CS69000-DPL Spring 2021 Homework	— Homework 5—	Last update: April 24, 2021
https://www.cs.purdue.edu/homes/ribeirob/courses/Spring2021/index.html		Due: **6:00am**, Saturday, April 24th

Instructions and Policy: Each student should write up their own solutions independently, no copying of any form is allowed. You MUST indicate the names of the people you discussed a problem with; ideally you should discuss with no more than two other people.
Chowdhury Mohammad Rakin Haider
You need to submit your answer in PDF. LaTeX is typesetting is encouraged but not required. Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.

Q0 (0pts correct answer, -1,000pts incorrect answer: (0,-1,000) pts): A correct answer to the following questions is worth 0pts. An incorrect answer is worth -1,000pts, which carries over to other homeworks and exams, and can result in an F grade in the course.

1. Student interaction with other students / individuals:
   (a) I have copied part of my homework from another student or another person (plagiarism).
   (b) Yes, I discussed the homework with another person but came up with my own answers. Their name(s) is (are) ____________________________
   (c) No, I did not discuss the homework with anyone
2. On using online resources:
   (a) I have copied one of my answers directly from a website (plagiarism).
   (b) I have used online resources to help me answer this question, but I came up with my own answers (you are allowed to use online resources as long as the answer is your own). Here is a list of the websites I have used in this homework:
   Wikipedia, MLWiki, StackOverflow.______________________
   (c) I have not used any online resources except the ones provided in the course website.

Homework 5

Q1: Conceptual Questions (5pts)

1. (**1.0 pt**) Detail how we can replace the equivariant representation of the Transformer architecture with an *equivariant representation* using an LSTM architecture. Specifically, replace the representation of the i-th word of the m-th self-attention head, $z_i^{(m)}$, of our Transformer lecture with a corresponding LSTM output.
   Hint: Make sure the representation coming out of the LSTM, $h_1, \ldots, h_n$ for a n-th word sentence, is equivariant (see Backpropagation-through-time lecture where we define the LSTM variables and our lecture on set representation for a procedure to make set representations out of LSTMs).

   Solution: We can replace the representation of the i-th word of the m-th self attention model $z_i^{(m)}$ of the Transformer with the hidden states, $h_1, h_2, \ldots, h_n$ of the LSTM. But the problem is that the representation of ith word of the mth self-attention head need to be a permutation equivariant representation. However, the representation obtained from LSTM are sequence representaiton and therefore they are not permutation equivariant. To solve this problem we can use Janossy Pooling to convert the sequence representation to a permutation equivariant representation. That means, given a sequence $x_1, x_2, \ldots, x_n$, we feed this all permutation of this sequence through LSTM and obtain the average of the hidden states to obtain the equivariant representation of the ith word of the mth self-attention head.

$$z_i^{(m)} = \sum_{\pi \in \Pi} LSTM^{(m)}(x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_n}).h_{\pi_i}$$

   In the equation $LSTM(x).h_i$ means the ith hidden state of the output of LSTM after feeding x. The equation means, we obtain each sequence representation of the ith word for each permutation of the sequence x. Then we obtain the average to find the equivariant representation of the ith word.

2. (**1.0 pt**) (a) Explain one of the main shortcomings of RNNs and how Bi-directional RNNs help ameliorate this shortcoming. (b) Are Bi-directional LSTMs able to represent well dependencies between the first and the last elements of a very long sequence? Why or why not?

   Solution: The main shortcomings of RNNs is that the forward pass of RNN is not a proper inference of the sequence because the forward pass of RNN cannot handle future dependencies. That means, in the unrolled HMM representation of the RNN, every hidden state H_t depends on the observations X_{t+1} to X_n. However, the forward pass cannot only account for the previous dependencies i.e. dependency of H_t on $X_{<start>}$ to X_t. The bidirectional RNN ameliorates this shortcoming by maintaining a forward and a backward RNN so that they can model the forward and backward dependencies.

   No, Bi-directional LSTMs are not able to represent dependencies between first and last elements of very long sequence well. This is because, the large sequence of multiplication of gradients results in vanishing or exploding gradient problems and therefore, the Bi-directional RNNs are not able to learn dependencies between first and last elements of very long sequences well.

3. (**1.0 pt**) Describe two common fixes used to treat class imbalances in supervised learning tasks as domain adaptation techniques: (i) loss re-weighting for full batch training and (ii) sampling the same number of class examples for SGD. What is the assumption they make about the test data?

   Solution: To treat class imbalances in supervised learning task as domain adaptation, loss re-weighting for full batch training modifies the average loss over samples of $X \sim p(x)$ and $y \sim p^{(tr)(y)}$ into an average loss over samples from $X \sim p(x)$ and $Y \sim p^{(te)(y)}$ as follows,

$$\mathcal{L}'(\mathcal{D}^{(tr)}) \propto \frac{1}{n^{(tr)}} \sum_{i=1}^{n^{(tr)}} \frac{p^{(te)}(y_i^{(tr)})}{p^{(tr)}(y_i^{(tr)})} \mathcal{L}(y_i^{(tr)}, g(\Gamma(x_i^{(tr)}))$$

   The assumption of using this method is that we know the class distribution in test set, $p^{te}(y)$, at the training time.

   On the other hand, treating class imbalances in supervised learning task as domain adaptation, can also be performed by sampling. In this technique, the same number of examples are sampled from the majority and minority classes by either replicating minority class or discarding members of majority class is another technique to handle class imbalance. The assumption of this method is all the classes in the test dataset has equal probability. That is $p^{(te)}(y) = \frac{1}{|\mathbf{Y}|}$.

4. (**1.0 pt**) (Supervised learning) Explain adversarial learning as a domain adaptation problem. That is, show that any defense against adversarial attacks is effectively adapting to a test data distribution different from the training data distribution. Explain why protecting against attacks without knowing the resulting test distribution is likely going to fail (the classifier will give significantly more errors in the adversarial test data than if the test data were sampled from the training data distribution).
Hint: Interesting extreme cases are (i) when for a trained model $\mathbf{W}^\star$ there exists an x in the domain for which, if y is the correct class of x, then $p(y|x;\mathbf{W}^\star) < 1/C$, where C is the number of classes; and (ii) when for all x in the domain then $\exists y$ for which $p(y|x) = 1$.

Solution: In adversarial learning, the defender trains the model where,

Training data: $\mathcal{D}^{(tr)} = \{(x_i^{(tr)}, y_i^{(tr)})\}_{i=1}^{N^{(tr)}}$ sampled from $(x, y) \sim p(y|x)p^{(tr)}(x)$

The adversary's goal is to modify the test samples so that the defenders model is fooled. So the adversary modifies the distribution from which the test samples are drawn. That means, the test data is as follow.

Test Data: $\mathcal{D}^{(te)} = \{x_i^{(te)}\}_{i=1}^{N^{(te)}}$ sampled from $x \sim p^{(te)}(x)$.

In most cases, $p^{(te)}(x) \approx p^{(tr)}(x)$. Or often the adversarial samples are obtained by transforming the image samples i.e $p^{(te)}(x) \approx p^{(tr)}(T^{-1}(x))$.

The goal of adversarial learning is to train robust models such that the prediction are invariant of the transformation. That means, $P(y|x^{(tr)}; W^*) = P(y|T(x^{(tr)}); W^*)$. In other words, if $P(y|x; W^*) = g(\Gamma(x; W_\Gamma^*); W_g^*)$, then we want to learn $\Gamma(x; W_\Gamma^*) = \Gamma(T(x); W_\Gamma^*)$ equivalently $\Gamma(X^{(tr)}; W_\Gamma^*) \overset{d}{=} \Gamma(X^{(te)}; W_\Gamma^*)$ where $X^{(tr)} \sim p^{(tr)}(x)$ and $X^{(te)} \sim p^{(te)}(x)$. This is similar to the goal of covariate shift adaption task.

5. **(1.0 pt)** Unsupervised learning seeks to learn the distribution that generated the training data.

(a) (**0.5 pt**) Give the pseudocode of rejection sampling. Now use your normalization factor (M in the algorithm described in class) to explain why a good proposal distribution is important for performance.

(b) (**0.5 pt**) A GAN is a type of rejection-sampling algorithm using a classifier for the rejection. Answer the following three questions:
(a) Why are GANs more efficient at generating observations than traditional rejection sampling methods? Give a mathematical argument.
Hint (a). Focus on the acceptance probability M of rejection sampling.

(b) Give one drawback of GANs compared to a traditional rejection sampling algorithm (describe the statistical problem)?
Hint (b): The mode collapsing problem of GANs can be described by the probability of sampling x. Describe the probability of sampling an example X and then explain why the GAN formulation does not prevent mode collapse w.r.t. the support of X.

(c) Propose a procedure to fight mode collapse on GANs.
Hint (c): You need to find a way to penalize the proposal if it can't produce adequate support.
Solution: The following are answer in the context of unsupervised learning which seeks learn the distribution that generated the training data.

(a) The pseudocode of rejection sampling is as follows.

```
Algorithm 1: Rejection Sampling
  Input   : q(x): proposal distribution; p(x): target distribution; M: constant such that $M \geq \sup \frac{p(x)}{q(x)}$
  Output: Generated Sample
  Sample $x \sim q(x)$;
  Sample $u \sim Uniform(0, 1)$;
  while $u > \frac{p(x)}{Mq(x)}$ do
      Sample $x \sim q(x)$;
      Sample $u \sim Uniform(0, 1)$;
  end
  return $x$
```

In this algorithm, if M is very high that means the ratio $\dfrac{p(x)}{Mq(x)}$ is likely to be very small. In this case, we will be rejecting most of the samples generated. Therefore, to get an accepted sample using the rejection sampling we will have to sample a large number of times. The value of M can be large if the proposal distribution is far from the target distribution. Therefore, in order to obtain good performance from rejection sampling we need to start with a proposal distribution that is close to the target distribution.

(b) (a) GAN uses a parameterized proposal distribution and optimizes the proposal distribution in a way so that it mimics the target distribution. Therefore, even if the proposal distribution is initially very bad and M is very high, it improves the proposal distribution is consecutive iteration and gets better proposal distribution and lowers the value of M. Therefore, after a few iteration GAN finds a proposal distribution such that $q(x) \sim p(x)$ and thus it will require $M \sim 1$. In this situation, we will need on average 1 sample to obtain an accepted sample. This is why GAN is more efficient than rejection sampling.

(b) A problem of GAN is mode collapse. Mode collapse is the problem where GAN fails to generate some sample in the support of $p(x)$. This happens when we start with a proposal distribution such that support of target distribution is not a subset of the support of the proposal distribution. Let, $x \in supp(p)$ but $x \notin supp(q)$. Then, the proposal distribution will not generate any samples of x and therefore, there will be no samples of x for which the classifier makes a mistake. As a result there will be no gradient to update $q(x) = 0$. And $q(x)$ will always remain zero. So, GAN will not be able to generate any sample of x.

(c) In order to fight mode collapse we can use some divergence metric as part of the loss. That means, if p(x) and q(x; w*) are far from each other then the divergence will be high and loss will be high. Using the gradient of loss, q(x; w*) will be modified in a way to be more similar to p(x).

Q2: Word Embeddings using 2nd order Markov Chains (1pt)

In class, we have seen that a word2vec model will define word embeddings using the SKIPGRAM model with window K (assumed to be an odd number) as a 1st order Markov chain. The negative log-likelihood (NLL) is (not accounting for the corner cases at the end and beginning of a sentence):

$$\mathcal{L}_{\text{word2vec}} = -\sum_{t=(K-1)/2}^{n-(K-1)/2} \sum_{k=1}^{(K-1)/2} (\log p(x_{t-k}|x_t;\mathbf{U},\mathbf{V}) + \log p(x_{t+k}|x_t;\mathbf{U},\mathbf{V})),$$

and the probability function is

$$p(x|x_t;\mathbf{U},\mathbf{V}) = \frac{1}{Z(x_t;\mathbf{U},\mathbf{V})}\exp(\langle \mathbf{U}x, \mathbf{V}x_t\rangle),$$

with appropriately-defined one-hot encoding of x_i, $i = 1,\dots,n$, and $Z(x_t;\mathbf{U},\mathbf{V})$ is a normalization function.

Task (1pt): Write down the NLL loss function and the *probability function* of a word embedding model that, again using a window of size K similar to the original SKIPGRAM approach, can embed words based on a 2nd order Markov chain.

Hint 1: We must change the original SKIPGRAM model since the dataset it creates is for a 1st order Markov chain.
Hint 2: There are multiple ways to define the model p. Note that you will need more parameters. Also pay attention that your 2nd order Markov chain cannot be described by a 1st order Markov chain.

Solution: For dataset created by the SKIPGRAM mode using 2nd order markov chain will have the following structure.

$\mathcal{D} = \{(x_{i-K}, x_{i-K+1}, \dots x_{i-1}, x_{i+2}, \dots, x_{i+K+1}), x_i, x_{i+1}\}_{i=1}^{N}$

If we further break it down to 2 element input output pairs we will get,

$\mathcal{D} = \{((x_i, x_{i+1}), (x_{i-K}, x_{i-K+1})), ((x_i, x_{i+1}), (x_{i-K+1}, x_{i-K+2})), \dots, ((x_i, x_{i+1}), (x_{i+K}, x_{i+K+1})\}_{i=1}^{N}$

The SKIPGRAM 2nd order Markov chain can be represented as a graph where each pair (x_i, x_{i+1}) are nodes of the graph and the edge weights are transition probabilities. We can model the 2nd-order markov chain as,

$$p(x_{t+1}|x_t, x_{t-1}, x_{t-2}; U, V) = \frac{exp(< U(x_{t+1}, x_t), V(x_{t-1}, x_{t-2}) >)}{Z(x_t, x_{t-1}, x_{t-2}; U, V)}$$

Here, U and V are parameters of size $n^2 \times d$ where d is the size of the embedding and Z is a normalizing factor..

And the NLL loss function is as follows,

$$\mathcal{L}_{\text{word2vec}} = -\sum_{t=(K-1)/2}^{n-(K-2)/2} \sum_{k=1}^{(K-1)/2} (\log p(x_{t-k}|x_{t-k+1}, x_t, x_{t+1};\mathbf{U},\mathbf{V}) + \log p(x_{t+k}|x_{t+k-1}, x_t, x_{t+1};\mathbf{U},\mathbf{V})),$$

Q3: Domain Adaptation with Maximum Mean Discrepancy (MMD) (4pts)

In this homework, you are faced with the problem of unsupervised domain adaptation. You are provided a data set of labeled images (CIFAR-10), which we'll call the *source* domain. Your task is to predict the labels for a *target* domain, in which the images have their color shifted. See an example in Figure 1.

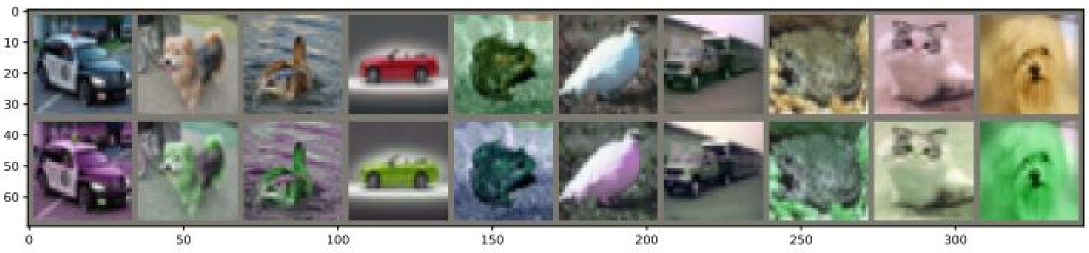


Figure 1: Example of images. *Top row*: Source domain, CIFAR-100. *Bottom row*: Target domain, shifted hue.

To aid you in this task, you also have access to *unlabeled* images from the target domain to use during training. In a more formal description:

- Training data: $\mathcal{D}^{(\text{tr})} = \{(x_i^{(\text{source})}, y_i^{(\text{source})})\}_{i=1}^m \cup \{x_i^{(\text{target})}\}_{i=1}^n$, where:
  - $(x_i^{(\text{source})}, y_i^{(\text{source})})$ are sampled iid from $p^{(\text{source})}(x)p(y|x)$
  - $x_i^{(\text{target})}$ are sampled iid from $p^{(\text{target})}(x)$
- Test data: $\mathcal{D}^{(\text{te})} = \{(x_i^{(\text{target})}, y_i^{(\text{target})})\}_{i=1}^{n^{(\text{te})}}$, where:
  - (x_i, y_i) are sampled iid from $p^{(\text{target})}(x)p(y|x)$.

For this homework, we will use the Maximum Mean Discrepancy (MMD) divergence to help us learn representations which are useful in both domains. Denote by $f(\Gamma(x; \boldsymbol{W}_\Gamma); \boldsymbol{W}_f)$ our neural network classifier, where $\Gamma(x; \boldsymbol{W}_\Gamma)$ is the part of the neural network which is tasked with learning the representations and $f(\cdot, \boldsymbol{W}_f)$ is the part of the neural network which is tasked on classifying an image given its representation.

We modify our loss function to incorporate the MMD as a regularization term:

$$\mathcal{L}(\mathcal{D}^{(\text{tr})}; \boldsymbol{W}_f, \boldsymbol{W}_\Gamma) = \mathcal{L}_{CE}(\mathcal{D}^{(\text{tr})}; \boldsymbol{W}_f, \boldsymbol{W}_\Gamma) + \lambda \cdot \text{MMD}^2(\mathcal{D}^{(\text{tr})}; \boldsymbol{W}_\Gamma),$$

where $\mathcal{L}_{CE}$ is the usual cross-entropy loss and λ is a hyperparameter.

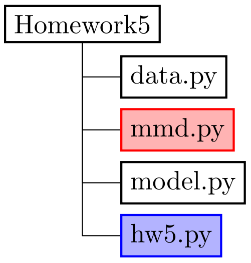
We will be implementing MMD with the Gaussian kernel:

$$k(\boldsymbol{z}, \boldsymbol{z}') = \exp\left(-\frac{\|\boldsymbol{z} - \boldsymbol{z}'\|_2^2}{2\sigma^2}\right),$$

where σ is a hyperparameter of the kernel. We will apply the MMD to the representations computed by $\Gamma(\cdot; \boldsymbol{W}_\Gamma)$ for both domains, with the goal of learning a representation that is invariant to both domains. To compute the MMD term, we will use the unbiased estimator of **?** (Eq. 3, Lemma 6).

Skeleton

This HW is a combination of two distinct tasks. You are going to fill in a few new components to the HW5 package given on Piazza.
Here is the folder structure that you should use:



- **Homework5:** the top-level folder that contains all the files required in this homework.
- **hw5.py:** The main executable to run training domain adaptation. **You need to implement in this file between line 90 to 130 to collect and visualize essential log (e.g., avg_batch_loss, source_validation_accuracy, target_validation_accuracy).**
- **data.py:** The module defines the CIFAR-10 dataset used for this homework. It will automatically download the raw data to working directory. The training/validation/test split is defined, thus there is no need to apply k-fold in this homework.
- **model.py:** The module defines a simple CNN for this homework.
- **mmd.py: You need to implement MMD loss function in this file between line 62 and 63.** Function argument details (e.g., shape) are provided in the docstring.

Action Items:

In your report, include the answer of the following questions:

1. (1 pt) Is this problem an example of *covariate shift* or *label shift*? Why? (describe through equations)
2. (1 pt) In your report, write the MMD equation below, replacing the question marks by the appropriate variables. Let Γ be the representation function and $\mathcal{D}^{(\text{tr})}$ the set of training examples with n target distribution examples (test distribution examples) and m source examples (training distribution examples), denoted $\boldsymbol{x}^{(\text{target})}$ and $\boldsymbol{x}^{(\text{source})}$ respectively. That is, assume a batch of m examples from the source domain and a batch of n examples from the target domain. Your answer *must* use the variables

$\boldsymbol{x}^{(\text{source})}, \boldsymbol{x}^{(\text{target})}, m, n.$

$$\begin{aligned}
&\widehat{\text{MMD}}^2(\mathcal{D}^{(\text{tr})}; \boldsymbol{W}_\Gamma) = \\
&= \frac{1}{?(?-1)} \sum_{i=1}^{?} \sum_{j \neq i}^{?} k(\Gamma(?_i; \boldsymbol{W}_\Gamma), \Gamma(?_j; \boldsymbol{W}_\Gamma)) \\
&\quad + \frac{1}{?(?-1)} \sum_{i=1}^{?} \sum_{j \neq i}^{?} k(\Gamma(?_i; \boldsymbol{W}_\Gamma), \Gamma(?_j; \boldsymbol{W}_\Gamma)) \\
&\quad - \frac{2}{??} \sum_{i=1}^{?} \sum_{j=1}^{?} k(\Gamma(?_i; \boldsymbol{W}_\Gamma), \Gamma(?_j; \boldsymbol{W}_\Gamma)).
\end{aligned}$$

3. (2 pts) Implement the MMD estimator in the file `mmd.py`, visualization in `hw5.py` and:
   (a) (1 pt) Run the code without using MMD, through the command `python hw5.py --reg_str 0`. Include in your report:
      i. Curve of training loss for each epoch.
      ii. Curve of validation accuracy for each epoch, for both source and target domain.
      iii. Final test accuracy of both source and target domain, for the best model.
   (b) (1 pt) Run the code with MMD, through the command `python hw5.py --reg_str 10 --kernel_sigma 20`. Include in your report:
      i. Curve of training loss for each epoch.
      ii. Curve of validation accuracy for each epoch, for both source and target domain.
      iii. Final test accuracy of both source and target domain, for the best model.

   Hint 1: If you have GPUs available, pass the option `--gpu 0` to use GPU 0. The code will run much faster on a GPU. The scholar cluster has GPUs, see the Piazza post on how to use it.
   Hint 2: Both runs are estimated to take at least 1 hour on scholar even with GPU. For CPU, it is expected to take 2 hours. Pay attention to specify a proper time limit on using `sbatch` on scholar.

The following action items are performed for this question.

1. This problem is an example of *covariate shift.* This is because we are only changing the training distribution of x, $p^{(tr)}(x)$ to a new distribution $p^{(te)}(x)$. However, given an image x, the probability of the class $p(y|x)$ remains unchanged in both training and testing dataset i.e. $p^{(tr)}(y|x) = p^{(te)}(y|x)$.
2. The modified MMD equation is as follows,

$$\begin{aligned}
&\widehat{\text{MMD}}^2(\mathcal{D}^{(\text{tr})}; \boldsymbol{W}_\Gamma) = \\
&= \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(\Gamma(\boldsymbol{x}_i^{(\text{target})}; \boldsymbol{W}_\Gamma), \Gamma(\boldsymbol{x}_j^{(\text{target})}; \boldsymbol{W}_\Gamma)) \\
&\quad + \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k(\Gamma(\boldsymbol{x}_i^{(\text{source})}; \boldsymbol{W}_\Gamma), \Gamma(\boldsymbol{x}_j^{(\text{source})}; \boldsymbol{W}_\Gamma)) \\
&\quad - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(\Gamma(\boldsymbol{x}_i^{(\text{target})}; \boldsymbol{W}_\Gamma), \Gamma(\boldsymbol{x}_j^{(\text{source})}; \boldsymbol{W}_\Gamma)).
\end{aligned}$$

3. The MMD estimator is implemented in `mmd.py` and visualization is implemented in `visualize.py`.

   (a) The code is run without using MMD, through the command `python hw5.py --reg_str 0`.

      i. The curve of training loss for each epoch is shown in 2.

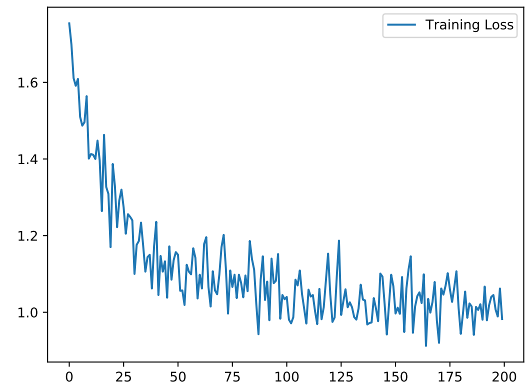


Figure 2: Training Loss

      ii. The curve of validation accuracy for each epoch, for both source and target domain is shown in 3.

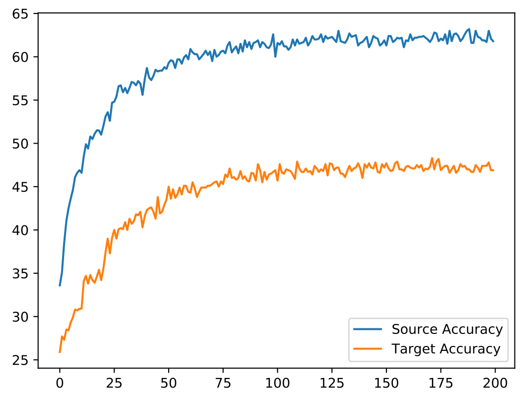


Figure 3: Validation accuracy for both source and target domain.

      iii. Final test accuracy of both source and target domain, for the best model, are 65.5% and 48.8% respectively.

   (b) The code is run with MMD, through the command `python hw5.py --reg_str 10 --kernel_sigma 20`.

      i. The curve of training loss for each epoch is shown in 4.

      ii. The curve of validation accuracy for each epoch, for both source and target domain is shown in 5.

      iii. Final test accuracy of both source and target domain, for the best model, are 65.2% and 56.7% respectively.

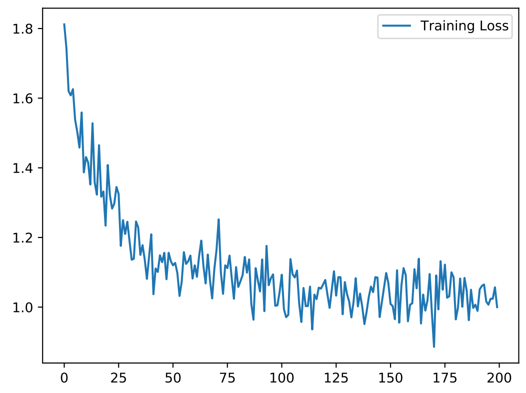


Figure 4: Training Loss

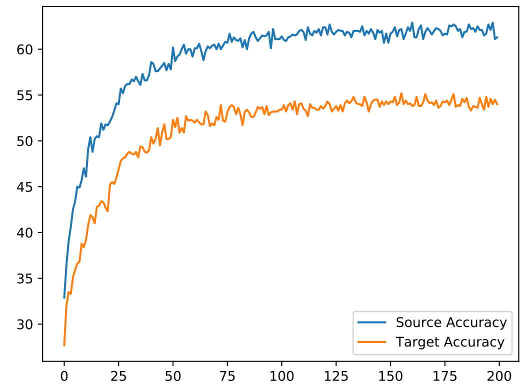


Figure 5: Validation accuracy for both source and target domain.

Hint 1: If you have GPUs available, pass the option `--gpu 0` to use GPU 0. The code will run much faster on a GPU. The scholar cluster has GPUs, see the Piazza post on how to use it.
Hint 2: Both runs are estimated to take at least 1 hour on scholar even with GPU. For CPU, it is expected to take 2 hours. Pay attention to specify a proper time limit on using `sbatch` on scholar.

Submission Instructions

Please read the instructions carefully. Failing to follow the instructions might lead to loss of points.

Naming convention: [your_purdue_login]_hw-5

All your submission files, including a ReadMe, and codes, should be included in one folder. The folder should be named with the above naming convention. For example, if my purdue account is “jsmith123”, then for Homework 5 I should name my folder as “jsmith123_hw-5”.

Remove any unnecessary files in your folder, such as training datasets (Data folder). Make sure your folder is structured as the tree shown in Overview section.

Submit: **TURNIN INSTRUCTIONS**

Please submit your homework files on **data.cs.purdue.edu** using the turnin command, e.g.:

```
turnin -c cs690-dpl -p hw-5 jsmith123_hw-5.
```

Please make sure you didn’t use any library/source explicitly forbidden to use. If any such library/-source code is used, you will get 0 pt for the coding part of the assignment. If your code doesn’t run on scholar.rcac.purdue.edu, then even if it compiles in another computer, your code will still be considered not-running and the respective part of the assignment will receive 0 pt.