# ChiNet

**Nil Adell** **Joseph Cornelius** **Alexander Nedergaard** **Lama Saouma**

## 1 Introduction

The Story Cloze Task (SCT) is a recent dataset for machine comprehension. The test data consists of 5-sentence short stories with two different possible endings, and a label indicating which ending is the correct one. Stories are deliberately constructed such that determining the correct ending requires understanding the semantic content, a very challenging task for machines that humans can nonetheless complete with perfect accuracy. Another challenge of the dataset is that training data only includes correct endings, so it is not trivial to train a supervised model to solve the task.

## 2 Methodology

We attempted to reproduce results from a recent paper that uses a conditional generative adversarial network (CGAN) to solve the Story Cloze Task (SCT) (Wang et al, 2017). CGANs extend generative adversarial networks (GANs) (Goodfellow et al, 2014) to generate content conditioned on a context. The idea applied to SCT is to use a generatative neural network (the "generator") to generate fake story endings conditioned previous sentences in the story, and a discriminative neural network (the "discriminator") to discriminate between real and fake story endings. Trained together in an unsupervised and competitive manner using only stories with real endings, the generator and discriminator become increasingly good at generating and discriminating story endings. After training, the discriminator can then be used by itself to determine the real ending to a story. Thus, our approach is technically discriminitative but has a generative nature as it relies on a generative model during training.

## 3 Model

Our model consists of a sentence RNN, a document RNN, a generator RNN and a discriminator. The sentence RNN takes an embedded sentence as input and output a distilled representation of the sentence. Distilled sentences can then be fed to the document RNN to get a representation of the document context. The generator RNN takes in a document context and generates an embedded sentence. The discriminator takes as inputs a document context and a distilled sentence and outputs the probability that the distilled sentence is the real ending given the document context. Thus, the sentence and document RNNs are shared by the discriminator and generator, but for training we consider the document and sentence RNNs as part of the discriminator.

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

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

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

Determining the most the most likely word $y_i$ from our hidden state $h_i^g$ would usually involve an $argmax$ operation. However, as the $argmax$ operation is not continuous and we require end-to-end differentiability for backpropagation, we use the Gumbel-Softmax trick to perform reparameterisation with continous relaxation. Specifically, we determine the generated word from the generator hidden state as follows:

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

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

$$u \sim Uniform(0, 1)$$

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

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

$$y_i = p_i W_e$$

where $W_{d\to e}$ 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. With large $t_i$ we get an embeddings that are the average of all embedded words in the vocabulary, and as $t_i$ approaches 0 we get exact embeddings. We continue to generate words until the stop-word is generated (we threshold $\pi_i$ at 0.9 to check) or the maximum sentence length is reached. We then stack the words to obtain the generated embedded sentence $\bar{s}$.

We use attention to weigh 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.

The discriminator score is defined as

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

where $W_{d\to s}$ 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 are the most likely ending given the document context $r_d$ and 0 to the least likely.

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

We train our model by generating fake endings using the generator and passing them into the discriminator. The discriminator tries to discriminate real and fake endings, and the generator tries to trick the discriminator. The generator and discriminator are trained together, and we additionally pretrain the generator to improve performance.

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

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

$$similarity(s, \bar{s}) = 1 - \frac{s \cdot \bar{s}}{\|s\|\|\bar{s}\|}$$

where $\bar{D}$ denotes the discriminator score of the generated sentence, $s$ denotes the ground truth target and $\bar{s}$ denotes the generated target 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 discriminator assigns a score 1 to the generated ending and the generated ending is maximally similar to the target ending.

For each training epoch, the discriminator is trained with $n_D$ batches and the generator with $n_G$ batches, and the number of training batches are 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 are averaged over the batches and the number of training batches are clipped to the range $[1, 20]$. The initial number of training batches is set to 1. We add 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 intersecting support of the generator and true data distributions****.

Pretraining is performed to improve the performance of the generator. During pretraining, the generator is not conditioned on a document context and instead only receives random noise, much like a traditional GAN. Specifically, the generator hidden state is initialized as

$$h_0^g = z$$

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

and the pretrain generator loss is defined as

$$L_G^{pretrain} = similarity(s, \bar{s})$$

We use the AdaDelta optimizer with $p = 0.999$ and $\epsilon = 1e - 5$. Learning rate is set to $1e - 3$ and batch size to 32. For embedding, we use pretrained word2vec weights with embedding size 300 (link***) and do not update these during training. During data preprocessing, we add <BOS> and <EOS> tokens to the beginning and ending of sentences, pad sentences shorter than the max sentence length with <PAD> and replace words not in our vocabulary with <UNK>. We use a vocabulary size of 20000 and a max sentence length of 50. All trainable parameters are initialized using Xavier initialization. We use 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.