CS231N A2 - PDF

Pei - Chen Wu

TOTAL POINTS

9 / 16

QUESTION 1

1 Fully Connected Net - Inline Q1 1 / 1

✓ **- 0 pts Correct**

- 0.5 pts Missing ReLU

- 0.5 pts Missing Sigmoid

- 0.5 pts Incorrect reason for Leaky ReLU

- 0.5 pts Incorrect reason for Sigmoid

- 0.5 pts Missing reasons (what types of input would lead to this behavior)

- 0.5 pts Incorrect reason for ReLU

- 0.5 pts Page not selected

QUESTION 2

2 Fully Connected Net - Inline Q2 1 / 1

✓ **- 0 pts Correct**

- 0.5 pts Improper explanation

- 1 pts Incorrect

- 0.5 pts Page not selected

QUESTION 3

3 Fully Connected Net - Inline Q3 2 / 2

✓ **- 0 pts Correct**

- 1 pts Missing analysis for Adam

- 1 pts Incorrect explanation for AdaGrad

- 2 pts Incorrect

- 1 pts Incorrect explanation for Adam

- 1 pts Page not selected

QUESTION 4

4 Batch Norm - Inline Q1 1 / 2

- 0 pts Correct

- 1 pts not mentioning insensitivity

✓ **- 1 pts no justification**

- 1 pts error in justification

- 1 pts insufficient justification

- 1 pts not tagging the answer correctly

QUESTION 5

5 Batch Norm - Inline Q2 1 / 2

- 0 pts Correct

✓ **- 1 pts Incorrect/insufficient justification**

- 2 pts Incorrect

QUESTION 6

6 Batch Norm - Inline Q3 2 / 3

- 0 pts Correct

✓ **- 1 pts select (1) or (4) for BN/LN**

- 1 pts not stating that (2) is analogous to LN

- 1 pts not stating that (3) is analogous to BN

- 3 pts didn't answer

- 1.5 pts didn't select correct page

QUESTION 7

7 Batch Norm - Inline Q4 1 / 2

- 0 pts Correct

✓ **- 1 pts didn't choose (2)**

- 1 pts chose (1)

- 1 pts didn't provide valid justification

- 1 pts didn't choose correct page

- 2 pts didn't provide answer

QUESTION 8

8 Dropout - Inline Q1 0 / 1

- 0 pts Correct

✓ **- 1 pts Incorrect/Incomplete answer - predictions at test-time will be incorrect.**

QUESTION 9

9 Dropout - Inline Q2 0 / 1

- 0 pts Correct

✓ **- 1 pts Incorrect/insufficient explanation**

QUESTION 10

10 Dropout - Inline Q3 0 / 1

- **0 pts** Correct
- **0.5 pts** Didn't mention loss of capacity while decreasing hidden units
- **0.5 pts** Partially correct answer

✓ **- 1 pts** **Incorrect/Insufficient answer**

```
Testing relu_backward function:
dx error:  3.2756349136310288e-12
```

5.1 Inline Question 1:

We've only asked you to implement ReLU, but there are a number of different activation functions that one could use in neural networks, each with its pros and cons. In particular, an issue commonly seen with activation functions is getting zero (or close to zero) gradient flow during backpropagation. Which of the following activation functions have this problem? If you consider these functions in the one dimensional case, what types of input would lead to this behaviour? 1. Sigmoid 2. ReLU 3. Leaky ReLU

5.2 Answer:

1. Sigmoid: Saturated neurons kill the gradients grad = 0 in saturated area.
2. ReLU: grad = 0 when the input value is negative.
3. Leaky ReLU: No grad = 0 situation.

6 "Sandwich" layers

There are some common patterns of layers that are frequently used in neural nets. For example, affine layers are frequently followed by a ReLU nonlinearity. To make these common patterns easy, we define several convenience layers in the file `cs231n/layer_utils.py`.

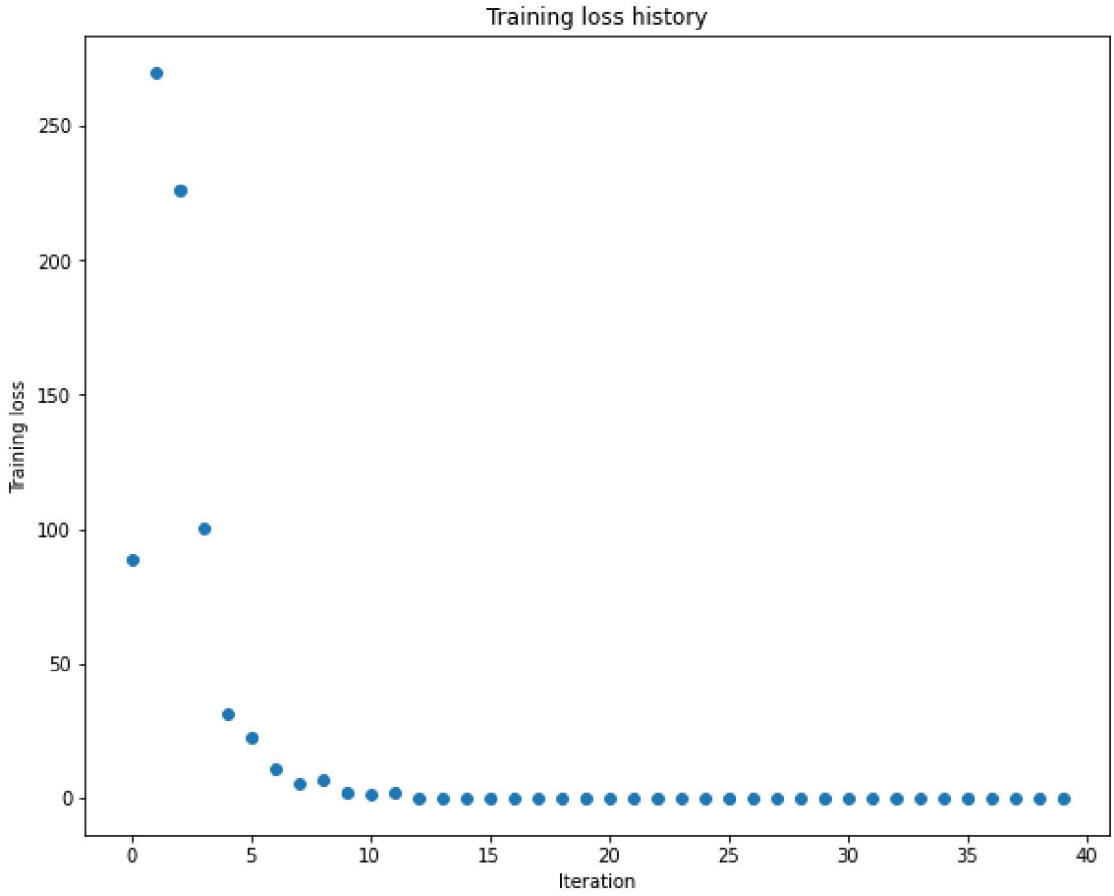
For now take a look at the `affine_relu_forward` and `affine_relu_backward` functions, and run the following to numerically gradient check the backward pass:

[20]:
```
from cs231n.layer_utils import affine_relu_forward, affine_relu_backward
np.random.seed(231)
x = np.random.randn(2, 3, 4)
w = np.random.randn(12, 10)
b = np.random.randn(10)
dout = np.random.randn(2, 10)

out, cache = affine_relu_forward(x, w, b)
dx, dw, db = affine_relu_backward(dout, cache)

dx_num = eval_numerical_gradient_array(lambda x: affine_relu_forward(x, w,␣
 ↪b)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: affine_relu_forward(x, w,␣
 ↪b)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: affine_relu_forward(x, w,␣
 ↪b)[0], b, dout)

# Relative error should be around e-10 or less
print('Testing affine_relu_forward and affine_relu_backward:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

1 Fully Connected Net - Inline Q1 1 / 1

✓ - 0 pts **Correct**

- **0.5 pts** Missing ReLU
- **0.5 pts** Missing Sigmoid
- **0.5 pts** Incorrect reason for Leaky ReLU
- **0.5 pts** Incorrect reason for Sigmoid
- **0.5 pts** Missing reasons (what types of input would lead to this behavior)
- **0.5 pts** Incorrect reason for ReLU
- **0.5 pts** Page not selected



10.2 Inline Question 2:

Did you notice anything about the comparative difficulty of training the three-layer net vs training the five layer net? In particular, based on your experience, which network seemed more sensitive to the initialization scale? Why do you think that is the case?

10.3 Answer:

It seems five layer net will need larger weight_scale to acchieve 100% accuracy on training set. Since deeper layer net will cause weights to vanish. Five layer net seems to be more sensitive to teh initialization scale.

11 Update rules

So far we have used vanilla stochastic gradient descent (SGD) as our update rule. More sophisticated update rules can make it easier to train deep networks. We will implement a few of the most commonly used update rules and compare them to vanilla SGD.

2 Fully Connected Net - Inline Q2 1 / 1

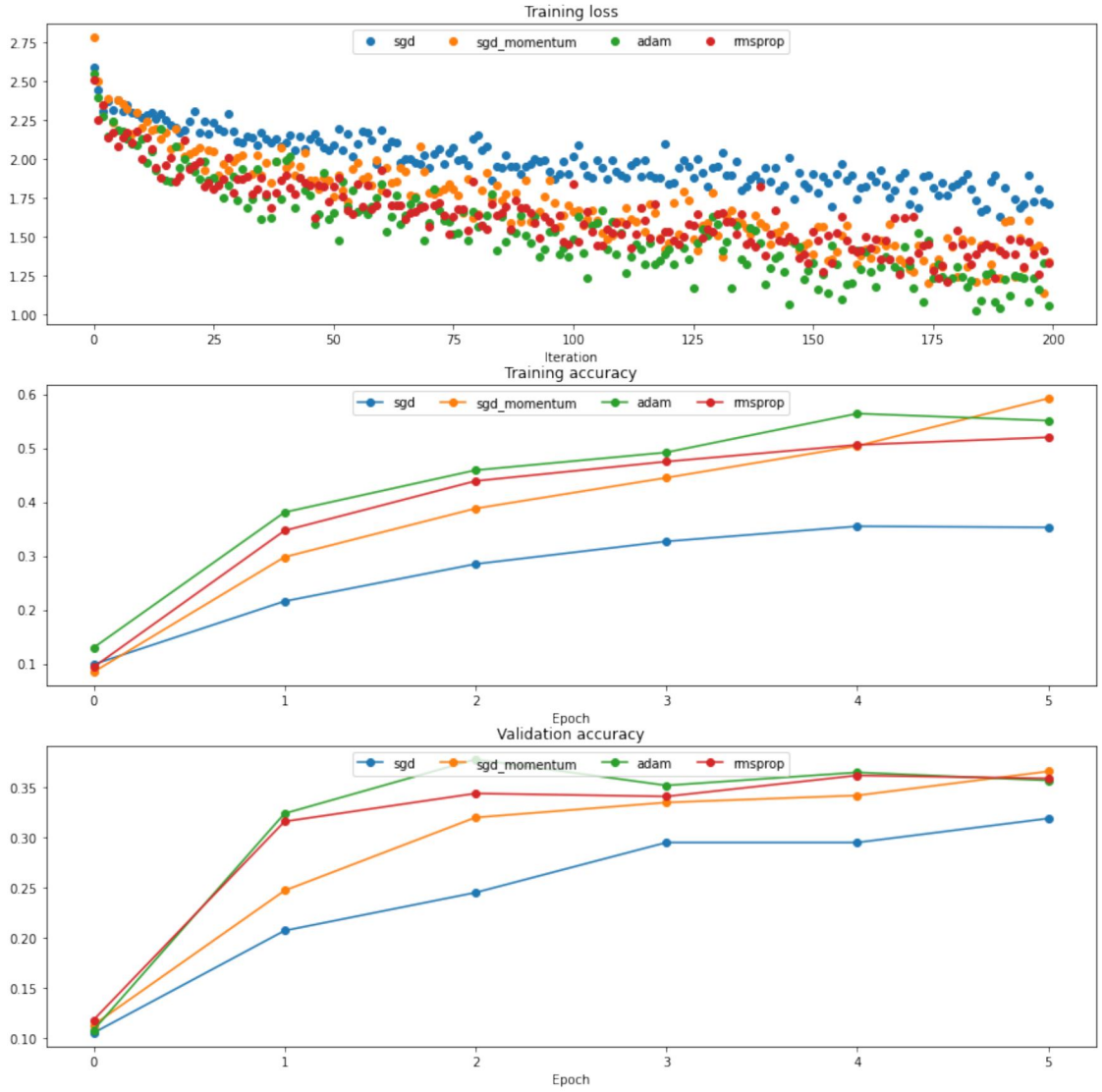
✓ **- 0 pts** **Correct**

- 0.5 pts Improper explanation

- 1 pts Incorrect

- 0.5 pts Page not selected

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:40:
MatplotlibDeprecationWarning: Adding an axes using the same arguments as a
previous axes currently reuses the earlier instance.  In a future version, a new
instance will always be created and returned.  Meanwhile, this warning can be
suppressed, and the future behavior ensured, by passing a unique label to each
axes instance.
```



13.1 Inline Question 3:

AdaGrad, like Adam, is a per-parameter optimization method that uses the following update rule:

```
cache += dw**2
w += - learning_rate * dw / (np.sqrt(cache) + eps)
```

John notices that when he was training a network with AdaGrad that the updates became very small, and that his network was learning slowly. Using your knowledge of the AdaGrad update rule, why do you think the updates would become very small? Would Adam have the same issue?

13.2 Answer:

AdaGrad will be slower than Adam, since it adds dw**2, this will cause cache value to become large and the update in the second equation will be slow. Adam doesn't have this issue, which update the weights based on bias correction and RMS prop. This controls the weight update pretty well.

14 Train a good model!

Train the best fully-connected model that you can on CIFAR-10, storing your best model in the `best_model` variable. We require you to get at least 50% accuracy on the validation set using a fully-connected net.

If you are careful it should be possible to get accuracies above 55%, but we don't require it for this part and won't assign extra credit for doing so. Later in the assignment we will ask you to train the best convolutional network that you can on CIFAR-10, and we would prefer that you spend your effort working on convolutional nets rather than fully-connected nets.

You might find it useful to complete the `BatchNormalization.ipynb` and `Dropout.ipynb` notebooks before completing this part, since those techniques can help you train powerful models.

[80]:
```
best_model = None
################################################################################
# TODO: Train the best FullyConnectedNet that you can on CIFAR-10. You might   ␣
 ↪#
# find batch/layer normalization and dropout useful. Store your best model in ␣
 ↪#
# the best_model variable.                                                     ␣
 ↪#
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
#results = {}
best_val = -1
hidden_dims = [100, 100, 100, 100]
learning_rate_decay=0.95


weight_scales = [1e-2, 1e-3]
learning_rates = [1e-3, 1e-4]
regularization_strengths = [1e-4]

for learning_rate in learning_rates:
  for reg in regularization_strengths:
    for wc in weight_scales:
```

3 Fully Connected Net - Inline Q3 2 / 2

✓ **- 0 pts** **Correct**

- 1 pts Missing analysis for Adam

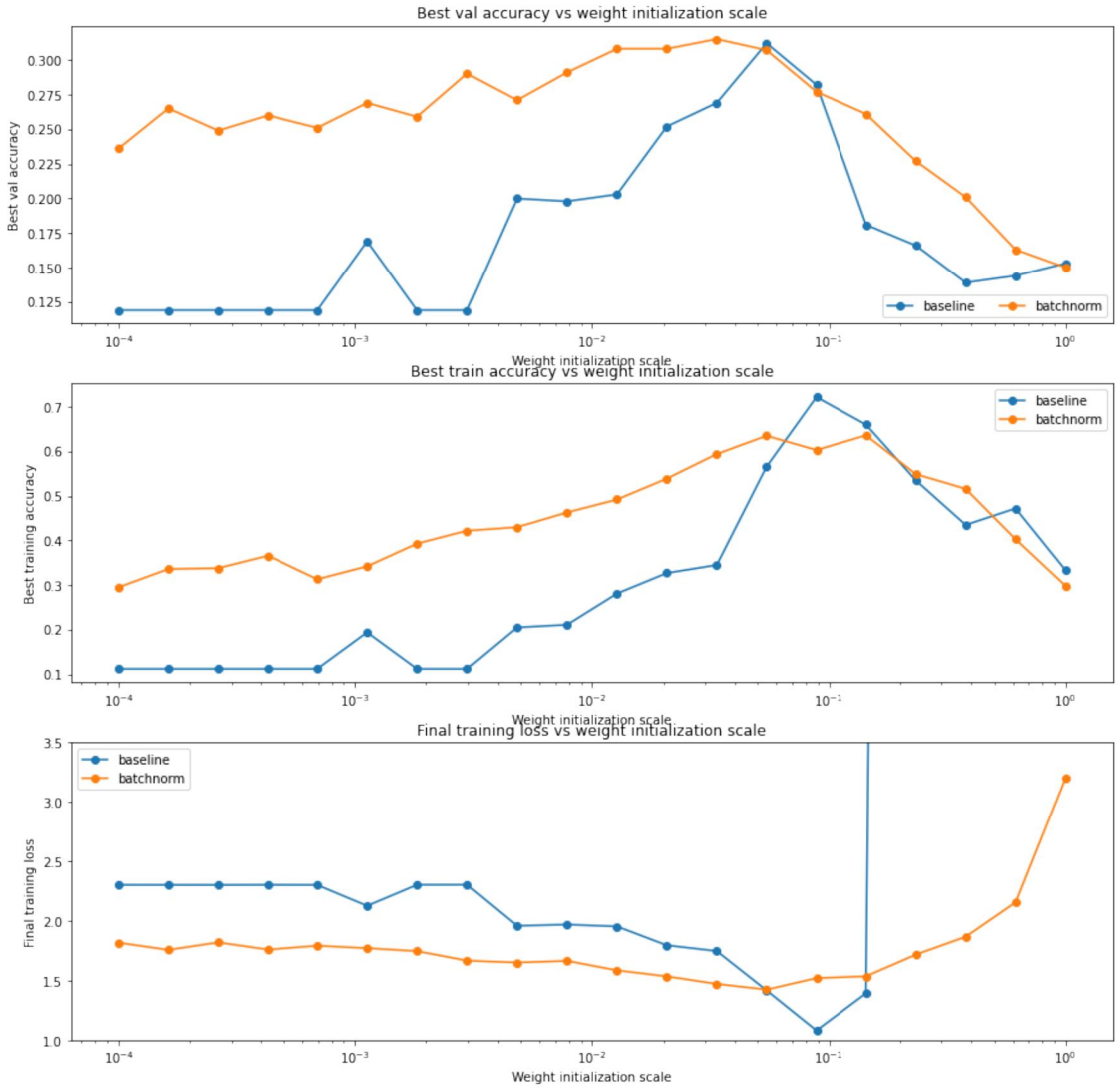
- 1 pts Incorrect explanation for AdaGrad

- 2 pts Incorrect

- 1 pts Incorrect explanation for Adam

- 1 pts Page not selected

```
plt.legend()
plt.gca().set_ylim(1.0, 3.5)

plt.gcf().set_size_inches(15, 15)
plt.show()
```



3.1 Inline Question 1:

Describe the results of this experiment. How does the scale of weight initialization affect models with/without batch normalization differently, and why?

3.2 Answer:

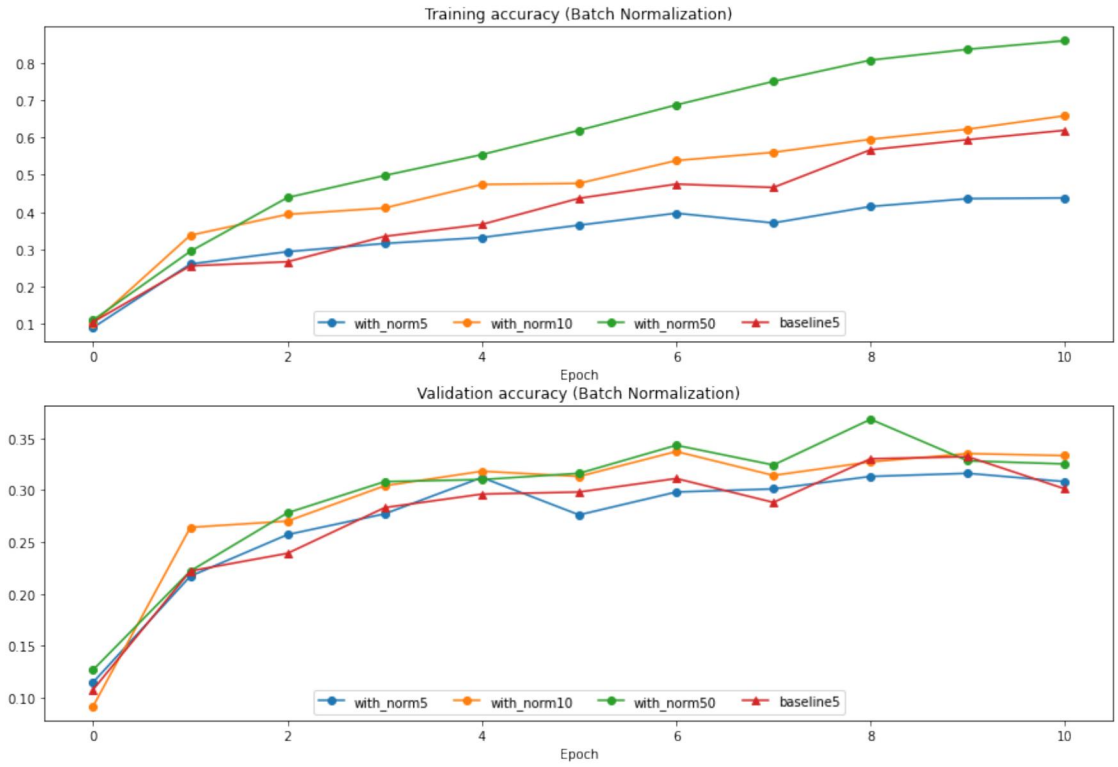
After adding batchnorm layer, weight initialization effect becomes less significant.

4 Batch Norm - Inline Q1 1 / 2

- **0 pts** Correct
- **1 pts** not mentioning insensitivity

✓ **- 1 pts no justification**

- **1 pts** error in justification
- **1 pts** insufficient justification
- **1 pts** not tagging the answer correctly



4.1 Inline Question 2:

Describe the results of this experiment. What does this imply about the relationship between batch normalization and batch size? Why is this relationship observed?

4.2 Answer:

Training loss converges faster with batch norm, training accuracy becomes high with higher batch norm. However, there is no significant difference between batch norm and baseline for test accuracy. with_norm5 smaller norm might be worse than no norm.

5 Layer Normalization

Batch normalization has proved to be effective in making networks easier to train, but the dependency on batch size makes it less useful in complex networks which have a cap on the input batch size due to hardware limitations.

Several alternatives to batch normalization have been proposed to mitigate this problem; one such technique is Layer Normalization [2]. Instead of normalizing over the batch, we normalize over the features. In other words, when using Layer Normalization, each feature vector corresponding to a single datapoint is normalized based on the sum of all terms within that feature vector.

[2] [Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization." stat 1050 (2016): 21.](https://arxiv.org/pdf/1607.06450.pdf)

5 Batch Norm - Inline Q2 1 / 2

- **0 pts** Correct

✓ **- 1 pts** **Incorrect/insufficient justification**

- **2 pts** Incorrect

5.1 Inline Question 3:

Which of these data preprocessing steps is analogous to batch normalization, and which is analogous to layer normalization?

1. Scaling each image in the dataset, so that the RGB channels for each row of pixels within an image sums up to 1.
2. Scaling each image in the dataset, so that the RGB channels for all pixels within an image sums up to 1.

3. Subtracting the mean image of the dataset from each image in the dataset.
4. Setting all RGB values to either 0 or 1 depending on a given threshold.

5.2 Answer:

1 and 2 is layer norm. 3 is batch norm.

6 Layer Normalization: Implementation

Now you'll implement layer normalization. This step should be relatively straightforward, as conceptually the implementation is almost identical to that of batch normalization. One significant difference though is that for layer normalization, we do not keep track of the moving moments, and the testing phase is identical to the training phase, where the mean and variance are directly calculated per datapoint.

Here's what you need to do:

- In `cs231n/layers.py`, implement the forward pass for layer normalization in the function `layernorm_forward`.

Run the cell below to check your results. * In `cs231n/layers.py`, implement the backward pass for layer normalization in the function `layernorm_backward`.

Run the second cell below to check your results. * Modify `cs231n/classifiers/fc_net.py` to add layer normalization to the `FullyConnectedNet`. When the `normalization` flag is set to `"layernorm"` in the constructor, you should insert a layer normalization layer before each ReLU nonlinearity.

Run the third cell below to run the batch size experiment on layer normalization.

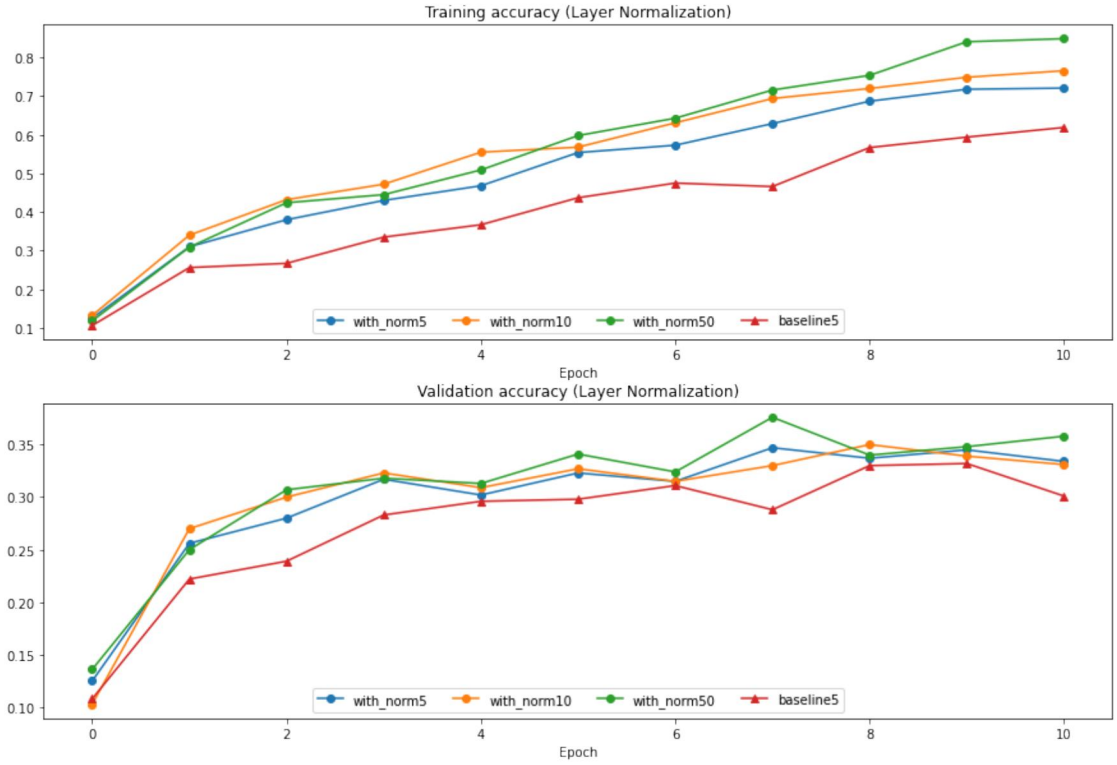
[48]:
```
# Check the training-time forward pass by checking means and variances
# of features both before and after layer normalization

# Simulate the forward pass for a two-layer network
np.random.seed(231)
N, D1, D2, D3 =4, 50, 60, 3
X = np.random.randn(N, D1)
W1 = np.random.randn(D1, D2)
W2 = np.random.randn(D2, D3)
a = np.maximum(0, X.dot(W1)).dot(W2)

print('Before layer normalization:')
```

6 Batch Norm - Inline Q3 2 / 3

- **0 pts** Correct

✓ - **1 pts** **select (1) or (4) for BN/LN**

- **1 pts** not stating that (2) is analogous to LN
- **1 pts** not stating that (3) is analogous to BN
- **3 pts** didn't answer
- **1.5 pts** didn't select correct page



7.1 Inline Question 4:

When is layer normalization likely to not work well, and why?

1. Using it in a very deep network
2. Having a very small dimension of features
3. Having a high regularization term

7.2 Answer:

3. Having a high regularization term will cause layer normalization likely to not work well. Since if reg is very large, the weights will be close to zero and minimizes the layer normalization.

7 Batch Norm - Inline Q4 1 / 2

- **0 pts** Correct

✓ - **1 pts** **didn't choose (2)**

- **1 pts** chose (1)
- **1 pts** didn't provide valid justification
- **1 pts** didn't choose correct page
- **2 pts** didn't provide answer

```
Fraction of test-time output set to zero:  0.0
```

3 Dropout backward pass

In the file `cs231n/layers.py`, implement the backward pass for dropout. After doing so, run the following cell to numerically gradient-check your implementation.

[6]:
```
np.random.seed(231)
x = np.random.randn(10, 10) + 10
dout = np.random.randn(*x.shape)

dropout_param = {'mode': 'train', 'p': 0.2, 'seed': 123}
out, cache = dropout_forward(x, dropout_param)
dx = dropout_backward(dout, cache)
dx_num = eval_numerical_gradient_array(lambda xx: dropout_forward(xx,␣
 ↪dropout_param)[0], x, dout)

# Error should be around e-10 or less
print('dx relative error: ', rel_error(dx, dx_num))
```

```
dx relative error:  5.44560814873387e-11
```

3.1 Inline Question 1:

What happens if we do not divide the values being passed through inverse dropout by `p` in the dropout layer? Why does that happen?

3.2 Answer:

If we didn't divided by p, the average value of the output will become p times the average value of the input, and dropout aims to keep the average value of the input and output remain the same.

4 Fully-connected nets with Dropout

In the file `cs231n/classifiers/fc_net.py`, modify your implementation to use dropout. Specifically, if the constructor of the network receives a value that is not 1 for the `dropout` parameter, then the net should add a dropout layer immediately after every ReLU nonlinearity. After doing so, run the following to numerically gradient-check your implementation.

[7]:
```
np.random.seed(231)
N, D, H1, H2, C = 2, 15, 20, 30, 10
X = np.random.randn(N, D)
y = np.random.randint(C, size=(N,))

for dropout in [1, 0.75, 0.5]:
  print('Running check with dropout = ', dropout)
  model = FullyConnectedNet([H1, H2], input_dim=D, num_classes=C,
```

8 Dropout - Inline Q1 0 / 1

- **0 pts** Correct

✓ **- 1 pts** **Incorrect/Incomplete answer - predictions at test-time will be incorrect.**

5.1 Inline Question 2:

Compare the validation and training accuracies with and without dropout -- what do your results suggest about dropout as a regularizer?

5.2 Answer:

Dropout improves training accuracy but not on validation set.

5.3 Inline Question 3:

Suppose we are training a deep fully-connected network for image classification, with dropout after hidden layers (parameterized by keep probability p). If we are concerned about overfitting, how should we modify p (if at all) when we decide to decrease the size of the hidden layers (that is, the number of nodes in each layer)?

5.4 Answer:

In order to prevent overfitting, we can decrease p, the probability for keeping the nodes. Therefore, less nodes will be kept.

9 Dropout - Inline Q2 0 / 1

- **0 pts** Correct

✓ **- 1 pts** **Incorrect/insufficient explanation**

5.1 Inline Question 2:

Compare the validation and training accuracies with and without dropout -- what do your results suggest about dropout as a regularizer?

5.2 Answer:

Dropout improves training accuracy but not on validation set.

5.3 Inline Question 3:

Suppose we are training a deep fully-connected network for image classification, with dropout after hidden layers (parameterized by keep probability p). If we are concerned about overfitting, how should we modify p (if at all) when we decide to decrease the size of the hidden layers (that is, the number of nodes in each layer)?

5.4 Answer:

In order to prevent overfitting, we can decrease p, the probability for keeping the nodes. Therefore, less nodes will be kept.

10 Dropout - Inline Q3 0 / 1

- **0 pts** Correct
- **0.5 pts** Didn't mention loss of capacity while decreasing hidden units
- **0.5 pts** Partially correct answer

✓ **- 1 pts** **Incorrect/Insufficient answer**