Deep Learning CS69000 Homework 5

Tunazzina Islam, islam32@purdue.edu

March 2020

Q0

1. Student interaction with other students / individuals:

   (c) No, I did not discuss the homework with anyone.

2. On using online resources:

   (b) I have used online resources to help me answer this question, but I came up with my own answers.

   Here is a list of the websites I have used in this homework:

   - pytorch forum discussion
   - http://www.jmlr.org/papers/volume13/gretton12a/gretton12a.pdf

Q1: Theoretical Questions

1. We can replace the equivariant representation of the Transformer architecture with an equivariant representation using Janossy pooling and an LSTM architecture.

   To use the LSTM as a neighbor aggregator, we will have a sequence of neighbor embeddings $\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t}$ (where $t = |N_u|$ is the number of neighbors to aggregate), which we'll give as the input of the LSTM. To avoid confusion, we'll call the short term outputs from the LSTM by $\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_t$, so the final output of the LSTM is going to be $\boldsymbol{c}_t, \boldsymbol{m}_t = \vec{f}(\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t})$.

   Janossy pooling can be a method to get a permutation invariant function $\overline{\overline{f}}$ from permutation sensitive function $\vec{f}$ by

   $$\overline{\overline{f}}(\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t}) = \frac{1}{t!} \sum_{\pi \in \Pi(t)} \vec{f}(\boldsymbol{h}_{v_{\pi(1)}}, \boldsymbol{h}_{v_{\pi(2)}}, \cdots, \boldsymbol{h}_{v_{\pi(t)}}) \quad (1)$$

   where $\Pi(t)$ will be all possible permutations to length t.

Now consider creating a new sequence $\boldsymbol{h}_{v'_1}, \boldsymbol{h}_{v'_2}, \cdots, \boldsymbol{h}_{v'_t}$ by adding a positional feature. Here, $\boldsymbol{h}_{v'_i} = (\boldsymbol{h}_{v_i}, i)$.

Now, a permutation-equivariant representation,

$$\overline{\overline{f}}(\boldsymbol{h}_{v'_1}, \boldsymbol{h}_{v'_2}, \cdots, \boldsymbol{h}_{v'_t})_i = \overline{\overline{f}}(\boldsymbol{h}_{v'_{\pi(1)}}, \boldsymbol{h}_{v'_{\pi(2)}}, \cdots, \boldsymbol{h}_{v'_{\pi(t)}})_{\pi_i} \tag{2}$$

2. (a) One of the main shortcomings of RNNs is it cannot consider any future outputs for the past hidden states. The correct inference algorithm for the foward pass uses the Baum-Welch algorithm, as the future outputs should influence the past hidden states. Because of this approximation, at inference time, the future outputs do not influence the past hidden states.

   Bi-directional RNNs help ameliorate this shortcoming by going backward in time. we explicitly model both backward in time and forward in time in order to make predictions.

   (b) Answer of this question depends on length of the sequence. If the length of the sequence is short, bi-directional LSTMs are able to represent short range dependencies between x_i and x_{i+1}. Because Bi-LSTMs have an extra hidden state C_t with gates that determine how the information must be propagated through time. These gates allow the network to remember or to forget information. If the sequence length is long, bi-directional LSTMs aren't be able to represent well dependencies between the first and the last elements of a sequence. LSTM's sequential computation inhibits parallelization of the gradient computation. Number of layers for an interaction between x_1 and x_n can be large and is a function of n. It will be slower.

Q2: Word Embeddings using 2nd order Markov Chains

For word embeddings using 1st order Markov Chains, we have two embedding matrices V and U each of them contains all different word vectors for all different words. V for center words and U for context words.

For word embeddings using 2nd order Markov Chains, we have two center words. But the act is different like $p(c|a, b) \neq p(c|b, a)$. To deal with this, we create two embedding matrices for the first and the second conditional words. In total, we have three embedding matrices $V1$ for first center words, $V2$ for second center words and U for context words.

The negative log-likelihood (NLL) is (not accounting for the corner cases at the end and beginning of a sentence):

$$\mathcal{L}_{\text{word2vec}} = -\sum_{t=k}^{n-k-1} \sum_{k=1}^{(K-1)/2} (\log p(x_{t-k}|x_t, x_{t+1}; \mathbf{U}, \mathbf{V1}, \mathbf{V2}) + \log p(x_{t+1+k}|x_t, x_{t+1}; \mathbf{U}, \mathbf{V1}, \mathbf{V2})),$$

Probability function is

$$p(x|x_t, x_{t+1}; \mathbf{U}, \mathbf{V1}, \mathbf{V2}) = \frac{1}{Z(x_t, x_{t+1}; \mathbf{U}, \mathbf{V1}, \mathbf{V2})} \exp(\langle \mathbf{U}x, \langle \mathbf{V1}x_t, \mathbf{V2}x_{t+1}\rangle\rangle),$$

where $Z = \sum_{x' \in \Omega} \exp(\langle \mathbf{U}x', \langle \mathbf{V1}x_t, \mathbf{V2}x_{t+1}\rangle\rangle)$, here Ω = Vocabulary size

Q3: Domain Adaptation with Maximum Mean Discrepancy (MMD)

1. The given problem is an example of *covariate shift*. Because only the distribution of X changes between training and testing.

$$p^{(tr)}(x, y) = p(y|x)p^{(tr)}(x)$$

$$p^{(te)}(x, y) = p(y|x)p^{(te)}(x)$$

   Extra data given in the form $\{x_j^{(te)}\}_j$ i.i.d. sampled from $p^{(te)}(x)$

2. MMD equation: Assume a batch of m examples from the source domain and a batch of n examples from the target domain.

$$\begin{aligned}
&\widehat{\mathrm{MMD}}^2(\mathcal{D}; \boldsymbol{W}_\Gamma) = \\
&= \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k(\Gamma(x_i^{source}; \boldsymbol{W}_\Gamma), \Gamma(x_j^{source}; \boldsymbol{W}_\Gamma)) \\
&\quad + \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(\Gamma(x_i^{target}; \boldsymbol{W}_\Gamma), \Gamma(x_j^{target}; \boldsymbol{W}_\Gamma)) \\
&\quad - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(\Gamma(x_i^{source}; \boldsymbol{W}_\Gamma), \Gamma(x_j^{target}; \boldsymbol{W}_\Gamma))
\end{aligned}$$

3. **Run the code without using MMD:**

   (i) Curve of training loss for each epoch

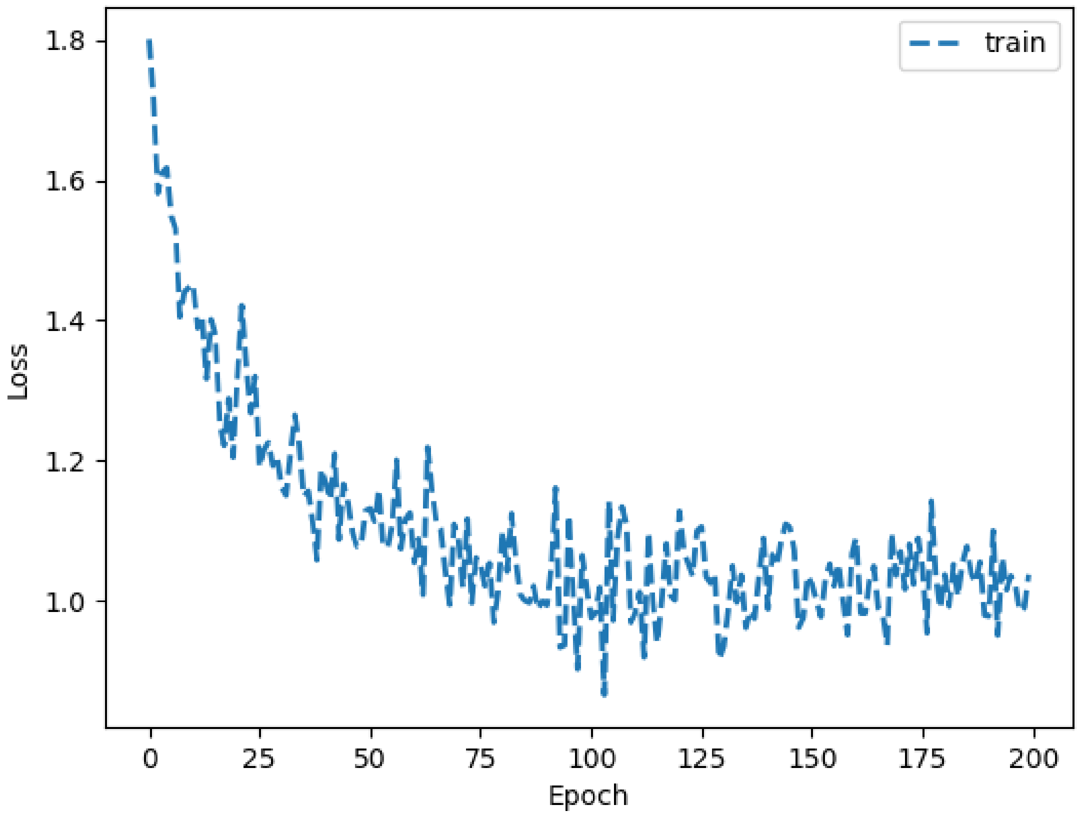

Figure 1: Curve of training loss for each epoch without MMD.

(ii) Curve of validation accuracy for each epoch, for both source and target domain.

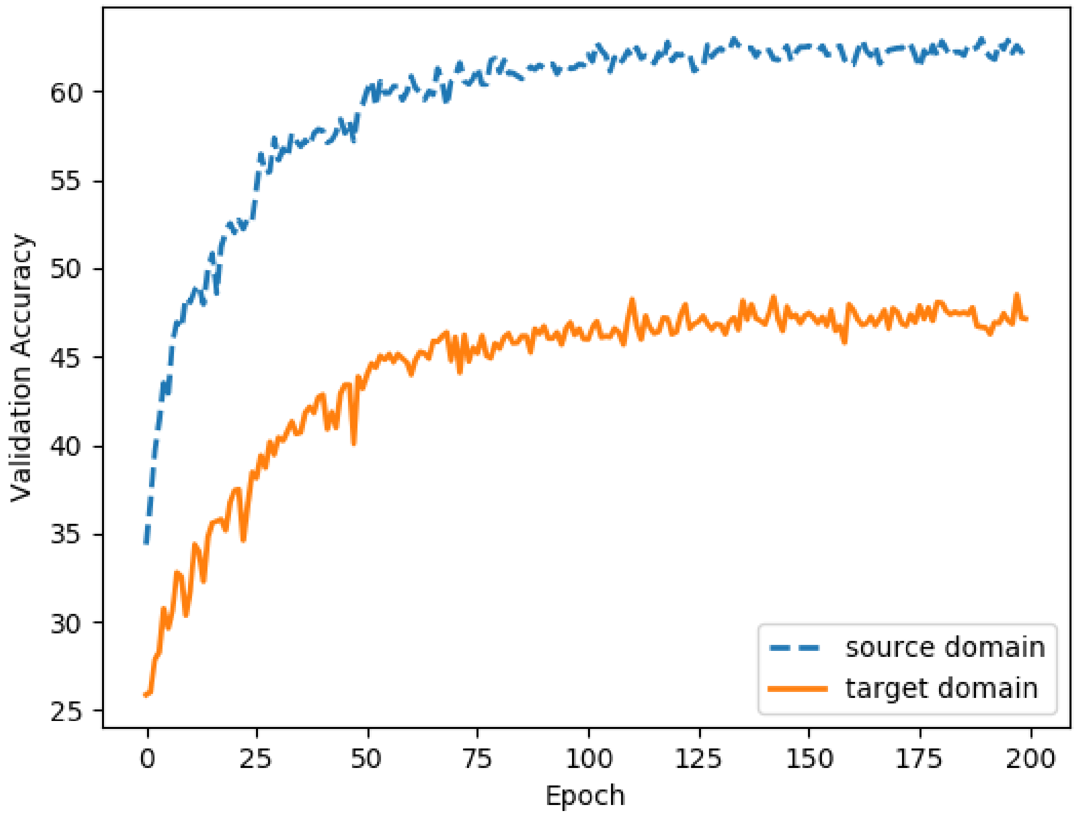

Figure 2: Curve of validation accuracy for each epoch, for both source and target domain without MMD.

(iii) Best model found at epoch 197, with target validation accuracy = 48.5%

Final Test Accuracy:

Source Domain: 65.6%

Target Domain: 49.2%

4. **Run the code with MMD:**

   (i) Curve of training loss for each epoch

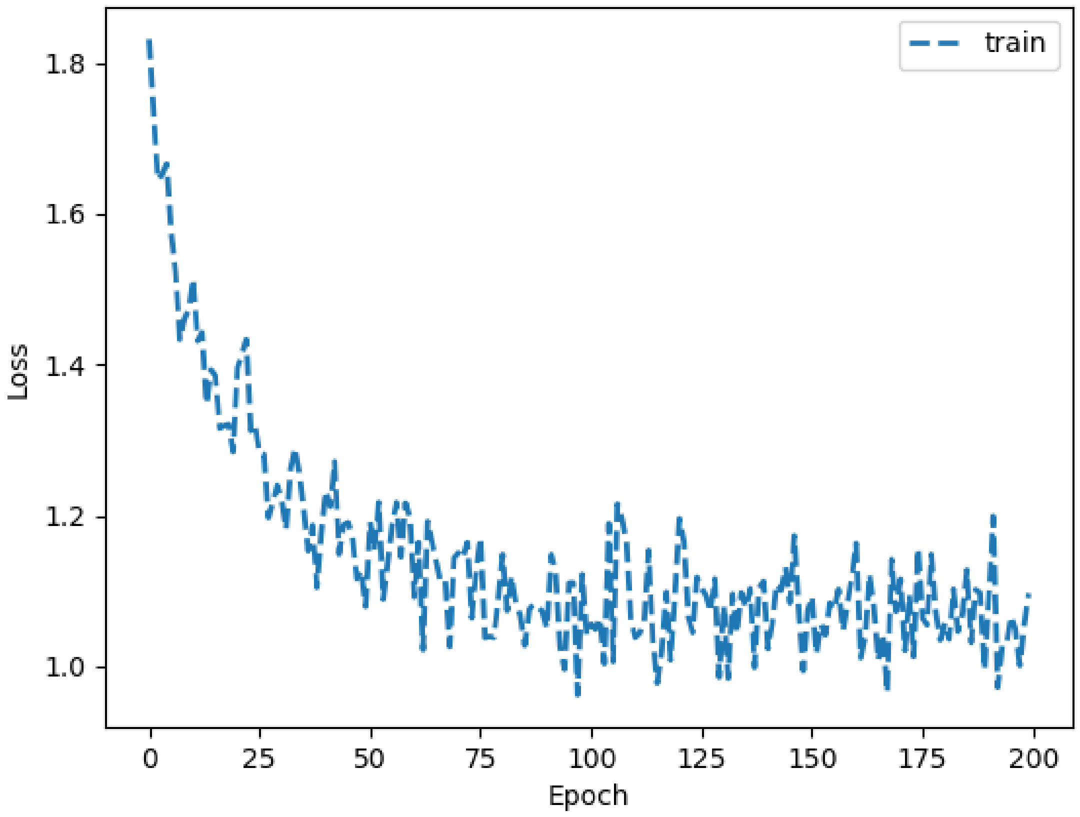

Figure 3: Curve of training loss for each epoch using MMD.

(ii) Curve of validation accuracy for each epoch, for both source and target domain.

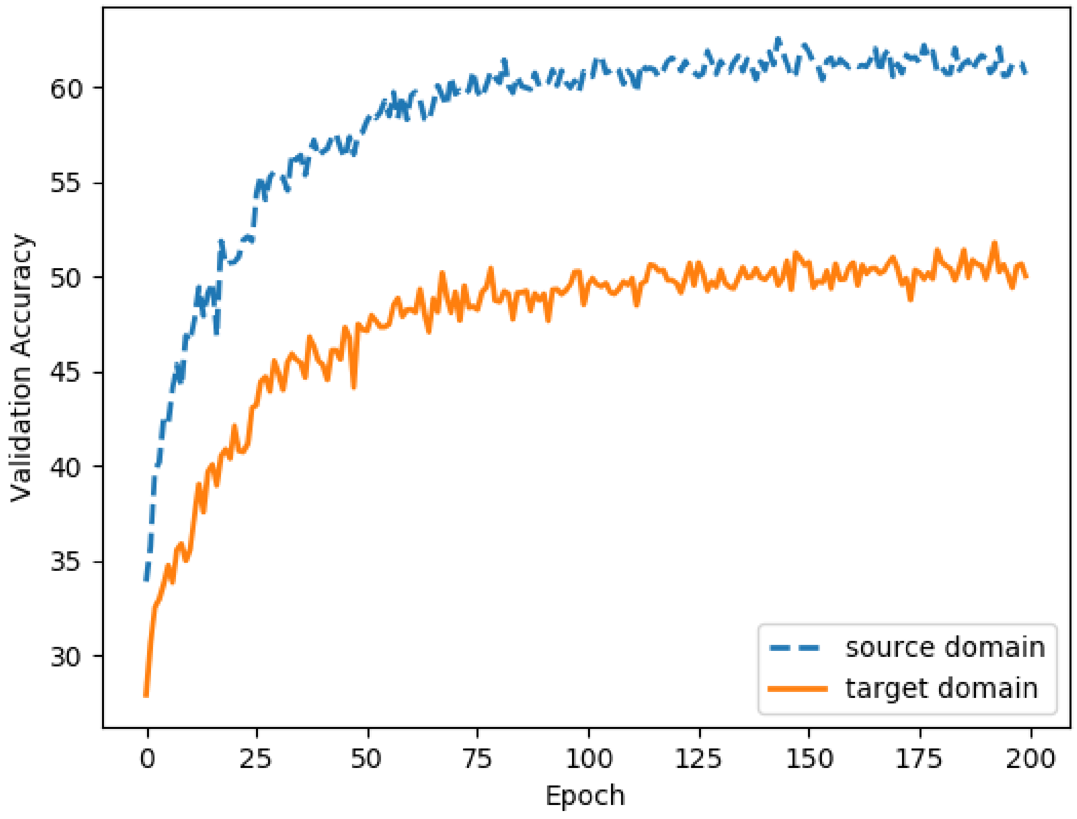

Figure 4: Curve of validation accuracy for each epoch, for both source and target domain using MMD.

(iii) Best model found at epoch 192, with target validation accuracy = 51.8%

Final Test Accuracy:

Source Domain: 64.6%

Target Domain: 53.0%