Regularization Methods LSTMS & RNNS

Jason Durant – jd3720 & Ashutosh Rawat - at4171

Executive Summary

Goal: Identify regularization methods to increase performance of for RNNs and LSTMS

Solution : Train Two RNN and LSTM Models with varying regularization methods and hyperparameters (e.g. Optimizer, Epochs, Batch Size) in order to construct models with increased robustness.

Value : Overfitting on a variety of tasks can be reduced and more complex RNN and LSTM Models can be utilized for increased performance on a whole host of tasks

Problem Motivation

Often Regularization is applied naively to LSTM and RNN models

- Most existing methods of regularization only provide a small gain in (Dropout & Batch Normalization)
- Large RNNS tends to overfit, as such practical applications of RNNs are small
- By identifying regularization techniques which lead to large performance RNNs and LSTMs can have extended use on a wide range of problems

Background Work

- Recurrent Neural Network Regularization[1]
  - Simple regularization technique for LSTM Unites
- Regularizing and Optimizing LSTM Language Models[2]
  - DropConnect on hidden to hide
- Recurrent Dropout without Memory Loss[3]
  - Drop neurons directly in connections w/o loss of long term memory
- Weight Regularization with LSTM Networks[4]
- Recurrent Batch Normalization[5]
  - For RNNs and LSTM Networks
- Layer Normalization[6]
  - Effective at stabilizing hidden state dynamics in a RNN

Background Work

- Review of Dropout as applied to RNNS[7]
  - Various approaches to Dropout for RNNs reviewed
- Pytorch Implementation of Recurrent Dropout[8]

Model Architectures

- PredRNN: Official Implementation Pytorch[9]
- ConvLSTM Keras Implementation[10]

Technical Challenges

- RNNs are super susceptible to overfitting
- Naively applying regularization methods give relatively small improvements for RNNs
- Applying conventional dropout does not work well with RNNs because the recurrence amplifies noise
- In order to solve this problem dropout is applied only to non recurrent layers
- This is done by masking the inputs and outputs to the layer dropout is applied to

Technical Challenges

- Choosing the correct metrics by which to analyze the performance of these regularization methods
- Using certain variations of Dropout in the LSTM Cells led to memory loss and corruption in LSTM cells
- Various regularization methods which could be affected by changing hyperparameters of the model were constrained by computer resources (e.g. batch size)
- Regularization methods such as L2 Weight Decay among other have their own hyperparameters which needed to be tested as well adding to the number of combinations

Approach

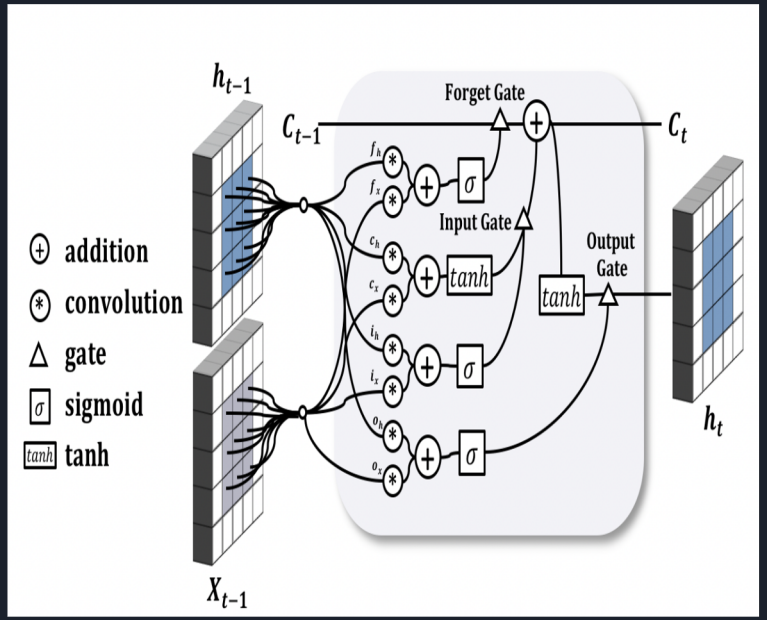
- Trained two RNN and LSTM network :
  - PredRNN++
  - ConvLSTM
- Moving MNIST dataset is used for training and validation
- Dataset contains 10,000 sequences each of length 20 showing 2 digits in 64x64 frames
- A subset of the above dataset is used for training due to the computational constraints

Approach

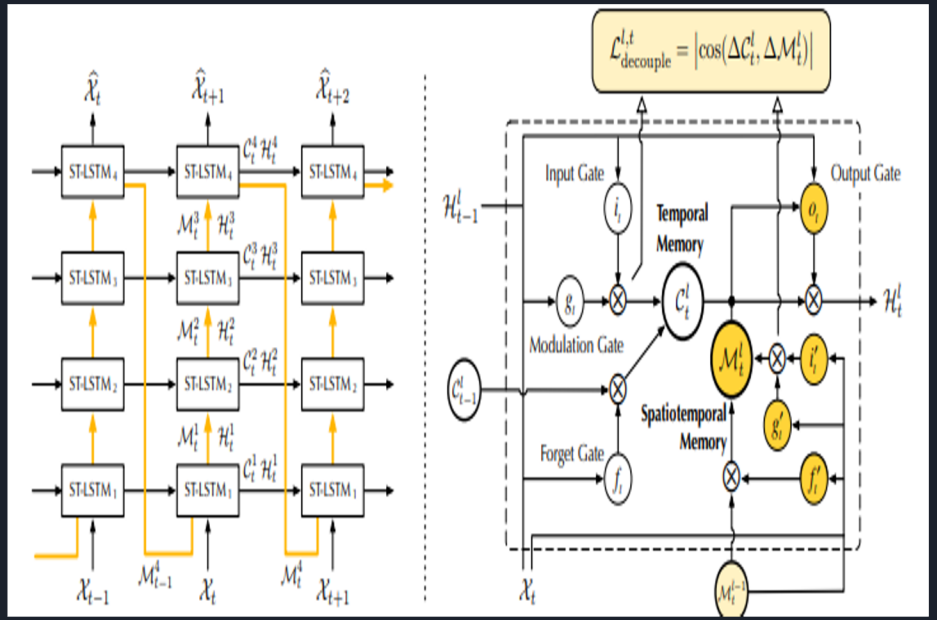
- For quantitative comparison we used the following metics:
  - MSE (Mean Square Error)
  - SSIM (Structural Similarity Index)
  - PNSL (Peak Signal to Noise Ratio)
- For both the network architecture learning is performed by optimizing the MSE loss
- We also focused on different optimization algorithms, as use of regularizers can impede the learning process
- Optimizers that performed best were further used to study the effect of regularizers

Architecture

ConvLSTM



PredRNN++



Implementation

- The PredRNN++ was implemented in Pytorch and ConvLSTM was implemented using Tensorflow
- Both the network were trained on GCP using K80 GPU
- The networks were trained on Moving MNIST dataset for 20 Epochs, with a batch size of 5
- The network were trained for only 20 Epochs because of hardware and time constraints
- Lower batch size were used because of memory constraints of the GPU

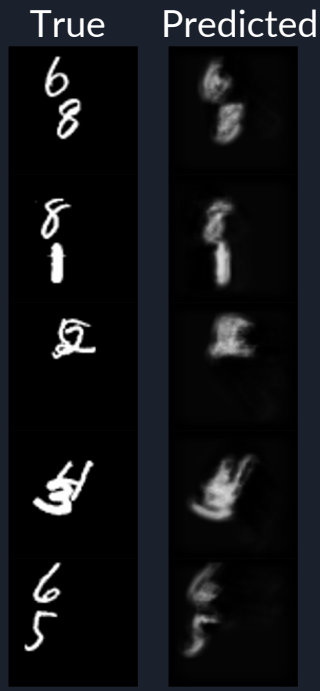
Experiment Design Flow

1. Base ConvLSTM and PredRNN models were selected as the model to utilize for analysis
2. MSE, SSIM, and PSNR metrics were selected to compare the effects of regularization on RNNs and LSTMS based models
3. Regularization methods were then applied to each of the two models
    a. Metrics were then collected and compared
    b. Qualitative image frames were also retrieved from the regularized models.
4. Metrics were then evaluated to compare performance between the various regularization methods.

Experimental Evaluation

Qualitative analysis of the prediction made using ConvLSTM

- The results contains the true 10 frames which are made into a gif, it is compared against the predicted frames
- The predicted frames are bit blurry but they are able to capture the important information



Experimental Evaluation

Performance of the base network:

	MSE	PSNR (db)	SSIM
PredRNN++	218.36	16.0535	0.7808
ConvLSTM	0.118	42.7138	0.9880

* Low value of MSE in ConvLSTM because the images were normalized (0-1)

Experimental Evaluation

Performance of the different regularization approach on PredRNN++:

	MSE	SSIM	PSNR
Base	218.36	0.7808	16.0535
Layer Norm	280.00	0.7055	15.0658
Dropout	216.23	0.7901	16.0300
Recurrent Dropout	215.78	0.8012	16.0122

Experimental Evaluation

Performance of the different regularization approach on ConvLSTM:

	MSE	SSIM	PSNR
Base	0.118	0.9880	42.7138
Kernel Regularizer (L2)	0.119	0.9662	31.09837
Dropout	0.117	0.9892	43.7330
Recurrent Dropout	0.118	0.9884	43.7605

Conclusion

Through experimentation with various regularization methods, variations of Dropout regularizations performed the best compared to other methods such as weight decay, batch normalization, and layer normalization.

Various methods such as Layer Normalization actually performed significantly worse than the base model performance indicating that not all regularization methods lead to increased performance.

By experimenting with Dropout variations for RNNs and LTSMs in the future there is a possibility that larger and more complex models which take into account memory can be used without concern for overfitting

References

[1] Zaremba, Wojciech, et al. “Recurrent Neural Network Regularization.” ArXiv:1409.2329 [Cs], Feb. 2015. arXiv.org, http://arxiv.org/abs/1409.2329.

[2]Merity, Stephen, et al. “Regularizing and Optimizing LSTM Language Models.” ArXiv:1708.02182 [Cs], Aug. 2017. arXiv.org, http://arxiv.org/abs/1708.02182.

[3]Semeniuta, Stanislau, et al. “Recurrent Dropout without Memory Loss.” ArXiv:1603.05118 [Cs], Aug. 2016. arXiv.org, http://arxiv.org/abs/1603.05118.

[4] Weight Regularization with LSTM Networks for Time Series Forecasting , https://machinelearningmastery.com/use-weight-regularization-lstm-networks-time-series-forecasting/

[5]Cooijmans, Tim, et al. “Recurrent Batch Normalization.” ArXiv:1603.09025 [Cs], Feb. 2017. arXiv.org, http://arxiv.org/abs/1603.09025.

[6]Ba, Jimmy Lei, et al. “Layer Normalization.” ArXiv:1607.06450 [Cs, Stat], July 2016. arXiv.org, http://arxiv.org/abs/1607.06450.

[7]G, Adrian. “A Review of Dropout as Applied to RNNs.” Medium, 24 June 2018, https://adriangcoder.medium.com/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b.

[8]“Implementing Recurrent Dropout.” PyTorch Forums, 26 July 2017, https://discuss.pytorch.org/t/implementing-recurrent-dropout/5343

References

[9]https://github.com/thuml/predrnn-pytorch

[10]https://github.com/keras-team/keras-io/blob/master/examples/vision/conv_lstm.py?fbclid=IwAR3XrzP4jI-Q0O5K-IqquDHpvBekQr-FN4zlJxVvjLp0ic_EWxGDuQF-9Cs

Thank You