

---

# ChiNet

---

Nil Adell    Joseph Cornelius    Alexander Nedergaard    Lama Saouma

Feel free to add more sections but those listed here are strongly recommended.

## 1 Introduction

You can keep this short. Ideally you introduce the task already in a way that highlights the difficulties your method will tackle.

## 2 Methodology

Your idea. You can rename this section if you like. Early on in this section – but not necessarily first – make clear what category your method falls into: Is it generative? Discriminative? Is there a particular additional data source you want to use?

## 3 Model

The math/architecture of your model. This should formally describe your idea from above. If you really want to, you can merge the two sections. **Xander is on this**

We define our sentence RNN as

$$h_i^s = GRU(s_i; h_{i-1}^s)$$

where  $s$  denotes the embedded sentence. We denote the final hidden state of the sentence RNN as  $r^s$ .

We then define our document RNN as

$$h_i^d = GRU(r_i^s; h_{i-1}^d)$$

and similarly denote the final hidden state of the document RNN as  $r^d$ .

Now, we define our generator RNN as

$$h_i^g = GRU(y_i; h_{i-1}^g)$$

where  $y_i$  is the embedded word generated at the previous time step. We set  $y_0$  to the embedded stop-word. Unlike the sentence and document RNN, where the initial hidden states  $h_0^s$  and  $h_0^d$  are set to 0, we initialize our generator hidden state as

$$h_0^g = r^d + z$$

$$z \sim \mathcal{N}(0, 1)$$

We determine the generated word from the generator hidden state using Gumbel-Softmax:

$$\pi_i = \text{softmax}(h_i^g W_{d \rightarrow e} W_e^T)$$

$$g_i = -\log(-\log(u))$$

$$u \sim \text{Uniform}(0, 1)$$

$$t_i = \text{relu}(h_i^g W_t) + \epsilon$$

$$p_i = \text{softmax}\left(\frac{\log(\pi_i) + g_i}{t_i}\right)$$

$$y_i = p_i W_e$$

where  $W_{d \rightarrow e}$  is a transformation matrix from document to embedding space,  $W_e$  is the embedding matrix,  $W_t$  is a matrix used to determine the temperature  $t_i$  and  $\epsilon$  is a small number to ensure that  $t_i$  is positive. We continue to generate words until the stop-word is generated (we use  $\pi_i$  to check) or the maximum sentence length is reached. We then stack the words to obtain the generated embedded sentence  $\bar{s}$ .

The discriminator score is defined as

$$D = \sigma(r^d W_{d \rightarrow s} (r_t^s)^T)$$

where  $W_{d \rightarrow s}$  denotes a transformation matrix from document space to sentence space and  $\sigma$  is the sigmoid function. The discriminator assigns the score 1 to target sentences  $r_t^s$  that it considers the most likely ending given the document context  $r_d$  and 0 to the least likely.

We use attention to assign weights to the inputs of the document RNN, based on their similarity to the target sentence. The weighted sentences are determined as

$$\tilde{r}_i^s = r_i^s \cdot a_i$$

$$a_i = r_t^s W_A (r_i^s)^T$$

where  $r_t^s$  denotes the target sentence,  $W_A$  is an attention matrix and  $\cdot$  denotes the scalar product. We do not use attention when determining the document context for sentence generation, as this would provide information about the ground truth target sentence to the generator.

Now, given a story without the ending (distilled to a single document representation using the sentence and document RNNs) and two endings (both distilled to sentence representations using the sentence RNN), we can determine the most likely ending as having the highest discriminator score  $D$ .

## 4 Training

What is your objective? How do you optimize it? **Xander is on this**

The discriminator loss is given by

$$L_D = -\log(D) - \log(1 - \bar{D})$$

where  $D$  denotes the discriminator score of the ground truth sentence and  $\bar{D}$  denotes the discriminator score of the generated sentence. The discriminator loss is minimized when the discriminator assigns a score 1 to the ground truth ending and a score 0 to the generated ending.

The generator loss is given by

$$L_G = -\log(\bar{D}) + \text{similarity}(r_t^s, r_g^s)$$

$$\text{similarity}(r_t^s, r_g^s) = 1 - \frac{r_t^s \cdot r_g^s}{\|r_t^s\| \|r_g^s\|}$$

where  $\bar{D}$  denotes the discriminator score of the generated sentence,  $r_t^s$  denotes the sentence representation of the target sentence and  $r_g^s$  denotes the target representation of the generated sentence. The second term in the loss is not traditionally used in GANs, but was added to improve the performance of the generator. The generator loss is minimized when the generator assigns a score 1 to the generated ending and the generated ending is maximally similar to the target ending.

During each epoch of training, the discriminator was trained for  $n_D$  batches and the generator for  $n_G$  batches, and the number of training batches were updated according to the ratio of the discriminator and generator losses:

$$n_D = \frac{L_D}{L_G}$$

$$n_G = \frac{L_G}{L_D}$$

The losses were averaged over the batches and the number of training batches were clipped to the range  $[1, 20]$ . The initial number of training batches were set to 1. We added noise to the generated sentences during discriminator training after 20\*\*\*\* training epochs. It has been proposed that adding noise to the discriminator input can theoretically improve GAN training by increasing the support intersection of the generator and true data distributions.

To improve the performance of the generator, pretraining was performed. During pretraining, the generator was not conditioned on a document context, much like a standard GAN. Specifically, the generator hidden state was initialized as

$$h_0^g = z$$

$$z \sim \mathcal{N}(0, 1)$$

and the pretrain generator loss is defined as

$$L_G^{pretrain} = similarity(r_t^s, r_g^s)$$

We used the AdaDelta optimizer with  $p = 0.999$  and  $\epsilon = 1e - 5$ , a learning rate of  $1e - 3$  and batch size 32. For embedding, we used pretrained word2vec weights (link\*\*\*) and did not update these during training. During data preprocessing, we added <BOS> and <EOS> tokens beginning and ends of sentences, padded sentences shorter than the max sentence length with <PAD> and replaced words not in our vocabulary with <UNK>. We used a vocabulary size of 25000\*\*\*\* and a max sentence length of 50. All trainable parameters were initialized using Xavier initialization. We used hidden state sizes of 128, 150 and 256 for the sentence, document and generator RNNs respectively.

## 5 Experiments

This **must** at least include the accuracy of your method on the validation set.

## 6 Conclusion

You can keep this short, too.