Homework 1
Background, MLP, and CNN[1]

CMU 10-707: Advanced Deep Learning (Spring 2024)

https://machinelearningcmu.github.io/S24-10707

OUT: Wednesday, January 31st, 2024
DUE: Wednesday, February 14th, 2024, 11:59pm
TAs: Jiatai Li, Kaiwen Geng, Torin Kovach

START HERE: Instructions

Homework 1 covers topics on overfitting, dropout, backpropagation and neural networks, and convolutional neural networks basics. The homework includes short answer questions, derivation questions, and coding tasks.

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: https://machinelearningcmu.github.io/S24-10707/index.html#7-academic-integrity-policies
- **Late Submission Policy:** See the late submission policy here: https://machinelearningcmu.github.io/S24-10707/index.html#policies
- **Submitting your work:**
  - **Gradescope:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (https://gradescope.com/). Please write your solution in the LaTeX files provided in the assignment and submit in a PDF form. Put your answers in the question boxes (between `\begin{soln}` and `\end{soln}`) below each problem. Please make sure you complete your answers within the given size of the question boxes. **Handwritten solutions are not accepted and will receive zero credit.** Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. For more information about how to submit your assignment, see the following tutorial (note that

[1] Compiled on Wednesday 31st January, 2024 at 23:13

even though the assignment in the tutorial is handwritten, submissions must be typed): https://www.youtube.com/watch?v=KMPoby5g_nE

- **Code submission:** All code must be submitted to a Gradescope autograder named as "S24 Homework 1: Programming". **If you do not submit your code here, you will not receive any credit for your assignment.** Gradescope grader will be used to check for plagiarism. Please make sure you familiarize yourself with the academic integrity information for this course.

Important Notes on the Programming Problems:

- Do not use any toolboxes except those already imported in the code template.
- Read the doc-strings/comments in the template very carefully before you start.
- Reach out for help on Piazza or during office hours when you struggle.
- Do not change any function signatures because your code will be auto-graded.
- Try to vectorize the computation as much as possible (e.g. compute in the form of matrix multiplication, utilize numpy functions instead of loops, etc.)
- Use Python 3.6 or above, and the latest version of numpy.

Problem 1 Regularization (5 pts)

Consider a dataset $\mathcal{D}$ of N training points $\left(\mathbf{x}^{(n)}, y^{(n)}\right)$ where $\mathbf{x}^{(n)} \in \mathbb{R}^D$ and $y^{(n)} \in \mathbb{R}$, a linear model of the form

$$f(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{D} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b \tag{1}$$

together with a sum-of-squares error function of the form

$$L(\mathbf{w}, \mathcal{D}) = \sum_{n=1}^{N} \left(f(\mathbf{x}^{(n)}, \mathbf{w}) - y^{(n)}\right)^2 \tag{2}$$

Now suppose that Gaussian noise $\epsilon_i \sim N(0, \sigma^2)$ is added independently to each of the input variables x_i. Show that minimizing $L(\mathbf{w})$ averaged over the noise distribution is equivalent to minimizing the sum-of-squares error for noise-free input variables with the addition of a weight-decay regularization term with a certain coefficient λ (that you should find).

In other words, for a perturbed dataset $\mathcal{D}'$ with data points $\mathbf{x}' = \mathbf{x} + \epsilon$ where $\epsilon \sim N(0, \sigma^2 \mathcal{I})$, you need to show

$$\mathbb{E}_\epsilon \left[L(\mathbf{w}, \mathcal{D}')\right] = L(\mathbf{w}, \mathcal{D}) + \lambda \left\|\mathbf{w}\right\|^2 \tag{3}$$

Solution

Problem 2 Dropout (8 pts)

Consider once again a dataset $\mathcal{D}$ of N training points $\left(\mathbf{x}^{(n)}, y^{(n)}\right)$ where $\mathbf{x}^{(n)} \in \mathbb{R}^D$ and $y^{(n)} \in \mathbb{R}$. For this question it will be easier to adopt a matrix notation: let $X \in \mathbb{R}^{N \times D}$ be the usual design matrix containing datapoints as rows, and $\mathbf{y} \in \mathbb{R}^N$ the target vector. Our sum-of-squares loss for our neural network is now written compactly as

$$L(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|^2 \tag{4}$$

Recall the dropout scheme seen in class: any input dimension is retained with probability p and the input can be expressed as $R \odot X$ where $R \in \{0,1\}^{N \times D}$ is a random matrix with $R_{ij} \sim \text{Bernoulli}(p)$ and $\odot$ denotes an element-wise product. Marginalizing the noise, our loss function becomes

$$L_{\text{dropout}}(\mathbf{w}) = \mathbb{E}_R\left[\|\mathbf{y} - (R \odot X)\mathbf{w}\|^2\right] \tag{5}$$

Problem 2.1 (2 pts)

Let $N = D = 1$, so that X and $\mathbf{y}$ are just scalar values, and thus $\mathbf{w}$ is also scalar. Show that dropout with linear regression is equivalent in expectation to a certain form of ridge regression. More specifically, you should show that

$$L_{\text{dropout}}(\mathbf{w}) = \|\mathbf{y} - pX\mathbf{w}\|^2 + p(1-p)\|X\mathbf{w}\|^2 \tag{6}$$

Solution

Problem 2.2 (6 pts)

Show the same statement, but for arbitrary values of N and D. That is, show that

$$L_{\text{dropout}}(\mathbf{w}) = \|\mathbf{y} - pX\mathbf{w}\|^2 + p(1-p)\|\Gamma\mathbf{w}\|^2 \tag{7}$$

where $\Gamma = (\text{diag}(X^T X))^{1/2}$.

Hint: Try proving the case for $N = 1$ and arbitrary values of D, and extend that to datasets of N points.

Solution

Problem 3 Back-propagation (12 pts)

Introduction and Notation

In this question, you will derive the necessary back-propagation operations for an efficient implementation of a feed-forward neural network for classification in Problem 6. Remember that the back-propagation algorithm calculates the gradient of each of the network's parameters to determine by how much to change them to achieve a better loss.

Let $f(x_1, x_2, x_3, \ldots, x_n) = f(\mathbf{x})$ be a scalar output function of multiple scalar inputs, or a scalar output function of a single vector input. Recall the operator ∇, defined as

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{8}$$

In this homework, we will abuse the notation and extend ∇. First let W be a $r \times c$ matrix and $g(W)$ be a scalar output function. Define

$$\nabla_W[g] = \begin{bmatrix} \frac{\partial g}{\partial W_{11}} & \cdots & \frac{\partial g}{\partial W_{1c}} \\ \cdots & & \cdots \\ \frac{\partial g}{\partial W_{r1}} & \cdots & \frac{\partial g}{\partial W_{rc}} \end{bmatrix} \tag{9}$$

(Note, this is not the Hessian, this is just a way to write and refer to each of the partial derivatives.) In addition, suppose $h(\mathbf{x}, \mathbf{y}, W)$ is a scalar function of vectors $\mathbf{x}$,$\mathbf{y}$, and a matrix W. Define

$$\nabla_{\mathbf{x}}[h] = \begin{bmatrix} \frac{\partial h}{\partial x_1} \\ \cdots \\ \frac{\partial h}{\partial x_n} \end{bmatrix} \tag{10}$$

and similarly for $\nabla_{\mathbf{y}}[h]$ and $\nabla_W[h]$.

With these constructs at hand, let us derive back-propagation for a one hidden layer neural network with a softmax output and cross-entropy loss function. Let column vectors $\mathbf{x} \in \mathbb{R}^D$ be a data-point and $\mathbf{y} \in \mathbb{R}^M$ be a one-hot encoding of the the corresponding label. Consider the neural network defined by the following equations.

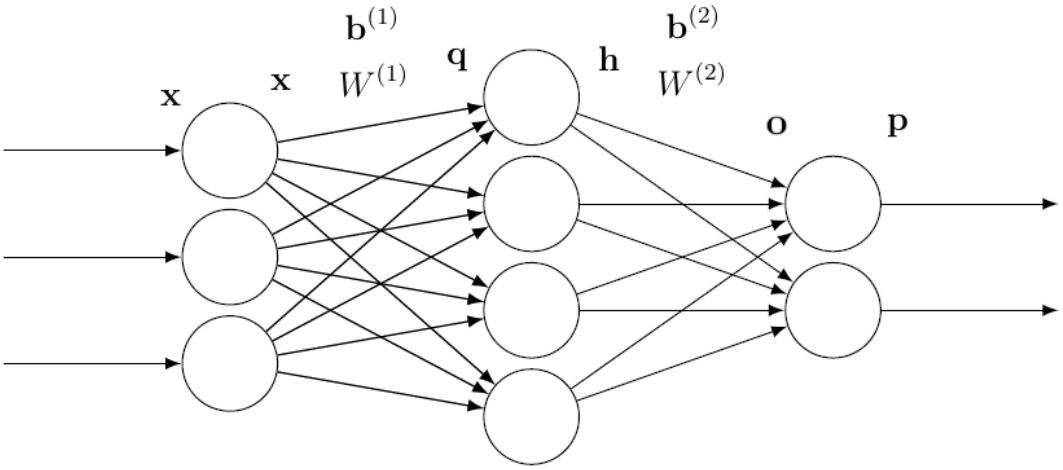


Figure 1: One layer fully connected neural network

$$\mathbf{q} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{11}$$

$$\mathbf{h} = \mathrm{ReLU}(\mathbf{q}) = \max(0, \mathbf{q}) \qquad \text{which is applied element-wise} \tag{12}$$

$$\mathbf{o} = W^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \tag{13}$$

$$\mathbf{p} = \mathrm{softmax}(\mathbf{o}) \qquad \text{which is defined as } p_i = \frac{e^{o_i}}{\sum_{k=1}^{M} e^{o_k}} \tag{14}$$

$$L(\mathbf{p}, \mathbf{y}) = -\sum_{i=1}^{M} y_i \log(p_i) \tag{15}$$

Note that $W^{(1)} \in \mathbb{R}^{H \times D}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{H}$, $W^{(2)} \in \mathbb{R}^{M \times H}$ and $\mathbf{b}^{(2)} \in \mathbb{R}^{M}$.

Our ultimate goal is to calculate the gradients of the loss function with respect to the parameters $W^{(1)}, \mathbf{b}^{(1)}, W^{(2)}, \mathbf{b}^{(2)}$.

Problem 3.1 (3 pts)

In these sections, you may find it helpful to use the Kronecker delta (https://en.wikipedia.org/wiki/Kronecker_delta) as a shorthand. First, derive each of the following using chain rule:

$$\frac{\partial p_i}{\partial o_j}, \frac{\partial L}{\partial o_j}, \frac{\partial o_i}{\partial b_j^{(2)}}, \frac{\partial L}{\partial b_j^{(2)}} \tag{16}$$

Solution

Then, show that (by showing each element of the vectors are equal on both sides)

$$\nabla_{\mathbf{o}}[L] = \mathbf{p} - \mathbf{y} \tag{17}$$
$$\nabla_{\mathbf{b}^{(2)}}[L] = \mathbf{p} - \mathbf{y} \tag{18}$$

Solution

Problem 3.2 (3 pts)

Derive the following using chain rule

$$\frac{\partial o_i}{\partial h_j}, \frac{\partial L}{\partial h_j} \tag{19}$$

Solution

Then, show that (by showing each element of the vectors are equal on both sides)

$$\nabla_{\mathbf{h}}[L] = W^{(2),\top} \nabla_{\mathbf{o}}[L] \tag{20}$$

Note $W^{(2),\top}$ is the transpose of $W^{(2)}$

Solution

Problem 3.3 (3 pts)

Derive the following using chain rule

$$\frac{\partial o_k}{\partial W_{ij}^{(2)}}, \frac{\partial L}{\partial W_{ij}^{(2)}} \tag{21}$$

Solution

Then, show that (by showing each element of the matrices are equal on both sides)

$$\nabla_{W^{(2)}}[L] = \nabla_{\mathbf{o}}[L]\mathbf{h}^\top \tag{22}$$

Solution

Problem 3.4 (3 pts)

Derive the following using chain rule. The second one should be in terms of $\frac{\partial L}{\partial h_i}$

$$\frac{\partial h_i}{\partial q_j}, \frac{\partial L}{\partial q_j} \tag{23}$$

Solution

With these expressions at hand, you should be equipped to implement Problem 6 efficiently. The derivative of $\mathbf{q}$ with respect to $\mathbf{x}$, $W^{(1)}$ and $\mathbf{b}^{(1)}$ follows in the same way as $\mathbf{o}$ with respect to $\mathbf{h}$, $W^{(2)}$ and $\mathbf{b}^{(2)}$.

Problem 4 Convolution and Pooling (10 Points)

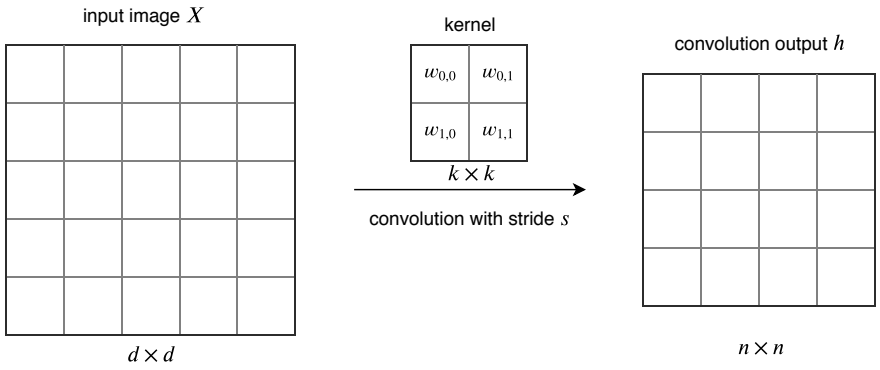


Figure 2: A simple example of convolution.

In this problem, you will work on derivations of convolution. You will also get familiar with `im2col` (image to column), a popular technique to accelerate convolution. For this problem, assume padding = 0.

Problem 4.1 (3 pts)

In lecture, you learned about two types of pooling as forms of downsampling: Max Pooling and Average Pooling

1. Please show that **Average Pooling** is a linear operator by describing how it can be represented as a Convolution. In particular, for a square pooling field of shape (p, p), please define the kernel matrix A and stride s needed to **implement Average Pooling as Convolution** for a one-channel input.
2. Additionally, please explain why **Max Pooling** cannot be implemented as a convolution.
3. Please say which you think would perform better and why: a CNN made up of **Convolutions and Max Pools** or a CNN made up of **Convolutions and Average Pools**.

Solution

Problem 4.2 (4 pts)

Let the input $X \in \mathbb{R}^{d \times d}$ be one single image, and is convolved by a $k \times k$ kernel with stride s. A simple example is shown in Figure 2.

a. **(1 pts)** The convolution output is in the shape of $n \times n$. Can you represent the output dimensionality n in terms of d, k and s?

Solution

b. **(1 pts)** Denote the loss as L, convolution output as h, and kernel parameters as w. During backpropagation, given $\frac{\partial L}{\partial h_{i,j}}$ and input X, please derive $\frac{\partial L}{\partial w_{a,b}}$.

Solution

c. **(2 pts)** Can the above derivative be represented as a convolution? If so, write down its expression and define the corresponding kernel.

Solution

Problem 4.3 (3 pts)

In this part, you will be walked through `im2col` (image to column), and you are encouraged to use this technique in the programming exercise. Naively, an simple implementation of convolution would be sliding the kernel over the feature map. However, this would result in loops of all kernels and all locations, which is slow and can be accelerated by constructing two big matrices and going through one single matrix multiplication.

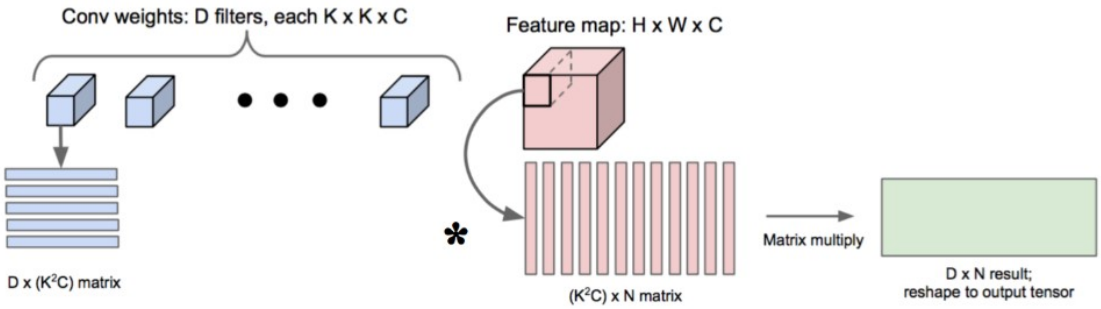


Figure 3: Illustration of im2col.

An illustration of im2col is shown in Figure 3. A patch inside the feature map corresponds to a possible location of the kernel. Each patch is of the same size as the convolution kernel, K^2C, and N is the number of all possible locations, or in other words, the spatial size of the output. We can take all possible patches, and reshape them into column vectors, then concatenate the vectors into one big matrix. Similarly, we can reshape the kernels into row vectors, and concatenate them into another one. Then we only need to go through one single matrix multiplication. And reshape the result to output shape.

For example, if the input feature map is of shape $32 \times 32 \times 3$, and we apply 5 filters each of size $5 \times 5 \times 3$ with stride=1 and padding=2, then the filter matrix would be of size 5×75, and image matrix would be of size 75×1024. The multiplication result would be of size 5×1024, and we need to reshape the result to shape $32 \times 32 \times 5$. (Hint: be very careful during reshaping.)

a. **(2 pts)** Consider a toy example where the input feature map is of size 3×3, and convolved by 2 kernels of size 2×2 with stride $s = 1$, as shown in Figure 4, please write the two constructed matrices with im2col.

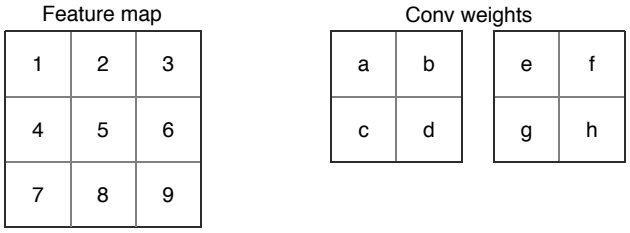


Figure 4: A toy example of im2col

Solution

b. **(1 pts)** Give one significant drawback of im2col.

Solution

Problem 5 MultiLayer Perceptron (30 pts)

In this problem we will implement multi-layer perceptron for an image classification task. The data we use here is a subset of the CIFAR-10 dataset[2], where there are 10 classes of 32×32 images.

Warning: It takes *multiple hours* to train all of the networks. Please start early and leave ample time for training/debugging. Be sure to *vectorize* all of your computations to ensure they can run fast enough (so make computations in the form of vector and matrix multiplications as much as possible instead of for loops).

In **Problem 5 and 6**, you will first implement several classes of important neural network modules, and then use these modules to build up the networks. For this part, we provide a template code. Please FOLLOW the templates, and implement the classes methods. Do NOT change the interface. The classes will be used for auto-grading. Code that does not pass the autograder WILL NOT be graded.

Format of the data: The dataset are stored in a pickle file (`cifar10-subset.pkl`), which can be downloaded from the "Resources" tab on piazza. There are 5000 images for training and 2000 images for testing, each corresponding to an "integer" label (in range 0 to 9). Once you download the dataset, replace `CIFAR_FILENAME` in the main function by the relative path you store the pickle file. `trainX` is a numpy array of dimension $(5000, 3072)$ representing the 5000 training images, and `trainy` is a $(5000, 10)$ dimensional numpy array of one-hot vectors representing the corresponding label information. Similarly, `testX` and `testy` contain 2000 data points and use the same format as the training dataset (i.e, $(2000, 3072)$ dimensional array and $(2000, 10)$ dimensional array). Normalizing the input (`trainX`, `testX`) to [0, 1] and converting the input to numpy array of float32 can accelerate training.

The label mappings are:

```
{
  0: 'Airplane',
  1: 'Automobile',
  2: 'Bird',
  3: 'Cat',
  4: 'Deer',
  5: 'Dog',
  6: 'Frog',
  7: 'Horse',
  8: 'Ship',
  9: 'Truck',
}
```

As a warm up practice (not graded), we recommend loading the data and plotting a few examples.

[2] https://www.cs.toronto.edu/~kriz/cifar.html

For problems 5 and 6, submit `mlp.py` and `cnn.py` only to gradescope (select them simultaneously and drag them to upload).

Problem 5.1 Implementation (15 pts)

In the Problem 5 folder, **mlp.py** is the only file you need to work on. **tests.py** and **tests.pk** are to help you test your implementation locally. Passing these local tests does not guarantee you passing the final online auto-grading tests, but failing locally very likely means also failing online. The following are terminal commands to run single module test and all modules test.

```
python -m unittest tests.TestReLU
python -m unittest tests
```

In **mlp.py**, there is a base class called `Transform` which you do not need to change. The `Transform` class represents a procedure done to some input X. In symbolic terms, $out = f(X)$, where f represents the transformation, y the output, x the input. We will be implementing the basic layers (e.g. `ReLU`, `LinearMap`, etc.) of neural networks by inheriting this `Transform` base class.

Below is a list of classes we will implement. Implement wherever the code template says `pass`. Reading `tests.py` might help you debug your implementation.

- **ReLU** (**1 point**): The layer of Rectified Linear Unit, a popular activation function. Available unit tests: `TestReLU`;
- **LinearMap** (**3 points**): Linear Transformation layer, i.e. fully-connected layer. In parameter update function `step()`, parameters should be updated using gradient descent with momentum. Available unit tests: `TestLinearMap`;
- **SoftmaxCrossEntropyLoss** (**2 points**): The layer of softmax and cross-entropy loss. The input is the pre-softmax logits, and the output is the **mean** cross-entropy loss across samples in a batch. Available unit tests: `TestLoss`;
- **SingleLayerMLP** (**3 points**): This is a nerual network with one hidden layer. The output is the logits(pre-softmax) for the classification tasks. Available unit tests: `TestSingleLayerMLP`;
- **TwoLayerMLP** (**3 points**): This is a neural network with two hidden layers. The output is the logits(pre-softmax) for the classification tasks. Available unit tests: `TestTwoLayerMLP`.
- **Dropout** (**1 point**): The dropout layer. In this assignment, we apply mask and scaling during training and do nothing at test time. Available unit tests: `TestDropout`.
- **BatchNorm** (**2 points**): The batch normalization layer. Please refer to the batch normalization paper or this[3] for implementation. Available unit tests: `TestBatchNorm`.

[3] https://agustinus.kristia.de/techblog/2016/07/04/batchnorm/

Tips:

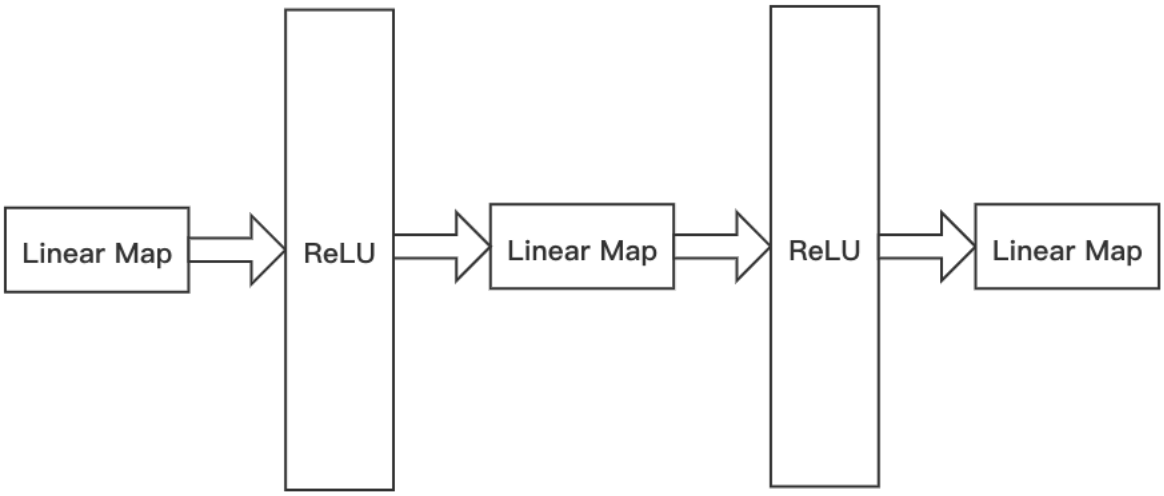
- LinearMap layer weights should be initialized using Xavier Initialization:

$$W_{i,j}^{k} \sim \texttt{Uniform}(-b, b) \ , \quad b = \sqrt{\frac{6}{m+n}}$$

  where k, i, j are indices for network layer and nodes, m and n are the input and output dimension;

- Special practices might be necessary for the numerical stability of the Softmax function;

This is what the TwoLayer network would look like:



Problem 5.2 Experiments (15 pts)

In this section, we will train and test the two networks we implemented, and evaluate their performances.

The network models will take images as inputs and give softmax output over 10 classes. During training, we will perform gradient descent with momentum to minimize Cross-Entropy Loss. Run the optimization for 50 epochs each time. Using the following hyperparameters parameters for training: batch size = 128, learning rate = 0.001, momentum = 0.9.

For every architecture, plot the train and test loss together on one plot, with loss on the y-axis against epoch number on x-axis. Similarly, plot the train and test accuracy after every epoch. Label each curve and all axes. Report the best loss and accuracy for training and testing achieved.

1. **(5 pts)** Train a **SingleLayerMLP** with 800 hidden nodes, show plots of loss and accuracy, and report the best loss and accuracy achieved.

Solution

2. **(5 pts)** Train a **TwoLayerMLP** with $(800, 800)$ hidden nodes, show plots of loss and accuracy, and report the best loss and accuracy achieved.

Solution

3. **(3 pts)** Batch Normalization: To the **SingleLayerMLP** in Problem 1, implement batch normalization using the same batch size as in Problem 1. Describe how batch norm helps (or doesn't help) in terms of speed and accuracy (train and validation).

Solution

4. **(2 pts)** Dropout: Now, to the **SingleLayerMLP** of Problem 1, add a dropout to the output of layer 1 with a probability of 0.5 and report your findings. Do you observe any changes in terms of performance? Does a model trained with dropout perform better or worse than the SingleLayerMLP? Briefly explain why you think this happens.

Solution

Problem 6 Convolutional Neural Network (35 pts)

In this problem we will implement convolutional neural networks for an image classification task. The data is the same as in the previous problem.

Again, the dataset is stored in a pickle file, with 5000 images for training and 2000 for testing. Once you download the dataset, replace `CIFAR_FILENAME` in the main function with the path where you stored the dataset and it would load the data for you. This time, the data is prepared in the following format:

`trainX` / `testX` – $(5000/2000, 3, 32, 32)$ numpy array of images normalized to [0, 1].
`trainy` / `testy` – $(5000/2000, 10)$ numpy array of one-hot vectors representing labels.

Problem 6.1 Implementation (16 pts)

In the Problem 6 folder, **cnn.py** is the only file you need to work on. **tests.py** and **tests.pk** are to help you test your implementation locally. Passing these local tests does not guarantee you passing the final online auto-grading tests, but failing locally very likely means also failing online. The following are terminal commands to run single module test and all modules test.

```
python -m unittest tests.TestReLU
python -m unittest tests
```

Below is a list of classes we will implement. Implement wherever the code template says `pass`. Reading `tests.py` might help you debug your implementation.

- **ReLU** (**0 points**): ReLU layer. Available unit tests: `TestReLU`.
- **Conv** (**7 points**): Convolutional layer. You will also need to implement functions **im2col** and **im2col_bw** in order to vectorize the forward and backward computation of convolutional layer [4]. Available unit tests: `TestIm2Col`, `TestConvWeightsBias`, `TestConvForward`, `TestConvBackward` and `TestConvUpdate`.
- **Maxpool** (**4 points**): Max Pooling layer. Available unit tests: `TestMaxPoolForward` and `TestMaxPoolBackward`.
- **LinearLayer** (**2 points**): Fully Connected layer, or Linear Layer. Available unit tests: `TestFCWeightsBias`, `TestLinearForward`, `TestLinearBackward` and `TestLinearUpdate`.
- **SoftMaxCrossEntropyLoss** (**0 points**): The layer of softmax and cross-entropy loss. The input is the pre-softmax logits, and the output is the **mean** of cross-entropy loss across samples in a batch.
- **ConvNet** (**3 points**): This is where you will initialize and call all the previous layers to implement the convolutional network. Available unit tests: `TestConvNet`.

[4] The following links from the cs231n course at Stanford are very helpful: http://cs231n.stanford.edu/slides/2016/winter1516_lecture11.pdf(starting on slide 66) and http://cs231n.github.io/convolutional-networks/

Tips:

- Linear layer weights initialization is the same with Problem 5. Convolutional layer weights should be initialized as:

$$W_{i,j}^{k} \sim \texttt{Uniform}(-b, b) \ , \quad b = \sqrt{\frac{6}{(f+c) * h * w}},$$

  where k, i, j are indices for network layer and nodes, f the number of filters (the number of output channels), c the number of input channels, h and w the filter height and width.

Problem 6.2 Experiments (19 pts)

In this section, you will implement different network architectures to see how they affect the performance.

The network models will take images as inputs and give softmax output over 10 classes. During training, we will perform gradient descent with momentum to minimize Cross-Entropy Loss. Run the optimization for 50 epochs each time. Using the following hyperparameters parameters for training: batch size = 128, learning rate = 0.001, momentum = 0.9.

1. **(12 pts)** For every architecture, plot the train and test loss together on one plot, with loss on the y-axis against epoch number on x-axis. Similarly, plot the train and test accuracy after every epoch. Label each curve and all axes. Report the best loss and accuracy for training and testing achieved.

   For those with CNN layers, the stride is 1 and the padding size is 2.

(a) **1conv1filter (3 pts)** A convolutional layer with one 6×6 filter and ReLU activation, a 2×2 max pooling layer, a linear layer, a softmax layer.

Hint: This simple architecture is mostly for helping you debug your convolutional layer. There is no need to worry about poor performance as long as it significantly improves for the next two architectures.

Solution

(b) **1conv5filter (3 pts)** A convolutional layer with five 4×4 filters and ReLU activation, a 2×2 max pooling layer, a linear layer, a softmax layer.

Solution

(c) **3conv5filter (3 pts)** A convolutional layer with five 4×4 filter and ReLU activation, a 2×2 max pooling layer, a convolutional layer with five 4×4 filter and ReLU activation, a convolutional layer with five 4×4 filter and ReLU activation, a linear layer, a softmax layer.

Solution

(d) **Full comparison between MLP and CNN (3 pts)** Plot the test accuracy for all the experiments you have done so far for both MLP and CNN. Also, explain why you think one has better performance than the other.
Hint: Look at the number of parameters of each network.

Solution

2. **(7 pts)** Compare different architectures based on your own results.

   (a) Compare (a) and (b). (2 pts)

   Solution

   (b) Compare (b) and (c). (2 pts)

   Solution

   (c) Compare the best CNN model with the best MLP model. (2 pts)

   Solution

   (d) Which one is your best performing CNN network (on the test dataset)? (1 pt)

   Solution

Collaboration Questions Please answer the following:

After you have completed all other components of this assignment, report your answers to the collaboration policy questions detailed in the Academic Integrity Policies found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? Is so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? Is so, include full details.
3. Did you find or come across code that implements any part of this assignment ? If so, include full details even if you have not used that portion of the code.

Solution