Crowd-sourced Solutions for CS231n Sample Midterm
Spring 2017

Disclaimer: These solutions were curated by students in CS231n. It is possible there are mistakes! If you see a mistake, please fix it. If you see something that might be a mistake, please add a comment on the side.

Motivation: Why work together to come up with solutions? Trying to learn course material from practice exams without solutions is like trying to train a CNN using forward passes but no backward passes. By attempting the practice midterm on our own, then by checking our best guesses with the correct answers, we can learn from our mistakes. Thanks everyone for helping put this together!

1 MULTIPLE CHOICE (20 points)

1. You start training your Neural Network but the loss is almost completely flat. 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

(TA Verified)

TA Explanation: The answer is A and D. The regularization strength being too high is not an issue because your loss will still go down. The *regularization* loss will be flat because the weights will be pushed to zero, but the *data* loss will still be changing and will not necessarily be flat. Regarding why D is a correct answer, there is a slide where the loss is almost totally flat and it says “weight initialization a prime suspect”

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. If we tried to run the VGGNet on a 32x32 input (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

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.
(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)

(TA Verified)

TA Explanation: The answer is NOT (d) because the vanishing gradient issue is not necessarily affected by the max pooling layer. The max pooling layer simply routes the maximums and the gradients there will be 1. So it doesn't contribute to the vanishing gradient issue. If there IS a vanishing gradient issue, it would be caused by the upstream gradients and the max pooling layer wouldn't affect that. TA verified the answer is only (c). --3:30 Office hours)
Additional TA Explanation: Nonlinearity of max pool makes (c) the answer

2 TRUE / FALSE (20 points)

1. True or **False**: Your Neural Network is not gradient checking. It could be that it's because you're using the AdaGrad update instead of Vanilla SGD.
(TA Verified)
"Don't blame that update rule bruh </3"

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

TA Explanation: Consider the case where the biases are nonzero.

3. **True** or False: Turning off L2 weight regularization will likely lead to higher accuracy on the training set.
(TA Verified)

4. **True** or False: It's sufficient for symmetry breaking in a Neural Network to initialize all weights to 0, provided that the biases are random
(TA Verified)

TA Explanation: If we have a 3 layer network with an input x, and we have some affine layer W1*x + b1, with a relu activation, and then we have another affine_relu layer, and so on, if we initialize all the weights to zero but the biases to non-zero values, then this final computation z3 = relu(W3*z2 + b3) will be b3. Because we initialized all the weights W3 to be zero, the W3*z2 term is zero. So our output will be b3 and we'll get some sort of loss from that (softmax for example). This z3 output will be non-zero. Thus the loss with respect to z3 will also be non-zero. This local gradient will be non-zero but the above layers dL/dz2 will be zero because W3 kills it (since its zero). But the gradient dL/dW3 will be NON ZERO because z2 has a bias term. Which means that when we do the gradient update, the gradient update for the weight will be non zero even though it was initialized to zero! Which is good. However on this first pass, the gradient with respect to W2 will be zero :(. However because the dL/dW3 was non zero, on the NEXT iteration the dL/dW2 term will be non-zero and so on). Also, **breaking symmetry** = "we don't want all the neurons to do the same thing because then we're only really learning like one feature, really. But if you initialize them randomly, each neuron will learn to specialize in different things. The reason we initialize them randomly is so the different neurons can learn different things and the different layers can specialize in different things"

5. True or **False**: 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.

TA Explanation: gradient means what will happen to the loss with a tiny increase of the weight. Thus, if the gradient is -3, a small increase of epsilon in the weight will increase the loss by -3*epsilon, or, in other words, decrease the loss by 3*epsilon. In this problem, decreasing the weight will increase the loss.

6. **True** or False: Regularizing the biases in a neural network is not as important because they do not interact multiplicatively with the inputs.
(TA Verified)

TA Explanation: We add biases to take into account the priors of the input data to that layer. We don't want to impose any penalty on making this large; if an input always has a lot more red, we want the bias term to take care of that. (Edited from False to True based on https://piazza.com/class/j0vi72697xc49k?cid=1473)

7. **True** or False: 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 ∂z/∂x = 1 − z^2).

TA Explanation: Recall that tanh(x) only has a range between -1 and 1, so 1-z^2 will only have the range between zero and 1.

8. **True** or False: During backpropagation, as the gradient flows backwards through any of sigmoid/tanh/ReLU non-linearities, it cannot change sign.
(TA Verified)

TA Explanation: ReLU, sigmoid, and tanh are nondecreasing functions; therefore, the gradient is always nonnegative

9. True or False: If a neuron with the ReLU activation function receives input 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).

This question is really ambiguous because they didn't specify whether ReLU is implemented with checking if element > 0 or element >= 0. If you were doing elem > 0, then the answer is **True.** if you were implementing your relu layer by checking if elem >= 0, then the answer is **False.** Let's say our ReLU layer is implemented with checking if an element is strictly > 0. If we receive activations like this [0 0 0; 0 0 0; 0 0 0] then our gradients for ReLu would also be set to zero and then the final gradient will also be zero. On the other hand, if our Relu layer is implemented with checking if an element is greater than *or equal to zero* >= 0, then the gradients for an input of all zeros will be set to 1's and then the final gradients will NOT be zero. Whew! Nice!

10. True or **False**: The loss function will always decrease after a parameter update when performing Vanilla Gradient Descent on the full objective (no mini-batches).
(TA Verified)

TA Explanation: It's also dependent on the learning rate because it could overshoot.

11. True or **False**: 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 in the objective. Recall: the centered formula is f 0 (x) = (f(x + h) − f(x − h))/2h instead of f 0 (x) = (f(x + h) − f(x))/h.
(TA Verified)

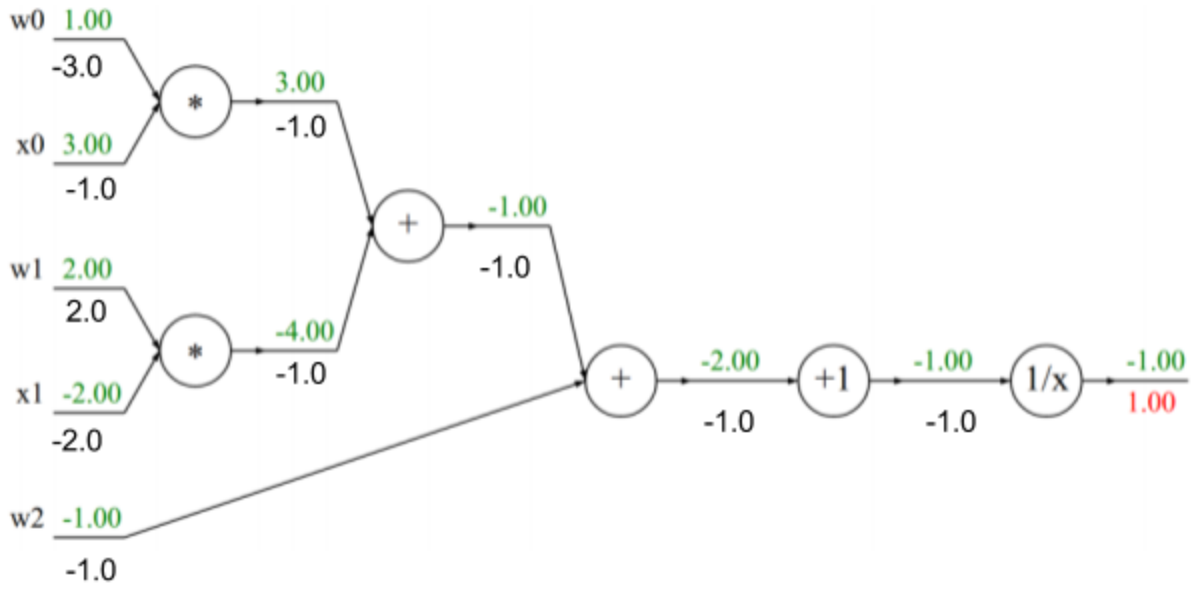
TA Explanation: Piazza.

3 SHORT ANSWER (60 points)

3.1 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 circuit, and has already been filled in.

[Double-click the diagram below to edit it.]



3.2 Convolutional Architectures

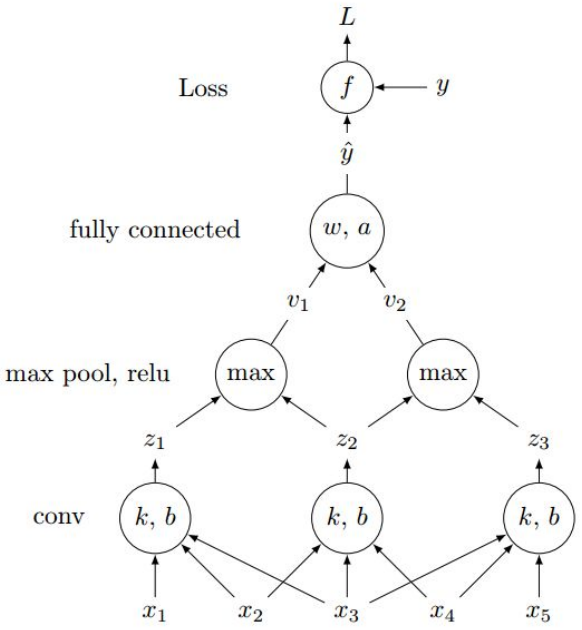
Consider the convolutional network defined by the layers in the left column below. Fill in the size of the activation volumes at each layer, and the number of parameters at each layer. You can write your answer as a multiplication (e.g. 128x128x3).

- CONV5-N denotes a convolutional layer with N neurons, each having 5x5xD filters, 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 neurons.

Layer	**Activation Volume Dimensions (memory)**	**Number of parameters**
INPUT	32x32x1	0

CONV5-10	32x32x10	10*(5x5x1 + 1)
POOL2	16x16x10	0
CONV5-10	16x16x10	10(5x5x10 + 1)
POOL2	8x8x10	0
FC-10	10x1	10(8x8x10 + 1)

3.3 Simple ConvNet (12 points)



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

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

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

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

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

a, w1, w2, b, k1, k2, k3 (Piazza: w, a, k, b is also fine)

(b) (3 points) Determine the following

∂L / ∂w1 = $(\hat{y} - y) * v\_1$

∂L / ∂w2 = $(\hat{y} - y) * v\_2$

∂L / ∂a = $(\hat{y} - y)$

(c) (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

$$\partial L / \partial v1 = \delta 1$$
$$\partial L / \partial v2 = \delta 2$$

Determine the following:

$\partial L / \partial z1 = I(z_1 \geq z_2) * I(z_1 \geq 0) * \delta 1$
$\partial L / \partial z2 = I(z_2 > z_1) * I(z_2 \geq 0) * \delta 1 + I(z_2 \geq z_3) * I(z_2 \geq 0) * \delta 2$
$\partial L / \partial z3 = I(z_3 > z_2) * I(z_3 \geq 0) * \delta 2$
Where $I(\,)$ is the indicator function

(d) (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

$$\partial L \, \partial z1 = \delta 1$$
$$\partial L \, \partial z2 = \delta 2$$
$$\partial L \, \partial z3 = \delta 3$$

Determine the following:

$\partial L / \partial k1 = \delta 1 * x_1 + \delta 2 * x_2 + \delta 3 * x_3$
$\partial L / \partial k2 = \delta 1 * x_2 + \delta 2 * x_3 + \delta 3 * x_4$
$\partial L / \partial k3 = \delta 1 * x_3 + \delta 2 * x_4 + \delta 3 * x_5$
$\partial L / \partial b = \delta 1 + \delta 2 + \delta 3$

(e) (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 $\partial L / \partial zi = \delta i$.

Determine the following:

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

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

RNNs and LSTMs are fair game for the midterm! (anything that was on lecture slides)

Difference between RNN hidden state and CNN hidden layer:
At test time, the weights of a CNN hidden layer don't change; for one input, there will always be one output
However, the output of an RNN DOES change for a given input, since the RNN also depends on the order of inputs (i.e. what the last n inputs were).