Homework #4

CSE 446: Machine Learning
Prof. Byron Boots
Due: **Friday** 03/19/2021 11:59 PM — **NO LATE DAYS ALLOWED**

Please review all homework guidance posted on the website before submitting to Gradescope.

- Turning in this homework late will result in a **ZERO** for the assignment. We have made the deadline as late as possible given when we must report grades to the university. Please plan to finish early.
- Please provide succinct answers along with succinct reasoning for all your answers. Points may be deducted if long answers demonstrate a lack of clarity. Similarly, when discussing the experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. In other words, all your explanations, tables, and figures for any particular part of a question must be grouped together.
- When submitting to gradescope, please link each question from the homework in gradescope to the location of its answer in your homework PDF. Failure to do so may result in point deductions. For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.

For the below problems, 1 through 5 are **required**. You may choose **only one BLUE problem from {6, 7}** to complete (please do not turn in both).

Conceptual Questions

1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

   a. *[2 points]* True or False: Given a data matrix $X \in R^{n \times d}$ where d is much smaller than n and $k = \text{rank}(X)$, if we project our data onto a k dimensional subspace using PCA, our projection will have zero reconstruction error (in other words, we find a perfect representation of our data, with no information loss).

   b. *[2 points]* True or False: Suppose that an $n \times n$ matrix X has a singular value decomposition of $USV^\top$, where S is a diagonal $n \times n$ matrix. Then, the rows of V are equal to the eigenvectors of $X^\top X$.

   c. *[2 points]* True or False: choosing k to minimize the k-means objective (see Equation (1) below) is a good way to find meaningful clusters.

   d. *[2 points]* True or False: The singular value decomposition of a matrix is unique.

   e. *[2 points]* True or False: The rank of a square matrix equals the number of its nonzero eigenvalues.

   f. *[2 points]* True or False: Autoencoders, where the encoder and decoder functions are both neural networks with nonlinear activations, can capture more variance of the data in its encoded representation than PCA using the same number of dimensions.

Basics of SVD

2. Given $X \in \mathbb{R}^{m\times n}$, recall that its Singular Value Decomposition (SVD) gives us a factorization of a matrix $X = U\Sigma V^\top$ such that $U \in \mathbb{R}^{m\times m}, V^\top \in \mathbb{R}^{n\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix representing the singular values of X. Show the following.

   a. *[3 points]* Let $\widehat{w}$ be the solution to the regression problem $\min_w \|Xw - y\|_2^2$. Let $\widehat{w}_R$ be the solution to the ridge regression problem $\min_w \|Xw - y\|_2^2 + \lambda\|w\|_2^2$. Let $X = U\Sigma V^\top$ be a singular value decomposition of X. Using this decomposition, explain why the solution $\widehat{w}_R$ to the ridge regression problem "shrinks" as compared to the solution $\widehat{w}$ of the standard regression problem.

   b. *[3 points]* Let $U \in \mathbb{R}^{n\times n}$ be a matrix with singular values all equal to one. Show that $UU^\top = U^\top U = I$.

   c. *[3 points]* Now use the above result to show that U preserves Euclidean norms. In other words, $\|Ux\|_2 = \|x\|_2$ for any $x \in \mathbb{R}^n$.

k-means clustering

3. Given a dataset $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$ and an integer $1 \le k \le n$, recall the following k-means objective function

$$\min_{\pi_1,\dots,\pi_k} \sum_{i=1}^{k} \sum_{j\in\pi_i} \|\mathbf{x}_j - \mu_i\|_2^2 \ , \quad \mu_i = \frac{1}{|\pi_i|}\sum_{j\in\pi_i} \mathbf{x}_j \ . \tag{1}$$

Above, $\{\pi_i\}_{i=1}^k$ is a partition of $\{1, 2, ..., n\}$. The objective (1) is NP-hard[1] to find a global minimizer of. Nevertheless the commonly-used algorithm we discussed in lecture (Lloyd's algorithm), typically works well in practice.

   a. *[5 points]* Implement Lloyd's algorithm for solving the k-means objective (1). Do not use any off-the-shelf implementations, such as those found in `scikit-learn`. Include your code in your submission.

   b. *[5 points]* Run the algorithm on the *training* dataset of MNIST with $k = 10$, plotting the objective function (1) as a function of the iteration number. Visualize (and include in your report) the cluster centers as a 28×28 image.

   c. *[5 points]* For $k = \{2, 4, 8, 16, 32, 64\}$ run the algorithm on the *training* dataset to obtain centers $\{\mu_i\}_{i=1}^k$. If $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and $\{(\mathbf{x}'_i, y'_i)\}_{i=1}^m$ denote the training and test sets, respectively, plot the training error $\frac{1}{n}\sum_{i=1}^n \min_{j=1,\dots,k} \|\mu_j - \mathbf{x}_i\|_2^2$ and test error $\frac{1}{m}\sum_{i=1}^m \min_{j=1,\dots,k} \|\mu_j - \mathbf{x}'_i\|_2^2$ as a function of k on the same plot.

PCA

4. Let's do PCA on MNIST dataset and reconstruct the digits in the dimensionality-reduced PCA basis. You will actually compute your PCA basis using the training dataset only, and evaluate the quality of the basis on the test set, similar to the k-means reconstructions of above. We have $n_{train} = 50,000$ training examples of size 28×28. Begin by flattening each example to a vector to obtain $X_{train} \in \mathbb{R}^{50,000\times d}$ and $X_{test} \in \mathbb{R}^{10,000\times d}$ for $d := 784$.

Let $\mu \in \mathbb{R}^d$ denote the average of the training examples in X_{train}, i.e., $\mu = \frac{1}{n_{train}} X_{train}^\top \mathbf{1}^\top$. Now let $\Sigma = (X_{train} - \mathbf{1}\mu^\top)^\top (X_{train} - \mathbf{1}\mu^\top)/50000$ denote the sample covariance matrix of the training examples, and let $\Sigma = UDU^T$ denote the eigenvalue decomposition of Σ.

[1]To be more precise, it is both NP-hard in d when $k = 2$ and k when $d = 2$. See the references on the wikipedia page for k-means for more details.

a. *[2 points]* If λ_i denotes the ith largest eigenvalue of Σ, what are the eigenvalues λ_1, λ_2, λ_{10}, λ_{30}, and λ_{50}? What is the sum of eigenvalues $\sum_{i=1}^{d} \lambda_i$?

b. *[5 points]* Let $x \in \mathbb{R}^d$ and $k \in 1, 2, \ldots, d$. Write a formula for the rank-k PCA approximation of x.

c. *[5 points]* Using this approximation, plot the reconstruction error from $k = 1$ to 100 (the X-axis is k and the Y-axis is the mean-squared error reconstruction error) on the training set and the test set (using the μ and the basis learned from the training set). On a separate plot, plot $1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$ from $k = 1$ to 100.

d. *[3 points]* Now let us get a sense of what the top PCA directions are capturing. Display the first 10 eigenvectors as images, and provide a brief interpretation of what you think they capture.

e. *[3 points]* Finally, visualize a set of reconstructed digits from the training set for different values of k. In particular provide the reconstructions for digits $2, 6, 7$ with values $k = 5, 15, 40, 100$ (just choose an image from each digit arbitrarily). Show the original image side-by-side with its reconstruction. Provide a brief interpretation, in terms of your perceptions of the quality of these reconstructions and the dimensionality you used.

Unsupervised Learning with Autoencoders

5. In this exercise, we will train two simple autoencoders to perform dimensionality reduction on MNIST. As discussed in lecture, autoencoders are a long-studied neural network architecture comprised of an encoder component to summarize the latent features of input data and a decoder component to try and reconstruct the original data from the latent features.

Weight Initialization and PyTorch

Last assignment, we had you refrain from using `torch.nn` modules. For this assignment, we recommend using `nn.Linear` for your linear layers. You will not need to initialize the weights yourself; the default He/Kaiming uniform initialization in PyTorch will be sufficient for this problem. *Hint: we also recommend using the `nn.Sequential` module to organize your network class and simplify the process of writing the forward pass. However, you may choose to organize your code however you'd like.*

Training

Use `optim.Adam` for this question. Experiment with different learning rates. Use mean squared error (`nn.MSELoss()` or `F.mse_loss()`) for the loss function.

a. *[10 points]* Use a network with a single linear layer. Let $W_{\mathrm{e}} \in \mathbb{R}^{h \times d}$ and $W_{\mathrm{d}} \in \mathbb{R}^{d \times h}$. Given some $x \in \mathbb{R}^d$, the forward pass is formulated as

$$\mathcal{F}_1(x) = W_{\mathrm{d}} W_{\mathrm{e}} x.$$

Run experiments for $h \in \{32, 64, 128\}$. For each of the different h values, report your final error and visualize a set of 10 reconstructed digits, side-by-side with the original image. *Note:* we omit the bias term in the formulation for notational convenience since `nn.Linear` learns bias parameters alongside weight parameters by default.

b. *[10 points]* Use a single-layer network with non-linearity. Let $W_{\mathrm{e}} \in \mathbb{R}^{h \times d}$, $W_{\mathrm{d}} \in \mathbb{R}^{d \times h}$, and activation $\sigma : \mathbb{R} \longmapsto \mathbb{R}$, where σ is the ReLU function. Given some $x \in \mathbb{R}^d$, the forward pass is formulated as

$$\mathcal{F}_2(x) = \sigma(W_{\mathrm{d}} \sigma(W_{\mathrm{e}} x)).$$

Report the same findings as asked for in part a (for $h \in \{32, 64, 128\}$).

c. *[5 points]* Now, evaluate $\mathcal{F}_1(x)$ and $\mathcal{F}_2(x)$ (use $h = 128$ here) on the test set. Provide the test reconstruction errors in a table.

d. *[5 points]* In a few sentences, compare the quality of the reconstructions from these two autoencoders with those of PCA from problem 4. You may need to re-run your code for PCA using the ranks $k \in \{32, 64, 128\}$ to match the h values used above.

Image Classification on CIFAR-10

6. In this problem we will explore different deep learning architectures for image classification on the CIFAR-10 dataset. Make sure that you are familiar with tensors, two-dimensional convolutions (`nn.Conv2d`) and fully-connected layers (`nn.Linear`), ReLU non-linearities (`F.relu`), pooling (`nn.MaxPool2d`), and tensor reshaping (`view`).

A few preliminaries:

- Each network f maps an image $x^{in} \in \mathbb{R}^{32\times32\times3}$ (3 channels for RGB) to an output $f(x^{in}) = x^{out} \in \mathbb{R}^{10}$. The class label is predicted as $\arg\max_{i=0,1,\ldots,9} x_i^{out}$. An error occurs if the predicted label differs from the true label for a given image.
- The network is trained via multiclass cross-entropy loss.
- Create a validation dataset by appropriately partitioning the train dataset. *Hint*: look at the documentation for `torch.utils.data.random_split`. Make sure to tune hyperparameters like network architecture and step size on the validation dataset. Do **NOT** validate your hyperparameters on the test dataset.
- At the end of each epoch (one pass over the training data), compute and print the training and validation classification accuracy.
- While one would usually train a network for hundreds of epochs to reach convergence and maximize accuracy, this can be prohibitively time-consuming, so feel free to train for just a dozen or so epochs.

For parts (a)–(c), apply a hyperparameter tuning method (grid search, random search, etc.) using the validation set, report the hyperparameter configurations you evaluated and the best set of hyperparameters from this set, and plot the training and validation classification accuracy as a function of epochs. Produce a separate line or plot for each hyperparameter configuration evaluated. Finally, evaluate your best set of hyperparameters on the test data and report the test accuracy.

Here are the network architectures you will construct and compare.

a. *[10 points]* **Fully-connected output, 0 hidden layers (logistic regression):** this network has no hidden layers and linearly maps the input layer to the output layer. This can be written as

$$x^{out} = W\text{vec}(x^{in}) + b$$

where $x^{out} \in \mathbb{R}^{10}$, $x^{in} \in \mathbb{R}^{32\times32\times3}$, $W \in \mathbb{R}^{10\times3072}$, $b \in \mathbb{R}^{10}$ since $3072 = 32\cdot32\cdot3$. For a tensor $x \in \mathbb{R}^{a\times b\times c}$, we let $\text{vec}(x) \in \mathbb{R}^{abc}$ be the reshaped form of the tensor into a vector (in an arbitrary but consistent pattern). There is no required benchmark validation accuracy for this part.

b. *[15 points]* **Fully-connected output, 1 fully-connected hidden layer:** this network has one hidden layer denoted as $x^{hidden} \in \mathbb{R}^M$ where M will be a hyperparameter you choose (M could be in the hundreds). The nonlinearity applied to the hidden layer will be the `relu` ($\text{relu}(x) = \max\{0, x\}$. This network can be written as

$$x^{out} = W_2\text{relu}(W_1\text{vec}(x^{in}) + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{M\times3072}$, $b_1 \in \mathbb{R}^M$, $W_2 \in \mathbb{R}^{10\times M}$, $b_2 \in \mathbb{R}^{10}$. Tune the different hyperparameters and train for a sufficient number of epochs to achieve a *validation accuracy* of at least 50%. Provide the hyperparameter configuration used to achieve this performance.

c. *[15 points]* **Convolutional layer with max-pool and fully-connected output:** for a convolutional layer W_1 with filters of size $k\times k\times3$, and M filters (reasonable choices are $M = 100$, $k = 5$), we have that $\text{Conv2d}(x^{in}, W_1) \in \mathbb{R}^{(33-k)\times(33-k)\times M}$.

   - Each convolution will have its own offset applied to each of the output pixels of the convolution; we denote this as $\text{Conv2d}(x^{in}, W) + b_1$ where b_1 is parameterized in $\mathbb{R}^M$. Apply a `relu` activation to the result of the convolutional layer.

- Next, use a max-pool of size $N \times N$ (a reasonable choice is $N = 14$ to pool to 2×2 with $k = 5$) we have that $\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{in}, W_1) + b_1)) \in \mathbb{R}^{\lfloor \frac{33-k}{N} \rfloor \times \lfloor \frac{33-k}{N} \rfloor \times M}$.
- We will then apply a fully-connected layer to the output to get a final network given as

$$x^{output} = W_2 \text{vec}(\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{input}, W_1) + b_1))) + b_2$$

where $W_2 \in \mathbb{R}^{10 \times M(\lfloor \frac{33-k}{N} \rfloor)^2}$, $b_2 \in \mathbb{R}^{10}$.

The parameters M, k, N (in addition to the step size and momentum) are all hyperparameters, but you can choose a reasonable value. Tune the different hyperparameters (number of convolutional filters, filter sizes, dimensionality of the fully-connected layers, stepsize, etc.) and train for a sufficient number of epochs to achieve a *validation accuracy* of at least 70%. Provide the hyperparameter configuration used to achieve this performance. Make sure to save this model so that you can do the next part.

The number of hyperparameters to tune, combined with the slow training times, will hopefully give you a taste of how difficult it is to construct networks with good generalization performance. State-of-the-art networks can have dozens of layers, each with their own hyperparameters to tune. Additional hyperparameters you are welcome to play with if you are so inclined, include: changing the activation function, replace max-pool with average-pool, adding more convolutional or fully connected layers, and experimenting with batch normalization or dropout.

d. *[10 points]* **Adversarial Attacks:** Modern deep neural networks are brittle and susceptible to small perturbations to their inputs. This gives rise to *adversarial examples*, which are nearly indistinguishable to the human eye but somehow "fool" neural networks into making drastically wrong predictions.

One algorithm to generate such examples is the untargeted fast gradient sign method (FGSM) attack, which can be described as follows: let x be an input image with label y, $\mathcal{F}$ be a neural network, and ϵ be a small value (intuitively, an attack rate).

$$\begin{aligned} \hat{y} &= \mathcal{F}(x) \\ \mathcal{L} &= \text{CrossEntropy}(\hat{y}, y) \\ x' &= x + \epsilon * \text{sign}\left(\nabla_x \mathcal{L}\right) \end{aligned}$$

where $\text{sign}(t) = \frac{t}{|t|}$. We then use x' as an input to the network. Note that the calculation for x' loosely resembles gradient descent. Intuitively, we are slightly adjusting the input image so that the model is less likely to predict its true class.

For this part, use your classifier from part (c) to do the following steps. As always, please provide all code and plots.

1. Select four images from the train set that have been correctly classified. Visualize them and provide their labels.
2. Implement the untargeted FGSM algorithm. Run one iteration on these images and visualize them: they should look like the originals.
3. Provide the predicted labels for your attacked images. You should have at least one image that is incorrectly classified. **Remark:** FGSM is a simple attack, but it's not always effective. In order to generate successful adversarial examples, you may need to try different values of ϵ or even different images, depending on where your classifier excels.
4. Explain the significance of the existence of such adversarial examples.

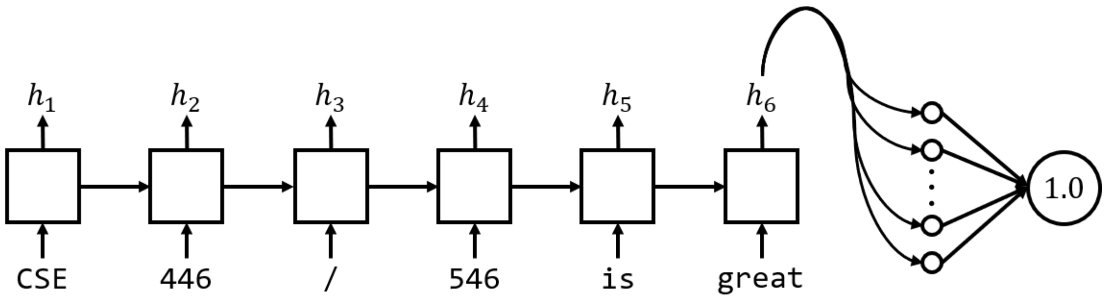
Text classification on SST-2

7. The Stanford Sentiment Treebank (SST-2) is a dataset of movie reviews. Each review is annotated with a label indicating whether the sentiment of the review is positive or negative. Below are some examples from the dataset. Note that often times the reviews are only partial sentences or even single words.

Sequence	Sentiment
is one big , dumb action movie .	Negative
perfectly executed and wonderfully sympathetic characters ,	Positive
until then there 's always these rehashes to feed to the younger generations	Negative
is like nothing we westerners have seen before .	Positive

In this problem you will use a Recurrent Neural Network (RNN) for classifying reviews as either Positive (1) or Negative (0).

Using an RNN for binary classification



Above is a simplified visualization of the RNN you will be building. Each token of the input sequence (*CSE*, *446*, . . .) is fed into the network sequentially. Note that in reality, we convert each token to some integer index. But training with discrete values does not work well, so we also "embed" the words in a high-dimensional continuous space. We already provided an `nn.Embedding` layer to you to do this. Each RNN cell (squares above) generates a hidden state h_i. We then feed the last hidden state into a simple fully-connected layer, which then produces a single prediction between 0 and 1.

Setup

1. Download the GloVe embeddings from `http://nlp.stanford.edu/data/glove.6B.zip`. Extract the zip file and move the file `glove.6B.50d.txt` into the same folder as the starter code (`util.py, train.py, hw4_p7.py`).

2. Download the SST-2 dataset from here `https://gluebenchmark.com/tasks` (Click on download next to `The Stanford Sentiment Treebank`). Extract the zip file into the same folder as above.

You only need to modify `hw4_p7.py`, however you are free to also modify the other two files. You only need to submit `hw4_p7.py`, but if you make changes to the other files you should also include them in your submission.

Problems

a. *[10 points]* In Natural Language Processing (NLP), we usually have to deal with variable length data. In SST-2, each data point corresponds to a review, and reviews often have different lengths. The issue is that to efficiently train a model with textual data, we need to create batches of data that are fixed size matrices. In order to store a batch of sequences in a single matrix, we add padding to shorter sequences so that each sequence has the same length. Given a list of N sequences, we:

   1. Find the length of the longest sequence in the batch, call it `max_sequence_length`

2. Append padding tokens to the end of each sequence so that each sequence has length `max_sequence_length`
3. Stack the sequences into a matrix of size `(N, max_sequence_length)`

In this process, words are mapped to integer ids, so in the above process we actually store integers rather than strings. For the padding token, we simply use id 0. In the file `hw4_a2.py`, fill out `collate_fn` to perform the above batching process. Details are provided in the comment of the function.

b. *[15 points]* Implement the constructor and forward method for `RNNBinaryClassificationModel`. You will use three different types of recurrent neural networks: the vanilla RNN (`nn.RNN`), Long Short-Term Memory (`nn.LSTM`) and Gated Recurrent Units (`nn.GRU`). For the hidden size, use 64 (Usually this is a hyperparameter you need to tune). We have already provided you with the embedding layer to turn token indices into continuous vectors of size 50, but you will need a linear layer to transform the last hidden state of the recurrent layer to a shape that can be interpreted as a label prediction. The code for each of the three networks will only differ by the recurrent layer you use. In your code submission, use any of the three layers.

c. *[5 points]* Implement the `loss` method of `RNNBinaryClassificationModel`, which should compute the binary cross-entropy loss between the predictions and the target labels. Also implemented the `accuracy` method, which given the predictions and the target labels should return the accuracy.

d. *[15 points]* We have already provided all of the data loading, training and validation code for you. Choose an appropriate batch size, learning rate and number of epochs and set the constants `TRAINING_BATCH_SIZE`, `NUM_EPOCHS`, `LEARNING_RATE` accordingly. With a good learning rate, you shouldn't have to train for more than 16 epochs. Report your best validation loss and corresponding validation accuracy, corresponding training loss and training accuracy for each of the three types of recurrent neural networks.

e. *[5 points]* Currently we are only using the final hidden state of the RNN to classify the entire sequence as being either positive or negative sentiment. But can we make use of the other hidden states? Suppose you wanted to use the same architecture for a related task called tagging. For tagging, the goal is to predict a tag for each token of the sequence. For instance, we might want predict the part of speech tag (verb, noun, adjective etc.) of each token. In a few sentences, describe how you would modify the current architecture to predict a tag for each token.