Introduction to Deep Learning (I2DL)

Solutions to exercise 1 and introduction to exercise 2

Today

- Discussion of 1st Exercise Set
  - All solutions in slides but we won't cover everything in detail here
  - Disclaimer: Small text incoming
- Introduction to 2nd Exercise Set

Announcements

- Number of submissions reduced
  - Exercise 2 will only have one submission
  - New goal for bonus: 6/7 submissions
  - (Advice: don't skip this exercise entirely)
- Next exercise session: Dec 19th (next Thursday)
  - Discussion of 2nd Exercise Set
    - Deadline for submission: 18 December 2019, 6pm
  - Introduction to 3rd Exercise Set

Disclaimers

- Deadlines
  - Be more careful with your time management!
- There will be a disconnection between lecture and exercises
  - Advice: revisit the exercise material once you obtained more knowledge due the lectures
- Any details unclear? -> Moodle/office hours ☺
- Everything presented in class or exercise sessions is relevant for the exam

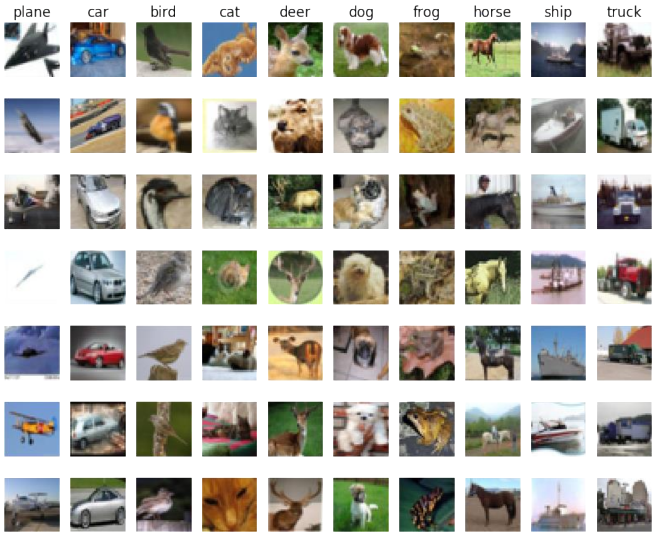
Solutions: Exercise 1

Goals

- Proficiency in Numpy
  - Vectorization
  - Will be relevant for every future project
- Introduction to basic pipeline

Data Pipeline

- Read, Parse & Visualize
  - Easy here as we use images
- Split into Train/Test
  - Validation also important but there are multiple ways one could do this
  - Generally: single validation set
- Normalize



Softmax Classifier

- Given input X, ground truth y, parameters W, what is the cross entropy loss?
- Naïve – with loops

Softmax Classifier

Cross-entropy loss
≈ softmax loss

- Cross-entropy loss naive

```
num_features, num_classes = W.shape
num_train = X.shape[0]
for n in range(num_train):
    x = X[n]
    scores = x.dot(W)
    max_score = np.max(scores)
    exp_scores = np.exp(scores - max_score)
    summedExponentialScores = np.sum(exp_scores)
    probs = exp_scores / summedExponentialScores
    loss += -np.log(probs[y[n]])

    for i in range(num_features):
        for j in range(num_classes):
            dW[i, j] += (probs[j] - (j == y[n])) * x[i]

# Right now the loss is a sum over all training examples, but we want it
# to be an average instead so we divide by num_train.
loss /= num_train
dW /= num_train

# Add regularization to the loss.
loss += 0.5 * reg * np.sum(W * W)
dW += reg * W
```

$$s'_i = x.W$$

http://www.deeplearningbook.org/contents/numerical.html

$$y'_{ic} = \frac{\exp(s'_{ic})}{\sum \exp(s'_{ij})} = \frac{\exp(s'_{ic} - K)}{\sum \exp(s'_{ij} - K)}$$

Can make the exp overflow or underflow

$$-\sum y_{ic} log y'_{ic}$$

Prediction probability

Ground truth probability: 0 or 1

$$dW_{ij} = \sum\sum(y_{ij} - \delta_{yi})xi$$

https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Softmax Classifier

- Always test your functions
- **Question:** why do we expect a loss of –log(0.1) without training?
- Answer: Random weight matrix -> all classes equally likely -> probability of correct class 1/n and n=10 here

$$-\frac{1}{N}\sum_N y_{ic} log y'_{ic}$$

Softmax Classifier

- Vectorized

```
num_train = X.shape[0]
scores = X.dot(W)
exponentialScores_stable = np.exp(scores - np.max(scores, axis=1,
                                  keepdims=True))
probs = exponentialScores_stable / np.sum(exponentialScores_stable, axis=1,
                                          keepdims=True)

loss = - 1.0 * np.sum(np.log(probs[list(range(num_train)), y])) / \
    num_train + 0.5 * reg * np.sum(W * W)

#compute gradient
dscores = probs
dscores[list(range(num_train)), y] -= 1
dscores /= num_train

dW = (X.T).dot(dscores) + reg * W
```

How is this stable?
Use: softmax(x) = softmax(x+c)
Here: c = -max(scores)

SGD

Always overfit on a small subset first!

```
randIdx = np.random.choice(num_train, batch_size)
X_batch = X[randIdx]
y_batch = y[randIdx]
#########################################################################
#                           END OF YOUR CODE                            #
#########################################################################

# evaluate loss and gradient
loss, grad = self.loss(X_batch, y_batch, reg)
loss_history.append(loss)

# perform parameter update
#########################################################################
# TODO:                                                                 #
# Update the weights using the gradient and the learning rate.          #
#########################################################################
self.W = self.W - learning_rate * grad
```

Hyperparameter Tuning using Grid Search

```
for learning_rate in learning_rates:
    for reg in regularization_strengths:
        print('train lr %e reg %e ' % (learning_rate, reg))
        softmax = SoftmaxClassifier()
        loss_hist = softmax.train(X_train, y_train,
                                  learning_rate=learning_rate, reg=reg,
                                  num_iters=500, verbose=False)
        y_train_pred = softmax.predict(X_train)
        trainingAccuracy = np.mean(y_train == y_train_pred)
        # print('training accuracy: %f' % trainingAccuracy, )
        y_val_pred = softmax.predict(X_val)
        validationAccuracy = np.mean(y_val == y_val_pred)
        # print('validation accuracy: %f' % validationAccuracy, )
        results[(learning_rate, reg)] = (trainingAccuracy,
                                         validationAccuracy)
        # save all classifiers and keep track of the best one
        all_classifiers.append((softmax, validationAccuracy, learning_rate,
                                reg))
        if validationAccuracy > best_val:
            best_val = validationAccuracy
            best_softmax = softmax
```

More -> lecture

Hyperparameter Tuning using Grid Search: Results

- Evaluate until you can't improve

```
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.269333 val accuracy: 0.257000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.314063 val accuracy: 0.309000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.350687 val accuracy: 0.342000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.321979 val accuracy: 0.316000
```

- Test best result

```python
# evaluate on test set
# Evaluate the best softmax on test set
y_test_pred = best_softmax.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: 0.360000
```

Neural Net

- Forward

```
H = np.maximum(X.dot(W1) + b1, 0)
scores = H.dot(W2) + b2
```

- Loss

```
scores_exp = np.exp(scores)
#data loss
loss = np.sum(-np.log(scores_exp[list(range(N)), y] / \
    np.sum(scores_exp, axis=1, keepdims=False))) / N
#regularizer loss
loss += 0.5 * reg * (np.sum(W1 * W1) + np.sum(W2 * W2))
```

Neural Net

- Backward

```
probs = np.exp(scores) / np.sum(scores_exp, axis=1, keepdims=True)
dscores = probs
dscores[list(range(N)), y] -= 1
dscores /= N
# print "Shape of dscores: " + str(dscores.shape)

###### propagate to seconds weights #####
# data gradient contribution
dW2 = (H.T).dot(dscores)
db2 = np.sum(dscores, axis=0, keepdims=False)
# regularization gradient contribution
dW2 += reg * W2

##### propagate to hidden layer #####
dH = dscores.dot(W2.T)
dH[H <= 0] = 0

##### propagate to first weights #####
dW1 = (X.T).dot(dH)
db1 = np.sum(dH, axis=0, keepdims=False)
# regularization gradient
dW1 += reg * W1

grads['W1'] = dW1
grads['b1'] = db1.T
grads['W2'] = dW2
grads['b2'] = db2.T
```

Neural Net

- Train

```
#########################################################################
# TODO: Create a random minibatch of training data and labels,          #
# storing hem in X_batch and y_batch respectively.                      #
#########################################################################
randIdx = np.random.choice(num_train, batch_size)
X_batch = X[randIdx]
y_batch = y[randIdx]
#########################################################################
#                              END OF YOUR CODE                         #
#########################################################################

# Compute loss and gradients using the current minibatch
loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
loss_history.append(loss)

#########################################################################
# TODO: Use the gradients in the grads dictionary to update the         #
# parameters of the network (stored in the dictionary self.params)      #
# using stochastic gradient descent. You'll need to use the             #
# gradients stored in the grads dictionary defined above.               #
#########################################################################
self.params['W1'] += -learning_rate * grads['W1']
self.params['b1'] += -learning_rate * grads['b1']
self.params['W2'] += -learning_rate * grads['W2']
self.params['b2'] += -learning_rate * grads['b2']
```

Neural Net

- Predict

```
W1, b1 = self.params['W1'], self.params['b1']
W2, b2 = self.params['W2'], self.params['b2']
H = np.maximum(X.dot(W1) + b1, 0)
scores = H.dot(W2) + b2
y_pred = np.argmax(scores, axis=1)
```

- What do weights look like?
  - More distinctive featues possible due to higher non-linearity and depth
  - More in the lectures ;)

Tuning

```
num_classes = 10
input_size = 32 * 32 * 3
num_iters=10000
batch_size=256
learning_rates = [1e-3]
learning_rate_decay=0.95
regularization_strengths = [2.0]
hidden_size = 150


results = {}
best_val = -1   # The highest validation accuracy that we have seen so far.
best_stats = None

for learning_rate in learning_rates:
    for reg in regularization_strengths:
        net = TwoLayerNet(input_size, hidden_size, num_classes)
        # Train the network
        print(X_train.shape)
        print(y_train.shape)
        print(X_val.shape)
        print(y_val.shape)
        stats = net.train(X_train, y_train, X_val, y_val,
        num_iters=num_iters, batch_size=batch_size,
        learning_rate=learning_rate,
        learning_rate_decay=learning_rate_decay,
        reg=reg, verbose=True)

        # Predict on the validation set
        val_acc = (net.predict(X_val) == y_val).mean()
        print('Validation accuracy: ', val_acc)
        results[(learning_rate, reg)] = stats
        if val_acc > best_val:
            best_val = val_acc
            best_net = net
            best_stats = stats
```

Note:
Training neural networks will get easier with more and more tricks but you can never know the best parameters. In the end this is a question of computation resources

Features

- Lesson
  - Sometimes it is important to use existing tools and new is not always better

- Interpretation
  - Useful to get a feeling what happens
  - More on that in the lectures

Outlook: Exercise 2

Exercise 2

- Deadline: December 18th, 6pm
  - don't wait until the end of the deadline...
- Only one notebook counts for the bonus: 1_FullyConnectedNets.ipynb
- 2 optional notebooks
  - 2_BatchNormalization-optional.ipynb
  - 3_Dropout-optional.ipynb

Exercise 2

- Make sure you have the latest zip file
- We had a few problems regarding the zip file. If your submission fails to upload, please try using the latest create_submission.sh script. If it still doesn't work, copy all your code from the solution blocks to the latest exercise files.

Exercise 2: Submission

- Implement modular layers
  - Fully connected layer
  - Softmax layer (Loss layer)
  - ReLu Layer (Activation layer)
- Contains Forward/Backward
  - Sandwichable
- Solver (SGD, Momentum)

Exercise 2: Optional Content

- Advanced layers
  - Dropout
  - Batch Normalization
  - Will be discussed in detail in the lectures

- Help you train more powerful networks

Questions?