

6.867: Homework 3

This homework will be structured in the same way as homeworks 1 and 2. Details of the submission and grading process are repeated at the end.

- We encourage you to do this assignment in groups of two students, though you may do it individually if you wish. However, all write-ups must be submitted individually and all code and figures should be your own
- If partners submit identical PDFs, both will receive no credit
- You submit your paper via the CMT site by 11:59 PM on Tuesday, November 14.
- After submission on CMT, you will be assigned 3 papers to review.
- Reviews must be entered on the CMT site by 11:59PM on Sunday, November 19.
- All reviews will be made visible shortly thereafter, and students will have a two day period in which to enter rebuttals of their reviews into CMT if they wish.

Introduction

1. Section 1 asks you to implement a “traditional” fully-connected neural network and test it on several data sets. Although it’s tempting to use an existing implementation, you won’t really understand the operation of these networks unless you do an implementation from scratch.
2. Section 2 asks you to experiment with an existing implementation of convolutional neural nets so that you can get some idea of the space of “architectural” decisions one faces when applying this approach.
3. Section 3 asks you to perform an unsupervised learning task by using an LDA model for topic modeling on a given corpus. You will examine the topics this approach discovers as the algorithm’s parameters are varied.

1 Neural Networks

In this question, we will implement backpropagation in a vector/matrix style, as presented in class. You should review the neural network basics/backpropagation lecture notes.

Use Matlab or Python to implement a program to train fully-connected neural nets using stochastic gradient descent. Do your implementation in sufficient generality so that you can vary the number of nodes in each hidden layer and the number of hidden layers. The gradient computation for gradient descent is given by the pseudo-code for backprop in the lecture slides. Make sure that you understand the dimensions of each of the vectors and matrices in the pseudo-code. When properly implemented, you should be able to do essentially all the computations layer-wise in matrix form.

First test your implementation on the simple two-dimensional, 3-class data set we provided you in (data/data_3class.csv). Try networks with one and two hidden layers. This is primarily for debugging, you do not need to report on this in detail.

1. ReLU + Softmax:

Assume that the **hidden** units are all ReLU; whose activation functions are $f(z) = \max(0, z)$. Also assume that the output layer is a Softmax layer for k classes. A Softmax layer maps a vector of k inputs (the outputs of the previous layer) to a vector of k “probabilities” that add up to one. The activation function for each output is:

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \text{ for } i = 1 \dots k$$

We assume that the target outputs are represented as one-hot vectors. Let the loss function be cross entropy, that is, for a particular training point (x, y) , the loss is $-\sum_{i=1}^k y_i \log(f(z)_i)$ where $f(z)$ is the Softmax output; see also equations 5.24 and 5.25 in Bishop.

Describe how the choice of a Softmax output layer and cross-entropy loss is incorporated into your implementation.

2. Initialization:

It has long been known that proper weight initialization is paramount for network convergence. A commonly used technique for weight initialization is Xavier Initialization¹. The goal is to randomly assign the initial weights so that the initial variance of any unit’s value (before applying the non-linearity) is independent of the size of the previous layer. This is especially important for saturating non-linearities where it is important to keep values close to 0. If weights are assigned as a zero mean Gaussian distribution, what should the variance be to achieve the desired property? For your analysis you only need to consider the forward pass. For your implementation you can initialize biases as 0 (why can you not do this for weights?).

¹<http://proceedings.mlr.press/v9/glorot10a.html>

3. Regularization:

If we wanted to add weight regularization during training:

$$J(w) = \ell(w) + \lambda(\|w^{(1)}\|_F^2 + \|w^{(2)}\|_F^2)$$

where $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$ is the matrix Frobenius norm. How would this affect the pseudo-code? Be specific. You don't need to implement and test this (unless you want to).

4. Binary classification:

Test your implementation on the four 2D data sets from HW 2. During the training process, use the training set to fit the weights and use the validation set to decide when to stop training. Report the final accuracy on the training data and on the test data.

Pay careful attention to the form of the target values you need for a Softmax output unit. You will need to convert the data in the files to that form.

Experiment with some architectures (single small hidden layer, single large hidden layer, two small hidden layers, two large hidden layers) and report your results. You will need to experiment with learning rates to find ones that work for the individual architectures. Discuss any trends that you see and compare your best results to those you obtained on these data sets in HW 2. You should be able to adapt the basic data reading and display code from HW 2.

5. Multi-class classification:

Repeat this process for the 10-class digit classification problem, using the MNIST data from HW 2. Note that in HW 2 we addressed several binary classification problems, such as odd vs even, with the MNIST data, now we are asking you to predict the digit directly. Remember to normalize the data as you did in HW 2 (what happens if you don't?). Try it with at least 200 samples per class but try more if you can.

For debugging: for a small single-layer neural net with tuned learning rate and a training set of 200 samples per class, your model accuracy on the training set should be over 80% after 10 epochs (i.e. 10 full passes of the training set), and the accuracy on the validation set should be over 90% after ~ 40 epochs.

2 Convolutional Neural Networks

In this question, you will be asked to experiment with a convolutional neural network implementation to perform image classification. The dataset we will use for this assignment was created by Zoya Bylinskii, and contains 451 works of art from 11 different artists all downsampled and padded to the same size. The task is to identify which artist produced each image. The original images can be found in the `art_data/artists` directory included with the assignment. The composition of the dataset and a sample painting from each artist are shown in Table 1.

Table 1: Artists and number of paintings in the dataset










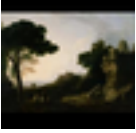
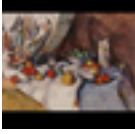
Artist	#	Example	Artist	#	Example
Canaletto	20		Paul Gauguin	37	
Claude Monet	54		Paul Sandby	36	
George Romney	63		Peter Paul Rubens	28	
J. M. W. Turner	70		Rembrandt	40	
John Robert Cozens	40		Richard Wilson	23	
Paul Cezanne	40				

Figure 1 shows an example of the type of convolutional architecture typically employed for similar image recognition problems. Convolutional layers apply filters to the image, and produce layers of feature maps. Often, the convolutional layers are interspersed with pooling layers. The final layers of the network are fully connected, and lead to an output layer with one node for each

of the K classes the network is trying to detect. We will use a similar architecture for our network.

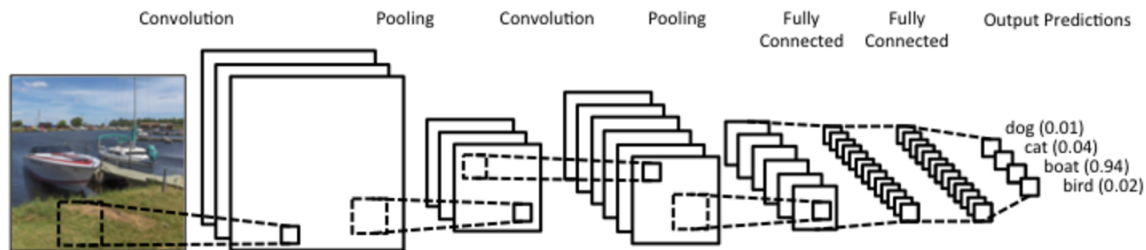


Figure 1: A typical convolutional architecture. Image taken from <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

For this question you will explore the use of a convolutional neural network to perform this artist identification task. To help you out, we have provided code you can use to perform the data processing and network training steps. The conv net code is implemented in [Tensorflow](#), a popular library for training deep neural networks. However, you are not required to use the code we have provided and are free to use any deep learning library you like.

In the code, the file `make_datasets.py` creates a dataset from the artists' images by downsampling them to 50x50 pixels, and transforming the RGB values to lie within the range $[-0.5, 0.5]$. The resulting dataset is saved in the file `art_data.pickle`. The provided code to train the network can be found in `conv.py`². You do not need to understand all of this code to complete the assignment, but you will need to make changes to the hyperparameters and network structure found in the function `build_graph`.

1. Convolutional filter receptive field

First, it is important to develop an intuition for how a convolutional layer affects the feature representations that the network learns. Assume that you have a network in which the first convolutional layer applies a 5x5 patch to the image, producing a feature map Z_1 . The next layer of the network is also convolutional; in this case, a 3x3 patch is applied to the feature map Z_1 to produce a new feature map, Z_2 . Assume the stride used in both cases is 1. Let the *receptive field* of a node in this network be the portion of the original image that contributes information to the node (that it can, through the filters of the network, “see”). What are the dimensions of the receptive field for a node in Z_2 ? Note that you can ignore padding, and just consider patches in the middle of the image and Z_1 . Thinking about your answer, why is it effective to build convolutional networks deeper (i.e. with more layers)?

²Note that this code was optimized for simplicity, not best programming practices

2. Run the conv net

Implement a conv net or use the code we have provided. If you do choose to use our implementation, examine the provided code in the *build_graph* function and make sure you can answer the following questions about the initial implementation (if you aren't using our code, make sure you can still answer these questions for your own implementation!):

- How many layers are there? Are they all convolutional? If not, what structure do they have?
- Which activation function is used on the hidden nodes?
- What loss function is being used to train the network?
- How is the loss being minimized?

The provided network can be trained by simply running the command *python conv.py*. It should take between 30-60 seconds to train your network for 1500 steps on a CPU. What is the training accuracy for your network after training? What is the validation accuracy? What do these two numbers tell you about what your network is doing?

3. Experiment with network features

The following is a list of common techniques used to boost CNN performance in practice. Many of these will have some regularization effect (among other effects) which is important on such a small dataset. Experiment with each one individually and discuss how it affects the network performance.




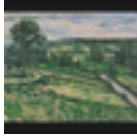

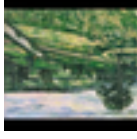


- **Early stopping:** Stop training your model after your validation accuracy starts to plateau or decrease (so that you do not overtrain your model). The number of steps can be controlled through the *NUM_TRAINING_STEPS* hyperparameter.
- **Pooling layers:** We will now add max pooling layers after our convolutional layers. This code has already been provided for you; all you need to do is switch the *POOL1* and/or *POOL2* flag(s) in the hyperparameters to **True**, and choose different values for the pooling filter size and stride.
- **Network shape (optional):** Experiment with different numbers of convolutional layers, different filter sizes for the layers, different stride lengths and different layer depths. Should later layers be shaped differently than earlier layers?
- **Dropout (optional):** Dropout is a form of regularization where during training units are randomly "dropped out" meaning their activations are set to 0 with some probability. Dropout has already been integrated into the *conv.py* code. To activate it, simply set the *DROPOUT_RATE* hyperparameter to some value > 0.0 . This value is the probability with which units are dropped out. You should explore different values to find one that works well.

- **Weight regularization (optional):** As in the last question, we will add an additional term to the loss function to penalize the norm of your network's weights. Again, this has already been implemented, and you simply need to find a good value for the *L2.CONST* hyperparameter.
- **Data augmentation (optional):** Another regularization technique is to augment the data with transformed versions of the original samples. The file *augmented_art_data.pickle* contains randomly flipped and distorted versions of the original training images (note that the validation images remain unchanged). Change the *DATA_FILE* constant at the top of your *conv.py* code to point to the augmented dataset, and try running your code on this new data. The dataset is 4 times as large, so you should increase the number of training steps accordingly.
- **Batch normalization (optional):** Batch normalization can be added after a network layer to normalize the activations before passing them to the next layer. This safeguards against internal covariate shift allowing for larger learning rates and much faster training times. Batch normalization has already been implemented in the provided code and can be toggled by the *BN* flag.

4. Test your architecture on variations of the data

Choose an optimized architecture using any number of the techniques above and any others you wish to try. You will then test this architecture on distorted versions of the images. The file *invariance_art_data.pickle* contains versions of the validation images that have been distorted using the transformations shown in Table 2. To test your code on this data, run the command `python conv.py invariance`. Report your performance on each of the transformed datasets. Are you surprised by any of the results? Which transformations is your network most invariant to, and which lead it to be unable to recognize the images? What does that tell you about the features your network has learned to use to recognize artists' images? Do certain techniques from question 2.3 make your model more or less invariant to certain transformations?

Table 2: Translations made to images

Transformation	Example	Transformation	Example
Normal		High contrast	
Translated		Low contrast	
Brightened		Flipped upside-down	
Darkened		Colors inverted	

3 Topic Models

In this question, you will be performing an unsupervised learning task on real world data. More specifically, you will be using the latent Dirichlet allocation (LDA) model to perform topic modeling on a corpus of text.

On a high level, LDA views documents as being generated from a mixture of latent topic variables that have an associated distribution over word probabilities. When learning these topics LDA further assumes a sparse Dirichlet prior for these distributions which models the fact that documents generally cover a small number of topics which in turn are primarily associated with a small number of words.

You are encouraged to read the review article by David Blei on topic models³ to gain familiarity with this subject. Those who are interested can also read the seminal paper by Blei et al. introducing LDA⁴.

In `hw3_resources.zip`, we provide you with the *20Newsgroups* data set⁵, which consists of 19,997 articles for 20 categories taken from the Usenet newsgroups collection.

1. **Train an LDA model:** Learn an LDA model over all 19,997 articles (i.e., ignoring the known categories) with 100 topics. Use the implementation of LDA provided by Mallet⁶ with the default parameters.
2. **Explore the topics:** Qualitatively describe the topics that your model has discovered. In addition, choose two of the 20 article categories and report the top three most represented topics in each of them, by averaging the topic distributions of the documents in the category. Do they make sense?
3. **Experiment with parameters:** Retrain the model with various numbers of topics. Report how decreasing the number of topics affected the topics that were discovered.

³<http://www.cs.columbia.edu/~blei/papers/Blei2012.pdf>

⁴<http://www.cs.columbia.edu/~blei/papers/BleiNgJordan2003.pdf>

⁵<http://www.ai.mit.edu/people/jrennie/20Newsgroups/>

⁶<http://mallet.cs.umass.edu/>

Procedure, grading, etc.

You will find a zip file with some useful code and data in the Stellar Materials page. You can do these assignments in any computational system that you are used to. We recommend Matlab or the Pylab/Numpy/Scipy/Matplotlib cluster of packages and we'll try to provide help for those two systems. If you use anything else, you're on your own...

You will be turning in a single, readable "paper" (a single PDF file) with your solutions. In the Stellar Materials page we have provided some sample solutions from a previous year. The content in these examples is not directly relevant, but note that there are coherent explanations and nice illustrations of results in figures and tables. You should write your paper as if it is going to be read by someone who has taken a class in machine learning but has not read this assignment handout.

We will be emulating the process of submitting papers for publication to a conference. We will be using an actual conference review system (CMT) to have these papers peer reviewed (by the other students). This means that your answers have to be readable and understandable to your peers and, where possible, interesting. Note that when explanations are called for, you will need to convince the reviewers that you understand what you're talking about. The course staff will serve as the Program Committee for the conference and make all final decisions.

Grading rubric

Your paper must be anonymous (no identifying information should appear in the PDF file). If it is not, it will automatically receive a 20% deduction, and will be graded by a grumpy staff member.

The paper must be no more than 6 pages long in a font no smaller than 10 point. It should include whatever tables, graphs, plots, etc., are necessary to demonstrate your work and conclusions. *It should not include code.*

Each of the **three** parts of the assignment will be graded on a scale from 0 to 5 (where 0 is failing and 5 is an A) on two aspects:

- **Content:** Did the solution answer the questions posed? Were the answers correct? Were the experiments well-designed or examples well chosen?
- **Clarity:** Were the results written up clearly? Were the plots labeled appropriately and described well? Did the plots support the points in the paper? Did the discussion in the paper illuminate the plots?

As a reviewer, you will be asked to provide a score for each section, and at at least two paragraphs of feedback, per review, explaining things that were done well and things that could have been improved upon.

Your overall score for this assignment will be:

- **80%:** The average of all 8 scores on your assignment given by all three reviewers.
- **20%:** A score for the quality of your reviews. This will be full credit, by default. But we will skim reviews and examine some carefully and may reduce this grade for review commentary that is sloppy or wrong.

The course staff will spot-check submissions and reviews, paying careful attention to cases where there were rebuttals. The staff will act as the program committee and determine a final score. Our overall goals in this process are:

- To motivate you to work seriously on the problems and learn something about the machine learning material in the process
- To engage you in thinking critically and learning from other students' solutions to the problems

We will arrange to give full credit to anyone who submits a serious and careful solution to the problems and who gives evidence of having read carefully the solutions they were assigned and who writes thoughtful reviews of them.

The questions are the points that your paper should cover in order to receive full credit. Your presentation should roughly follow the order of these questions so that your reviewers can see what you're doing.