PreCNetLM: Language Modeling with Predictive Coding Framework

Zeyu Wang *

Paul G. Allen School of Computer Science and Engineering, University of Washington

December 2020

1 Introduction

Since the introduction of the predictive coding framework in Rao and Ballard (1999), there have been recent advances (Lotter et al., 2017, 2020; Straka et al., 2020) made in integrating modern deep learning techniques with the predictive coding framework for the video next frame prediction task. Specifically Lotter et al. (2017) and Lotter et al. (2020), introduced Predictive Coding Network (PredNet), which is a predictive coding inspired deep learning model architecture that is able learn to next frame prediction of videos of moving and or rotating objects. However, PredNet does not closely follow the predictive coding framework as pointed out in Rane et al. (2020) and Straka et al. (2020). The latter work took a step further and proposed the Predictive Coding Network (PreCNet) and showed that the model is able to perform better than PredNet in both next frame and multiple frame prediction on multiple datasets (Straka et al., 2020).

In this paper, I explore using the PreCNet architecture to model language (PreCNetLM) [1]. I modify the PreCNet model architecture from using CovLSTM with vanilla LSTM and removing / modifying any up-sample / down-sample / pooling operations. I evaluate the model against a vanilla LSTM model on perplexity.

The original authors of PredNet mentioned that "the architecture is general with respect to the kinds of data it models" while only focused on testing the model on image (video) data (Lotter et al., 2017). As using PredNet or PreCNet on language is yet to be explored, this paper tests the generality of such models.

*Email: `zwan48@cs.washington.edu`.

[1]The code used for replicating this paper is available on GitHub at `https://github.com/BBBBlarry/PreCNetLM`

2 Related work

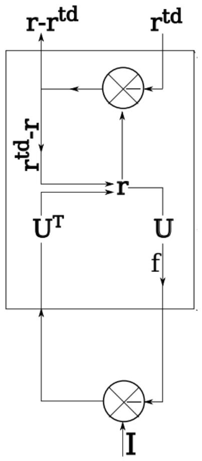
2.1 Predictive Coding

Rao and Ballard (1999) introduced and popularized the predictive coding framework. The development of the framework is inspired by the observation of the endstopping phenomenon in the animal visual cortex. Essentially, the phenomenon is observed when a line segment that would optimally excite a neuron goes beyond the classical reception field of a neuron, the neuron's response is reduced (see Figure 1b) (Rao and Ballard, 1999). The predictive coding framework proposes that the excitation (or inhibition) of these visual cortical neurons convey error signals rather than the what is within their reception field. Essentially, the neuron gets two inputs – the prediction of what would the signal be and what actually the signal is, and it outputs the error signal and pass it on to the higher level neurons in the cortex (see Figure 1a). When the line segment goes beyond a neuron's reception field, the cortex could predict that there is likely a line inside of the reception field; therefore the prediction error is reduced and the the output from the neuron is reduced. When given an input, the dynamics of the predictive schema is described as follows:

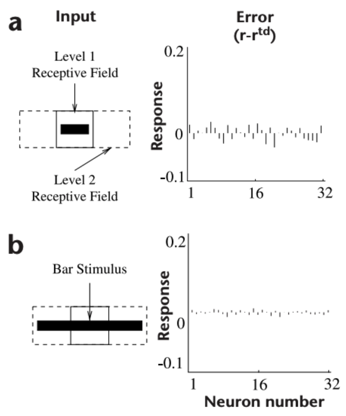
> ... the initial predictions at any given level are based on an arbitrary random combination of the basis vectors, giving large error signals. To minimize this error, the network converges to the responses that best predict the current input by subtracting the prediction from the input (via inhibition) and propagating the residual error signal to the neurons at the next level, which integrate this error and generate a better prediction. (Rao and Ballard, 1999)

2.2 PredNet

PredNet (Lotter et al., 2017, 2020) is an attempt at adopting the predictive coding framework in the modern neural network paradigm. The model architecture is shown in Figure 2a. In the paper, the model is used to predict the next frames in a video stream. On a high level, the model gets an input (A), compares to a prediction ($\hat{A}$), and pass the error signal (E) to both the representation layer (R) and to the next level above as input. To train the model, the PredNet objective minimizes the error (E). Lotter et al. (2017) found that minimizing the error (E) for all levels is better than only mini-



(a) The predictive coding schema. The input I is transformed to the representation r which is then compared to the prediction r^{td}. The error $r - r^{td}$ is passed on to a higher level and the prediction of the input $f(rU)$ is pass down to the lower level.



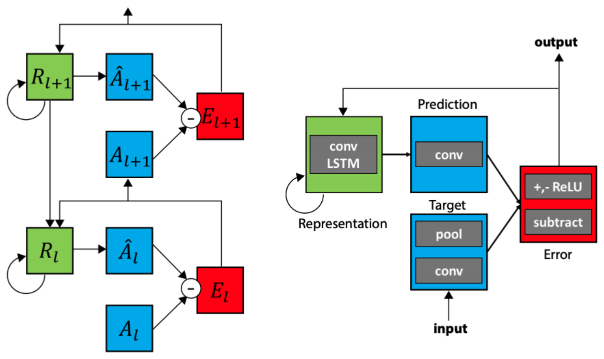
(b) The endstopping phenomenon observed in a model implemented predictive coding. When the visual signal (the bar) is just within the reception field of the Level 1 neurons, their outputs are large (b) in comparison to that of the neurons when the bar extends beyond their reception field.

Figure 1: The model used (1a) and endstopping phenomenon observed (1b) in Rao and Ballard (1999).

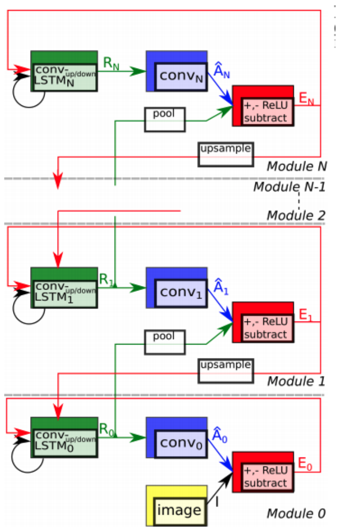
mizing the error for the bottom level on a toy dataset of rotating faces but the opposite is true when modeling the CalTech Pedestrian Dataset (Dollár et al., 2009). Notably, the PredNet model was able to achieve better performance (both mean squared error and SSMI) than the state-of-the-art model in the latter task.

2.3 PreCNet

PreCNet (Straka et al., 2020) improves upon PredNet by following the model proposed in Rao and Ballard (1999) more closely (see Figure 2b). At each level, instead of predicting the lower level error E like PredNet, PreCNet predicts the lower level representation R. In addition, the representation layers at different level do not directly communicate with one another. Straka et al. (2020) found that PreCNet outperforms PredNet in the Caltech Pedestrian Dataset in next frame and multiple frame predictions for most time steps, but falls short comparing to the state-of-the-art models: RC-GAN (Kwon and Park, 2019) and CrevNet (Yu et al., 2020). In addition, they found that minimizing only the bottom level error improves the model performance,



(a) An illustration of PredNet (Lotter et al., 2017, 2020). Notably, the PredNet model differ from the original predictive coding model in two ways: 1) the model predicts the lower level error rather than the representation, and 2) the representation layers are directly connected to one another rather than not.



(b) An illustration of PreCNet (Straka et al., 2020).

Figure 2: Illustrations of deep learning models from previous works that uses the predictive coding framework.

similar to PredNet on the Caltech Pedestrian Dataset.

3 Methodology

3.1 Model Architecture

PreCNetLM adopts PreCNet's model training and inference algorithms but modified the architecture by replacing the vision-specific components to components that are used in general settings (see Figure 3). Specifically, I replaced all of the convolutional layers and upsampling layer with feedforward layers. I also replaced the convolutional LSTM layers with vanilla LSTM layers. Lastly, I removed all of the pooling layers.

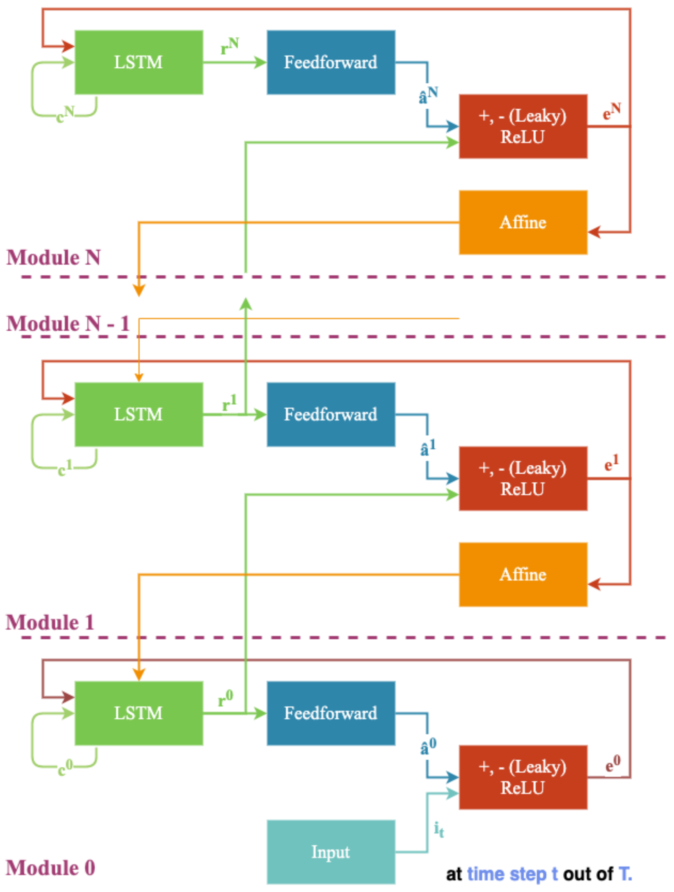


Figure 3: The architecture of PreC-NetLM. Each module contains representation layer (outputs $\mathbf{r}$), decoding layer (outputs $\hat{\mathbf{a}}$), error representation ($\mathbf{e}$). $\mathbf{c}$ is the cell state / hidden state that is fed back to the representation layers at every time step. The affine transformations convert the dimension of the error representation of the module to that of the lower level module. The representation layer takes either the the error representation $\mathbf{e}^l$ from the same level (l) or that of the level above $\mathbf{e}^{l+1}$ passed through the aforementioned affine transformation. The decoding layer at level l takes $\mathbf{e}^l$ and outputs a prediction $\hat{\mathbf{a}}^l$ of lower level's representation layer output r^{l-1} (or the input i_t if l $==$ 0). Lastly, the error representation is the concatenation of +/- prediction errors passed through a (leaky) ReLU unit. See Algorithm 1 for more details.

Algorithm 1: Calculate PreCNetLM states at time step t, assume $N > 0$.

```
Require: Input vector $\mathbf{i}_t$, previous hidden / cell states $\mathbf{c}^l_{t-1}$ of the
         representation layers $l \in \{0, 1, ..., L\}$, previous error state
         $\mathbf{e}^N_{t-1}$ of the top level module $N$.
for $j \leftarrow N$ to $0$ by $-1$ do // iterate top down
    if $l == N$ then // top level module
        $\mathbf{r}^l_t, \mathbf{c}^l_t \leftarrow \mathrm{LSTM}^l(\mathbf{e}^l_{t-1}, \mathbf{c}^l_{t-1})$;
        $\hat{\mathbf{a}}^l_t \leftarrow \mathrm{Feedforward}^l(\mathbf{r}^l_t)$;
        $\mathbf{e}^l_t \leftarrow \mathrm{ReLU}(\mathrm{concat}(\hat{\mathbf{a}}^l_t - \mathbf{r}^{l-1}_{t-1}), (\mathbf{r}^{l-1}_{t-1} - \hat{\mathbf{a}}^l_t))$;
    else if $l < N$ and $l > 0$ then// middle level modules
        $\mathbf{r}^l_t, \mathbf{c}^l_t \leftarrow \mathrm{LSTM}^l(\mathrm{affine}^{l+1}(\mathbf{e}^{l+1}_{t-1}), \mathbf{c}^l_t)$;
        $\hat{\mathbf{a}}^l_t \leftarrow \mathrm{Feedforward}^l(\mathbf{r}^l_t)$;
        $\mathbf{e}^l_t \leftarrow \mathrm{ReLU}(\mathrm{concat}(\hat{\mathbf{a}}^l_t - \mathbf{r}^{l-1}_{t-1}), (\mathbf{r}^{l-1}_{t-1} - \hat{\mathbf{a}}^l_t))$;
    else // bottom level module
        $\mathbf{r}^l_t, \mathbf{c}^l_t \leftarrow \mathrm{LSTM}^l(\mathrm{affine}^{l+1}(\mathbf{e}^{l+1}_{t-1}), \mathbf{c}^l_t)$;
        $\hat{\mathbf{a}}^l_t \leftarrow \mathrm{Softmax}(\mathrm{Feedforward}^l(\mathbf{r}^l_t))$;
        $\mathbf{e}^l_t \leftarrow \mathrm{ReLU}(\mathrm{concat}(\hat{\mathbf{a}}^l_t - \mathbf{i}_t), (\mathbf{i}_t - \hat{\mathbf{a}}^l_t))$;
    end
end
for $j \leftarrow 0$ to $N$ by $1$ do // iterate bottom up
    if $l == 0$ then // bottom level module
        $\mathbf{r}^l_t, \mathbf{c}^l_t \leftarrow \mathrm{LSTM}^l(\mathbf{e}^l_t, \mathbf{c}^l_t)$;
    else if $l < N$ and $l > 0$ then// middle level modules
        $\mathbf{e}^l_t \leftarrow \mathrm{ReLU}(\mathrm{concat}(\hat{\mathbf{a}}^l_t - \mathbf{r}^{l-1}_t), (\mathbf{r}^{l-1}_t - \hat{\mathbf{a}}^l_t))$;
        $\mathbf{r}^l_t, \mathbf{c}^l_t \leftarrow \mathrm{LSTM}^l(\mathbf{e}^l_t, \mathbf{c}^l_t)$;
    else // top level module
        $\mathbf{e}^l_t \leftarrow \mathrm{ReLU}(\mathrm{concat}(\hat{\mathbf{a}}^l_t - \mathbf{r}^{l-1}_t), (\mathbf{r}^{l-1}_t - \hat{\mathbf{a}}^l_t))$;
    end
end
```

3.2 Inference Procedure

The inference procedure of PreCNetLM is the same as that of the PreCNet. Essentially, at each time step t with an input sequence $\mathbf{i}_t$, the model performs a 1) prediction and a 2) correction. At prediction phase, the model tried to predict the input $\mathbf{i}_t$; at correction phase, the model propagates the prediction error $\mathbf{e}_0$ upwards. The more details, see Algorithm 1.

When performing inference beyond one time step, following Straka et al. (2020), the input $\mathbf{i}_t$ is replaced with the model's prediction $\hat{\mathbf{a}}_t^0$, i.e. the prediction error at the bottom ($\mathbf{e}_0$) will be a zero vector.

3.3 Training Procedure

The training objective is to minimized the weighted prediction error.

$$Loss(\mathbf{I}) = \sum_{t=1}^{T} \sum_{l=0}^{L} \frac{\mu_l}{n_l} \sum_{j=0}^{n_l} \mathbf{e}_{t,j}^l \tag{1}$$

In Equation 1, $\mathbf{I}$ is the the input matrix where each row ($\mathbf{i}_t$) is the one hot encoding of the input sequence at each time step t. T is the total number of time steps in $\mathbf{I}$. L is the number of levels (modules) in the model. $\mathbf{e}_t^l$ is a vector of error representation at level l at time step t when modeling input $\mathbf{I}$. n_l is the magnitude of the vector $\mathbf{e}_t^l$. Lastly, μ_l is module level weighting factor. Note that $\mathbf{e}_t^l$ is calculated based on $\mathbf{I}$ (see Algorithm 1).

4 Experiments

I run language modeling task with PreCNetLM on several data sets. A set of simple experiments are used to evaluate the basic learning capabilities of the model. I also run it the text of the entire volume of Harry Potter [2] and a data set of tweets from 2014 US Gubernatorial Election candidates (Wang, 2020).

[2] https://courses.cs.washington.edu/courses/cse599g1/19au/files/homework3.tar.gz

Task Name	Description	Example
Repeating Single Digit	Repeat a random token t in the vocabulary V.	[1, 1, 1, 1, 1, 1, ...]
Sequence of Numbers	Repeat a sequence of numbers from 0 to $\|V\| - 1$.	[0, 1, 2, 3, 0, 1, ...]
Sequence of Repeating Numbers	Repeat a sequence of repeating numbers from 0 to $\|V\| - 1$.	[0, 0, 1, 1, 2, 2, 3, 3, 0, 0, ...]
Binary Counting	Prompted by a number n, then repeats 1 for n times and 0 for n times, etc.	[3, 1, 1, 1, 0, 0, 0, 1, 1, 1, ...]

Table 1: A summary of all of the simple experiment tasks.

4.1 Simple Experiments

The simple experiments are consisted of a set of artificial toy data sets. The tasks that are summarized in Table 1.

The first three tasks should be easy enough for a model to learn because the model does not need to "look" beyond the last time steps. However, binary counting is difficult because 1) the model needs to remember the prompt, 2) it also needs to keep track of how many 1s or 0s it has predicted, and 3) it needs to model the relationship between the prompt and the concept of counting.

4.2 Penalization Experiments

In previous literature, the authors found that penalizing the upper level E unit outputs improves results. In this work, I experiment with whether this effect is still true in with PreCNetLM. I run the same set of simple experiments on two different versions of PreCNetLM — one with upper level penalization and one without. In the penalization version, the E unit outputs of the lowest level is penalized with $\mu_0 = 1.0$ and the rest of the levels are penalized with $\mu_l = 0.1$. The no-upper-level-penalization version only has $\mu_0 = 1.0$ for the lowest level and $\mu_l = 0.0$ for the rest. The full specification of all of the experiments are presented in the Section 7.

4.3 Harry Potter Texts

In this experiment, the text of entire volume of Harry Potter is used for the language modeling task at the character level. The data is divided into sequences of 100 characters. I split the data into train and dev sets in a 90%-10% split and is fed into the models in batches of 4. Each character is encoded with one-hot encoding.

4.4 Twitter Texts

Similar to the above experiment, in this experiment, I use tweets from the U.S. 2014 Gubernatorial elections (Wang, 2020) for language modeling task at the character level. These tweets were collected from all the candidates' campaign twitter account's timeline during the three months leading up to the election. During data loading, the tweets are sorted by length and each batch's sequence length is truncated into the shortest tweet in the batch. The data is divided into train and dev sets in a 90%-10% split and is fed into the models in batches of 4. Each character is encoded with one-hot encoding.

4.5 Evaluation

For each of the Experiment 4.1, 4.3, and 4.4, I compare the perplexity of the PreCNetLM with a similar-sized vanilla LSTM. For Experiment 4.2, I compare the perplexity of the two different versions of the model. Note that we cannot meaningfully compare the loss because the penalization scheme is slightly different for the two versions of the PreCNet. However, we can use training loss over batches to compare the speed of convergence.

5 Results

First of all, the penalization experiment (Table 2) shows that penalizing on the bottom level improves the perplexity of PreCNetLM for 3 out of the 4 simple experiment tasks. I observe a performance gain in the sequence of numbers task where using the bottom-only penalization scheme improved perplexity by a large margin. In the only task that bottom-only penalization schema failed to outperform the other, binary counting, neither model converged. In addition, Figure 4 shows that the model converge around the same time despite the penalization scheme. These results suggest that penalizing only the bottom level is a good way to improve model performance but not convergence speed.

PreCNetLM falls short on beating LSTM in perplexity in all of the simple experiments and on modeling both the Harry Potter text and Twitter text dataset (see Table 3, 4). I observe that when PreCNet is able to converge, it does not fall significantly behind LSTM. This is observed in repeating single digit, sequence of numbers, and sequence of repeating numbers task. However, when PreCNetLM cannot learn the task (i.e. binary counting, Harry Potter text, Twitter text) the perplexity becomes really large.

Lastly, it is worth noting that PreCNetLM runs about 3 times slower than the LSTM model with a comparable number of parameters due to the complexity of the inner control structure of PreCNetLM. To train for Harry Potter text, the model takes 6 hours per epoch while LSTM takes about 2 hours; to train for the Twitter text, PreCNet takes about 3 hours per epoch and LSTM takes about 1 hour.

6 Discussion and Future Works

Even though the results from vision makes PreCNet a promising candidate for language modeling, my work shows that this is not true. The model fails to converge on the the language modeling tasks and the perplexity is far to great for the model to be useful.

There are several potential reasons that PreCNetLM did not work for more complicated data. Firstly, the optimization objective for PreCNetLM does not directly optimize for perplexity; instead, it optimizes for difference between prediction and input. Sequence-to-sequence models in deep learning like LSTM uses cross entropy loss as the optimization objective, which is directly related to perplexity. Another reason could be that PreCNetLM does not take advantage of embeddings that are commonly used in natural language processing. Because the model tries to predict exactly as the data comes in, it is difficult to incorporate a word (or character) embedding into the framework. Therefore, I encourage future research to be done on

Task	Penalize Upper	Dev Perplexity (Lower the better)
Repeating Single Digit	True	1.30
Repeating Single Digit	False	**1.14**
Sequence of Numbers	True	6.05
Sequence of Numbers	False	**1.14**
Sequence of Repeating Numbers	True	1.56
Sequence of Repeating Numbers	False	**1.14**
Binary Counting	True	**41.3**
Binary Counting	False	48.6

Table 2: The results of the penalization experiments. I evaluated perplexity for the two versions of PreCNetLM – one with upper level penalization and one without (see Section 4.2). The results show that penalizing upper level negatively impacts the performance for 3 out of the 4 tasks. For the binary counting task, both version failed to converge.

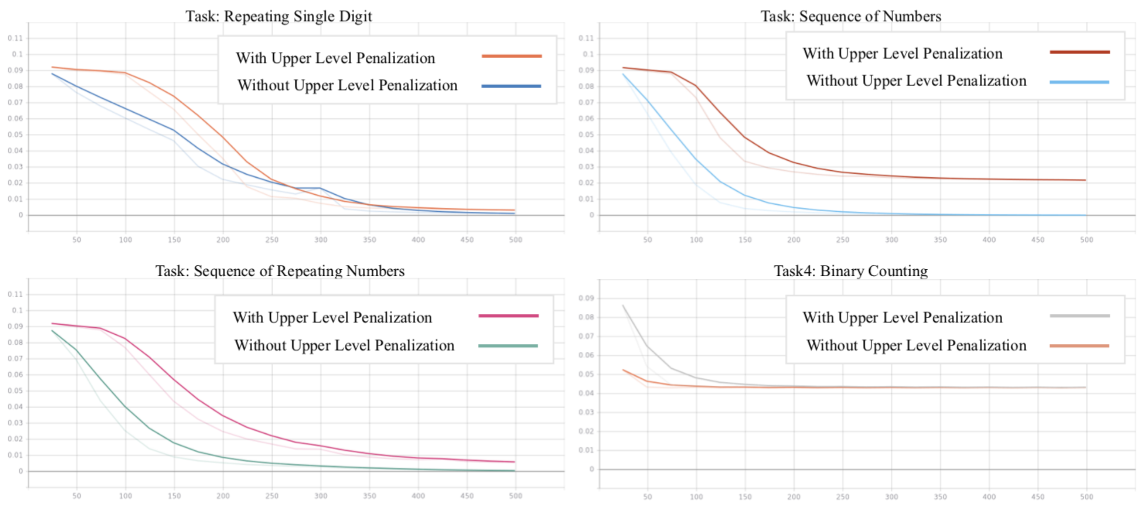


Figure 4: The training losses of the penalization experiments. Even though the two losses are not comparable due to different training objectives, we can observe that both converge at around the same time.

Task	Model	Dev Perplexity (Lower the better)
Repeating Single Digit	PreCNetLM	1.97
Repeating Single Digit	LSTM	**1.01**
Sequence of Numbers	PreCNetLM	1.13
Sequence of Numbers	LSTM	**1.02**
Sequence of Repeating Numbers	PreCNetLM	1.14
Sequence of Repeating Numbers	LSTM	**1.04**
Binary Counting	PreCNetLM	40.1
Binary Counting	LSTM	**1.42**

Table 3: The results of the simple experiment comparisons between PreCNetLM and LSTM. LSTM outperforms PreCNetLM at every task in perplexity. Notably, while PreCNetLM did not converge in the binary counting task, LSTM was able to converge.

Task	Model	Dev Perplexity (Lower the better)
Harry Potter	PreCNetLM	1.22×10^7
Harry Potter	LSTM	**3.92**
Twitter	PreCNetLM	3.91×10^{10}
Twitter	LSTM	**35.9**

Table 4: The results of character level language modeling using both PreCNetLM and LSTM. The perplexities are measured on character level.

incorporating cross entropy loss in PreCNetLM and explore options to add embeddings to the model.

7 Appendix

7.1 Experiment specifications

7.1.1 Simple Experiments and Penalization Experiments

The vocabulary for all of the simple experiment is $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The data is fed in with a sequence length of 20 and batch size of 16 and 10 batches in total. The PreCNetLM model[3] contains 3 level of modules, each has 1 $\hat{\mathbf{a}}$ layer of size 64 and 2 $\mathbf{r}$ layers of size 32 (total parameters: 59600). It was trained with $\mu = [1.0, 1.0, 1.0]$. For penalization setup the configuration for the *penalize_upper* model is $\mu = [1.0, 0.1, 0.1]$. The LSTM model contains 2 layers of hidden layers 64 (total parameters: 53350). Both models are trained for 50 epochs.

7.1.2 Harry Potter and Twitter Experiments

The PreCNetLM model contains 3 level of modules and 1 $\hat{\mathbf{a}}$ layer of size 128 and $[3, 2, 1]$ $\mathbf{r}$ layers of size 128 (total parameters: 917000 for Harry Potter, and 1000000 for Twitter). They were both trained with $\mu = [1.0, 1.0, 1.0]$. For LSTM, it contains 7 hidden layers of size 128 (total parameters: 915500 for Harry Potter, and 963000 for Twitter). Both models are trained for 5 epochs.

Acknowledgement

Thank you Prof. Rajesh P. N. Rao and Beibin Li for providing support and teaching throughout the Autumn Quarter of 2020. Also a big shout out to Falcon (2019) for providing PyTorch Lightning which significantly accelerated the development process for this paper.

[3]The specifications here do not show all layers in the model, but rather the configurable hyperparameters.

References

Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, 2009. ISBN 9781424439935. doi: 10.1109/CVPRW.2009.5206631.

WA Falcon. PyTorch Lightning, 2019. URL https://github.com/PyTorchLightning/pytorch-lightning.

Yong Hoon Kwon and Min Gyu Park. Predicting future frames using retrospective cycle gan. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019. ISBN 9781728132938. doi: 10.1109/CVPR.2019.00191.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–18, 2017.

William Lotter, Gabriel Kreiman, and David Cox. A neural network trained for prediction mimics diverse features of biological neurons and perception. *Nature Machine Intelligence*, 2(4):210–219, 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-0170-9. URL http://dx.doi.org/10.1038/s42256-020-0170-9.

Roshan Prakash Rane, Edit Szügyi, Vageesh Saxena, André Ofner, and Sebastian Stober. PredNet and predictive coding: A critical review. *ICMR 2020 - Proceedings of the 2020 International Conference on Multimedia Retrieval*, pages 233–241, 2020. doi: 10.1145/3372278.3390694.

Rajesh P.N. Rao and Dana H. Ballard. Hierarchical Predictive Coding Model Hierarchical Predictive Coding of Natural Images. *Nature neuroscience*, 2 (1):79, 1999. URL http://neurosci.nature.com.

Zdenek Straka, Tomas Svoboda, and Matej Hoffmann. PreCNet: Next Frame Video Prediction Based on Predictive Coding. 2020. URL http://arxiv.org/abs/2004.14878.

Zeyu Wang. Us v.s. Them in 280 Characters: Why Political Polarization Fuels Vicious Attacks on Twitter. pages 1–37, 2020. URL https://github.com/BBBBlarry/US-v.s.-Them-in-280-Characters/blob/master/paper/Wang_2020.pdf.

Wei Yu, Yichao Lu, Steve Easterbrook, and Sanja Fidler. Efficient and information-preserving future frame prediction and beyond. *Iclr*, (2016): 1–14, 2020.