UROP Report – Conditional Sequence to Sequence for Neural Dialog Generation

Contribution: Fang Haoyang (50%), Gao Tong (50%)

Introduction

Neural dialog system has been an active area of research, as a dialog system makes it possible for human to interact with computer using everyday language, lowering the threshold to fully utilize devices' functionalities. One basic dialog model is called sequence-to-sequence, which encodes the question into a context vector, and use a decoder to reconstruct the corresponding answer given that context vector. However, this model does not consider the topic clusters of answers thus producing monotonous responses. In our project, we present a modified version of sequence to sequence model, aiming at improving the quality of replying questions with certain latent topics. This model is combined from ideas of VAE and our supervisor.

We implemented two improved models to answer questions with specific latent topics. In this report, we first introduce the intuition and mathematic theory behind two models, then present our idea with important code snippets. Finally, we propose some possible ways for further improvement on this model.

Motivation

1. Answer Clustering:

   Most of QA pairs in human dialogue is attached to a certain topic c, and those words related to certain topic c will have higher probability to appear under such scenario. Therefore, it's possible to use a model to learn the latent topic

c behind each QA pair, and generate answer with words that have higher correlation to the latent topic c coming with that question.

2. Sentence Vector:

   In order to find out the topic c, extracting the meaning of questions and clustering it to their most related centroids are necessary. Former research on word vector shows that after few batch of training, word vectors with similar meaning can have less cosine similarity, hence we decide to generate sentence vectors from word vectors, use them as our unbiased estimator of questions, and clustering them with unsupervised classification algorithm to get our reasonable target topics. The idea of converting word vectors to sentence vector is proposed by [1], who suggested a computationally efficient algorithm with incredible performance. We integrated this algorithm into our preprocess.

3. Latent First Word Picker:

   VAE has been a popular model since it was introduced in 2013 [2]. It largely improves the diversity as well as the noise resistance of the output by adding a latent variable z with prior Q(z). In past models/papers about VAE, the latent variable z always participates in generating the whole output sequence. However, while latent variable brings diversity to answers, it may cause inconsistency (or grammer faults) inside the sentence, since generation of every word is affected by the latent variable.

By reading some question-answer pairs, we find that the diversity of the answers largely depend on the diversity of its first word. We believe that use VAE to generate the first word can promise similar diversity for answers. And passing the first word to the encoder to generate the whole answer also promises the consistency of the answer.

Implementation of the Conditional Sequence-to-Sequence Model

The conditional sequence-to-sequence model contains five parst:

1. Classifier: First we convert embedded answers (sequences of word vectors) to sentences vectors, using the idea from [1]. It generates the resulting sentence vector with different weights on different words (measured by word occurrence), and removes unrelated part in sentence vector using PCA. This results our ideal sentence vectors.

   Given sentence vectors, we use k-means algorithm to discover inter-connections among sentences, and generate latent topic of each sentence. We used scikit-learn, a popular machine learning library in python, to help us achieve this goal with only few lines of code.

2. Encoder: A bidirectional GRU, encoding question Q into hidden variable h by concatenating outputs of GRU and passing through a non-linear layer

(sinh).

```
#net1
#BiGRU+concat+nonlinear
class Encoder(nn.Module):
    def __init__(self, emb_sz, hidden_size, n_layers=1):
        super(Encoder, self).__init__()
        self.hidden_size = hidden_size
        self.n_layers = n_layers

        self.gru = nn.GRU(emb_sz, hidden_size, dropout=0.2, batch_first=True, bidirectional=True)
        for w in self.gru.parameters(): # initialize the gate weights with orthogonal\
            if w.dim()>1:
                nn.init.orthogonal(w, sqrt(2))

    #embedded: [batch_sz, seq_len, emb_sz]
    #out: [batch_sz, seq_len, hid_sz*2] (biRNN)   hidden: [batch_sz, 2*hid_sz]
    def forward(self, embedded):
        output, hidden = self.gru(embedded) #hidden: [2, batch_sz, hid_sz] (2=fw&bw)
        N = list(hidden.size())[1]
        hidden = torch.transpose(hidden, 0, 1).contiguous() #hidden: [batch_sz, 2, hid_sz] (2=fw&bw)
        hidden = hidden.view(N,-1) #hidden: [batch_sz, 2*hid_sz] (2=fw&bw)
        hidden = torch.sinh(hidden) #hidden: [batch_sz, 2*hid_sz] (2=fw&bw)
        return output, hidden
```

3. Topic Picker: Two fully connected layers with PReLU [3] as activation function. Use dropout layer to avoid overfitting and softmax layer to get score of each topic.

```
#net2
#used to pick topic
#dropout+dense+relu+dense+softmax
class Picker(nn.Module):
    def __init__(self, batch_sz, hidden_size, relu_size, output_size):
        super(Picker, self).__init__()
        self.relu = nn.PReLU()
        self.l1 = nn.Linear(hidden_size*2, relu_size)
        self.l2 = nn.Linear(relu_size, output_size)
        self.drop = nn.Dropout(p=0.2)
        self.sm = nn.Softmax(dim=1)

    #input hidden: [batch_sz, 2*hidden_size]
    #output score: [batch_sz, score_size]
    def forward(self, hidden):
        hidden = self.drop(hidden)
        hidden = self.l1(hidden) #hidden: [batch_sz, relu_size]
        hidden = self.relu(hidden) #hidden: [batch_sz, relu_size]
        score = self.l2(hidden)
        score = self.sm(score)
        return score
```

4. First Word Picker: Same as the decoder in VAE [2]. TBC

```
#net4
#used to pick first word of answer
#same as the decoder of variational auto-encoder (VAE)
class LatentPicker(nn.Module):
    def __init__(self, batch_sz, hidden_size, latent_size, relu_size, output_size, training=False):
        super(LatentPicker, self).__init__()
        self.relu = nn.PReLU()
        self.l_mu = nn.Linear(hidden_size*2, latent_size)
        self.l_logvar = nn.Linear(hidden_size*2, latent_size)
        self.l1 = nn.Linear(hidden_size*2 + latent_size, relu_size)
        self.l2 = nn.Linear(relu_size, output_size)
        self.sm = nn.Softmax(dim=1)
        self.drop = nn.Dropout(p=0.2)
        self.training = training

    def reparameterize(self, mu, logvar):
        if self.training:
            std = logvar.mul(0.5).exp_()
            eps = Variable(std.data.new(std.size()).normal_())
            return eps.mul(std).add_(mu)
        else:
            return mu

    #input hidden: [batch_sz, 2*hidden_size]
    #output score: [batch_sz, score_size]
    def forward(self, hidden):
        mu = self.l_mu(hidden)
        logvar = self.l_logvar(hidden)
        z = self.reparameterize(mu, logvar) #z: [batch_sz, latent_size]
        hidden = torch.cat( (hidden, z) , dim=1) #hidden: [batch_sz, 2*hidden_size+latent_size]
        hidden = self.drop(hidden) #hidden: [batch_sz, 2*hidden_size+latent_size]
        hidden = self.l1(hidden) #hidden: [batch_sz, relu_size]
        hidden = self.relu(hidden) #hidden: [batch_sz, relu_size]
        score = self.l2(hidden)
        score = self.sm(score)
        return score
```

5. Decoder: Similar to standard decoder in seq2seq models, but instead of starting with <SOS> (or <GO>), the first actual word of the answer is given to start with.

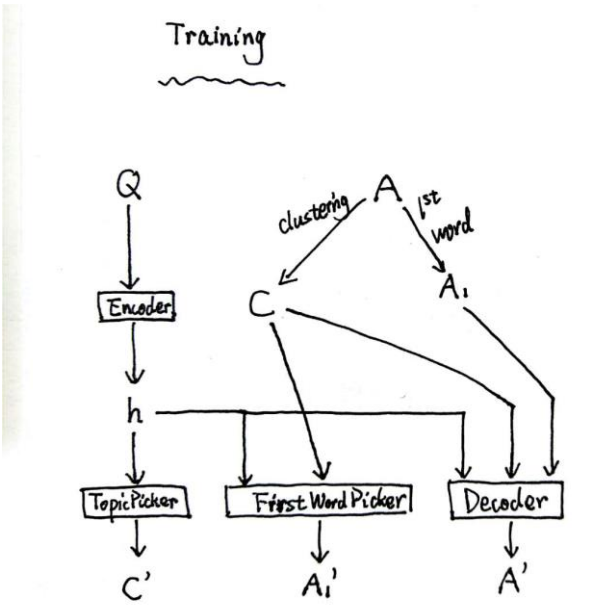
At training time:

Given questions Q and answers A.

Special Preprocessing: We find corresponding topic C for each answer A.

Training:

1. Pass Q into Encoder and get latent variable h as output.
2. Pass h into Topic Picker and get predicted topic C' as output.
3. Pass h and C into First Word Picker and get predicted first word A_1' as output.
4. Pass h, C and actual first word A_1 into Decoder and get predicted answer A as output.

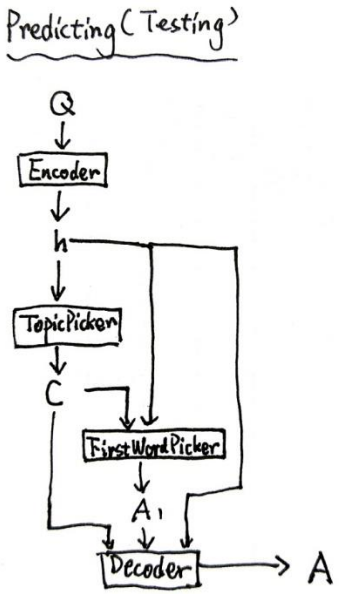


At testing time:

Given question Q.

Predicting:

1. Pass Q into Encoder and get latent variable h as output.
2. Pass h into Topic Picker and get topic C as output.
3. Pass h and C into First Word Picker and get first word A_1 as output.
4. Pass h, C and A_1 into decoder and get answer A.



Theoretical Proof

Our task is to model the probability of the answer Y given question X, i.e. P(Y|X).

First, we introduce topic c for each Y, so we have:

$$\mathrm{P(Y|X)} = P(Y|X,c) * P(c|X).$$

Then we do some separations on Y. Denote $\mathrm{Y} = [w_1,\ w_2,\ w_3, \ldots]$ where w_i is the i[th] word in the answer Y. Let $Y_0 = [w_1],\ Y_1 = [w_2, w_3, \ldots]$. Since Y is sequentially generated in RNN-based models, we have:

$$\mathrm{P(Y|X,c)} = P(Y_0|X,c) * P(Y_1|X,c,Y_0).$$

Combining the equations above:

$$\mathrm{P(Y|X)} = P(c|X) * P(Y_0|X,c) * P(Y_1|X,c,Y_0)$$

Then we have the likelihood:

$$\mathrm{logP(Y|X)} = logP(c|X) + logP(Y_0|X,c) + logP(Y_1|X,c,Y_0)$$

We further introduce latent variable z into $P(Y_0|X,c)$, according to basic knowledge of VAE, we have the inequality:

$$logP(Y_0|X,c) \ \geq\ \mathrm{E}(\log P(Y_0|z,X,c)) -\ \mathrm{KL(Q(z|}X,c,Y_0)||\mathrm{P(z))}$$

, where $\mathrm{Q(z|}X,c,Y_0)$ is a proposal distribution to approximate the posterior $\mathrm{P(z|}X,c,Y_0)$ and in this case, it is a diagonal Gaussian whose parameters depend on X, c and Y_0.

We thus have the following evidence lower bound (ELBO) for the likelihood of the data:

$$\mathrm{logP(Y|X)} \geq logP(c|X) + logP(Y_1|X,c,Y_0) +\ \mathrm{E}(\log P(Y_0|z,X,c))$$

$$-\ \mathrm{KL(Q(z|}X,c,Y_0)||\mathrm{P(z))}$$

Note that this loss decomposes into 4 parts:

1. the loss of the topic picker
2. the loss of decoder
3. the cross-entropy loss between first word of the model and of the data
4. the KL divergence between the approximate posterior and the prior, and that this lower bound is much closer to our likelihood comparing with traditional VAE models.

Future Plan

1. Use pre-trained word2vec e.g. GloVe [4], to calculate sentence vector:

   Pre-trained word2vec has better structure in their vector space, e.g. meaningful neighbors and linear structures. These features can be brought to the sentence vector, resulting a good structure in sentence vector space, so that the topics, clusters of sentence vector, have more sentimental meaning.
2. Try replacing the dissimilarity measurement with cosine similarity:

   Traditional k-means use Euclidean distance to measure the dissimilarity between two points, which doesn't really make sense on clustering sentence vector. As research have shown that it is meaningful to use cosine similarity to measure the distance between two word vectors, customizing k-means with cosine similarity to cluster sentence vectors should be better.
3. Use silhouette to decide k [5]:

   Assuming sentences are well clustered in the vector space (distance between clusters are relatively high comparing with distance between points inside a

cluster), when different k are selected on the same magnitude, the clustering result could have a far cry from each other even if they have little difference, say, 1 or 2. Silhouette method aim to interpret, visualize and validate consistency within clusters of data. By applying this method it's more possible to find the ideal number of clusters (topics).

Reference

[1] Arora, S., Liang, Y., & Ma, T. (2016). A simple but tough-to-beat baseline for sentence embeddings.

[2] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).

[4] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

[5] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics, 20, 53-65.