Assignment 1
COL 775: Deep Learning. Semester II, 2022-23.
Due Date: Tuesday March 21, 2023. 11:50 pm.

March 10, 2023

1 ResNet over Convolutional Networks and different Normalization schemes

Residual Networks (ResNet) [He et al., 2016] present a very simply idea to introduce identity mappings via residual connections. They are shown to significantly improve the quality of training (and generalization) in deeper networks. We covered the core idea in class. Before starting this part of the assignment, you should thoroughly read the ResNet paper. Specifically, we will implement the ResNet [He et al., 2016] architecture, and study the effect of different normalisation schemes, viz. Batch Normalization [Ioffe and Szegedy, 2015], Instance Normalization [Ulyanov et al., 2016], Batch-Instance Normalization [Nam and Kim, 2018], Layer Normalization [Ba et al., 2016], and Group Normalization [Wu and He, 2020] within ResNet. We will experiment with CIFAR 10 dataset as described in the ResNet paper.

1.1 Image Classification using Residual Network

This sub-part will implement ResNet for Image Classification.

1. You will implement a ResNet architecture from scratch in PyTorch. Assume that the total number of layers in the network is given by 6n+2. This includes the first hidden (convolution) layer processing the input of size 32×32. This is followed by n layers with feature map size 32×32, followed by n layers with feature map size 16×16, n layers with feature map size given by 8×8, and finally a fully connected output layer with r units, r being number of classes. The number of filters in the 3 sets of n hidden layers (after the first convoluational layer) are 16, 32, 64, respectively. There are residual connections between each block of 2 layers, except for the first convolutional layer and the output layer. All the convolutions use a filter size of 3×3 inspired by the VGG net architecture [Simonyan and Zisserman, 2015]. Whenever down-sampling, we use the convolutional

layer with stride of 2. Appropriate zero padding is done at each layer so that there is no change in size due to boundary effects. The final hidden layer does a mean pool over all the features before feeding into the output layer. Refer to Section 4.2 in the ResNet paper for more details of the architecture. Your program should take n as input. It should also take r as input denoting the total number of classes.

2. Train a ResNet architecture with $n = 2$ as described above on the CIFAR 10 dataset. Use a batch size of 128 and train for 100 epochs. For CIFAR 10, $r = 10$. Use SGD optimizer with initial learning rate of 0.1. Decay or schedule the learning rate as appropriate. Feel free to experiment with different optimizers other than SGD.

3. The train data has $50,000$ images. Randomly select any $10,000$ of these as validation data. Use validation data for early stopping. NOTE: DO NOT use test data split at all during the training process. Report the following statistics / analysis:

   - Accuracy, Micro F1 and Macro F1 on Train, Val and Test splits.
   - Plot the error curves for both the train and the val data.

1.2 Impact of Normalization

The standard implementation of ResNet uses Batch Normalization [Ioffe and Szegedy, 2015]. In this part of the assignment, we will replace Batch Normalization with various other normalization schemes and study their impact.

1. Implement from scratch the following normalization schemes. They should be implemented as a sub-class of nn.Module.

   (a) Batch Normalization (BN) [Ioffe and Szegedy, 2015]

   (b) Instance Normalization (IN) [Ulyanov et al., 2016]

   (c) Batch-Instance Normalization (BIN) [Nam and Kim, 2018]

   (d) Layer Normalization (LN) [Ba et al., 2016]

   (e) Group Normalization (GN) [Wu and He, 2020]

2. In your implementation of ResNet in Section 1.1, replace the Pytorch's inbuilt Batch Normalization (nn.BatchNorm2d) with the 5 normalization schemes that you implemented above, giving you 5 new variants of the model. Note that normalization is done immediately after the convolution layer. For comparison, remove all normalization from the architecture, giving you a No Normalization (NN) variant as a baseline to compare with. In total, we have 6 new variants (BN, IN, BIN, LN, GN, and NN).

3. Train the 6 new variants on the CIFAR 10 dataset, as done in Section 1.1.

4. As a sanity check, compare the error curves and performance statistics of the model trained in Section 1.1 with the BN variant trained in this part. It should be identical (almost).

5. Compare the error curves and performance statistics (accuracy, micro F1, macro F1 on train / val / test splits) of all the six models.

6. **Impact of Batch Size:** [Wu and He, 2020] claim that one of the advantages of GN over BN is its insensitivity to batch size. Retrain the BN and GN variants of the model with Batch Size 8 and compare them with the same variants trained with batch Size 128. Note that reducing the batch size will significantly increase the training time. To reduce the time taken, run it for 100 epochs and early stop based on validation accuracy.

7. **Evolution of feature distributions:** Pretrained models are commonly used as feature extractors for transfer learning. In this sub-part, we will qualitatively analyze the features extracted from different variants of the ResNet model. Plot the 1^{st}, 20^{th}, 80^{th}, and 99^{th} quantile of the features as a function of the training epoch. You may compute these statistics over the val data at the end of each epoch. You may use the activations at the end of penultimate layer as features.

2 Text-to-SQL: A challenging Sequence to Sequence task.

Semantic Parsing is the task of converting natural language into a logical and structured form such as code or SQL queries. It is a critical task which is required for building a lot of Deep Learning systems like voice assistants, search engines, and interactive robots. Generating queries from their textual description will involve using an **Encoder-Decoder architecture**. For this part of the assignment, the database and code have been provided at `https://github.com/daman1209arora/Text-To-SQL-COL775`. The dataset has been pre-processed from the Spider dataset[Yu et al., 2018] Firstly, let us understand the format of the data given to us.

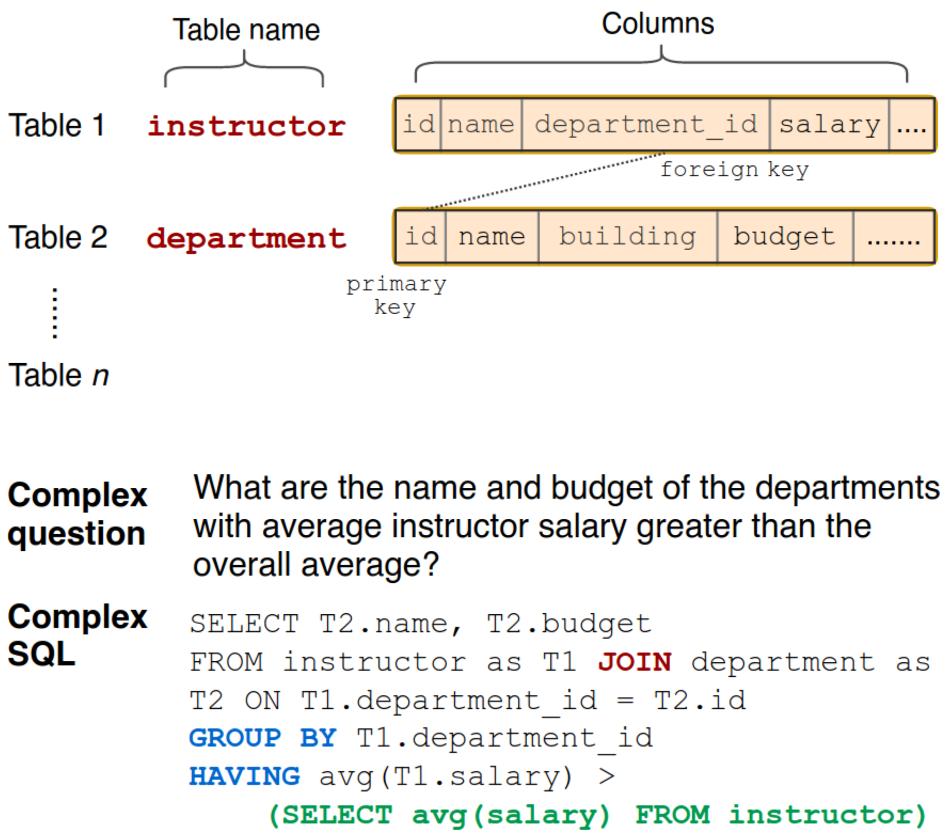
2.1 The Data

Each textual question q has an associated database D, which consists of various tables $T_1, T_2 ... T_n$. Each table consists of some columns $T_i = [C_{i,1}, C_{i,2} ... C_{i,m_i}]$ and associated schemas. Since there are multiple databases present, there are two ways of evaluating a DL system on this task.

1. **Intra-domain transfer**: How does a model generalize to new queries assuming it has seen a database in the training data?

2. **Inter-domain transfer**: How does a model generalize to queries on databases which it has not seen in the training data?

An example data point would look like:



Complex question What are the name and budget of the departments with average instructor salary greater than the overall average?

Complex SQL

```
SELECT T2.name, T2.budget
FROM instructor as T1 JOIN department as
T2 ON T1.department_id = T2.id
GROUP BY T1.department_id
HAVING avg(T1.salary) >
    (SELECT avg(salary) FROM instructor)
```

2.2 Metrics for evaluation:

Evaluation metrics are critical for any system to be evaluated. For this task we will use 2 different metrics. All of them have been implemented in the `evaluation.py` file given in the GitHub repository. You must use the same exact script for your evaluation on the dev set.

1. **Exact Match Accuracy:** For an SQL query, we break it down into components such as `SELECT, WHERE, GROUP BY` etc. Then, the components are broken into sets of sub-components. For example, for the query given in the example above, the sub-components of the following components would be:

   (a) `SELECT`: `(department.name, department.budget)`

   (b) `GROUP BY`: `(instructor.department_id)`

   (c) `HAVING`: `>(avg(T1.salary), SELECT avg(SALARY) FROM instructor)`

Note that sub-components can be nested queries themselves, for example, the sub-component of `HAVING` is itself a query. Then, to check if we have an exact match, we check whether the **set**(to eliminate any ordering issues) of sub-components of each component is an exact match. Only if all components match exactly, we count a query as an exact match.

2. **Execution Accuracy** When a query is executed, if all the outputs match with the output of the gold query, we count it as 1, otherwise 0.

For more details about the evaluation metrics, you can refer to the paper by Yu et al. [2018]. For exact implementation of these metrics, refer to the code which has been provided. Note that all of these metrics **can** indeed generate false positives, which is why it is good to have multiple metrics for generative tasks.

2.3 Models

For your experiments you are supposed to train the following models:

1. A Seq2Seq model with GloVe[Pennington et al., 2014] embeddings, using an LSTM encoder and an LSTM decoder. For details of an LSTM, refer to Goodfellow et al. [2016] [1].

2. A Seq2Seq+Attention model with GloVe embeddings, using an LSTM encoder and an LSTM decoder

3. A Seq2Seq+Attention model using a pre-trained **frozen** BERT-base-cased encoder and an LSTM decoder.

4. A Seq2Seq+Attention model with pre-trained BERT-base-cased encoder and an LSTM decoder, where the BERT encoder can now be fine-tuned along with the remaining network.

Keep in mind that you **have to incorporate the schema** into the learning algorithm so that it can generalize well. There can be multiple methods to do so. We leave it to your choice. For the decoder you must implement **beam search** from scratch. Note that training recurrent architectures like LSTMs can be slightly tricky. Therefore, look into various tricks that people have employed for this purpose. Some of them can be found in [Sutskever et al., 2014]. Try to optimize your training pipeline as much as possible. You must report the following:

1. Loss curves on the training and dev set

2. Exact Match Accuracy and Execution Accuracy (see evaluation metrics) on the dev set.

3. All hyper-parameter settings used in your experiments.

[1] https://www.deeplearningbook.org/contents/rnn.html (See Section 10.10.1)

4. Any findings observed from any potential grid search performed.
5. Any other insights that you gained from the training procedure and the outputs of your model.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. 2016. URL `http://arxiv.org/abs/1607.06450`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. `http://www.deeplearningbook.org`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL `https://doi.org/10.1109/CVPR.2016.90`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 32nd International Conference on Machine Learning, ICML 2015, 1:448–456, 2015.

Hyeonseob Nam and Hyo-Eun Kim. Batch-instance normalization for adaptively style-invariant neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pages 2563–2572, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/018b59ce1fd616d874afad0f44ba338d-Abstract.html`.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://aclanthology.org/D14-1162`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL `http://arxiv.org/abs/1409.1556`.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. CoRR, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. (2016), 2016. URL `http://arxiv.org/abs/1607.08022`.

Yuxin Wu and Kaiming He. Group Normalization. International Journal of Computer Vision, 128(3):742–755, 2020.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2018. URL `https://arxiv.org/abs/1809.08887`.