CS7643 Deep Learning: Homework 1

Yousef Emam[1*]

September 26, 2019

*[1]Y. Emam is with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA emamy@gatech.edu

1 Gradient Descent

1.1 Optimizer of Unconstrained Opt. Problem

Function to minimize:

$$g(w) = f(w^t) + (w - w^t)^\top \nabla f(w^t) + \frac{\lambda}{2}||w - w^t||^2$$

Take the gradient w.r.t. w and set it to 0:

$$\nabla g(w^*) = \nabla f(w^t) + \lambda(w^* - w^t) = 0,$$

which implies:

$$w^* = w^t - \frac{1}{\lambda}\nabla f(w^t).$$

This means that gradient descent is the optimal update with respect to the unconstrained problem if $\eta = \frac{1}{\lambda}$.

1.2 Prove Lemma

Q1.2

INCOMPLETE ANSWER

Let $f(v^{(t)}) = \frac{1}{2}\|v^{(t)}\|^2 \Rightarrow \nabla f(v^{(t)}) = v^{(t)}$

$$\Rightarrow f(w^*) \geq f(v^{(t)}) + \langle w^* - v^{(t)}, \nabla f(v^{(t)}) \rangle$$

$$\Rightarrow \frac{\|w^*\|^2}{2} \geq \frac{1}{2}\|v_t\|^2 + \langle w^* - v^{(t)}, v^{(t)} \rangle$$

Note that $v^{(t)} = \frac{-w^{(t)} + w^{(t-1)}}{\eta}$

$$\Rightarrow f(w^*) \geq \frac{1}{2}\|v^{(t)}\|^2 + \langle w^* + \frac{w^{(t)} - w^{(t-1)}}{\eta}, v^{(t)} \rangle$$

$$\Rightarrow T f(w^*) \geq \frac{1}{2}\sum_{t=1}^{T}\|v^{(t)}\|^2 + \sum_{t=1}^{T} \langle w^* + \frac{w^{(t)} - w^{(t-1)}}{\eta}, v^{(t)} \rangle$$

Figure 1: Question 1b- Scanned Answer

1.3 Bound Convergence of Gradient Descent

Q1. 3

$$\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$$

$$f(\bar{w}) - f(w^*) = f\left(\frac{1}{T}\sum_{t=1}^{T} w^{(t)}\right) - f(w^*) \leq \frac{1}{T}\sum_{t=1}^{T}\left(f(w^{(t)}) - f(w^*)\right) \quad (1)$$

$$f(w^*) \geq f(w^{(t)}) + \langle (w^* - w^{(t)}), \nabla f(w^{(t)}) \rangle$$

$$\Rightarrow -f(w^*) \leq -f(w^{(t)}) + \langle (w^* - w^{(t)}), \nabla f(w^{(t)}) \rangle \quad (2)$$

$\Rightarrow$ Plug (2) into RHS of (1):

$$\Rightarrow f(\bar{w}) - f(w^*) \leq \frac{1}{T}\sum_{t=1}^{T}\left(\langle (w^{(t)} - w^*), \nabla f(w^{(t)}) \rangle\right) \leftarrow \text{Desired Expression}$$

$$\leq \frac{1}{T}\left(\frac{\|w^*\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\|\nabla f(w^{(t)})\|^2\right)$$

Plug in upper bounds & $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$

$$\leq \frac{1}{T}\left(\frac{B\sqrt{\rho^2 T}}{2} + \frac{\rho\sqrt{T}}{2}\right)$$

$$= \boxed{\frac{1}{\sqrt{T}}\left(\frac{\rho}{2}(B+1)\right)}$$

Figure 2: Question 1c- Scanned Answer

1.4 SGD Improvement Guarantee

Given function:

$$f(w) = \frac{1}{2}(w-2)^2 + \frac{1}{2}(w+1)^2 = (w - \frac{1}{2})^2 + 2.25$$

Gradient is given by:

$$\nabla f(w) = (w-2) + (w+1)$$

Assume $w^t = 0$, and $N = 2$ is selected:

$$f(0) = (-\frac{1}{2})^2 + 2.25 = 2.5,$$

and

$$w^{t+1} = 0 - \eta \nabla f_2(0) = -\eta.$$

Then:

$$f(w^{t+1}) = (-1/2 - \eta)^2 + 2.25 \geq f(w^t = 0) = (-\frac{1}{2})^2 + 2.25 \, \forall \eta > 0.$$

The above provides a counter example proving that SGD is not guaranteed to decrease the overall loss function in every iteration.

2 Automatic Differentiation (Figure 7)

2.1 Compute the value of f at $\vec{w} = (1, 2)$

$$f(1,2) = [7.80207426 * 10^{24}, 2.73105858]$$

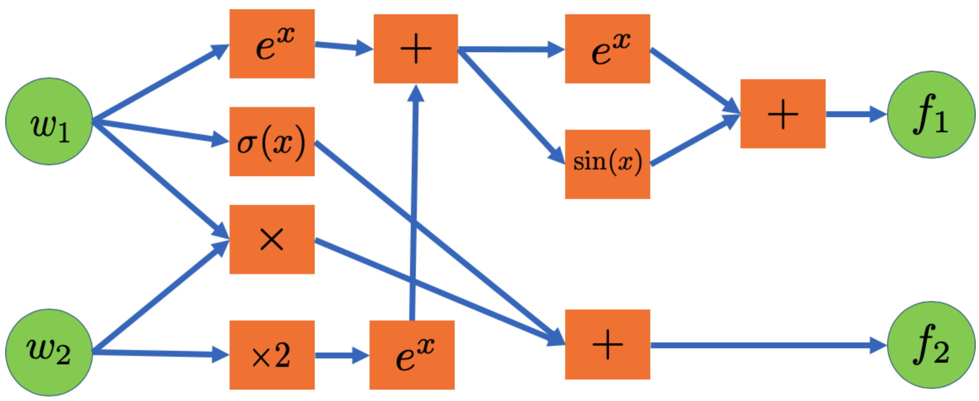


Figure 3: Question 2a- Computational Graph

```
def forward(w):
    w1 = w[0]
    w2 = w[1]
    temp = np.exp(w1)+np.exp(2*w2)
    f1 = np.exp(temp) + np.sin(temp)
    f2 = w1*w2 + sigmoid(w1)
    return np.array([f1,f2])
```

Figure 4: Question 2a- Forward Pass

2.2 Compute the Jacobian using num. diff. with $\Delta w = 0.01$

$$\frac{\partial f(1,2)}{\partial \vec{w}} \approx \begin{bmatrix} 2.18016202 * 10^{25}, 2.19655301 \\ 6.93442399 * 10^{30}, 1.00000000 \end{bmatrix}.$$

Computed using the finite distance formula. Specifically:

$$f'(x) = (f(w+h) - f(w-h))/2h.$$

```
# Q2.b
print("Numerical Differentation: ")
delta_w1 = np.array([0.1, 0])
delta_w2 = np.array([0, 0.1])
print((forward(w+delta_w1)-forward(w-delta_w1))/(2*0.1))
print((forward(w+delta_w2)-forward(w-delta_w2))/(2*0.1))
```

Figure 5: Question 2b

2.3 Compute the Jacobian using forward mode auto-diff.

$$\frac{\partial f(1,2)}{\partial \vec{w}} = \begin{bmatrix} 2.12082367 * 10^{25}, 2.19661193 \\ 8.51957643 * 10^{26}, 1.00000000 \end{bmatrix}.$$

```python
def forward_auto(w):
    w1 = w[0]
    w2 = w[1]
    dw1 = 1
    dw2 = 1
    temp = np.exp(w1)+np.exp(2*w2)
    dtemp_dw1 = np.exp(w1)*dw1
    dtemp_dw2 =  2*np.exp(2*w2)*dw2
    f = np.zeros((2,1))
    f[0] = np.exp(temp) + np.sin(temp)
    f[1] = w1*w2 + sigmoid(w1)
    df_dw = np.zeros((2,2))
    df_dw[0,0] = np.exp(temp)*dtemp_dw1 + np.cos(temp)*dtemp_dw1
    df_dw[1,0] = np.exp(temp)*dtemp_dw2 + np.cos(temp)*dtemp_dw2
    df_dw[0,1] = dw1*w2 + sigmoid(w1)*(1-sigmoid(w1))*dw1
    df_dw[1,1] = w1*dw2
    return (f, df_dw)
```

Figure 6: Question 2c- Forward Auto-Diff

2.4 Compute the Jacobian using backward mode auto-diff.

$$\frac{\partial f(1,2)}{\partial \vec{w}} = \begin{bmatrix} 2.12082367 * 10^{25}, 2.19661193 \\ 8.51957643 * 10^{26}, 1.00000000 \end{bmatrix}.$$

The fact that this is the same result obtained from the forward mode auto-differentiation is not surprising since both forward and backward mode auto-differentiation compute exact derivative.

```
def backward_auto(w):
    w1 = w[0]
    w2 = w[1]
    dw1 = 1
    dw2 = 1
    temp = np.exp(w1)+np.exp(2*w2)
    f = np.zeros((2,1))
    f[0] = np.exp(temp) + np.sin(temp)
    f[1] = w1*w2 + sigmoid(w1)

    df1_dtemp = np.exp(temp) + np.cos(temp)
    df1_w1 = df1_dtemp * np.exp(w1)
    df1_w2 = df1_dtemp * 2 * np.exp(2*w2)
    df_dw[0,0] = df1_w1
    df_dw[1,0] = df1_w2
    df_dw[0,1] = w2 + sigmoid(w1)*(1-sigmoid(w1))
    df_dw[1,1] = w1
    return (f, df_dw)
```

Figure 7: Question 2d- Backward Auto-Diff

2.5 Don't you love that software does this for us?

Yes.

3 Q3: Directed Acyclic Graphs

3.1 If G is a DAG, then G has a topological order

Using the Lemma that every DAG has at least 1 node with in-degree 0, we can construct a topological ordering using the following algorithm:

topOrder = $\{\}$
while G *is not empty* **do**
    temp $\leftarrow \{v \in V\colon \deg_{in}(v) = 0\}$
    topOrder $\leftarrow$ topOrder $\bigcup$ enumerate(temp)
    $G \leftarrow$ remove(G, $\{v \in V\colon \deg_{in}(v) = 0\}$)
end
return topOrder

Algorithm 1: Create Topological Order

The loop is guaranteed to terminate since when 0 in-degree nodes ($\{v \in V\colon \deg_{in}(v) = 0\}$) are removed from the graph, along with their edges, the resulting graph is also a DAG. Therefore, the resulting graph also has at least one node with in-degree 0. Moreover, in this topological ordering, if $n_{v_i} < n_{v_j}$ then there can exist a directed path from v_i to v_j but no path from v_j to v_i.

3.2 If G has a topological order, then G is a DAG

Since many topological orderings can be generated from a DAG, no algorithm can be use to reconstruct the exact DAG given a topological ordering. However, the statement can be proven using the fact that if $n_{v_i} < n_{v_j}$ then there can exist a directed path from v_i to v_j but no path from v_j to v_i. That is simply because if there existed a directed cycle in G, then there must exist two nodes, v_{k_1} and v_{k_2}, such that there is a directed graph from v_{k_1} to v_{k_2} and vice-versa. This in turn implies that $n_{v_{k_1}} > n_{v_{k_2}}$ and $n_{v_{k_2}} > n_{v_{k_1}}$ which is impossible.

softmax

September 24, 2019

1 Softmax Classifier

This exercise guides you through the process of classifying images using a Softmax classifier. As part of this you will:

- Implement a fully vectorized loss function for the Softmax classifier
- Calculate the analytical gradient using vectorized code
- Tune hyperparameters on a validation set
- Optimize the loss function with Stochastic Gradient Descent (SGD)
- Visualize the learned weights

[80]:
```
# start-up code!
import random

import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading extenrnal modules
# see http://stackoverflow.com/questions/1907993/
↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[81]:
```
from load_cifar10_tvt import load_cifar10_train_val

X_train, y_train, X_val, y_val, X_test, y_test = load_cifar10_train_val()
print("Train data shape: ", X_train.shape)
print("Train labels shape: ", y_train.shape)
print("Val data shape: ", X_val.shape)
print("Val labels shape: ", y_val.shape)
```

```
print("Test data shape: ", X_test.shape)
print("Test labels shape: ", y_test.shape)
```

```
Train, validation and testing sets have been created as
 X_i and y_i where i=train,val,test
Train data shape:  (3073, 49000)
Train labels shape:  (49000,)
Val data shape:  (3073, 1000)
Val labels shape:  (1000,)
Test data shape:  (3073, 1000)
Test labels shape:  (1000,)
```

Code for this section is to be written in `cs231n/classifiers/softmax.py`

[156]:
```
# Now, implement the vectorized version in softmax_loss_vectorized.

import time

from cs231n.classifiers.softmax import softmax_loss_vectorized

# gradient check.
from cs231n.gradient_check import grad_check_sparse

W = np.random.randn(10, 3073) * 0.0001

tic = time.time()
loss, grad = softmax_loss_vectorized(W, X_train, y_train, 0.00001)
toc = time.time()
print("vectorized loss: %e computed in %fs" % (loss, toc - tic))

# As a rough sanity check, our loss should be something close to -log(0.1).
print("loss: %f" % loss)
print("sanity check: %f" % (-np.log(0.1)))

f = lambda w: softmax_loss_vectorized(w, X_train, y_train, 0.0)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)
```

```
vectorized loss: 2.317796e+00 computed in 0.984168s
loss: 2.317796
sanity check: 2.302585
numerical: 2.060581 analytic: 2.060581, relative error: 8.199045e-09
numerical: -1.910819 analytic: -1.910819, relative error: 4.206250e-09
numerical: 0.747108 analytic: 0.747108, relative error: 2.444263e-08
numerical: 0.396134 analytic: 0.396134, relative error: 5.135847e-09
numerical: 1.727855 analytic: 1.727855, relative error: 5.932070e-08
numerical: 0.781361 analytic: 0.781361, relative error: 8.113437e-09
numerical: -2.858721 analytic: -2.858721, relative error: 8.045210e-09
numerical: 1.416608 analytic: 1.416608, relative error: 2.640299e-08
```

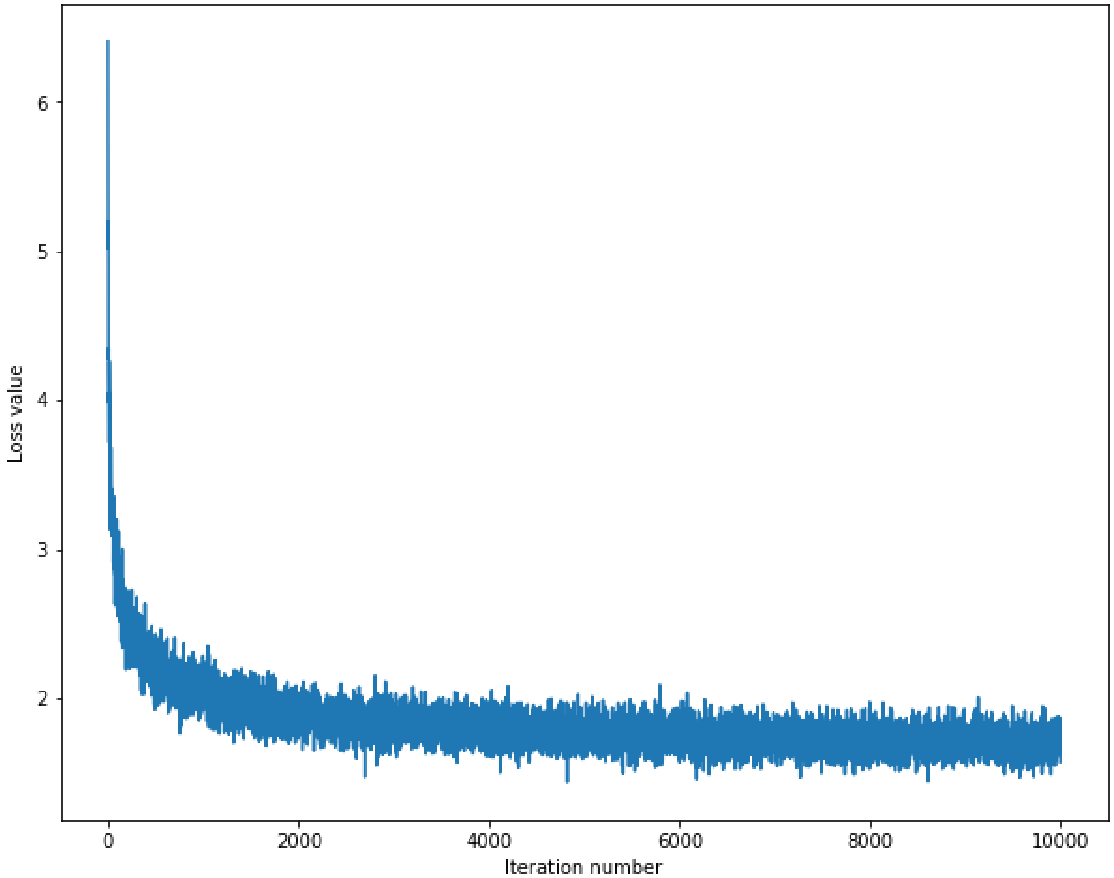
```
numerical: -3.852705 analytic: -3.852705, relative error: 1.780070e-09
numerical: 0.527616 analytic: 0.527616, relative error: 1.419871e-08
```

Code for this section is to be written in`cs231n/classifiers/linear_classifier.py`

[157]:
```
# Now that efficient implementations to calculate loss function and gradient of␣
↪the softmax are ready,
# use it to train the classifier on the cifar-10 data
# Complete the `train` function in cs231n/classifiers/linear_classifier.py

from cs231n.classifiers.linear_classifier import Softmax

classifier = Softmax()
loss_hist = classifier.train(
    X_train,
    y_train,
    learning_rate=1e-6,
    reg=1e-1,
    num_iters=10000,
    batch_size=200,
    verbose=False,
)

# Training Accuracy
y_train_pred = classifier.predict(X_train)
train_accuracy = np.mean(y_train == y_train_pred)
print("softmax on raw pixels training set accuracy: %f" % (train_accuracy,))

# Plot loss vs. iterations
plt.plot(loss_hist)
plt.xlabel("Iteration number")
plt.ylabel("Loss value")
```

```
softmax on raw pixels training set accuracy: 0.424755
```

[157]:
```
Text(0, 0.5, 'Loss value')
```

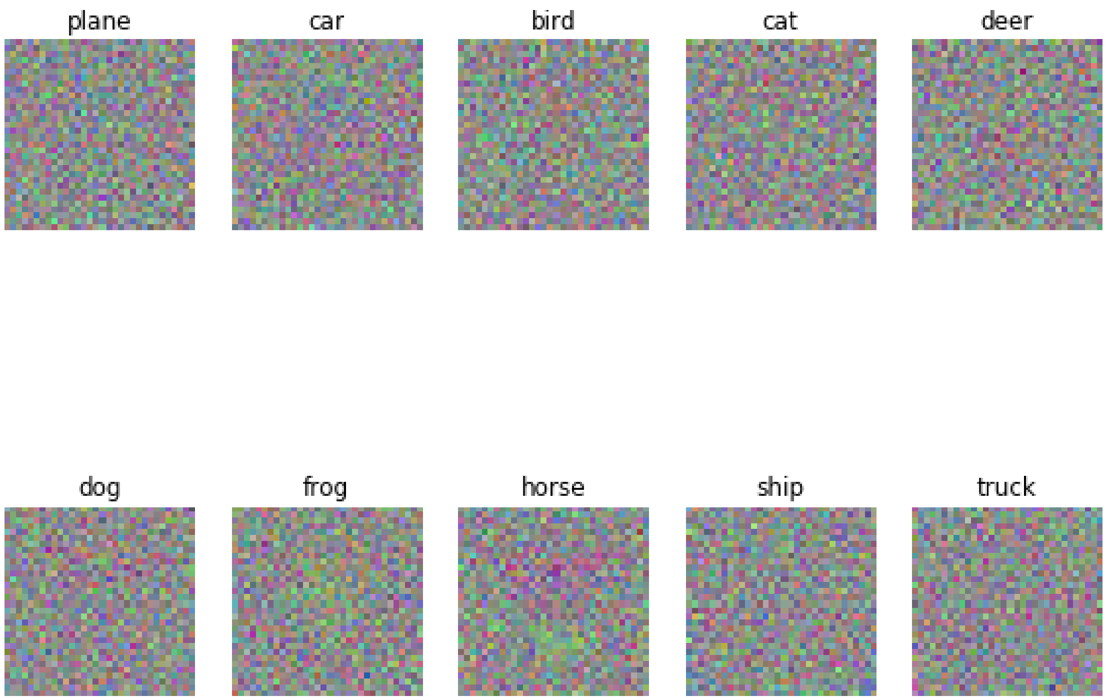


[158]:
```
# Complete the `predict` function in cs231n/classifiers/linear_classifier.py
# Evaluate on test set
y_test_pred = classifier.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print("softmax on raw pixels final test set accuracy: %f" % (test_accuracy,))
```

```
softmax on raw pixels final test set accuracy: 0.380000
```

[159]:
```
# Visualize the learned weights for each class
w = classifier.W[:, :-1]  # strip out the bias
w = w.reshape(10, 32, 32, 3)

w_min, w_max = np.min(w), np.max(w)

classes = [
    "plane",
    "car",
    "bird",
    "cat",
```

```
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[i].squeeze() - w_min) / (w_max - w_min)
    plt.imshow(wimg.astype("uint8"))
    plt.axis("off")
    plt.title(classes[i])
```



two_layer_net

September 24, 2019

1 Implementing a Neural Network

In this exercise we will develop a neural network with fully-connected layers to perform classification, and test it out on the CIFAR-10 dataset.

[2]:
```
# A bit of setup

import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

The neural network parameters will be stored in a dictionary (`model` below), where the keys are the parameter names and the values are numpy arrays. Below, we initialize toy data and a toy model that we will use to verify your implementations.

[19]:
```
# Create some toy data to check your implementations
input_size = 4
hidden_size = 10
num_classes = 3
num_inputs = 5

def init_toy_model():
  model = {}
```

```
    model['W1'] = np.linspace(-0.2, 0.6, num=input_size*hidden_size).
↪reshape(input_size, hidden_size)
    model['b1'] = np.linspace(-0.3, 0.7, num=hidden_size)
    model['W2'] = np.linspace(-0.4, 0.1, num=hidden_size*num_classes).
↪reshape(hidden_size, num_classes)
    model['b2'] = np.linspace(-0.5, 0.9, num=num_classes)
    return model

def init_toy_data():
    X = np.linspace(-0.2, 0.5, num=num_inputs*input_size).reshape(num_inputs,␣
↪input_size)
    y = np.array([0, 1, 2, 2, 1])
    return X, y

model = init_toy_model()
X, y = init_toy_data()
```

2 Forward pass: compute scores

Open the file `cs231n/classifiers/neural_net.py` and look at the function `two_layer_net`. This function is very similar to the loss functions you have written for the Softmax exercise in HW0: It takes the data and weights and computes the class scores, the loss, and the gradients on the parameters.

Implement the first part of the forward pass which uses the weights and biases to compute the scores for all inputs.

[18]:
```
from cs231n.classifiers.neural_net import two_layer_net

scores = two_layer_net(X, model)
print(scores)
correct_scores = [[-0.5328368, 0.20031504, 0.93346689],
 [-0.59412164, 0.15498488, 0.9040914 ],
 [-0.67658362, 0.08978957, 0.85616275],
 [-0.77092643, 0.01339997, 0.79772637],
 [-0.89110401, -0.08754544, 0.71601312]]

# the difference should be very small. We get 3e-8
print('Difference between your scores and correct scores:')
print(np.sum(np.abs(scores - correct_scores)))
```

```
[[-0.5328368   0.20031504  0.93346689]
 [-0.59412164  0.15498488  0.9040914 ]
 [-0.67658362  0.08978957  0.85616275]
 [-0.77092643  0.01339997  0.79772637]
 [-0.89110401 -0.08754544  0.71601312]]
Difference between your scores and correct scores:
```

3.848682278081994e-08

3 Forward pass: compute loss

In the same function, implement the second part that computes the data and regularizaion loss.

[17]:
```
reg = 0.1
loss, _ = two_layer_net(X, model, y, reg)
correct_loss = 1.38191946092

# should be very small, we get 5e-12
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

```
Difference between your loss and correct loss:
4.6769255135359344e-12
```

4 Backward pass

Implement the rest of the function. This will compute the gradient of the loss with respect to the variables `W1`, `b1`, `W2`, and `b2`. Now that you (hopefully!) have a correctly implemented forward pass, you can debug your backward pass using a numeric gradient check:

[16]:
```
from cs231n.gradient_check import eval_numerical_gradient

# Use numeric gradient checking to check your implementation of the backward␣
↪pass.
# If your implementation is correct, the difference between the numeric and
# analytic gradients should be less than 1e-8 for each of W1, W2, b1, and b2.

loss, grads = two_layer_net(X, model, y, reg)

# these should all be less than 1e-8 or so
for param_name in grads:
  param_grad_num = eval_numerical_gradient(lambda W: two_layer_net(X, model, y,␣
↪reg)[0], model[param_name], verbose=False)
  print('%s max relative error: %e' % (param_name, rel_error(param_grad_num,␣
↪grads[param_name])))
```

```
W1 max relative error: 4.426512e-09
b1 max relative error: 5.435432e-08
W2 max relative error: 8.023739e-10
b2 max relative error: 8.190173e-11
```

5 Train the network

To train the network we will use SGD with Momentum. Last assignment you implemented vanilla SGD. You will now implement the momentum update and the RMSProp update. Open the file `classifier_trainer.py` and familiarize yourself with the `ClassifierTrainer` class. It performs optimization given an arbitrary cost function data, and model. By default it uses vanilla SGD, which we have already implemented for you. First, run the optimization below using Vanilla SGD:

[20]:
```
from cs231n.classifier_trainer import ClassifierTrainer

model = init_toy_model()
trainer = ClassifierTrainer()
# call the trainer to optimize the loss
# Notice that we're using sample_batches=False, so we're performing Gradient␣
↪Descent (no sampled batches of data)
best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                             model, two_layer_net,
                                             reg=0.001,
                                             learning_rate=1e-1, momentum=0.0,␣
↪learning_rate_decay=1,
                                             update='sgd', sample_batches=False,
                                             num_epochs=100,
                                             verbose=False)
print('Final loss with vanilla SGD: %f' % (loss_history[-1], ))
```

```
starting iteration  0
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
Final loss with vanilla SGD: 0.940686
```

Now fill in the **momentum update** in the first missing code block inside the `train` function, and run the same optimization as above but with the momentum update. You should see a much better result in the final obtained loss:

[22]:
```
model = init_toy_model()
trainer = ClassifierTrainer()
# call the trainer to optimize the loss
# Notice that we're using sample_batches=False, so we're performing Gradient␣
↪Descent (no sampled batches of data)
best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                             model, two_layer_net,
```

```
                                                    reg=0.001,
                                                    learning_rate=1e-1, momentum=0.9,␣
 ↪learning_rate_decay=1,
                                                    update='momentum',␣
 ↪sample_batches=False,
                                                    num_epochs=100,
                                                    verbose=False)
correct_loss = 0.494394
print('Final loss with momentum SGD: %f. We get: %f' % (loss_history[-1],␣
 ↪correct_loss))
```

```
starting iteration  0
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
Final loss with momentum SGD: 0.494394. We get: 0.494394
```

The **RMSProp** update step is given as follows:

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-8)
```

Here, `decay_rate` is a hyperparameter and typical values are [0.9, 0.99, 0.999].

Implement the **RMSProp** update rule inside the `train` function and rerun the optimization:

[23]:
```
model = init_toy_model()
trainer = ClassifierTrainer()
# call the trainer to optimize the loss
# Notice that we're using sample_batches=False, so we're performing Gradient␣
 ↪Descent (no sampled batches of data)
best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                                    model, two_layer_net,
                                                    reg=0.001,
                                                    learning_rate=1e-1, momentum=0.9,␣
 ↪learning_rate_decay=1,
                                                    update='rmsprop',␣
 ↪sample_batches=False,
                                                    num_epochs=100,
                                                    verbose=False)
correct_loss = 0.439368
print('Final loss with RMSProp: %f. We get: %f' % (loss_history[-1],␣
 ↪correct_loss))
```

```
starting iteration  0
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
Final loss with RMSProp: 0.429848. We get: 0.439368
```

6 Load the data

Now that you have implemented a two-layer network that passes gradient checks, it's time to load up our favorite CIFAR-10 data so we can use it to train a classifier.

[24]:
```
from cs231n.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the two-layer neural net classifier.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = range(num_training, num_training + num_validation)
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = range(num_training)
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = range(num_test)
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Reshape data to rows
    X_train = X_train.reshape(num_training, -1)
```

```
    X_val = X_val.reshape(num_validation, -1)
    X_test = X_test.reshape(num_test, -1)

    return X_train, y_train, X_val, y_val, X_test, y_test


# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

7 Train a network

To train our network we will use SGD with momentum. In addition, we will adjust the learning rate with an exponential learning rate schedule as optimization proceeds; after each epoch, we will reduce the learning rate by multiplying it by a decay rate.

[48]:
```
from cs231n.classifiers.neural_net import init_two_layer_model

model = init_two_layer_model(32*32*3, 70, 10) # input size, hidden size, number␣
 ↪of classes
trainer = ClassifierTrainer()
best_model, loss_history, train_acc, val_acc = trainer.train(X_train, y_train,␣
 ↪X_val, y_val,
                                             model, two_layer_net,
                                             num_epochs=20, reg=1.0,
                                             momentum=0.90, learning_rate_decay␣
 ↪= 0.95,
                                             learning_rate=5e-5, verbose=True)
```

```
starting iteration  0
Finished epoch 0 / 20: cost 2.302596, train: 0.085000, val 0.092000, lr
5.000000e-05
starting iteration  10
starting iteration  20
starting iteration  30
```

```
starting iteration  40
starting iteration  50
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
starting iteration  100
starting iteration  110
starting iteration  120
starting iteration  130
starting iteration  140
starting iteration  150
starting iteration  160
starting iteration  170
starting iteration  180
starting iteration  190
starting iteration  200
starting iteration  210
starting iteration  220
starting iteration  230
starting iteration  240
starting iteration  250
starting iteration  260
starting iteration  270
starting iteration  280
starting iteration  290
starting iteration  300
starting iteration  310
starting iteration  320
starting iteration  330
starting iteration  340
starting iteration  350
starting iteration  360
starting iteration  370
starting iteration  380
starting iteration  390
starting iteration  400
starting iteration  410
starting iteration  420
starting iteration  430
starting iteration  440
starting iteration  450
starting iteration  460
starting iteration  470
starting iteration  480
Finished epoch 1 / 20: cost 1.703471, train: 0.370000, val 0.359000, lr
4.750000e-05
starting iteration  490
```

```
starting iteration  500
starting iteration  510
starting iteration  520
starting iteration  530
starting iteration  540
starting iteration  550
starting iteration  560
starting iteration  570
starting iteration  580
starting iteration  590
starting iteration  600
starting iteration  610
starting iteration  620
starting iteration  630
starting iteration  640
starting iteration  650
starting iteration  660
starting iteration  670
starting iteration  680
starting iteration  690
starting iteration  700
starting iteration  710
starting iteration  720
starting iteration  730
starting iteration  740
starting iteration  750
starting iteration  760
starting iteration  770
starting iteration  780
starting iteration  790
starting iteration  800
starting iteration  810
starting iteration  820
starting iteration  830
starting iteration  840
starting iteration  850
starting iteration  860
starting iteration  870
starting iteration  880
starting iteration  890
starting iteration  900
starting iteration  910
starting iteration  920
starting iteration  930
starting iteration  940
starting iteration  950
starting iteration  960
starting iteration  970
```

```
Finished epoch 2 / 20: cost 1.692839, train: 0.435000, val 0.444000, lr
4.512500e-05
starting iteration  980
starting iteration  990
starting iteration  1000
starting iteration  1010
starting iteration  1020
starting iteration  1030
starting iteration  1040
starting iteration  1050
starting iteration  1060
starting iteration  1070
starting iteration  1080
starting iteration  1090
starting iteration  1100
starting iteration  1110
starting iteration  1120
starting iteration  1130
starting iteration  1140
starting iteration  1150
starting iteration  1160
starting iteration  1170
starting iteration  1180
starting iteration  1190
starting iteration  1200
starting iteration  1210
starting iteration  1220
starting iteration  1230
starting iteration  1240
starting iteration  1250
starting iteration  1260
starting iteration  1270
starting iteration  1280
starting iteration  1290
starting iteration  1300
starting iteration  1310
starting iteration  1320
starting iteration  1330
starting iteration  1340
starting iteration  1350
starting iteration  1360
starting iteration  1370
starting iteration  1380
starting iteration  1390
starting iteration  1400
starting iteration  1410
starting iteration  1420
starting iteration  1430
```

```
starting iteration  1440
starting iteration  1450
starting iteration  1460
Finished epoch 3 / 20: cost 1.549167, train: 0.483000, val 0.451000, lr
4.286875e-05
starting iteration  1470
starting iteration  1480
starting iteration  1490
starting iteration  1500
starting iteration  1510
starting iteration  1520
starting iteration  1530
starting iteration  1540
starting iteration  1550
starting iteration  1560
starting iteration  1570
starting iteration  1580
starting iteration  1590
starting iteration  1600
starting iteration  1610
starting iteration  1620
starting iteration  1630
starting iteration  1640
starting iteration  1650
starting iteration  1660
starting iteration  1670
starting iteration  1680
starting iteration  1690
starting iteration  1700
starting iteration  1710
starting iteration  1720
starting iteration  1730
starting iteration  1740
starting iteration  1750
starting iteration  1760
starting iteration  1770
starting iteration  1780
starting iteration  1790
starting iteration  1800
starting iteration  1810
starting iteration  1820
starting iteration  1830
starting iteration  1840
starting iteration  1850
starting iteration  1860
starting iteration  1870
starting iteration  1880
starting iteration  1890
```

```
starting iteration  1900
starting iteration  1910
starting iteration  1920
starting iteration  1930
starting iteration  1940
starting iteration  1950
Finished epoch 4 / 20: cost 1.596957, train: 0.476000, val 0.456000, lr
4.072531e-05
starting iteration  1960
starting iteration  1970
starting iteration  1980
starting iteration  1990
starting iteration  2000
starting iteration  2010
starting iteration  2020
starting iteration  2030
starting iteration  2040
starting iteration  2050
starting iteration  2060
starting iteration  2070
starting iteration  2080
starting iteration  2090
starting iteration  2100
starting iteration  2110
starting iteration  2120
starting iteration  2130
starting iteration  2140
starting iteration  2150
starting iteration  2160
starting iteration  2170
starting iteration  2180
starting iteration  2190
starting iteration  2200
starting iteration  2210
starting iteration  2220
starting iteration  2230
starting iteration  2240
starting iteration  2250
starting iteration  2260
starting iteration  2270
starting iteration  2280
starting iteration  2290
starting iteration  2300
starting iteration  2310
starting iteration  2320
starting iteration  2330
starting iteration  2340
starting iteration  2350
```

```
starting iteration  2360
starting iteration  2370
starting iteration  2380
starting iteration  2390
starting iteration  2400
starting iteration  2410
starting iteration  2420
starting iteration  2430
starting iteration  2440
Finished epoch 5 / 20: cost 1.591155, train: 0.507000, val 0.465000, lr
3.868905e-05
starting iteration  2450
starting iteration  2460
starting iteration  2470
starting iteration  2480
starting iteration  2490
starting iteration  2500
starting iteration  2510
starting iteration  2520
starting iteration  2530
starting iteration  2540
starting iteration  2550
starting iteration  2560
starting iteration  2570
starting iteration  2580
starting iteration  2590
starting iteration  2600
starting iteration  2610
starting iteration  2620
starting iteration  2630
starting iteration  2640
starting iteration  2650
starting iteration  2660
starting iteration  2670
starting iteration  2680
starting iteration  2690
starting iteration  2700
starting iteration  2710
starting iteration  2720
starting iteration  2730
starting iteration  2740
starting iteration  2750
starting iteration  2760
starting iteration  2770
starting iteration  2780
starting iteration  2790
starting iteration  2800
starting iteration  2810
```

```
starting iteration  2820
starting iteration  2830
starting iteration  2840
starting iteration  2850
starting iteration  2860
starting iteration  2870
starting iteration  2880
starting iteration  2890
starting iteration  2900
starting iteration  2910
starting iteration  2920
starting iteration  2930
Finished epoch 6 / 20: cost 1.506842, train: 0.504000, val 0.461000, lr
3.675459e-05
starting iteration  2940
starting iteration  2950
starting iteration  2960
starting iteration  2970
starting iteration  2980
starting iteration  2990
starting iteration  3000
starting iteration  3010
starting iteration  3020
starting iteration  3030
starting iteration  3040
starting iteration  3050
starting iteration  3060
starting iteration  3070
starting iteration  3080
starting iteration  3090
starting iteration  3100
starting iteration  3110
starting iteration  3120
starting iteration  3130
starting iteration  3140
starting iteration  3150
starting iteration  3160
starting iteration  3170
starting iteration  3180
starting iteration  3190
starting iteration  3200
starting iteration  3210
starting iteration  3220
starting iteration  3230
starting iteration  3240
starting iteration  3250
starting iteration  3260
starting iteration  3270
```

```
starting iteration  3280
starting iteration  3290
starting iteration  3300
starting iteration  3310
starting iteration  3320
starting iteration  3330
starting iteration  3340
starting iteration  3350
starting iteration  3360
starting iteration  3370
starting iteration  3380
starting iteration  3390
starting iteration  3400
starting iteration  3410
starting iteration  3420
Finished epoch 7 / 20: cost 1.512307, train: 0.542000, val 0.484000, lr
3.491686e-05
starting iteration  3430
starting iteration  3440
starting iteration  3450
starting iteration  3460
starting iteration  3470
starting iteration  3480
starting iteration  3490
starting iteration  3500
starting iteration  3510
starting iteration  3520
starting iteration  3530
starting iteration  3540
starting iteration  3550
starting iteration  3560
starting iteration  3570
starting iteration  3580
starting iteration  3590
starting iteration  3600
starting iteration  3610
starting iteration  3620
starting iteration  3630
starting iteration  3640
starting iteration  3650
starting iteration  3660
starting iteration  3670
starting iteration  3680
starting iteration  3690
starting iteration  3700
starting iteration  3710
starting iteration  3720
starting iteration  3730
```

```
starting iteration  3740
starting iteration  3750
starting iteration  3760
starting iteration  3770
starting iteration  3780
starting iteration  3790
starting iteration  3800
starting iteration  3810
starting iteration  3820
starting iteration  3830
starting iteration  3840
starting iteration  3850
starting iteration  3860
starting iteration  3870
starting iteration  3880
starting iteration  3890
starting iteration  3900
starting iteration  3910
Finished epoch 8 / 20: cost 1.628261, train: 0.514000, val 0.492000, lr
3.317102e-05
starting iteration  3920
starting iteration  3930
starting iteration  3940
starting iteration  3950
starting iteration  3960
starting iteration  3970
starting iteration  3980
starting iteration  3990
starting iteration  4000
starting iteration  4010
starting iteration  4020
starting iteration  4030
starting iteration  4040
starting iteration  4050
starting iteration  4060
starting iteration  4070
starting iteration  4080
starting iteration  4090
starting iteration  4100
starting iteration  4110
starting iteration  4120
starting iteration  4130
starting iteration  4140
starting iteration  4150
starting iteration  4160
starting iteration  4170
starting iteration  4180
starting iteration  4190
```

```
starting iteration  4200
starting iteration  4210
starting iteration  4220
starting iteration  4230
starting iteration  4240
starting iteration  4250
starting iteration  4260
starting iteration  4270
starting iteration  4280
starting iteration  4290
starting iteration  4300
starting iteration  4310
starting iteration  4320
starting iteration  4330
starting iteration  4340
starting iteration  4350
starting iteration  4360
starting iteration  4370
starting iteration  4380
starting iteration  4390
starting iteration  4400
Finished epoch 9 / 20: cost 1.500103, train: 0.534000, val 0.484000, lr
3.151247e-05
starting iteration  4410
starting iteration  4420
starting iteration  4430
starting iteration  4440
starting iteration  4450
starting iteration  4460
starting iteration  4470
starting iteration  4480
starting iteration  4490
starting iteration  4500
starting iteration  4510
starting iteration  4520
starting iteration  4530
starting iteration  4540
starting iteration  4550
starting iteration  4560
starting iteration  4570
starting iteration  4580
starting iteration  4590
starting iteration  4600
starting iteration  4610
starting iteration  4620
starting iteration  4630
starting iteration  4640
starting iteration  4650
```

```
starting iteration  4660
starting iteration  4670
starting iteration  4680
starting iteration  4690
starting iteration  4700
starting iteration  4710
starting iteration  4720
starting iteration  4730
starting iteration  4740
starting iteration  4750
starting iteration  4760
starting iteration  4770
starting iteration  4780
starting iteration  4790
starting iteration  4800
starting iteration  4810
starting iteration  4820
starting iteration  4830
starting iteration  4840
starting iteration  4850
starting iteration  4860
starting iteration  4870
starting iteration  4880
starting iteration  4890
Finished epoch 10 / 20: cost 1.439580, train: 0.553000, val 0.493000, lr
2.993685e-05
starting iteration  4900
starting iteration  4910
starting iteration  4920
starting iteration  4930
starting iteration  4940
starting iteration  4950
starting iteration  4960
starting iteration  4970
starting iteration  4980
starting iteration  4990
starting iteration  5000
starting iteration  5010
starting iteration  5020
starting iteration  5030
starting iteration  5040
starting iteration  5050
starting iteration  5060
starting iteration  5070
starting iteration  5080
starting iteration  5090
starting iteration  5100
starting iteration  5110
```

```
starting iteration  5120
starting iteration  5130
starting iteration  5140
starting iteration  5150
starting iteration  5160
starting iteration  5170
starting iteration  5180
starting iteration  5190
starting iteration  5200
starting iteration  5210
starting iteration  5220
starting iteration  5230
starting iteration  5240
starting iteration  5250
starting iteration  5260
starting iteration  5270
starting iteration  5280
starting iteration  5290
starting iteration  5300
starting iteration  5310
starting iteration  5320
starting iteration  5330
starting iteration  5340
starting iteration  5350
starting iteration  5360
starting iteration  5370
starting iteration  5380
Finished epoch 11 / 20: cost 1.420972, train: 0.551000, val 0.485000, lr
2.844000e-05
starting iteration  5390
starting iteration  5400
starting iteration  5410
starting iteration  5420
starting iteration  5430
starting iteration  5440
starting iteration  5450
starting iteration  5460
starting iteration  5470
starting iteration  5480
starting iteration  5490
starting iteration  5500
starting iteration  5510
starting iteration  5520
starting iteration  5530
starting iteration  5540
starting iteration  5550
starting iteration  5560
starting iteration  5570
```

```
starting iteration  5580
starting iteration  5590
starting iteration  5600
starting iteration  5610
starting iteration  5620
starting iteration  5630
starting iteration  5640
starting iteration  5650
starting iteration  5660
starting iteration  5670
starting iteration  5680
starting iteration  5690
starting iteration  5700
starting iteration  5710
starting iteration  5720
starting iteration  5730
starting iteration  5740
starting iteration  5750
starting iteration  5760
starting iteration  5770
starting iteration  5780
starting iteration  5790
starting iteration  5800
starting iteration  5810
starting iteration  5820
starting iteration  5830
starting iteration  5840
starting iteration  5850
starting iteration  5860
starting iteration  5870
Finished epoch 12 / 20: cost 1.502760, train: 0.545000, val 0.500000, lr
2.701800e-05
starting iteration  5880
starting iteration  5890
starting iteration  5900
starting iteration  5910
starting iteration  5920
starting iteration  5930
starting iteration  5940
starting iteration  5950
starting iteration  5960
starting iteration  5970
starting iteration  5980
starting iteration  5990
starting iteration  6000
starting iteration  6010
starting iteration  6020
starting iteration  6030
```

```
starting iteration  6040
starting iteration  6050
starting iteration  6060
starting iteration  6070
starting iteration  6080
starting iteration  6090
starting iteration  6100
starting iteration  6110
starting iteration  6120
starting iteration  6130
starting iteration  6140
starting iteration  6150
starting iteration  6160
starting iteration  6170
starting iteration  6180
starting iteration  6190
starting iteration  6200
starting iteration  6210
starting iteration  6220
starting iteration  6230
starting iteration  6240
starting iteration  6250
starting iteration  6260
starting iteration  6270
starting iteration  6280
starting iteration  6290
starting iteration  6300
starting iteration  6310
starting iteration  6320
starting iteration  6330
starting iteration  6340
starting iteration  6350
starting iteration  6360
Finished epoch 13 / 20: cost 1.506627, train: 0.535000, val 0.511000, lr
2.566710e-05
starting iteration  6370
starting iteration  6380
starting iteration  6390
starting iteration  6400
starting iteration  6410
starting iteration  6420
starting iteration  6430
starting iteration  6440
starting iteration  6450
starting iteration  6460
starting iteration  6470
starting iteration  6480
starting iteration  6490
```

```
starting iteration  6500
starting iteration  6510
starting iteration  6520
starting iteration  6530
starting iteration  6540
starting iteration  6550
starting iteration  6560
starting iteration  6570
starting iteration  6580
starting iteration  6590
starting iteration  6600
starting iteration  6610
starting iteration  6620
starting iteration  6630
starting iteration  6640
starting iteration  6650
starting iteration  6660
starting iteration  6670
starting iteration  6680
starting iteration  6690
starting iteration  6700
starting iteration  6710
starting iteration  6720
starting iteration  6730
starting iteration  6740
starting iteration  6750
starting iteration  6760
starting iteration  6770
starting iteration  6780
starting iteration  6790
starting iteration  6800
starting iteration  6810
starting iteration  6820
starting iteration  6830
starting iteration  6840
starting iteration  6850
Finished epoch 14 / 20: cost 1.482915, train: 0.572000, val 0.489000, lr
2.438375e-05
starting iteration  6860
starting iteration  6870
starting iteration  6880
starting iteration  6890
starting iteration  6900
starting iteration  6910
starting iteration  6920
starting iteration  6930
starting iteration  6940
starting iteration  6950
```

```
starting iteration  6960
starting iteration  6970
starting iteration  6980
starting iteration  6990
starting iteration  7000
starting iteration  7010
starting iteration  7020
starting iteration  7030
starting iteration  7040
starting iteration  7050
starting iteration  7060
starting iteration  7070
starting iteration  7080
starting iteration  7090
starting iteration  7100
starting iteration  7110
starting iteration  7120
starting iteration  7130
starting iteration  7140
starting iteration  7150
starting iteration  7160
starting iteration  7170
starting iteration  7180
starting iteration  7190
starting iteration  7200
starting iteration  7210
starting iteration  7220
starting iteration  7230
starting iteration  7240
starting iteration  7250
starting iteration  7260
starting iteration  7270
starting iteration  7280
starting iteration  7290
starting iteration  7300
starting iteration  7310
starting iteration  7320
starting iteration  7330
starting iteration  7340
Finished epoch 15 / 20: cost 1.639811, train: 0.568000, val 0.502000, lr
2.316456e-05
starting iteration  7350
starting iteration  7360
starting iteration  7370
starting iteration  7380
starting iteration  7390
starting iteration  7400
starting iteration  7410
```

```
starting iteration  7420
starting iteration  7430
starting iteration  7440
starting iteration  7450
starting iteration  7460
starting iteration  7470
starting iteration  7480
starting iteration  7490
starting iteration  7500
starting iteration  7510
starting iteration  7520
starting iteration  7530
starting iteration  7540
starting iteration  7550
starting iteration  7560
starting iteration  7570
starting iteration  7580
starting iteration  7590
starting iteration  7600
starting iteration  7610
starting iteration  7620
starting iteration  7630
starting iteration  7640
starting iteration  7650
starting iteration  7660
starting iteration  7670
starting iteration  7680
starting iteration  7690
starting iteration  7700
starting iteration  7710
starting iteration  7720
starting iteration  7730
starting iteration  7740
starting iteration  7750
starting iteration  7760
starting iteration  7770
starting iteration  7780
starting iteration  7790
starting iteration  7800
starting iteration  7810
starting iteration  7820
starting iteration  7830
Finished epoch 16 / 20: cost 1.512649, train: 0.550000, val 0.506000, lr
2.200633e-05
starting iteration  7840
starting iteration  7850
starting iteration  7860
starting iteration  7870
```

```
starting iteration  7880
starting iteration  7890
starting iteration  7900
starting iteration  7910
starting iteration  7920
starting iteration  7930
starting iteration  7940
starting iteration  7950
starting iteration  7960
starting iteration  7970
starting iteration  7980
starting iteration  7990
starting iteration  8000
starting iteration  8010
starting iteration  8020
starting iteration  8030
starting iteration  8040
starting iteration  8050
starting iteration  8060
starting iteration  8070
starting iteration  8080
starting iteration  8090
starting iteration  8100
starting iteration  8110
starting iteration  8120
starting iteration  8130
starting iteration  8140
starting iteration  8150
starting iteration  8160
starting iteration  8170
starting iteration  8180
starting iteration  8190
starting iteration  8200
starting iteration  8210
starting iteration  8220
starting iteration  8230
starting iteration  8240
starting iteration  8250
starting iteration  8260
starting iteration  8270
starting iteration  8280
starting iteration  8290
starting iteration  8300
starting iteration  8310
starting iteration  8320
Finished epoch 17 / 20: cost 1.473777, train: 0.555000, val 0.517000, lr
2.090602e-05
starting iteration  8330
```

```
starting iteration  8340
starting iteration  8350
starting iteration  8360
starting iteration  8370
starting iteration  8380
starting iteration  8390
starting iteration  8400
starting iteration  8410
starting iteration  8420
starting iteration  8430
starting iteration  8440
starting iteration  8450
starting iteration  8460
starting iteration  8470
starting iteration  8480
starting iteration  8490
starting iteration  8500
starting iteration  8510
starting iteration  8520
starting iteration  8530
starting iteration  8540
starting iteration  8550
starting iteration  8560
starting iteration  8570
starting iteration  8580
starting iteration  8590
starting iteration  8600
starting iteration  8610
starting iteration  8620
starting iteration  8630
starting iteration  8640
starting iteration  8650
starting iteration  8660
starting iteration  8670
starting iteration  8680
starting iteration  8690
starting iteration  8700
starting iteration  8710
starting iteration  8720
starting iteration  8730
starting iteration  8740
starting iteration  8750
starting iteration  8760
starting iteration  8770
starting iteration  8780
starting iteration  8790
starting iteration  8800
starting iteration  8810
```

```
Finished epoch 18 / 20: cost 1.556926, train: 0.562000, val 0.509000, lr
1.986072e-05
starting iteration  8820
starting iteration  8830
starting iteration  8840
starting iteration  8850
starting iteration  8860
starting iteration  8870
starting iteration  8880
starting iteration  8890
starting iteration  8900
starting iteration  8910
starting iteration  8920
starting iteration  8930
starting iteration  8940
starting iteration  8950
starting iteration  8960
starting iteration  8970
starting iteration  8980
starting iteration  8990
starting iteration  9000
starting iteration  9010
starting iteration  9020
starting iteration  9030
starting iteration  9040
starting iteration  9050
starting iteration  9060
starting iteration  9070
starting iteration  9080
starting iteration  9090
starting iteration  9100
starting iteration  9110
starting iteration  9120
starting iteration  9130
starting iteration  9140
starting iteration  9150
starting iteration  9160
starting iteration  9170
starting iteration  9180
starting iteration  9190
starting iteration  9200
starting iteration  9210
starting iteration  9220
starting iteration  9230
starting iteration  9240
starting iteration  9250
starting iteration  9260
starting iteration  9270
```

```
starting iteration  9280
starting iteration  9290
starting iteration  9300
Finished epoch 19 / 20: cost 1.502153, train: 0.546000, val 0.513000, lr
1.886768e-05
starting iteration  9310
starting iteration  9320
starting iteration  9330
starting iteration  9340
starting iteration  9350
starting iteration  9360
starting iteration  9370
starting iteration  9380
starting iteration  9390
starting iteration  9400
starting iteration  9410
starting iteration  9420
starting iteration  9430
starting iteration  9440
starting iteration  9450
starting iteration  9460
starting iteration  9470
starting iteration  9480
starting iteration  9490
starting iteration  9500
starting iteration  9510
starting iteration  9520
starting iteration  9530
starting iteration  9540
starting iteration  9550
starting iteration  9560
starting iteration  9570
starting iteration  9580
starting iteration  9590
starting iteration  9600
starting iteration  9610
starting iteration  9620
starting iteration  9630
starting iteration  9640
starting iteration  9650
starting iteration  9660
starting iteration  9670
starting iteration  9680
starting iteration  9690
starting iteration  9700
starting iteration  9710
starting iteration  9720
starting iteration  9730
```

```
starting iteration  9740
starting iteration  9750
starting iteration  9760
starting iteration  9770
starting iteration  9780
starting iteration  9790
Finished epoch 20 / 20: cost 1.272936, train: 0.561000, val 0.521000, lr
1.792430e-05
finished optimization. best validation accuracy: 0.521000
```

8 Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.37 on the validation set. This isn't very good.

One strategy for getting insight into what's wrong is to plot the loss function and the accuracies on the training and validation sets during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

[49]:
```python
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(loss_history)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_acc)
plt.plot(val_acc)
plt.legend(['Training accuracy', 'Validation accuracy'], loc='lower right')
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
```

[49]:
```
Text(0, 0.5, 'Clasification accuracy')
```

[50]:
```
from cs231n.vis_utils import visualize_grid

# Visualize the weights of the network

def show_net_weights(model):
    plt.imshow(visualize_grid(model['W1'].T.reshape(-1, 32, 32, 3), padding=3).
 ↪astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(model)
```

9 Tune your hyperparameters

What's wrong?. Looking at the visualizations above, we see that the loss is decreasing more or less linearly, which seems to suggest that the learning rate may be too low. Moreover, there is no gap between the training and validation accuracy, suggesting that the model we used has low capacity, and that we should increase its size. On the other hand, with a very large model we would expect to see more overfitting, which would manifest itself as a very large gap between the training and validation accuracy.

Tuning. Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks, so we want you to get a lot of practice. Below, you should experiment with different values of the various hyperparameters, including hidden layer size, learning rate, numer of training epochs, and regularization strength. You might also consider tuning the momentum and learning rate decay parameters, but you should be able to get good performance using the default values.

Approximate results. You should be aim to achieve a classification accuracy of greater than 50% on the validation set. Our best network gets over 56% on the validation set.

Experiment: You goal in this exercise is to get as good of a result on CIFAR-10 as you can, with a fully-connected Neural Network. For every 1% above 56% on the Test set we will award you with one extra bonus point. Feel free implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.).

[52]:
```
best_model = None # store the best model into this


##################################################################################
```

```
# TODO: Tune hyperparameters using the validation set. Store your best trained ␣
↪#
# model in best_model.                                                         ␣
↪#
#                                                                              ␣
↪#
# To help debug your network, it may help to use visualizations similar to the ␣
↪#
# ones we used above; these visualizations will have significant qualitative   ␣
↪#
# differences from the ones we saw above for the poorly tuned network.         ␣
↪#
#                                                                              ␣
↪#
# Tweaking hyperparameters by hand can be fun, but you might find it useful to ␣
↪#
# write code to sweep through possible combinations of hyperparameters         ␣
↪#
# automatically like we did on the previous assignment.                        ␣
↪#
################################################################################
# input size, hidden size, number of classes
model = init_two_layer_model(32*32*3, 1000, 10)
trainer = ClassifierTrainer()
best_model, loss_history, train_acc, val_acc = trainer.train(X_train, y_train,␣
↪X_val, y_val,
                                             model, two_layer_net,
                                             num_epochs=20, reg=1.0,
                                             momentum=0.90, learning_rate_decay␣
↪= 0.95,
                                             learning_rate=5e-5, verbose=True)
################################################################################
#                              END OF YOUR CODE                                ␣
↪#
################################################################################
```

```
starting iteration  0
Finished epoch 0 / 20: cost 2.302740, train: 0.088000, val 0.085000, lr
5.000000e-05
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
starting iteration  60
starting iteration  70
```

```
starting iteration  80
starting iteration  90
starting iteration  100
starting iteration  110
starting iteration  120
starting iteration  130
starting iteration  140
starting iteration  150
starting iteration  160
starting iteration  170
starting iteration  180
starting iteration  190
starting iteration  200
starting iteration  210
starting iteration  220
starting iteration  230
starting iteration  240
starting iteration  250
starting iteration  260
starting iteration  270
starting iteration  280
starting iteration  290
starting iteration  300
starting iteration  310
starting iteration  320
starting iteration  330
starting iteration  340
starting iteration  350
starting iteration  360
starting iteration  370
starting iteration  380
starting iteration  390
starting iteration  400
starting iteration  410
starting iteration  420
starting iteration  430
starting iteration  440
starting iteration  450
starting iteration  460
starting iteration  470
starting iteration  480
Finished epoch 1 / 20: cost 1.800183, train: 0.359000, val 0.386000, lr
4.750000e-05
starting iteration  490
starting iteration  500
starting iteration  510
starting iteration  520
starting iteration  530
```

```
starting iteration  540
starting iteration  550
starting iteration  560
starting iteration  570
starting iteration  580
starting iteration  590
starting iteration  600
starting iteration  610
starting iteration  620
starting iteration  630
starting iteration  640
starting iteration  650
starting iteration  660
starting iteration  670
starting iteration  680
starting iteration  690
starting iteration  700
starting iteration  710
starting iteration  720
starting iteration  730
starting iteration  740
starting iteration  750
starting iteration  760
starting iteration  770
starting iteration  780
starting iteration  790
starting iteration  800
starting iteration  810
starting iteration  820
starting iteration  830
starting iteration  840
starting iteration  850
starting iteration  860
starting iteration  870
starting iteration  880
starting iteration  890
starting iteration  900
starting iteration  910
starting iteration  920
starting iteration  930
starting iteration  940
starting iteration  950
starting iteration  960
starting iteration  970
Finished epoch 2 / 20: cost 1.586629, train: 0.431000, val 0.452000, lr
4.512500e-05
starting iteration  980
starting iteration  990
```

```
starting iteration  1000
starting iteration  1010
starting iteration  1020
starting iteration  1030
starting iteration  1040
starting iteration  1050
starting iteration  1060
starting iteration  1070
starting iteration  1080
starting iteration  1090
starting iteration  1100
starting iteration  1110
starting iteration  1120
starting iteration  1130
starting iteration  1140
starting iteration  1150
starting iteration  1160
starting iteration  1170
starting iteration  1180
starting iteration  1190
starting iteration  1200
starting iteration  1210
starting iteration  1220
starting iteration  1230
starting iteration  1240
starting iteration  1250
starting iteration  1260
starting iteration  1270
starting iteration  1280
starting iteration  1290
starting iteration  1300
starting iteration  1310
starting iteration  1320
starting iteration  1330
starting iteration  1340
starting iteration  1350
starting iteration  1360
starting iteration  1370
starting iteration  1380
starting iteration  1390
starting iteration  1400
starting iteration  1410
starting iteration  1420
starting iteration  1430
starting iteration  1440
starting iteration  1450
starting iteration  1460
Finished epoch 3 / 20: cost 1.681464, train: 0.477000, val 0.466000, lr
```

```
4.286875e-05
starting iteration  1470
starting iteration  1480
starting iteration  1490
starting iteration  1500
starting iteration  1510
starting iteration  1520
starting iteration  1530
starting iteration  1540
starting iteration  1550
starting iteration  1560
starting iteration  1570
starting iteration  1580
starting iteration  1590
starting iteration  1600
starting iteration  1610
starting iteration  1620
starting iteration  1630
starting iteration  1640
starting iteration  1650
starting iteration  1660
starting iteration  1670
starting iteration  1680
starting iteration  1690
starting iteration  1700
starting iteration  1710
starting iteration  1720
starting iteration  1730
starting iteration  1740
starting iteration  1750
starting iteration  1760
starting iteration  1770
starting iteration  1780
starting iteration  1790
starting iteration  1800
starting iteration  1810
starting iteration  1820
starting iteration  1830
starting iteration  1840
starting iteration  1850
starting iteration  1860
starting iteration  1870
starting iteration  1880
starting iteration  1890
starting iteration  1900
starting iteration  1910
starting iteration  1920
starting iteration  1930
```

```
starting iteration  1940
starting iteration  1950
Finished epoch 4 / 20: cost 1.542658, train: 0.497000, val 0.479000, lr
4.072531e-05
starting iteration  1960
starting iteration  1970
starting iteration  1980
starting iteration  1990
starting iteration  2000
starting iteration  2010
starting iteration  2020
starting iteration  2030
starting iteration  2040
starting iteration  2050
starting iteration  2060
starting iteration  2070
starting iteration  2080
starting iteration  2090
starting iteration  2100
starting iteration  2110
starting iteration  2120
starting iteration  2130
starting iteration  2140
starting iteration  2150
starting iteration  2160
starting iteration  2170
starting iteration  2180
starting iteration  2190
starting iteration  2200
starting iteration  2210
starting iteration  2220
starting iteration  2230
starting iteration  2240
starting iteration  2250
starting iteration  2260
starting iteration  2270
starting iteration  2280
starting iteration  2290
starting iteration  2300
starting iteration  2310
starting iteration  2320
starting iteration  2330
starting iteration  2340
starting iteration  2350
starting iteration  2360
starting iteration  2370
starting iteration  2380
starting iteration  2390
```

```
starting iteration  2400
starting iteration  2410
starting iteration  2420
starting iteration  2430
starting iteration  2440
Finished epoch 5 / 20: cost 1.421326, train: 0.503000, val 0.498000, lr
3.868905e-05
starting iteration  2450
starting iteration  2460
starting iteration  2470
starting iteration  2480
starting iteration  2490
starting iteration  2500
starting iteration  2510
starting iteration  2520
starting iteration  2530
starting iteration  2540
starting iteration  2550
starting iteration  2560
starting iteration  2570
starting iteration  2580
starting iteration  2590
starting iteration  2600
starting iteration  2610
starting iteration  2620
starting iteration  2630
starting iteration  2640
starting iteration  2650
starting iteration  2660
starting iteration  2670
starting iteration  2680
starting iteration  2690
starting iteration  2700
starting iteration  2710
starting iteration  2720
starting iteration  2730
starting iteration  2740
starting iteration  2750
starting iteration  2760
starting iteration  2770
starting iteration  2780
starting iteration  2790
starting iteration  2800
starting iteration  2810
starting iteration  2820
starting iteration  2830
starting iteration  2840
starting iteration  2850
```

```
starting iteration  2860
starting iteration  2870
starting iteration  2880
starting iteration  2890
starting iteration  2900
starting iteration  2910
starting iteration  2920
starting iteration  2930
Finished epoch 6 / 20: cost 1.476008, train: 0.514000, val 0.500000, lr
3.675459e-05
starting iteration  2940
starting iteration  2950
starting iteration  2960
starting iteration  2970
starting iteration  2980
starting iteration  2990
starting iteration  3000
starting iteration  3010
starting iteration  3020
starting iteration  3030
starting iteration  3040
starting iteration  3050
starting iteration  3060
starting iteration  3070
starting iteration  3080
starting iteration  3090
starting iteration  3100
starting iteration  3110
starting iteration  3120
starting iteration  3130
starting iteration  3140
starting iteration  3150
starting iteration  3160
starting iteration  3170
starting iteration  3180
starting iteration  3190
starting iteration  3200
starting iteration  3210
starting iteration  3220
starting iteration  3230
starting iteration  3240
starting iteration  3250
starting iteration  3260
starting iteration  3270
starting iteration  3280
starting iteration  3290
starting iteration  3300
starting iteration  3310
```

```
starting iteration  3320
starting iteration  3330
starting iteration  3340
starting iteration  3350
starting iteration  3360
starting iteration  3370
starting iteration  3380
starting iteration  3390
starting iteration  3400
starting iteration  3410
starting iteration  3420
Finished epoch 7 / 20: cost 1.537347, train: 0.551000, val 0.528000, lr
3.491686e-05
starting iteration  3430
starting iteration  3440
starting iteration  3450
starting iteration  3460
starting iteration  3470
starting iteration  3480
starting iteration  3490
starting iteration  3500
starting iteration  3510
starting iteration  3520
starting iteration  3530
starting iteration  3540
starting iteration  3550
starting iteration  3560
starting iteration  3570
starting iteration  3580
starting iteration  3590
starting iteration  3600
starting iteration  3610
starting iteration  3620
starting iteration  3630
starting iteration  3640
starting iteration  3650
starting iteration  3660
starting iteration  3670
starting iteration  3680
starting iteration  3690
starting iteration  3700
starting iteration  3710
starting iteration  3720
starting iteration  3730
starting iteration  3740
starting iteration  3750
starting iteration  3760
starting iteration  3770
```

```
starting iteration  3780
starting iteration  3790
starting iteration  3800
starting iteration  3810
starting iteration  3820
starting iteration  3830
starting iteration  3840
starting iteration  3850
starting iteration  3860
starting iteration  3870
starting iteration  3880
starting iteration  3890
starting iteration  3900
starting iteration  3910
Finished epoch 8 / 20: cost 1.493411, train: 0.535000, val 0.482000, lr
3.317102e-05
starting iteration  3920
starting iteration  3930
starting iteration  3940
starting iteration  3950
starting iteration  3960
starting iteration  3970
starting iteration  3980
starting iteration  3990
starting iteration  4000
starting iteration  4010
starting iteration  4020
starting iteration  4030
starting iteration  4040
starting iteration  4050
starting iteration  4060
starting iteration  4070
starting iteration  4080
starting iteration  4090
starting iteration  4100
starting iteration  4110
starting iteration  4120
starting iteration  4130
starting iteration  4140
starting iteration  4150
starting iteration  4160
starting iteration  4170
starting iteration  4180
starting iteration  4190
starting iteration  4200
starting iteration  4210
starting iteration  4220
starting iteration  4230
```

```
starting iteration  4240
starting iteration  4250
starting iteration  4260
starting iteration  4270
starting iteration  4280
starting iteration  4290
starting iteration  4300
starting iteration  4310
starting iteration  4320
starting iteration  4330
starting iteration  4340
starting iteration  4350
starting iteration  4360
starting iteration  4370
starting iteration  4380
starting iteration  4390
starting iteration  4400
Finished epoch 9 / 20: cost 1.323490, train: 0.548000, val 0.510000, lr
3.151247e-05
starting iteration  4410
starting iteration  4420
starting iteration  4430
starting iteration  4440
starting iteration  4450
starting iteration  4460
starting iteration  4470
starting iteration  4480
starting iteration  4490
starting iteration  4500
starting iteration  4510
starting iteration  4520
starting iteration  4530
starting iteration  4540
starting iteration  4550
starting iteration  4560
starting iteration  4570
starting iteration  4580
starting iteration  4590
starting iteration  4600
starting iteration  4610
starting iteration  4620
starting iteration  4630
starting iteration  4640
starting iteration  4650
starting iteration  4660
starting iteration  4670
starting iteration  4680
starting iteration  4690
```

```
starting iteration  4700
starting iteration  4710
starting iteration  4720
starting iteration  4730
starting iteration  4740
starting iteration  4750
starting iteration  4760
starting iteration  4770
starting iteration  4780
starting iteration  4790
starting iteration  4800
starting iteration  4810
starting iteration  4820
starting iteration  4830
starting iteration  4840
starting iteration  4850
starting iteration  4860
starting iteration  4870
starting iteration  4880
starting iteration  4890
Finished epoch 10 / 20: cost 1.398117, train: 0.565000, val 0.520000, lr
2.993685e-05
starting iteration  4900
starting iteration  4910
starting iteration  4920
starting iteration  4930
starting iteration  4940
starting iteration  4950
starting iteration  4960
starting iteration  4970
starting iteration  4980
starting iteration  4990
starting iteration  5000
starting iteration  5010
starting iteration  5020
starting iteration  5030
starting iteration  5040
starting iteration  5050
starting iteration  5060
starting iteration  5070
starting iteration  5080
starting iteration  5090
starting iteration  5100
starting iteration  5110
starting iteration  5120
starting iteration  5130
starting iteration  5140
starting iteration  5150
```

```
starting iteration  5160
starting iteration  5170
starting iteration  5180
starting iteration  5190
starting iteration  5200
starting iteration  5210
starting iteration  5220
starting iteration  5230
starting iteration  5240
starting iteration  5250
starting iteration  5260
starting iteration  5270
starting iteration  5280
starting iteration  5290
starting iteration  5300
starting iteration  5310
starting iteration  5320
starting iteration  5330
starting iteration  5340
starting iteration  5350
starting iteration  5360
starting iteration  5370
starting iteration  5380
Finished epoch 11 / 20: cost 1.619243, train: 0.569000, val 0.508000, lr
2.844000e-05
starting iteration  5390
starting iteration  5400
starting iteration  5410
starting iteration  5420
starting iteration  5430
starting iteration  5440
starting iteration  5450
starting iteration  5460
starting iteration  5470
starting iteration  5480
starting iteration  5490
starting iteration  5500
starting iteration  5510
starting iteration  5520
starting iteration  5530
starting iteration  5540
starting iteration  5550
starting iteration  5560
starting iteration  5570
starting iteration  5580
starting iteration  5590
starting iteration  5600
starting iteration  5610
```

```
starting iteration  5620
starting iteration  5630
starting iteration  5640
starting iteration  5650
starting iteration  5660
starting iteration  5670
starting iteration  5680
starting iteration  5690
starting iteration  5700
starting iteration  5710
starting iteration  5720
starting iteration  5730
starting iteration  5740
starting iteration  5750
starting iteration  5760
starting iteration  5770
starting iteration  5780
starting iteration  5790
starting iteration  5800
starting iteration  5810
starting iteration  5820
starting iteration  5830
starting iteration  5840
starting iteration  5850
starting iteration  5860
starting iteration  5870
Finished epoch 12 / 20: cost 1.483477, train: 0.563000, val 0.523000, lr
2.701800e-05
starting iteration  5880
starting iteration  5890
starting iteration  5900
starting iteration  5910
starting iteration  5920
starting iteration  5930
starting iteration  5940
starting iteration  5950
starting iteration  5960
starting iteration  5970
starting iteration  5980
starting iteration  5990
starting iteration  6000
starting iteration  6010
starting iteration  6020
starting iteration  6030
starting iteration  6040
starting iteration  6050
starting iteration  6060
starting iteration  6070
```

```
starting iteration  6080
starting iteration  6090
starting iteration  6100
starting iteration  6110
starting iteration  6120
starting iteration  6130
starting iteration  6140
starting iteration  6150
starting iteration  6160
starting iteration  6170
starting iteration  6180
starting iteration  6190
starting iteration  6200
starting iteration  6210
starting iteration  6220
starting iteration  6230
starting iteration  6240
starting iteration  6250
starting iteration  6260
starting iteration  6270
starting iteration  6280
starting iteration  6290
starting iteration  6300
starting iteration  6310
starting iteration  6320
starting iteration  6330
starting iteration  6340
starting iteration  6350
starting iteration  6360
Finished epoch 13 / 20: cost 1.403796, train: 0.542000, val 0.523000, lr
2.566710e-05
starting iteration  6370
starting iteration  6380
starting iteration  6390
starting iteration  6400
starting iteration  6410
starting iteration  6420
starting iteration  6430
starting iteration  6440
starting iteration  6450
starting iteration  6460
starting iteration  6470
starting iteration  6480
starting iteration  6490
starting iteration  6500
starting iteration  6510
starting iteration  6520
starting iteration  6530
```

```
starting iteration  6540
starting iteration  6550
starting iteration  6560
starting iteration  6570
starting iteration  6580
starting iteration  6590
starting iteration  6600
starting iteration  6610
starting iteration  6620
starting iteration  6630
starting iteration  6640
starting iteration  6650
starting iteration  6660
starting iteration  6670
starting iteration  6680
starting iteration  6690
starting iteration  6700
starting iteration  6710
starting iteration  6720
starting iteration  6730
starting iteration  6740
starting iteration  6750
starting iteration  6760
starting iteration  6770
starting iteration  6780
starting iteration  6790
starting iteration  6800
starting iteration  6810
starting iteration  6820
starting iteration  6830
starting iteration  6840
starting iteration  6850
Finished epoch 14 / 20: cost 1.512430, train: 0.603000, val 0.525000, lr
2.438375e-05
starting iteration  6860
starting iteration  6870
starting iteration  6880
starting iteration  6890
starting iteration  6900
starting iteration  6910
starting iteration  6920
starting iteration  6930
starting iteration  6940
starting iteration  6950
starting iteration  6960
starting iteration  6970
starting iteration  6980
starting iteration  6990
```

```
starting iteration  7000
starting iteration  7010
starting iteration  7020
starting iteration  7030
starting iteration  7040
starting iteration  7050
starting iteration  7060
starting iteration  7070
starting iteration  7080
starting iteration  7090
starting iteration  7100
starting iteration  7110
starting iteration  7120
starting iteration  7130
starting iteration  7140
starting iteration  7150
starting iteration  7160
starting iteration  7170
starting iteration  7180
starting iteration  7190
starting iteration  7200
starting iteration  7210
starting iteration  7220
starting iteration  7230
starting iteration  7240
starting iteration  7250
starting iteration  7260
starting iteration  7270
starting iteration  7280
starting iteration  7290
starting iteration  7300
starting iteration  7310
starting iteration  7320
starting iteration  7330
starting iteration  7340
Finished epoch 15 / 20: cost 1.381355, train: 0.591000, val 0.541000, lr
2.316456e-05
starting iteration  7350
starting iteration  7360
starting iteration  7370
starting iteration  7380
starting iteration  7390
starting iteration  7400
starting iteration  7410
starting iteration  7420
starting iteration  7430
starting iteration  7440
starting iteration  7450
```

```
starting iteration  7460
starting iteration  7470
starting iteration  7480
starting iteration  7490
starting iteration  7500
starting iteration  7510
starting iteration  7520
starting iteration  7530
starting iteration  7540
starting iteration  7550
starting iteration  7560
starting iteration  7570
starting iteration  7580
starting iteration  7590
starting iteration  7600
starting iteration  7610
starting iteration  7620
starting iteration  7630
starting iteration  7640
starting iteration  7650
starting iteration  7660
starting iteration  7670
starting iteration  7680
starting iteration  7690
starting iteration  7700
starting iteration  7710
starting iteration  7720
starting iteration  7730
starting iteration  7740
starting iteration  7750
starting iteration  7760
starting iteration  7770
starting iteration  7780
starting iteration  7790
starting iteration  7800
starting iteration  7810
starting iteration  7820
starting iteration  7830
Finished epoch 16 / 20: cost 1.410587, train: 0.602000, val 0.540000, lr
2.200633e-05
starting iteration  7840
starting iteration  7850
starting iteration  7860
starting iteration  7870
starting iteration  7880
starting iteration  7890
starting iteration  7900
starting iteration  7910
```

```
starting iteration  7920
starting iteration  7930
starting iteration  7940
starting iteration  7950
starting iteration  7960
starting iteration  7970
starting iteration  7980
starting iteration  7990
starting iteration  8000
starting iteration  8010
starting iteration  8020
starting iteration  8030
starting iteration  8040
starting iteration  8050
starting iteration  8060
starting iteration  8070
starting iteration  8080
starting iteration  8090
starting iteration  8100
starting iteration  8110
starting iteration  8120
starting iteration  8130
starting iteration  8140
starting iteration  8150
starting iteration  8160
starting iteration  8170
starting iteration  8180
starting iteration  8190
starting iteration  8200
starting iteration  8210
starting iteration  8220
starting iteration  8230
starting iteration  8240
starting iteration  8250
starting iteration  8260
starting iteration  8270
starting iteration  8280
starting iteration  8290
starting iteration  8300
starting iteration  8310
starting iteration  8320
Finished epoch 17 / 20: cost 1.482933, train: 0.619000, val 0.549000, lr
2.090602e-05
starting iteration  8330
starting iteration  8340
starting iteration  8350
starting iteration  8360
starting iteration  8370
```

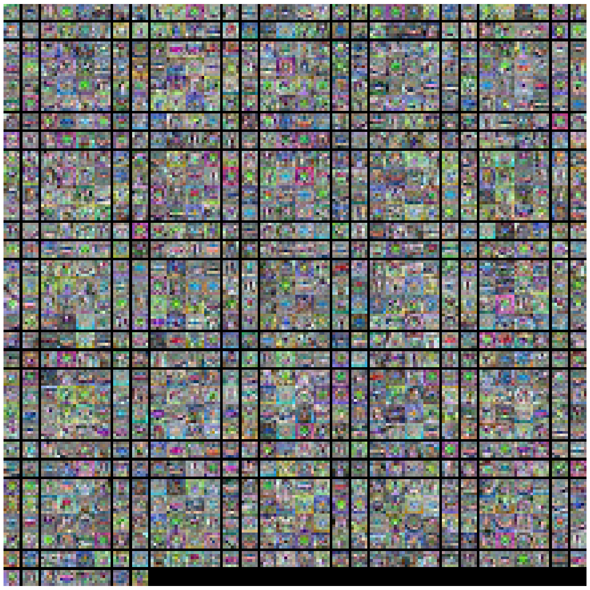
```
starting iteration  8380
starting iteration  8390
starting iteration  8400
starting iteration  8410
starting iteration  8420
starting iteration  8430
starting iteration  8440
starting iteration  8450
starting iteration  8460
starting iteration  8470
starting iteration  8480
starting iteration  8490
starting iteration  8500
starting iteration  8510
starting iteration  8520
starting iteration  8530
starting iteration  8540
starting iteration  8550
starting iteration  8560
starting iteration  8570
starting iteration  8580
starting iteration  8590
starting iteration  8600
starting iteration  8610
starting iteration  8620
starting iteration  8630
starting iteration  8640
starting iteration  8650
starting iteration  8660
starting iteration  8670
starting iteration  8680
starting iteration  8690
starting iteration  8700
starting iteration  8710
starting iteration  8720
starting iteration  8730
starting iteration  8740
starting iteration  8750
starting iteration  8760
starting iteration  8770
starting iteration  8780
starting iteration  8790
starting iteration  8800
starting iteration  8810
Finished epoch 18 / 20: cost 1.184236, train: 0.616000, val 0.528000, lr
1.986072e-05
starting iteration  8820
starting iteration  8830
```

```
starting iteration  8840
starting iteration  8850
starting iteration  8860
starting iteration  8870
starting iteration  8880
starting iteration  8890
starting iteration  8900
starting iteration  8910
starting iteration  8920
starting iteration  8930
starting iteration  8940
starting iteration  8950
starting iteration  8960
starting iteration  8970
starting iteration  8980
starting iteration  8990
starting iteration  9000
starting iteration  9010
starting iteration  9020
starting iteration  9030
starting iteration  9040
starting iteration  9050
starting iteration  9060
starting iteration  9070
starting iteration  9080
starting iteration  9090
starting iteration  9100
starting iteration  9110
starting iteration  9120
starting iteration  9130
starting iteration  9140
starting iteration  9150
starting iteration  9160
starting iteration  9170
starting iteration  9180
starting iteration  9190
starting iteration  9200
starting iteration  9210
starting iteration  9220
starting iteration  9230
starting iteration  9240
starting iteration  9250
starting iteration  9260
starting iteration  9270
starting iteration  9280
starting iteration  9290
starting iteration  9300
Finished epoch 19 / 20: cost 1.375873, train: 0.629000, val 0.545000, lr
```

```
1.886768e-05
starting iteration  9310
starting iteration  9320
starting iteration  9330
starting iteration  9340
starting iteration  9350
starting iteration  9360
starting iteration  9370
starting iteration  9380
starting iteration  9390
starting iteration  9400
starting iteration  9410
starting iteration  9420
starting iteration  9430
starting iteration  9440
starting iteration  9450
starting iteration  9460
starting iteration  9470
starting iteration  9480
starting iteration  9490
starting iteration  9500
starting iteration  9510
starting iteration  9520
starting iteration  9530
starting iteration  9540
starting iteration  9550
starting iteration  9560
starting iteration  9570
starting iteration  9580
starting iteration  9590
starting iteration  9600
starting iteration  9610
starting iteration  9620
starting iteration  9630
starting iteration  9640
starting iteration  9650
starting iteration  9660
starting iteration  9670
starting iteration  9680
starting iteration  9690
starting iteration  9700
starting iteration  9710
starting iteration  9720
starting iteration  9730
starting iteration  9740
starting iteration  9750
starting iteration  9760
starting iteration  9770
```

```
starting iteration  9780
starting iteration  9790
Finished epoch 20 / 20: cost 1.288014, train: 0.616000, val 0.535000, lr
1.792430e-05
finished optimization. best validation accuracy: 0.549000
```

[53]:
```python
# visualize the weights
show_net_weights(best_model)
```



10 Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set.

[54]:
```python
scores_test = two_layer_net(X_test, best_model)
print('Test accuracy: ', np.mean(np.argmax(scores_test, axis=1) == y_test))
```

```
Test accuracy:  0.536
```

[ ]:
```python

```

layers

September 24, 2019

1 Modular neural nets

In the previous exercise, we computed the loss and gradient for a two-layer neural network in a single monolithic function. This isn't very difficult for a small two-layer network, but would be tedious and error-prone for larger networks. Ideally we want to build networks using a more modular design so that we can snap together different types of layers and loss functions in order to quickly experiment with different architectures.

In this exercise we will implement this approach, and develop a number of different layer types in isolation that can then be easily plugged together. For each layer we will implement `forward` and `backward` functions. The `forward` function will receive data, weights, and other parameters, and will return both an output and a `cache` object that stores data needed for the backward pass. The `backward` function will recieve upstream derivatives and the cache object, and will return gradients with respect to the data and all of the weights. This will allow us to write code that looks like this:

```python
def two_layer_net(X, W1, b1, W2, b2, reg):
    # Forward pass; compute scores
    s1, fc1_cache = affine_forward(X, W1, b1)
    a1, relu_cache = relu_forward(s1)
    scores, fc2_cache = affine_forward(a1, W2, b2)

    # Loss functions return data loss and gradients on scores
    data_loss, dscores = svm_loss(scores, y)

    # Compute backward pass
    da1, dW2, db2 = affine_backward(dscores, fc2_cache)
    ds1 = relu_backward(da1, relu_cache)
    dX, dW1, db1 = affine_backward(ds1, fc1_cache)

    # A real network would add regularization here

    # Return loss and gradients
    return loss, dW1, db1, dW2, db2
```

[67]:
```python
# As usual, a bit of setup

import numpy as np
import matplotlib.pyplot as plt
```

```
from cs231n.gradient_check import eval_numerical_gradient_array,␣
↪eval_numerical_gradient
from cs231n.layers import *

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/
↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

2 Affine layer: forward

Open the file `cs231n/layers.py` and implement the `affine_forward` function.

Once you are done we will test your can test your implementation by running the following:

[68]:
```
# Test the affine_forward function

num_inputs = 2
input_shape = (4, 5, 6)
output_dim = 3

input_size = num_inputs * np.prod(input_shape)
weight_size = output_dim * np.prod(input_shape)

x = np.linspace(-0.1, 0.5, num=input_size).reshape(num_inputs, *input_shape)
w = np.linspace(-0.2, 0.3, num=weight_size).reshape(np.prod(input_shape),␣
↪output_dim)
b = np.linspace(-0.3, 0.1, num=output_dim)

out, _ = affine_forward(x, w, b)
correct_out = np.array([[ 1.49834967,  1.70660132,  1.91485297],
                        [ 3.25553199,  3.5141327,   3.77273342]])

# Compare your output with ours. The error should be around 1e-9.
```

```python
print('Testing affine_forward function:')
print('difference: ', rel_error(out, correct_out))
```

```
Testing affine_forward function:
difference:  9.769847728806635e-10
```

3 Affine layer: backward

Now implement the `affine_backward` function. You can test your implementation using numeric gradient checking.

[69]:
```python
# Test the affine_backward function

x = np.random.randn(10, 2, 3)
w = np.random.randn(6, 5)
b = np.random.randn(5)
dout = np.random.randn(10, 5)

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

_, cache = affine_forward(x, w, b)
dx, dw, db = affine_backward(dout, cache)

# The error should be less than 1e-10
print('Testing affine_backward function:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing affine_backward function:
dx error:  1.389772357243565e-10
dw error:  2.0802769210228722e-09
db error:  3.283181149785737e-11
```

4 ReLU layer: forward

Implement the `relu_forward` function and test your implementation by running the following:

[70]:
```python
# Test the relu_forward function

x = np.linspace(-0.5, 0.5, num=12).reshape(3, 4)
```

```
out, _ = relu_forward(x)
correct_out = np.array([[ 0.,          0.,          0.,          0.,         ],
                        [ 0.,          0.,          0.04545455,  0.13636364,],
                        [ 0.22727273,  0.31818182,  0.40909091,  0.5,        ]])

# Compare your output with ours. The error should be around 1e-8
print('Testing relu_forward function:')
print('difference: ', rel_error(out, correct_out))
```

```
Testing relu_forward function:
difference:  4.999999798022158e-08
```

5 ReLU layer: backward

Implement the `relu_backward` function and test your implementation using numeric gradient checking:

[71]:
```
x = np.random.randn(10, 10)
dout = np.random.randn(*x.shape)

dx_num = eval_numerical_gradient_array(lambda x: relu_forward(x)[0], x, dout)

_, cache = relu_forward(x)
dx = relu_backward(dout, cache)

# The error should be around 1e-12
print('Testing relu_backward function:')
print('dx error: ', rel_error(dx_num, dx))
```

```
Testing relu_backward function:
dx error:  3.2756297249955374e-12
```

6 Loss layers: Softmax and SVM

You implemented these loss functions in the last assignment, so we'll give them to you for free here. It's still a good idea to test them to make sure they work correctly.

[72]:
```
num_classes, num_inputs = 10, 50
x = 0.001 * np.random.randn(num_inputs, num_classes)
y = np.random.randint(num_classes, size=num_inputs)

dx_num = eval_numerical_gradient(lambda x: svm_loss(x, y)[0], x, verbose=False)
loss, dx = svm_loss(x, y)

# Test svm_loss function. Loss should be around 9 and dx error should be 1e-9
print('Testing svm_loss:')
print('loss: ', loss)
```

```
print('dx error: ', rel_error(dx_num, dx))

dx_num = eval_numerical_gradient(lambda x: softmax_loss(x, y)[0], x,␣
↪verbose=False)
loss, dx = softmax_loss(x, y)

# Test softmax_loss function. Loss should be 2.3 and dx error should be 1e-8
print('\nTesting softmax_loss:')
print('loss: ', loss)
print('dx error: ', rel_error(dx_num, dx))
```

```
Testing svm_loss:
loss:  8.99959980158041
dx error:  1.4021566006651672e-09

Testing softmax_loss:
loss:  2.302545580559075
dx error:  1.002114418846665e-08
```

7 Convolution layer: forward naive

We are now ready to implement the forward pass for a convolutional layer. Implement the function `conv_forward_naive` in the file `cs231n/layers.py`.

You don't have to worry too much about efficiency at this point; just write the code in whatever way you find most clear.

You can test your implementation by running the following:

[73]:
```
x_shape = (2, 3, 4, 4)
w_shape = (3, 3, 4, 4)
x = np.linspace(-0.1, 0.5, num=np.prod(x_shape)).reshape(x_shape)
w = np.linspace(-0.2, 0.3, num=np.prod(w_shape)).reshape(w_shape)
b = np.linspace(-0.1, 0.2, num=3)

conv_param = {'stride': 2, 'pad': 1}
out, _ = conv_forward_naive(x, w, b, conv_param)
correct_out = np.array([[[[-0.08759809, -0.10987781],
                           [-0.18387192, -0.2109216 ]],
                          [[ 0.21027089,  0.21661097],
                           [ 0.22847626,  0.23004637]],
                          [[ 0.50813986,  0.54309974],
                           [ 0.64082444,  0.67101435]]],
                         [[[-0.98053589, -1.03143541],
                           [-1.19128892, -1.24695841]],
                          [[ 0.69108355,  0.66880383],
                           [ 0.59480972,  0.56776003]],
                          [[ 2.36270298,  2.36904306],
```

```
                                [ 2.38090835,  2.38247847]]]]])

# Compare your output to ours; difference should be around 1e-8
print('Testing conv_forward_naive')
print('difference: ', rel_error(out, correct_out))
```

```
Testing conv_forward_naive
difference:  2.2121476417505994e-08
```

8 Aside: Image processing via convolutions

As fun way to both check your implementation and gain a better understanding of the type of operation that convolutional layers can perform, we will set up an input containing two images and manually set up filters that perform common image processing operations (grayscale conversion and edge detection). The convolution forward pass will apply these operations to each of the input images. We can then visualize the results as a sanity check.

[74]:
```
from scipy.misc import imread, imresize

kitten, puppy = imread('kitten.jpg'), imread('puppy.jpg')
# kitten is wide, and puppy is already square
d = kitten.shape[1] - kitten.shape[0]
kitten_cropped = kitten[:, d//2:-d//2, :]

img_size = 200   # Make this smaller if it runs too slow
x = np.zeros((2, 3, img_size, img_size))
x[0, :, :, :] = imresize(puppy, (img_size, img_size)).transpose((2, 0, 1))
x[1, :, :, :] = imresize(kitten_cropped, (img_size, img_size)).transpose((2, 0,␣
 ↪1))

# Set up a convolutional weights holding 2 filters, each 3x3
w = np.zeros((2, 3, 3, 3))

# The first filter converts the image to grayscale.
# Set up the red, green, and blue channels of the filter.
w[0, 0, :, :] = [[0, 0, 0], [0, 0.3, 0], [0, 0, 0]]
w[0, 1, :, :] = [[0, 0, 0], [0, 0.6, 0], [0, 0, 0]]
w[0, 2, :, :] = [[0, 0, 0], [0, 0.1, 0], [0, 0, 0]]

# Second filter detects horizontal edges in the blue channel.
w[1, 2, :, :] = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]

# Vector of biases. We don't need any bias for the grayscale
# filter, but for the edge detection filter we want to add 128
# to each output so that nothing is negative.
b = np.array([0, 128])
```

```python
# Compute the result of convolving each input in x with each filter in w,
# offsetting by b, and storing the results in out.
out, _ = conv_forward_naive(x, w, b, {'stride': 1, 'pad': 1})

def imshow_noax(img, normalize=True):
    """ Tiny helper to show images as uint8 and remove axis labels """
    if normalize:
        img_max, img_min = np.max(img), np.min(img)
        img = 255.0 * (img - img_min) / (img_max - img_min)
    plt.imshow(img.astype('uint8'))
    plt.gca().axis('off')

# Show the original images and the results of the conv operation
plt.subplot(2, 3, 1)
imshow_noax(puppy, normalize=False)
plt.title('Original image')
plt.subplot(2, 3, 2)
imshow_noax(out[0, 0])
plt.title('Grayscale')
plt.subplot(2, 3, 3)
imshow_noax(out[0, 1])
plt.title('Edges')
plt.subplot(2, 3, 4)
imshow_noax(kitten_cropped, normalize=False)
plt.subplot(2, 3, 5)
imshow_noax(out[1, 0])
plt.subplot(2, 3, 6)
imshow_noax(out[1, 1])
plt.show()
```

```
       ␣
↪---------------------------------------------------------------------------

       ImportError                               Traceback (most recent call␣
↪last)

       <ipython-input-74-92a8b739741c> in <module>
   ----> 1 from scipy.misc import imread, imresize
         2
         3 kitten, puppy = imread('kitten.jpg'), imread('puppy.jpg')
         4 # kitten is wide, and puppy is already square
         5 d = kitten.shape[1] - kitten.shape[0]


       ImportError: cannot import name 'imread'
```

9 Convolution layer: backward naive

Next you need to implement the function `conv_backward_naive` in the file `cs231n/layers.py`. As usual, we will check your implementation with numeric gradient checking.

[78]:
```
x = np.random.randn(4, 3, 5, 5)
w = np.random.randn(2, 3, 3, 3)
b = np.random.randn(2,)
dout = np.random.randn(4, 2, 5, 5)
conv_param = {'stride': 1, 'pad': 1}

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

out, cache = conv_forward_naive(x, w, b, conv_param)
dx, dw, db = conv_backward_naive(dout, cache)

# Your errors should be around 1e-9'
print('Testing conv_backward_naive function')
print('dx error: ', rel_error(dx, dx_num))
print('dw error: ', rel_error(dw, dw_num))
print('db error: ', rel_error(db, db_num))
```

```
Testing conv_backward_naive function
dx error:  2.6599967595550446e-09
dw error:  2.2541074080896546e-10
db error:  9.5467474823066e-12
```

10 Max pooling layer: forward naive

The last layer we need for a basic convolutional neural network is the max pooling layer. First implement the forward pass in the function `max_pool_forward_naive` in the file `cs231n/layers.py`.

[79]:
```
x_shape = (2, 3, 4, 4)
x = np.linspace(-0.3, 0.4, num=np.prod(x_shape)).reshape(x_shape)
pool_param = {'pool_width': 2, 'pool_height': 2, 'stride': 2}

out, _ = max_pool_forward_naive(x, pool_param)

correct_out = np.array([[[[-0.26315789, -0.24842105],
                          [-0.20421053, -0.18947368]],
                         [[-0.14526316, -0.13052632],
                          [-0.08631579, -0.07157895]],
```

```
                                 [[-0.02736842, -0.01263158],
                                  [ 0.03157895,  0.04631579]]],
                                [[[ 0.09052632,  0.10526316],
                                  [ 0.14947368,  0.16421053]],
                                 [[ 0.20842105,  0.22315789],
                                  [ 0.26736842,  0.28210526]],
                                 [[ 0.32631579,  0.34105263],
                                  [ 0.38526316,  0.4       ]]]])

# Compare your output with ours. Difference should be around 1e-8.
print('Testing max_pool_forward_naive function:')
print('difference: ', rel_error(out, correct_out))
```

```
Testing max_pool_forward_naive function:
difference:  4.1666665157267834e-08
```

11 Max pooling layer: backward naive

Implement the backward pass for a max pooling layer in the function `max_pool_backward_naive` in the file `cs231n/layers.py`. As always we check the correctness of the backward pass using numerical gradient checking.

[80]:
```
x = np.random.randn(3, 2, 8, 8)
dout = np.random.randn(3, 2, 4, 4)
pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}

dx_num = eval_numerical_gradient_array(lambda x: max_pool_forward_naive(x,␣
 ↪pool_param)[0], x, dout)

out, cache = max_pool_forward_naive(x, pool_param)
dx = max_pool_backward_naive(dout, cache)

# Your error should be around 1e-12
print('Testing max_pool_backward_naive function:')
print('dx error: ', rel_error(dx, dx_num))
```

```
Testing max_pool_backward_naive function:
dx error:  3.275642258527761e-12
```

12 Fast layers

Making convolution and pooling layers fast can be challenging. To spare you the pain, we've provided fast implementations of the forward and backward passes for convolution and pooling layers in the file `cs231n/fast_layers.py`.

The fast convolution implementation depends on a Cython extension; to compile it you need to run the following from the `cs231n` directory:

python setup.py build_ext --inplace

The API for the fast versions of the convolution and pooling layers is exactly the same as the naive versions that you implemented above: the forward pass receives data, weights, and parameters and produces outputs and a cache object; the backward pass recieves upstream derivatives and the cache object and produces gradients with respect to the data and weights.

NOTE: The fast implementation for pooling will only perform optimally if the pooling regions are non-overlapping and tile the input. If these conditions are not met then the fast pooling implementation will not be much faster than the naive implementation.

You can compare the performance of the naive and fast versions of these layers by running the following:

[81]:
```
from cs231n.fast_layers import conv_forward_fast, conv_backward_fast
from time import time

x = np.random.randn(100, 3, 31, 31)
w = np.random.randn(25, 3, 3, 3)
b = np.random.randn(25,)
dout = np.random.randn(100, 25, 16, 16)
conv_param = {'stride': 2, 'pad': 1}

t0 = time()
out_naive, cache_naive = conv_forward_naive(x, w, b, conv_param)
t1 = time()
out_fast, cache_fast = conv_forward_fast(x, w, b, conv_param)
t2 = time()

print('Testing conv_forward_fast:')
print('Naive: %fs' % (t1 - t0))
print('Fast: %fs' % (t2 - t1))
print('Speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('Difference: ', rel_error(out_naive, out_fast))

t0 = time()
dx_naive, dw_naive, db_naive = conv_backward_naive(dout, cache_naive)
t1 = time()
dx_fast, dw_fast, db_fast = conv_backward_fast(dout, cache_fast)
t2 = time()

print('\nTesting conv_backward_fast:')
print('Naive: %fs' % (t1 - t0))
print('Fast: %fs' % (t2 - t1))
print('Speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('dx difference: ', rel_error(dx_naive, dx_fast))
print('dw difference: ', rel_error(dw_naive, dw_fast))
print('db difference: ', rel_error(db_naive, db_fast))
```

```
Testing conv_forward_fast:
Naive: 4.739521s
Fast: 0.008533s
Speedup: 555.434255x
Difference:  7.053358279005642e-11

Testing conv_backward_fast:
Naive: 6.787412s
Fast: 0.011820s
Speedup: 574.238946x
dx difference:  5.415099860467299e-12
dw difference:  1.3570073863477166e-12
db difference:  2.3571176246445243e-14
```

[82]:

```
from cs231n.fast_layers import max_pool_forward_fast, max_pool_backward_fast

x = np.random.randn(100, 3, 32, 32)
dout = np.random.randn(100, 3, 16, 16)
pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}

t0 = time()
out_naive, cache_naive = max_pool_forward_naive(x, pool_param)
t1 = time()
out_fast, cache_fast = max_pool_forward_fast(x, pool_param)
t2 = time()

print('Testing pool_forward_fast:')
print('Naive: %fs' % (t1 - t0))
print('fast: %fs' % (t2 - t1))
print('speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('difference: ', rel_error(out_naive, out_fast))

t0 = time()
dx_naive = max_pool_backward_naive(dout, cache_naive)
t1 = time()
dx_fast = max_pool_backward_fast(dout, cache_fast)
t2 = time()

print('\nTesting pool_backward_fast:')
print('Naive: %fs' % (t1 - t0))
print('speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('dx difference: ', rel_error(dx_naive, dx_fast))
```

```
Testing pool_forward_fast:
Naive: 0.354986s
fast: 0.001795s
speedup: 197.757870x
difference:  0.0
```

```
Testing pool_backward_fast:
Naive: 0.366232s
speedup: 39.574572x
dx difference:  0.0
```

13 Sandwich layers

There are a couple common layer "sandwiches" that frequently appear in ConvNets. For example convolutional layers are frequently followed by ReLU and pooling, and affine layers are frequently followed by ReLU. To make it more convenient to use these common patterns, we have defined several convenience layers in the file `cs231n/layer_utils.py`. Lets grad-check them to make sure that they work correctly:

[83]:
```
from cs231n.layer_utils import conv_relu_pool_forward, conv_relu_pool_backward

x = np.random.randn(2, 3, 16, 16)
w = np.random.randn(3, 3, 3, 3)
b = np.random.randn(3,)
dout = np.random.randn(2, 3, 8, 8)
conv_param = {'stride': 1, 'pad': 1}
pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}

out, cache = conv_relu_pool_forward(x, w, b, conv_param, pool_param)
dx, dw, db = conv_relu_pool_backward(dout, cache)

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

print('Testing conv_relu_pool_forward:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing conv_relu_pool_forward:
dx error:  6.702093662796137e-09
dw error:  1.5446117758365741e-09
db error:  2.0456884116571853e-11
```

[84]:
```
from cs231n.layer_utils import conv_relu_forward, conv_relu_backward

x = np.random.randn(2, 3, 8, 8)
w = np.random.randn(3, 3, 3, 3)
```

```
b = np.random.randn(3,)
dout = np.random.randn(2, 3, 8, 8)
conv_param = {'stride': 1, 'pad': 1}

out, cache = conv_relu_forward(x, w, b, conv_param)
dx, dw, db = conv_relu_backward(dout, cache)

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

print('Testing conv_relu_forward:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing conv_relu_forward:
dx error:  2.0945883679826754e-08
dw error:  4.214221674271771e-09
db error:  6.630038270484846e-12
```

[85]:
```
from cs231n.layer_utils import affine_relu_forward, affine_relu_backward

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)

print('Testing affine_relu_forward:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing affine_relu_forward:
```

```
dx error:  3.141513976758779e-10
dw error:  6.355226245075724e-10
db error:  3.2755971928120225e-12
```

[ ]:

```

```

convnet

September 24, 2019

1 Train a ConvNet!

We now have a generic solver and a bunch of modularized layers. It's time to put it all together, and train a ConvNet to recognize the classes in CIFAR-10. In this notebook we will walk you through training a simple two-layer ConvNet and then set you free to build the best net that you can to perform well on CIFAR-10.

Open up the file `cs231n/classifiers/convnet.py`; you will see that the `two_layer_convnet` function computes the loss and gradients for a two-layer ConvNet. Note that this function uses the "sandwich" layers defined in `cs231n/layer_utils.py`.

[1]:
```
# As usual, a bit of setup

import numpy as np
import matplotlib.pyplot as plt
from cs231n.classifier_trainer import ClassifierTrainer
from cs231n.gradient_check import eval_numerical_gradient
from cs231n.classifiers.convnet import *

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

[2]:
```
from cs231n.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    """
```

```python
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the two-layer neural net classifier. These are the same steps as
    we used for the SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = range(num_training, num_training + num_validation)
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = range(num_training)
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = range(num_test)
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Transpose so that channels come first
    X_train = X_train.transpose(0, 3, 1, 2).copy()
    X_val = X_val.transpose(0, 3, 1, 2).copy()
    x_test = X_test.transpose(0, 3, 1, 2).copy()

    return X_train, y_train, X_val, y_val, X_test, y_test


# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 3, 32, 32)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3, 32, 32)
Validation labels shape:  (1000,)
Test data shape:  (1000, 32, 32, 3)
```

```
Test labels shape:  (1000,)
```

2 Sanity check loss

After you build a new network, one of the first things you should do is sanity check the loss. When we use the softmax loss, we expect the loss for random weights (and no regularization) to be about `log(C)` for `C` classes. When we add regularization this should go up.

[3]:
```
model = init_two_layer_convnet()

X = np.random.randn(100, 3, 32, 32)
y = np.random.randint(10, size=100)

loss, _ = two_layer_convnet(X, model, y, reg=0)

# Sanity check: Loss should be about log(10) = 2.3026
print('Sanity check loss (no regularization): ', loss)

# Sanity check: Loss should go up when you add regularization
loss, _ = two_layer_convnet(X, model, y, reg=1)
print('Sanity check loss (with regularization): ', loss)
```

```
Sanity check loss (no regularization):  2.302678587918066
Sanity check loss (with regularization):  2.3447926824948357
```

3 Gradient check

After the loss looks reasonable, you should always use numeric gradient checking to make sure that your backward pass is correct. When you use numeric gradient checking you should use a small amount of artifical data and a small number of neurons at each layer.

[4]:
```
num_inputs = 2
input_shape = (3, 16, 16)
reg = 0.0
num_classes = 10
X = np.random.randn(num_inputs, *input_shape)
y = np.random.randint(num_classes, size=num_inputs)

model = init_two_layer_convnet(num_filters=3, filter_size=3,␣
 ↪input_shape=input_shape)
loss, grads = two_layer_convnet(X, model, y)
for param_name in sorted(grads):
    f = lambda _: two_layer_convnet(X, model, y)[0]
    param_grad_num = eval_numerical_gradient(f, model[param_name],␣
 ↪verbose=False, h=1e-6)
    e = rel_error(param_grad_num, grads[param_name])
```

```
    print('%s max relative error: %e' % (param_name, rel_error(param_grad_num,␣
↪grads[param_name])))
```

```
W1 max relative error: 5.290520e-06
W2 max relative error: 1.848474e-05
b1 max relative error: 5.200847e-08
b2 max relative error: 6.433921e-10
```

4 Overfit small data

A nice trick is to train your model with just a few training samples. You should be able to overfit small datasets, which will result in very high training accuracy and comparatively low validation accuracy.

[5]:
```
# Use a two-layer ConvNet to overfit 50 training examples.

model = init_two_layer_convnet()
trainer = ClassifierTrainer()
best_model, loss_history, train_acc_history, val_acc_history = trainer.train(
          X_train[:50], y_train[:50], X_val, y_val, model, two_layer_convnet,
          reg=0.001, momentum=0.9, learning_rate=0.0001, batch_size=10,␣
↪num_epochs=10,
          verbose=True)
```

```
starting iteration  0
Finished epoch 0 / 10: cost 2.292191, train: 0.120000, val 0.118000, lr
1.000000e-04
Finished epoch 1 / 10: cost 2.201300, train: 0.260000, val 0.098000, lr
9.500000e-05
Finished epoch 2 / 10: cost 1.918935, train: 0.280000, val 0.096000, lr
9.025000e-05
starting iteration  10
Finished epoch 3 / 10: cost 1.232655, train: 0.400000, val 0.166000, lr
8.573750e-05
Finished epoch 4 / 10: cost 1.088217, train: 0.560000, val 0.156000, lr
8.145062e-05
starting iteration  20
Finished epoch 5 / 10: cost 0.828684, train: 0.680000, val 0.189000, lr
7.737809e-05
Finished epoch 6 / 10: cost 1.657847, train: 0.740000, val 0.166000, lr
7.350919e-05
starting iteration  30
Finished epoch 7 / 10: cost 0.608962, train: 0.860000, val 0.177000, lr
6.983373e-05
Finished epoch 8 / 10: cost 0.350635, train: 0.900000, val 0.146000, lr
6.634204e-05
starting iteration  40
```

```
Finished epoch 9 / 10: cost 0.259157, train: 0.900000, val 0.167000, lr
6.302494e-05
Finished epoch 10 / 10: cost 0.797573, train: 0.880000, val 0.161000, lr
5.987369e-05
finished optimization. best validation accuracy: 0.189000
```

Plotting the loss, training accuracy, and validation accuracy should show clear overfitting:

[ ]:
```
plt.subplot(2, 1, 1)
plt.plot(loss_history)
plt.xlabel('iteration')
plt.ylabel('loss')

plt.subplot(2, 1, 2)
plt.plot(train_acc_history)
plt.plot(val_acc_history)
plt.legend(['train', 'val'], loc='upper left')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```

5 Train the net

Once the above works, training the net is the next thing to try. You can set the `acc_frequency` parameter to change the frequency at which the training and validation set accuracies are tested. If your parameters are set properly, you should see the training and validation accuracy start to improve within a hundred iterations, and you should be able to train a reasonable model with just one epoch.

Using the parameters below you should be able to get around 50% accuracy on the validation set.

[6]:
```
model = init_two_layer_convnet(filter_size=7)
trainer = ClassifierTrainer()
best_model, loss_history, train_acc_history, val_acc_history = trainer.train(
          X_train, y_train, X_val, y_val, model, two_layer_convnet,
          reg=0.001, momentum=0.9, learning_rate=0.0001, batch_size=50,␣
 ↪num_epochs=1,
          acc_frequency=50, verbose=True)
```

```
starting iteration  0
Finished epoch 0 / 1: cost 2.326596, train: 0.115000, val 0.101000, lr
1.000000e-04
starting iteration  10
starting iteration  20
starting iteration  30
starting iteration  40
starting iteration  50
Finished epoch 0 / 1: cost 1.722926, train: 0.341000, val 0.349000, lr
```

```
1.000000e-04
starting iteration  60
starting iteration  70
starting iteration  80
starting iteration  90
starting iteration  100
Finished epoch 0 / 1: cost 1.792310, train: 0.334000, val 0.338000, lr
1.000000e-04
starting iteration  110
starting iteration  120
starting iteration  130
starting iteration  140
starting iteration  150
Finished epoch 0 / 1: cost 1.840112, train: 0.395000, val 0.363000, lr
1.000000e-04
starting iteration  160
starting iteration  170
starting iteration  180
starting iteration  190
starting iteration  200
Finished epoch 0 / 1: cost 1.764864, train: 0.379000, val 0.410000, lr
1.000000e-04
starting iteration  210
starting iteration  220
starting iteration  230
starting iteration  240
starting iteration  250
Finished epoch 0 / 1: cost 1.769096, train: 0.427000, val 0.419000, lr
1.000000e-04
starting iteration  260
starting iteration  270
starting iteration  280
starting iteration  290
starting iteration  300
Finished epoch 0 / 1: cost 1.524768, train: 0.435000, val 0.453000, lr
1.000000e-04
starting iteration  310
starting iteration  320
starting iteration  330
starting iteration  340
starting iteration  350
Finished epoch 0 / 1: cost 2.337207, train: 0.399000, val 0.367000, lr
1.000000e-04
starting iteration  360
starting iteration  370
starting iteration  380
starting iteration  390
starting iteration  400
```

```
Finished epoch 0 / 1: cost 1.473121, train: 0.465000, val 0.458000, lr
1.000000e-04
starting iteration  410
starting iteration  420
starting iteration  430
starting iteration  440
starting iteration  450
Finished epoch 0 / 1: cost 1.717805, train: 0.414000, val 0.457000, lr
1.000000e-04
starting iteration  460
starting iteration  470
starting iteration  480
starting iteration  490
starting iteration  500
Finished epoch 0 / 1: cost 1.587554, train: 0.513000, val 0.491000, lr
1.000000e-04
starting iteration  510
starting iteration  520
starting iteration  530
starting iteration  540
starting iteration  550
Finished epoch 0 / 1: cost 1.648589, train: 0.468000, val 0.452000, lr
1.000000e-04
starting iteration  560
starting iteration  570
starting iteration  580
starting iteration  590
starting iteration  600
Finished epoch 0 / 1: cost 1.650349, train: 0.415000, val 0.429000, lr
1.000000e-04
starting iteration  610
starting iteration  620
starting iteration  630
starting iteration  640
starting iteration  650
Finished epoch 0 / 1: cost 1.746694, train: 0.470000, val 0.497000, lr
1.000000e-04
starting iteration  660
starting iteration  670
starting iteration  680
starting iteration  690
starting iteration  700
Finished epoch 0 / 1: cost 1.624673, train: 0.495000, val 0.507000, lr
1.000000e-04
starting iteration  710
starting iteration  720
starting iteration  730
starting iteration  740
```

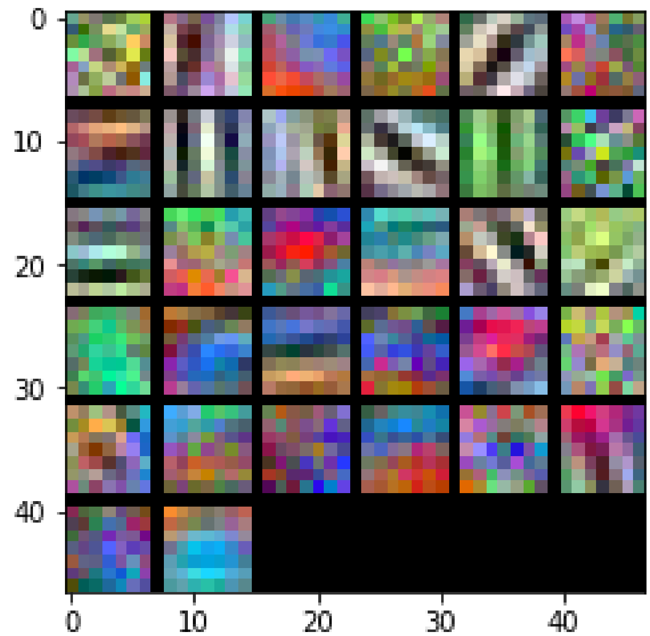
```
starting iteration  750
Finished epoch 0 / 1: cost 1.218479, train: 0.533000, val 0.478000, lr
1.000000e-04
starting iteration  760
starting iteration  770
starting iteration  780
starting iteration  790
starting iteration  800
Finished epoch 0 / 1: cost 1.675719, train: 0.486000, val 0.463000, lr
1.000000e-04
starting iteration  810
starting iteration  820
starting iteration  830
starting iteration  840
starting iteration  850
Finished epoch 0 / 1: cost 1.413279, train: 0.468000, val 0.486000, lr
1.000000e-04
starting iteration  860
starting iteration  870
starting iteration  880
starting iteration  890
starting iteration  900
Finished epoch 0 / 1: cost 1.548007, train: 0.471000, val 0.453000, lr
1.000000e-04
starting iteration  910
starting iteration  920
starting iteration  930
starting iteration  940
starting iteration  950
Finished epoch 0 / 1: cost 2.070196, train: 0.524000, val 0.469000, lr
1.000000e-04
starting iteration  960
starting iteration  970
Finished epoch 1 / 1: cost 1.489284, train: 0.429000, val 0.433000, lr
9.500000e-05
finished optimization. best validation accuracy: 0.507000
```

6 Visualize weights

We can visualize the convolutional weights from the first layer. If everything worked properly, these will usually be edges and blobs of various colors and orientations.

[7]:
```
from cs231n.vis_utils import visualize_grid

grid = visualize_grid(best_model['W1'].transpose(0, 2, 3, 1))
plt.imshow(grid.astype('uint8'))
```

[7]: <matplotlib.image.AxesImage at 0x11ba67e48>



```
[ ]:
```

softmax-classifier

September 26, 2019

0.1 PyTorch data

PyTorch comes with a nice paradigm for dealing with data which we'll use here. A PyTorch `Dataset` knows where to find data in its raw form (files on disk) and how to load individual examples into Python datastructures. A PyTorch `DataLoader` takes a dataset and offers a variety of ways to sample batches from that dataset.

Take a moment to browse through the `CIFAR10 Dataset` in `2_pytorch/cifar10.py`, read the `DataLoader` documentation linked above, and see how these are used in the section of `train.py` that loads data. Note that in the first part of the homework we subtracted a mean CIFAR10 image from every image before feeding it in to our models. Here we subtract a constant color instead. Both methods are seen in practice and work equally well.

PyTorch provides lots of vision datasets which can be imported directly from `torchvision.datasets`. Also see `torchtext` for natural language datasets.

0.2 Softmax Classifier in PyTorch

In PyTorch Deep Learning building blocks are implemented in the neural network module `torch.nn` (usually imported as `nn`). A PyTorch model is typically a subclass of `nn.Module` and thereby gains a multitude of features. Because your logistic regressor is an `nn.Module` all of its parameters and sub-modules are accessible through the `.parameters()` and `.modules()` methods.

Now implement a softmax classifier by filling in the marked sections of `models/softmax.py`.

The main driver for this question is `train.py`. It reads arguments and model hyperparameter from the command line, loads CIFAR10 data and the specified model (in this case, softmax). Using the optimizer initialized with appropriate hyperparameters, it trains the model and reports performance on test data.

Complete the following couple of sections in `train.py`: 1. Initialize an optimizer from the torch.optim package 2. Update the parameters in model using the optimizer initialized above

At this point all of the components required to train the softmax classifer are complete for the softmax classifier. Now run

```
$ run_softmax.sh
```

to train a model and save it to `softmax.pt`. This will also produce a `softmax.log` file which contains training details which we will visualize below.

Note: You may want to adjust the hyperparameters specified in `run_softmax.sh` to get reasonable performance.

0.3 Visualizing the PyTorch model

[6]:
```
# Assuming that you have completed training the classifer, let us plot the␣
↪training loss vs. iteration. This is an
# example to show a simple way to log and plot data from PyTorch.

# we neeed matplotlib to plot the graphs for us!
import matplotlib
# This is needed to save images
matplotlib.use('Agg')
import matplotlib.pyplot as plt
%matplotlib inline
```

```
//anaconda3/envs/cs7643/lib/python3.6/site-packages/ipykernel_launcher.py:7:
UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.

The backend was *originally* set to 'module://ipykernel.pylab.backend_inline' by
the following code:
  File "//anaconda3/envs/cs7643/lib/python3.6/runpy.py", line 193, in
_run_module_as_main
    "__main__", mod_spec)
  File "//anaconda3/envs/cs7643/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel_launcher.py", line 16, in <module>
    app.launch_new_instance()
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/traitlets/config/application.py", line 658, in launch_instance
    app.start()
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel/kernelapp.py", line 563, in start
    self.io_loop.start()
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/tornado/platform/asyncio.py", line 148, in start
    self.asyncio_loop.run_forever()
  File "//anaconda3/envs/cs7643/lib/python3.6/asyncio/base_events.py", line 421,
in run_forever
    self._run_once()
  File "//anaconda3/envs/cs7643/lib/python3.6/asyncio/base_events.py", line
1425, in _run_once
    handle._run()
  File "//anaconda3/envs/cs7643/lib/python3.6/asyncio/events.py", line 127, in
_run
    self._callback(*self._args)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/ioloop.py",
```

```
line 690, in <lambda>
    lambda f: self._run_callback(functools.partial(callback, future))
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/ioloop.py",
line 743, in _run_callback
    ret = callback()
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/gen.py",
line 787, in inner
    self.run()
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/gen.py",
line 748, in run
    yielded = self.gen.send(value)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel/kernelbase.py", line 365, in process_one
    yield gen.maybe_future(dispatch(*args))
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/gen.py",
line 209, in wrapper
    yielded = next(result)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel/kernelbase.py", line 272, in dispatch_shell
    yield gen.maybe_future(handler(stream, idents, msg))
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/gen.py",
line 209, in wrapper
    yielded = next(result)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel/kernelbase.py", line 542, in execute_request
    user_expressions, allow_stdin,
  File "//anaconda3/envs/cs7643/lib/python3.6/site-packages/tornado/gen.py",
line 209, in wrapper
    yielded = next(result)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel/ipkernel.py", line 294, in do_execute
    res = shell.run_cell(code, store_history=store_history, silent=silent)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/ipykernel/zmqshell.py", line 536, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/interactiveshell.py", line 2855, in run_cell
    raw_cell, store_history, silent, shell_futures)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/interactiveshell.py", line 2881, in _run_cell
    return runner(coro)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/async_helpers.py", line 68, in _pseudo_sync_runner
    coro.send(None)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/interactiveshell.py", line 3058, in run_cell_async
    interactivity=interactivity, compiler=compiler, result=result)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
```

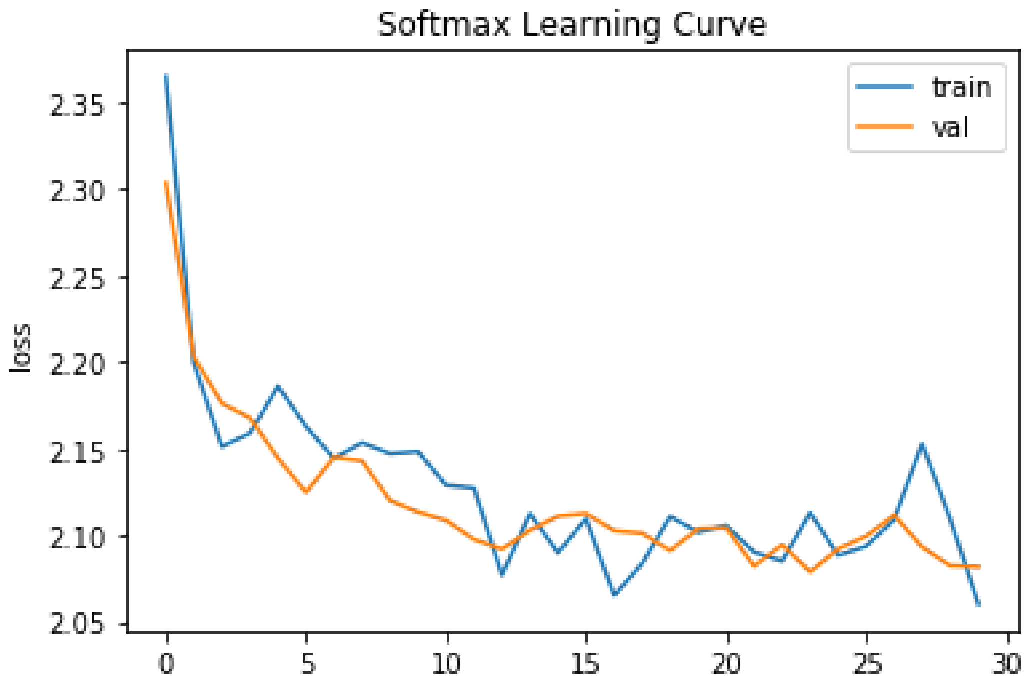
```
packages/IPython/core/interactiveshell.py", line 3249, in run_ast_nodes
    if (await self.run_code(code, result,  async_=asy)):
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/interactiveshell.py", line 3326, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-4-b4795f515c11>", line 9, in <module>
    get_ipython().run_line_magic('matplotlib', 'inline')
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/interactiveshell.py", line 2314, in run_line_magic
    result = fn(*args, **kwargs)
  File "<//anaconda3/envs/cs7643/lib/python3.6/site-
packages/decorator.py:decorator-gen-108>", line 2, in matplotlib
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/magic.py", line 187, in <lambda>
    call = lambda f, *a, **k: f(*a, **k)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/magics/pylab.py", line 99, in matplotlib
    gui, backend = self.shell.enable_matplotlib(args.gui.lower() if
isinstance(args.gui, str) else args.gui)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/interactiveshell.py", line 3414, in enable_matplotlib
    pt.activate_matplotlib(backend)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/IPython/core/pylabtools.py", line 314, in activate_matplotlib
    matplotlib.pyplot.switch_backend(backend)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/matplotlib/pyplot.py", line 231, in switch_backend
    matplotlib.use(newbackend, warn=False, force=True)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/matplotlib/__init__.py", line 1410, in use
    reload(sys.modules['matplotlib.backends'])
  File "//anaconda3/envs/cs7643/lib/python3.6/importlib/__init__.py", line 166,
in reload
    _bootstrap._exec(spec, module)
  File "//anaconda3/envs/cs7643/lib/python3.6/site-
packages/matplotlib/backends/__init__.py", line 16, in <module>
    line for line in traceback.format_stack()


  import sys
```

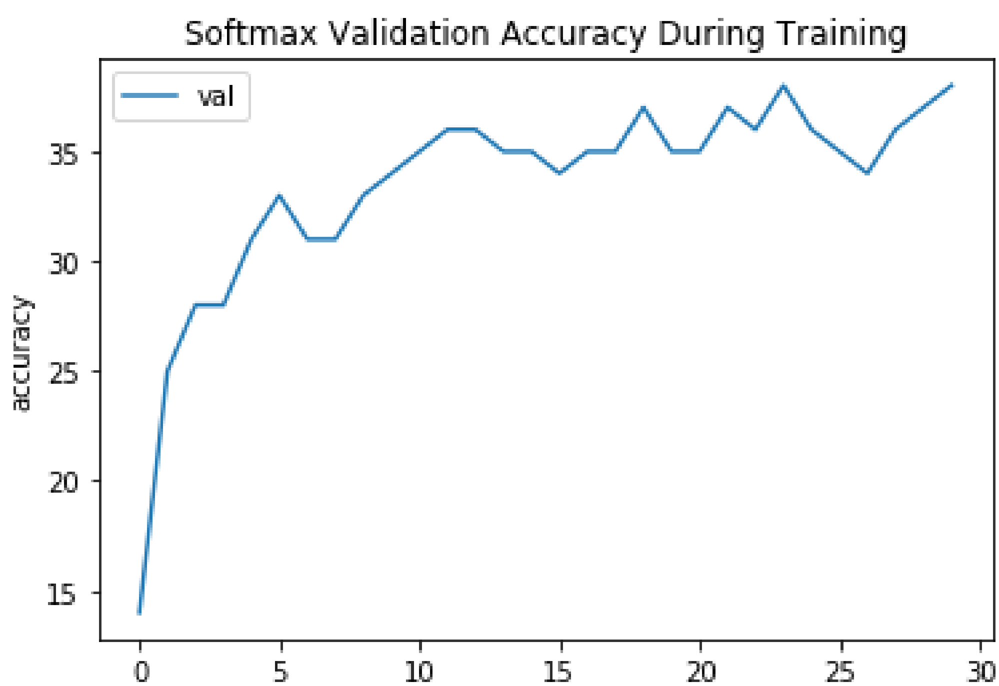
[8]:

```
# Parse the train and val losses one line at a time.
import re
# regexes to find train and val losses on a line
float_regex = r'[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?'
train_loss_re = re.compile('.*Train Loss: ({})'.format(float_regex))
val_loss_re = re.compile('.*Val Loss: ({})'.format(float_regex))
```

```python
val_acc_re = re.compile('.*Val Acc: ({})'.format(float_regex))
# extract one loss for each logged iteration
train_losses = []
val_losses = []
val_accs = []
# NOTE: You may need to change this file name.
with open('softmax.log', 'r') as f:
    for line in f:
        train_match = train_loss_re.match(line)
        val_match = val_loss_re.match(line)
        val_acc_match = val_acc_re.match(line)
        if train_match:
            train_losses.append(float(train_match.group(1)))
        if val_match:
            val_losses.append(float(val_match.group(1)))
        if val_acc_match:
            val_accs.append(float(val_acc_match.group(1)))
```

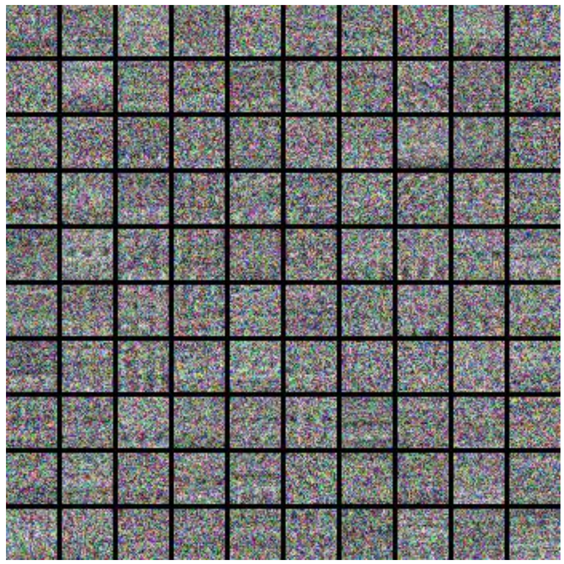
[9]:
```python
fig = plt.figure()
plt.plot(train_losses, label='train')
plt.plot(val_losses, label='val')
plt.title('Softmax Learning Curve')
plt.ylabel('loss')
plt.legend()
fig.savefig('softmax_lossvstrain.png')

fig = plt.figure()
plt.plot(val_accs, label='val')
plt.title('Softmax Validation Accuracy During Training')
plt.ylabel('accuracy')
plt.legend()
fig.savefig('softmax_valaccuracy.png')
```

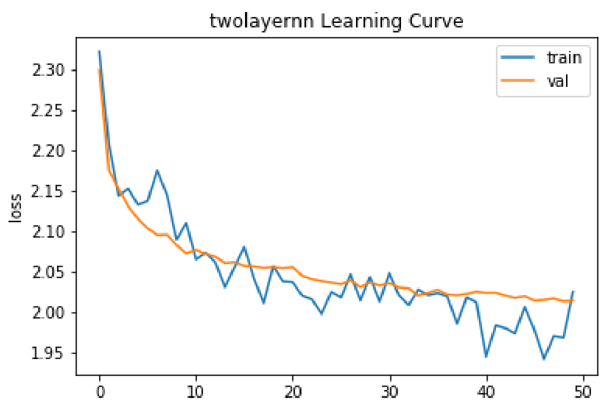

Softmax Learning Curve
train
val
loss
2.35
2.30
2.25
2.20
2.15
2.10
2.05
0
5
10
15
20
25
30

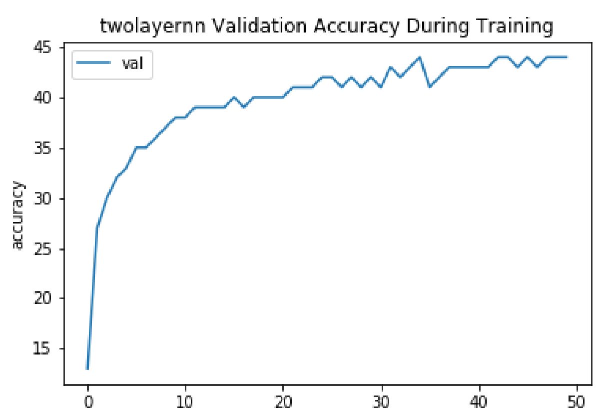

Softmax Validation Accuracy During Training
val
accuracy
35
30
25
20
15
0
5
10
15
20
25
30

[ ]:

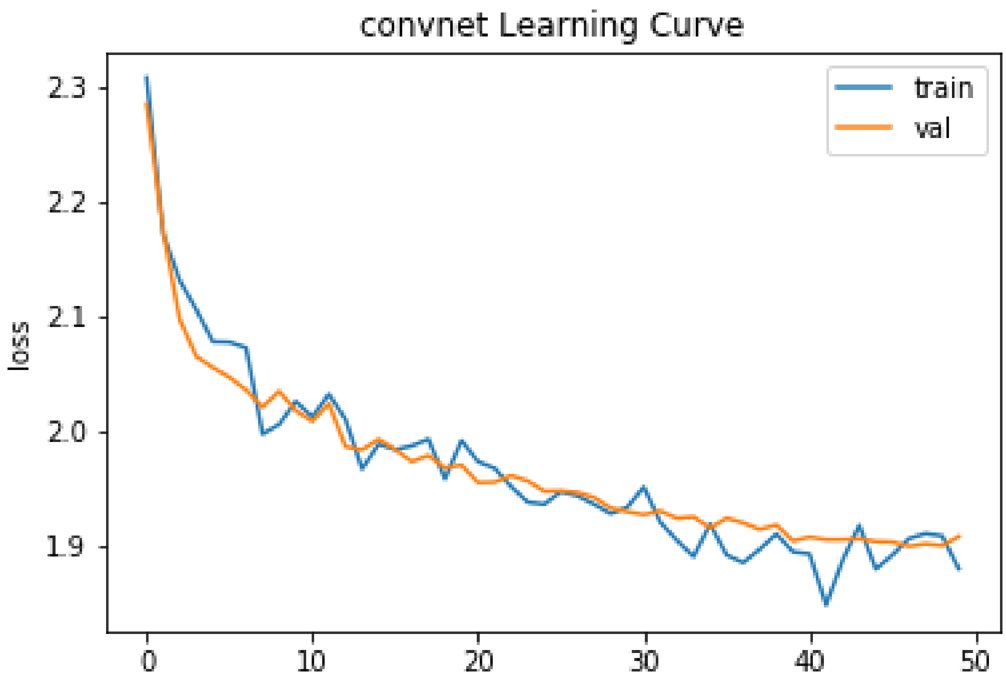
Q8.6 Two-Layer

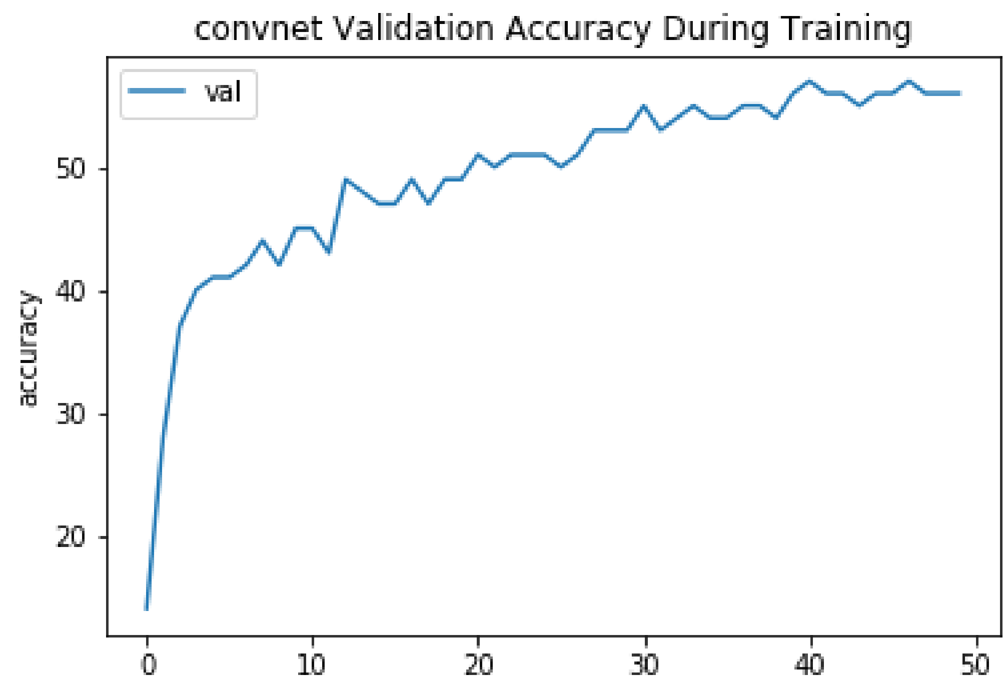






Q8.7 ConvNet


convnet Learning Curve
train
val
loss
2.3
2.2
2.1
2.0
1.9
0
10
20
30
40
50


convnet Validation Accuracy During Training
val
accuracy
50
40
30
20
0
10
20
30
40
50

Q8.8 Experiment