 pts)	

Welcome to the CS231N Midterm Exam!

- The exam is 1 hour 15 minutes.
- No notes or electronic devices are allowed.

I understand and agree to uphold the Stanford Honor Code during this exam.

Signature: ____________________________ Date: ________________

Good luck!

1 True / False (20 points)

Fill in the circle next to True or False, or fill in neither. Fill it in completely like this: ●. No explanations are required.

Scoring: Correct answer is worth 2 points. To discourage guessing, incorrect answers are worth -1 points. Leaving a question blank will give 0 points.

1. Your Neural Network is not gradient checking. It could be that it's because you're using the AdaGrad update instead of Vanilla SGD.
   ○ True
   ○ False

   SOLUTION:
   False. Gradient check happens before multiplication with learning rates.

2. If the input to a ConvNet is a zero image (all zeros), then the class probabilities will come out uniform.
   ○ True
   ○ False

   SOLUTION:
   False. May have nonuniform bias terms.

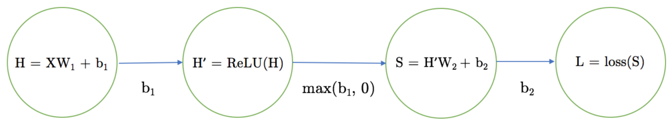
3. Turning off L2 weight regularization will likely lead to higher accuracy on the training set.
   ○ True
   ○ False

   SOLUTION:
   True. Turning off regularization increases chances of overfitting, which may give us higher accuracy on the training set.

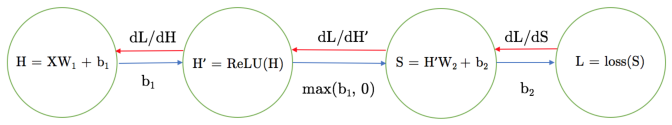
4. It's sufficient for symmetry breaking (i.e. there will be no symmetry in the weights after some updates) in a Neural Network to initialize all weights to 0, provided that the biases are random.
   ○ True
   ○ False

   SOLUTION:
   True. Non-zero asymmetric biases contribute to non-zero asymmetric upstream gradients, which when combined with asymmetric layer input gives us asymmetric gradients for the weights.See the following diagrams for a 2 layer neural network:

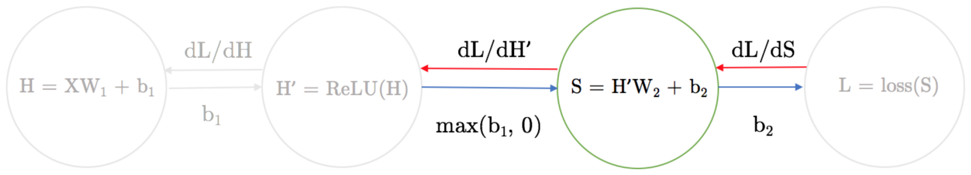
Forward Pass



Backward Pass



Backward Pass – First Iteration



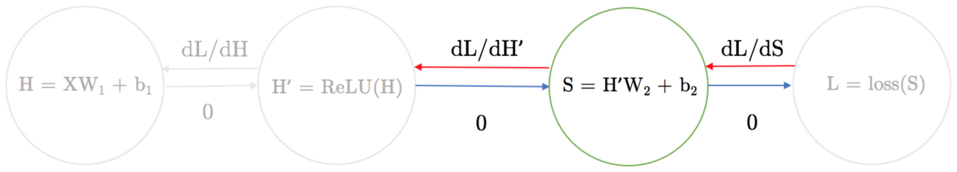
$$\frac{\partial L}{\partial W_2} = \mathrm{H'}^{\mathrm{T}}\,\frac{\partial L}{\partial S} = \max(\mathrm{b}_1, 0)^{\mathrm{T}}\,\frac{\partial L}{\partial S}$$

$$\frac{\partial L}{\partial H'} = \frac{\partial L}{\partial S}\,\mathrm{W_2}^{\mathrm{T}} = 0$$

Assuming $\frac{\partial L}{\partial S}$ is non-zero (which is very likely to be the case), and as the biases were randomly initialized, $\frac{\partial L}{\partial W_2}$ will be a non-zero, asymmetric update – breaking symmetry for W_2, which was initialized to zeros. However, $\frac{\partial L}{\partial H'}$ will be zero since W_2 was 0 in the forward pass for the first iteration. Hence, the upstream gradient for the earlier layers of the network will be 0, and none of the earlier parameters will update (not breaking symmetry for W_1).

By the second iteration however, W_2 is non-zero, and so $\frac{\partial L}{\partial H'}$ will be non-zero, and ultimately (similarly to the second layer) we'll have a non-zero asymmetric update for W_1, breaking all symmetry. In general, we will need N iterations for a N layered network to break symmetry for all zero initialized weights.

Backward Pass, Biases and Weights Zero Initialized – First Iteration



$$\frac{\partial L}{\partial W_2} = \mathrm{H'}^{\mathrm{T}}\,\frac{\partial L}{\partial S} = 0$$

$$\frac{\partial L}{\partial H'} = \frac{\partial L}{\partial S}\,\mathrm{W_2}^{\mathrm{T}} = 0$$

$$\frac{\partial L}{\partial b_2} = \sum_i \frac{\partial L}{\partial S_i}$$

The key before was that the biases were randomly initialized – what happens if they're zero initialized? Well, then $\frac{\partial L}{\partial W_2}$ becomes 0, and so we don't update our weights, unlike before. Additionally, even though we will have an update for b_2, b_1 will never update, and so H' will always be 0 and $\frac{\partial L}{\partial W_2}$ stays 0. This means W_2 stays 0, and so none of the earlier parameters of the network update either as $\frac{\partial L}{\partial H'}$ stays 0.

5. The derivative of the loss with respect to some weight in your network is -3. That means that decreasing this weight (by a tiny amount) would decrease the loss.
   ◯ True
   ◯ False

   SOLUTION:
   False. Increasing by tiny amount should decrease the loss, as the gradient is the direction of steepest ascent

6. During backpropagation, as the gradient flows backwards through a tanh non-linearity, it will always become smaller or equal in magnitude (Recall: if $z = tanh(x)$ then $\frac{\partial z}{\partial x} = 1 - z^2$).
   ◯ True
   ◯ False

   SOLUTION:
   True. $\frac{\partial z}{\partial x} \leq 1 \Rightarrow \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} \leq \frac{\partial L}{\partial z}$.

7. During backpropagation, as the gradient flows backwards through any of sigmoid/tanh/ReLU non-linearities, it cannot change sign.
   ◯ True
   ◯ False

   SOLUTION:
   True. $\frac{\partial f(x)}{\partial x} \geq 0$ for these activation functions.

8. If a neuron with the ReLU activation function ($y = relu(Wx+b)$) receives input x that is all zero, then the final (not local!) gradient on its weights and biases will also be zero (i.e. none of its parameters will update at all).
   ◯ True
   ◯ False

   SOLUTION:
   False, since $Wx + b \neq 0$.

9. The loss function will always decrease after a parameter update when performing Vanilla Gradient Descent on the full objective (no mini-batches).
   ◯ True
   ◯ False

   SOLUTION:
   False. If the learning rate is too high, this may not be the case.

10. One reason that the centered difference formula for the finite difference approximation of the gradient is preferable to the uncentered alternative is because it is better at avoiding kinks (non differentiabilities) in the objective. Recall: the centered formula is $f'(x) = (f(x+h) - f(x-h))/2h$ instead of $f'(x) = (f(x+h) - f(x))/h$.
    ◯ True
    ◯ False

SOLUTION:
False. Centered formula is good because it is more accurate (error is quadratic for C2 functions) but if the step size is doubled to 2h, it is less good at avoiding kinks.

2 Multiple Choices (40 points)

Fill in the circle next to the letter(s) of your choice (like this: ●). No explanations are required. **Choose ALL options that apply.**

Each question is worth 4 points and the answer may contain multiple options. Selecting all of the correct options will get full credits. Partial credits will be given to correct but incomplete options. Answers containing any incorrect options will get 0 points.

1. You start training your Neural Network but the total loss (cross entropy loss + regularization loss) is almost completely flat from the start. What could be the cause?

○ A: The learning rate could be too low

○ B: The regularization strength could be too high

○ C: The class distribution could be very uneven in the dataset

○ D: The weight initialization scale could be incorrectly set

SOLUTION:
A. D.
A. If the learning rate is too low, your loss will decrease very slowly.
B. High regularization terms will affect the loss value, but it shouldn't cause loss-curve issues. (In particular, with random initialization of weights, you should see a large initial loss that drops significantly right after the first training step.)
C. Uneven class distribution could cause overfitting, but shouldn't cause optimization-related problems.
D. If your weights are incorrectly set, you could run into issues with the gradients, such as saturation, that prevent your model from learning effectively

2. A VGGNet only uses a sequence of 3x3 CONV with stride 1 pad 1 and 2x2 POOL stride 2 pad 0 layers. It eventually transitions to Fully Connected layers and the classifier. There are 5 POOL layers in total. On ImageNet, the VGGNet takes 224x224 images as input. If we take this VGGNet **trained** on ImageNet, and try to run it at test time with a 32x32 input image (e.g. CIFAR-10 image):

○ A: The code would crash on the very first CONV layer because 3x3 filters with stride 1 pad 1 wouldn't "fit" across 32x32 input

○ B: The amount of memory needed to store the forward activations in the first CONV layer would be reduced by a factor of 7 (since 224/32 = 7)

○ C: The network would run fine until the very first Fully Connected layer, where it would crash

○ D: The network would run forward just fine but its predictions would, of course, be ImageNet class predictions

SOLUTION:
C.
A. Incorrect. (Sidenote: stride 1 generally has no problems with 'fitting'.)
B. Should be 7^2.
C. The Fully Connected layer's dimensions depends on the input dimensions.
D. See C.

3. A max pooling layer in a ConvNet:

○ A: Is approximately as fast to compute in both forward and backward pass as a CONV layer (with the same filter size and strides).

○ B: Is similar to batch normalization in that it will keep all of your neuron activities in a similar range.

○ C: Could contribute to difficulties during gradient checking (higher error than usual, as in the SVM).

○ D: Could contribute to the vanishing gradient problem (recall: this is a problem where by the end of a backward pass the gradients are very small)

SOLUTION:
C, D.
A. Maxpool is faster to compute.
B. Maxpool favors extreme activities instead of normalizing and constraining them.
C. It has the same problems as ReLU; the gradient is a step function with a jump/discontinuity.
D. Each maxpool layer will have $\frac{k^2-1}{k^2}$ gradients set to zero, where k is filter size, due to gradient only flowing through the activation with highest value. This leads to sparsening/thinning of gradients.

4. Which of the following are valid activation functions (elementwise non-linearities) you could use in a neural network? (That is, which functions could be effective when training a neural net in practice?)

○ A: $f(x) = \max(0.25x, 0.75x)$

○ B: $f(x) = \min(0, x)$

○ C: $f(x) = 0.7x$

○ D: $f(x) = \begin{cases} 1 & \text{if } x > 0.5 \\ -1 & \text{else} \end{cases}$

SOLUTION:
A, B.
A and B: correct since they are both non-linear functions that could plausibly be used to train a network.
C: just a linear function, which ultimately reduces a deep neural network to a series of linear operations, making it no more powerful than a linear model.
D: piecewise constant that has zero gradients.

3 Short Answer (40 points)

Answer each question in provided space.

3.1 Zero-One Loss

The Zero-One Loss is a simple classification loss function that intuitively counts how many mistakes the model makes. In this problem we will examine the Zero-One Loss in the context of binary classification, where the loss for a single example x_i with label $y_i \in \{0, 1\}$ is given by

$$L(x_i) = \mathbb{1}[s_{y_i} - s_{(1-y_i)} < 0]$$

where s_j represents the model's predicted score for the j-th class and $\mathbb{1}$ is an indicator function that is 1 if the condition inside the function is true, and 0 otherwise.

1. Suppose we are using zero-one loss for a network performing binary classification across N training examples with balanced classes. Roughly what value would we expect the *total loss* (i.e. loss summing over all examples) to be at the start of training (assuming we initialize our biases to zero and our weight matrices to small random numbers centered at 0)?

   SOLUTION:
   If the network is initialized randomly, we expect half the time we would predict class 0, and the other times we would predict class 1. This leads to an expected starting loss of 0.5 per example, and thus $N/2$ over the whole dataset.

2. A friend of yours suggests that you can train a neural network classifier by performing SGD over the zero-one loss function. Is this a good idea? Briefly explain why or why not.

   SOLUTION:
   This is a bad idea – if we visualize the zero-one loss, it is essentially a step function that jumps from a constant value of zero to a constant value of one, when the score for the incorrect class becomes higher than that of the correct class. This means that the gradients (even if we ignore the fact they are not well-defined when $s_{y_i} == s_{1-y_i}$) are zero everywhere leading to no training.

3. Suppose now you are training a Softmax classifier and have been initializing your weight matrices to random numbers between -0.001 and +0.001, but your friend also suggests that it is better to scale down the range to something much closer to zero; in particular, he recommends randomly choosing values between -1e-7 to +1e-7. Is this a good idea? Why or why not?

   SOLUTION:
   This is another bad idea – if your weight matrices are too small, then this easily leads to a vanishing

gradient problem, where as you pass gradients down from one hidden layer to the previous one, factoring in the W matrices makes the signal quickly go towards zero.

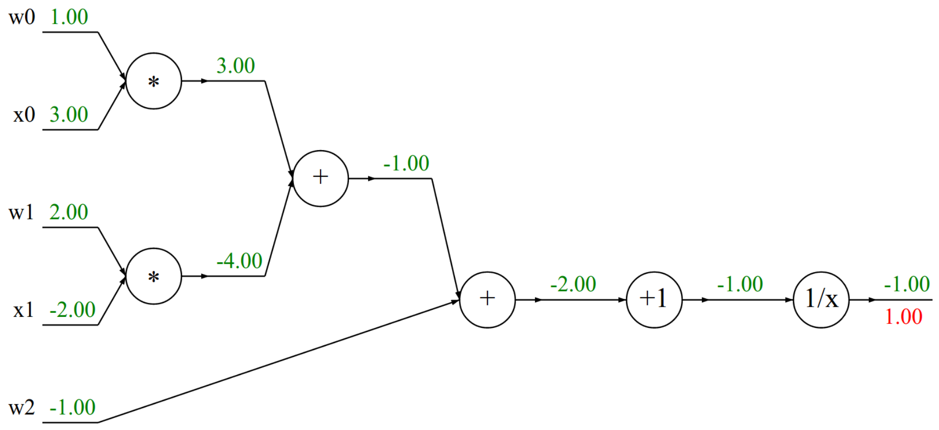
4. Consider optimizing the loss function $f(x) = \frac{1}{2}x^4$ where x is a scalar. If an optimization process starts with $x = 1$, and uses gradient descent with learning rate α to update x where the value of x at time t is x_t, will the optimization converge to the global minimum of x (i.e. $\lim_{t\to\infty} x_t = 0$)? If your answer depends on α, specify the range of α that leads to this behavior.

   SOLUTION:
   $\alpha < 1$: in order to converge, the requirement on α is $\alpha \cdot |2x^3| < 2|x|$. Other proofs are also possible.

   - $0 < \alpha < 1$: converge
   - $\alpha = 1$: oscillating between 1 and -1
   - $\alpha > 1$: diverge

3.2 Backpropagation

Fill in the missing gradients underneath the forward pass activations in each circuit diagram. The gradient of the output with respect to the loss is one (1.00) for every circut, and has already been filled in.



SOLUTION:
top-bottom, left to right: [-3,-1,2,-2,-1] [-1,-1] [-1] [-1] [-1]

3.3 Convolutional Architectures

Consider the convolutional network defined by the layers in the left column below. Fill in the size of the activation volumes at each layer, and the number of parameters at each layer. You can write your answer as a multiplication (e.g. 128x128x3).

- CONV5-N denotes a convolutional layer with N filters, each of size 5x5xD, where D is the depth of the activation volume at the previous layer. Padding is 2, and stride is 1.
- POOL2 denotes a 2x2 max-pooling layer with stride 2 (pad 0)
- FC-N denotes a fully-connected layer with N output neurons.

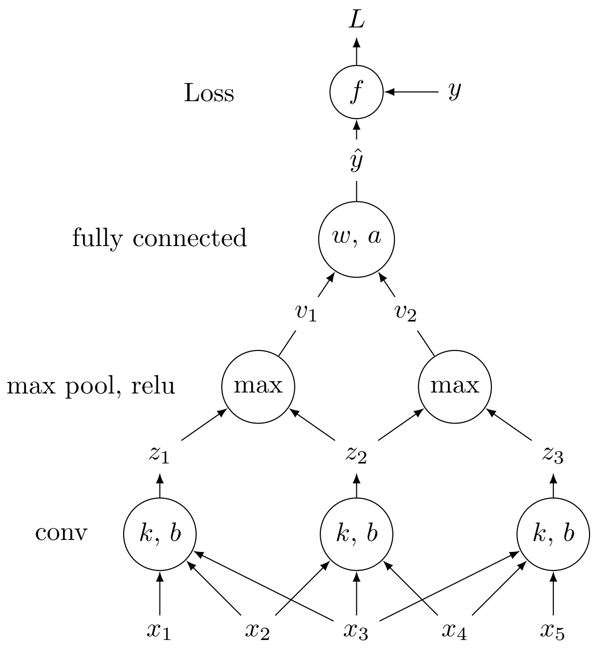
Layer	Activation Volume Dimensions (memory)	Number of parameters
INPUT	32x32x1	0
CONV5-10		
POOL2		
CONV5-10		
POOL2		
FC-10		

SOLUTION:

Layer	Activation Volume Dimensions (memory)	Number of parameters
INPUT	32x32x1	0
CONV5-10	32x32x10	(5x5x1+1)x10
POOL2	16x16x10	0
CONV5-10	16x16x10	(5x5x10+1)x10
POOL2	8x8x10	0
FC-10	10	(8x8x10+1)x10

3.4 Simple ConvNet (12 points)

Consider the following 1-dimensional ConvNet, where all variables are scalars:



$$L = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + a$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \max\{z_1, z_2, 0\} \\ \max\{z_2, z_3, 0\} \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

1. **(1 point)** List the parameters in this network.

   SOLUTION:

   $k_1, k_2, k_3, b, w_1, w_2, a$

2. **(3 points)** Determine the following

$$\frac{\partial L}{\partial w_1} =$$

$$\frac{\partial L}{\partial w_2} =$$

$$\frac{\partial L}{\partial a} =$$

   SOLUTION:

$$\frac{\partial L}{\partial w_1} = (\hat{y} - y)v_1$$

$$\frac{\partial L}{\partial w_2} = (\hat{y} - y)v_2$$

$$\frac{\partial L}{\partial a} = (\hat{y} - y)$$

3. **(3 points)** Given the gradients of the loss L with respect to the second layer activations v, derive the gradient of the loss with respect to the first layer activations z. More precisely, given

$$\frac{\partial L}{\partial v_1} = \delta_1 \qquad \frac{\partial L}{\partial v_2} = \delta_2$$

Determine the following

$$\frac{\partial L}{\partial z_1} =$$

$$\frac{\partial L}{\partial z_2} =$$

$$\frac{\partial L}{\partial z_3} =$$

SOLUTION:

$$\frac{\partial L}{\partial z_1} = \delta_1 \mathbb{1}_{z_1 > z_2 \& z_1 > 0}$$

$$\frac{\partial L}{\partial z_2} = \delta_1 \mathbb{1}_{z_2 > z_1 \& z_2 > 0} + \delta_2 \mathbb{1}_{z_2 > z_3 \& z_2 > 0}$$

$$\frac{\partial L}{\partial z_3} = \delta_2 \mathbb{1}_{z_3 > z_2 \& z_3 > 0}$$

4. **(3 points)** Given the gradients of the loss L with respect to the first layer activations z, derive the gradient of the loss with respect to the convolution filter k. More precisely, given

$$\frac{\partial L}{\partial z_1} = \delta_1 \qquad \frac{\partial L}{\partial z_2} = \delta_2 \qquad \frac{\partial L}{\partial z_3} = \delta_3$$

Determine the following

$$\frac{\partial L}{\partial k_1} =$$

$$\frac{\partial L}{\partial k_2} =$$

$$\frac{\partial L}{\partial k_3} =$$

$$\frac{\partial L}{\partial b} =$$

SOLUTION:

$$\frac{\partial L}{\partial k_1} = \delta_1 x_1 + \delta_2 x_2 + \delta_3 x_3 = \sum_{i=1}^{3} \delta_i x_i$$

$$\frac{\partial L}{\partial k_2} = \delta_1 x_2 + \delta_2 x_3 + \delta_3 x_4 = \sum_{i=1}^{3} \delta_i x_{i+1}$$

$$\frac{\partial L}{\partial k_3} = \delta_1 x_3 + \delta_2 x_4 + \delta_3 x_5 = \sum_{i=1}^{3} \delta_i x_{i+2}$$

$$\frac{\partial L}{\partial b} = \delta_1 + \delta_2 + \delta_3 = \sum_{i=1}^{3} \delta_i$$

5. **(2 points)** Suppose we have a general 1D convolution layer

$$\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} k_1 & \cdots & k_d & & & \\ & k_1 & \cdots & k_d & & \\ & & & \ddots & & \\ & & & k_1 & \cdots & k_d \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}$$

And we know that

$$\frac{\partial L}{\partial z_i} = \delta_i$$

Determine

$$\frac{\partial L}{\partial k_j} =$$

$$\frac{\partial L}{\partial b} =$$

SOLUTION:

$$\frac{\partial L}{\partial k_j} = \sum_{i=1}^{m} \delta_i x_{i+j-1}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} \delta_i$$