EECS 498/598 Deep Learning - Homework 3

March 2nd, 2019

Instructions

- This homework is Due March 26th at 11.59pm. Late submission policies apply.
- You will submit a write-up and your code for this homework.

1 [25 points] Text Classification using CNNs

In this problem, you will implement a CNN text classifier similar to the network of [1] for sentiment classification. The network has the following architecture.

`Embedding layer` $\rightarrow$ `1-d Convolution` $\rightarrow$ `Pooling` $\rightarrow$ `ReLU` $\rightarrow$ `Linear` $\rightarrow$ `Sigmoid`

Assume that we perform global average-pooling in the pooling layer.

Assume the input to the convolution layer is given by $X \in \mathbb{R}^{N \times H \times W}$. Further assume that the temporal dimension (or sequence dimension) is the second dimension, of size H. Consider a convolutional kernel $W^{\text{conv}} \in \mathbb{R}^{F \times H' \times W'}$ and a bias vector $b \in \mathbb{R}^F$. The output of 1-d convolution is given by $Y \in \mathbb{R}^{N \times F \times H'' \times W''}$.

1. For a 1-d convolution layer, what should be the values of W' and W''?
2. Express the output of the convolutional layer $Y_{n,f}$ as a function of X_n, W_f^{conv} and b_f based on the $*_{\text{filt}}$ notation defined in homework 1.
3. What is the size of $Y_{n,f}$ in terms of H, H', W, W' ?
4. What size is the output of the pooling layer ?
5. Implement the network as a `nn.Module` class called `CNN` in the emtpy file `cnn.py`.
   - Your module will need a `_init_()` and a `forward()` function.
   - The input to your `forward()` function will be a 2-d tensor of word ids of size $N \times H$. It will returns logits of size N.
   - Your will find `nn.Conv1d` useful in your implementation.
   - Use a fixed kernel size of $H' = 5$
6. Train your model on the sentiment classification task from homework 2. You should pad your sequences so that sequences in a batch have the same length. Report test accuracies for the following architectural choices and hyperparameters.
   - Global average-pooling, Global max-pooling
   - Kernel sizes: 5, 7

2 [10 points] Siamese Networks for Learning Embeddings.

In this problem, you will implement a Siamese network for face verification in `siamese_face.py`. The dataset is `att_faces.tar` and `lfw_faces.tar`

1. Implement the contrasive loss class `ContrasiveLoss`

$$L(x^{(1)}, x^{(2)}, y) = (1-y)\|f(x^{(1)}) - f(x^{(2)})\|_2^2 + y(max\{0, m - \|f(x^{(1)}) - f(x^{(2)})\|_2\})^2$$

   where m is the margin value, y is the label denoting whether $x^{(1)}$ and $x^{(2)}$ are from the same person. For more details, you may want to read `http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf`

2. Design architecture and build the embedding network in class `SiameseNetwork`, and train the siamese network for face images in att dataset. Report the learning curve of losses on training dataset and the qualitative result on training/testing dataset, i.e. the figures showing a pair of images and the dissimilarity measurement between these two pictures. You may want to adjust the optimizer, learning rate, number of epochs, network architecture, image pre-processing, batch size, margin value, etc. to get good performance. We expect the reported dissimilarity number is consistent with visual similarity.

3. **Extra credit:** Repeat the process in the above question to train siamese network for face images in lfw dataset.

3 [35 points] Conditional Variational Autoencoders.

In this problem, you will implement a conditional variational autoencoder (CVAE) from [2] and train it on the MNIST dataset.

1. Derive the variational lowerbound of a conditional variational autoencoder. Show that:

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{y}) &\geq \mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y}) \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})\right] - D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right),\end{aligned} \tag{1}$$

   where $\mathbf{x}$ is a binary vector of dimension d, $\mathbf{y}$ is a one-hot vector of dimension c defining a class, $\mathbf{z}$ is a vector of dimension m sampled from the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$. The posterior distribution is modeled by a neural network of parameters ϕ. The generative distribution $p_\theta(\mathbf{x}|\mathbf{y})$ is modeled by another neural network of parameters θ.

2. Derive the analytical solution to the KL-divergence between two Gaussian distributions $D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right)$. Let us assume that $p_\theta(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and show that:

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right) = -\frac{1}{2}\sum_{j=1}^{J}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right), \tag{2}$$

   where μ_j and σ_j are the outputs of the neural network that estimates the parameters of the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$.

3. Fill in code for CVAE network as a `nn.Module` class called `CVAE` in the file `cvae.py`

   - Implement the `recognition_model` function $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$.

- Implement the `generative_model` function $p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})$.
- Implement the `forward` function by inferring the Gaussian parameters using the recognition model, sampling a latent variable using the reparametrization trick and generating the data using the generative model.
- Implement the variational lowerbound `loss_function` $\mathcal{L}(\theta,\phi;\mathbf{x},\mathbf{y})$.
- Train the CVAE and visualize.

If trained successfully, you should be able to sample images $\mathbf{x}$ that reflect the given label $\mathbf{y}$ given the noise vector $\mathbf{z}$.

4 [30 points] Generative Adversarial Networks.

In this problem, you will implement generative adversarial networks and train it on the MNIST dataset. Specifically, you will implement the Deep Convolutional Generative Adversarial Networks (DCGAN) from [3]. In the generative adversarial networks formulation, we have a generator network G that takes in random vector $\mathbf{z}$ and a discriminator network D that takes in an input image $\mathbf{x}$. The parameters of G and D are optimized via the adversarial objective:

$$\min_G \max_D \quad \mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log(1 - D(G(\mathbf{z})))\Big], \tag{3}$$

In practice, we alternate between training D and G where we train G to maximize:

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log D(G(\mathbf{z}))\Big], \tag{4}$$

and we follow by training D to maximize:

$$\mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\Big[\log(1 - D(G(\mathbf{z})))\Big], \tag{5}$$

Therefore, the two separate optimizations make up one full training step. Given this information, you will do the following:

1. Fill in code for the DCGAN network in the `gan.py`. Descriptions of what should be filled in is writen as comments in the code itself.
    - Implement the `sample_noise` function.
    - Implement the `build_discriminator` function.
    - Implement the `build_generator` function.
    - Implement the `get_optimizer` function.
    - Implement the `bce_loss` function.
    - Use the previously implemented `bce_loss` to implement the `discriminator_loss` function.
    - Use the previously implemented `bce_loss` implement the `generator_loss` function.
    - Train your DCGAN!

If trained successfully, you should see the progression of sample quality getting better as training epochs increase.

References

[1] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[2] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *NeurIPS*. 2015.

[3] Alec Radford, Luke Me, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *ICLR*. 2016.