Variational Attention

Cyril de Lavergne

May 2020

1 Motivation

The main goal of this paper is to replicate the work of the following paper [2] and [3] for numerical time series data. This implies removing the embedding layer. Both experiments in [2] quote that VED+VAttn-h performs best according to the following table:

Model	Inference	BLEU-2	Entropy	Dist-1	Dist-2
DED+DAttn	MAP	1.84	–	–	–
VED+DAttn	MAP	1.68	–	–	–
	Sampling	1.68	2.113	0.311	0.450
VED+VAttn-$\bar{h}$	MAP	1.78	–	–	–
	Sampling	1.79	**2.167**	**0.324**	**0.467**

Table 4: Performance on conversation systems.

2 Feature

This paper use hourly bar data collected from Interactive brokers, a popular online broker. After collecting VIX prices, we create diverse technical indicators and generate random convolutional kernels at a number of 2000 as in [1], a new SOT algorithm on time series tasks. Note, we binarize the output with faster loading to conduct experiments with Numpy files.

Random convolational kernels are simulated based on:

- Length. Length is selected randomly from $7, 9, 11$ with equal probability,making kernels considerably shorter than input time series in most cases.
- Weights. The weights are sampled from a normal distribution, $w \sim \mathcal{N}(0, 1)$ and are mean centered after being set. As such, most weights are relatively small, but can take on larger magnitudes.

- Bias. Bias is sampled from a uniform distribution, $w \sim \mathcal{U}(-1, 1)$ Only positive values in the feature maps are used. Bias therefore has the effect that two otherwise similar kernels, but with different biases, can 'highlight' different aspects of the resulting feature maps by shifting the values in a feature map above or below zero by a fixed amount.

- Dilation. Dilation is sampled on an exponential scale, which ensures that the effective length of the kernel, including dilation, is up to the length of the input time series. Dilation allows otherwise similar kernels but with different dilations to match the same or similar patterns at different frequencies and scales.

- Padding. When each kernel is generated, a decision is made (at random, with equal probability) whether or not padding will be used when applying the kernel. If padding is used, an amount of zero padding is appended to the start and end of each time series when applying the kernel, such that the 'middle' element of the kernel is centered on every point in the time series. Without padding, kernels are not centered at the first and last points of the time series, and 'focus' on patterns in the central regions of time series whereas with padding, kernels also match patterns at the start or end of time series.

Each kernel is applied to each input time series, producing a feature map. The convolution operation involves a sliding dot product between a kernel and an input time series.

$$X_i * \omega = \sum_{j=0}^{l_{\text{kernel}}-1} X_{i+(j\times d)} \times \omega_j.$$

Note, we use the same convolutional kernels for validating and testing as in the training phase to avoid look-ahead bias. According to the paper [1] and for LSTM encoder and decoder stability, we proceed to scale in the range of 0 and 1 each feature generated and also the target.

The label in this experiment is the future log returns at time $t + 1$ of the VIX asset.

3 Model

The model is composed of an encoder and decoder with attention mechanism. The encoder is a bidirectional LSTM with 100 hidden units with a high dropout of 50% to avoid overfitting. The choice of these hyperparameters were chosen from experience. Between the encoder and decoder, we build a latent space of 100 dimensions sampled from a Gaussian distribution. The decoder is however build with an LSTM cell with Luong attention with a sampling temperature of

1. We pass to the attention mechanism the encoder output of each batch. At each training step of the decoder, we accumulate the context KL loss from time series inputs. We then proceed to concatenate to the next batch, latent vectors. Training logits of each batch are obtained after processing of the attention cell with the current states.

The inference part dynamically decode from the attention cell, the logits.

The loss function is computed based on the Kullback Leibler divergence given as follows:

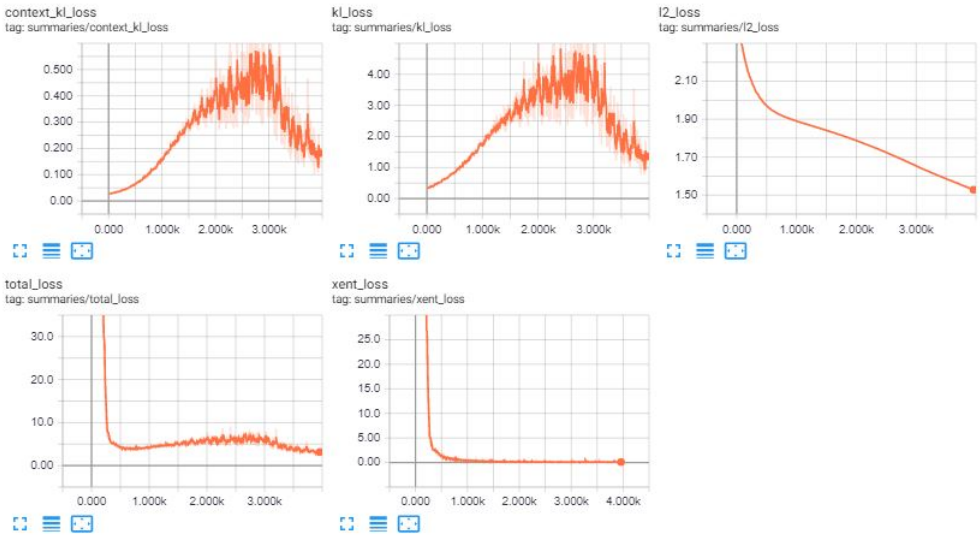
```
def calculate_kl_loss(self):
    """(Gaussian) Kullback-Leibler divergence KL(q||p), per training example"""
    # (tf.Tensor, tf.Tensor) -> tf.Tensor
    with tf.name_scope("KL_divergence"):
        return -0.5 * tf.reduce_sum(1.0 + 2 * self.z_log_sigma - self.z_mean ** 2 -
                                    tf.exp(2 * self.z_log_sigma), 1)

def build_decoder(self):
```

In addition we clip the mean squared error between 1. and -1. with the target and the training logits. The we compute the L2 loss regularization on the loss. Finally we add the context Kullback Leibler divergence and sum all these variables as the cost.

The training step involved KL Annealing after some iteration to help variables to converge as you can see it decreasing in Results section.

4 Results

Training results of the experiments are as follows:

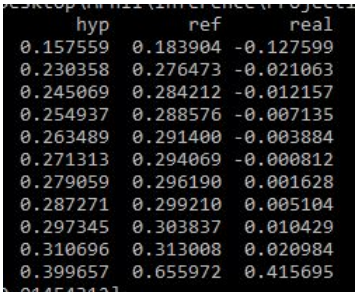


Note we validate at each step to check which iteration has best performed to use on the testing dataset. Output of the validation is mainly based on the correlation between target and predictions. A sample output is as follows:

```
Epoch  13/100 - Time   18.1 BLEU: -0.0005463078183658719
references_val 0.5970887459703875 -0.11168842427705722
hypotheses_val 0.38301307 0.13499907
mean_squared_error 0.12593087047814858
corr -0.011438556737663561
val_bleu_str -0.011438556737663561
Epoch  14/100 - Time   20.5 BLEU: -0.011438556737663561
references_val 0.5970887459703875 -0.11168842427705722
hypotheses_val 0.39957505 0.15109812
mean_squared_error 0.12536630487380743
corr 0.03649470273891831
val_bleu_str 0.03649470273891831
Epoch  15/100 - Time   23.3 BLEU: 0.03649470273891831
```

Hence, we take the epoch 15 as checkpoint to test our dataset and we can check the quantiles of the distribution of the hypothesis vs the label (standardized).

We can clearly see that the hypothesis are downward skewed probably due to some outliers in the training dataset. Therefore we fit an empirical distribution and long (buy) if the probability is above 0.5 or 0.9, and inversely short (sell) if the probability is below 0.5 or 0.1. In every case, the correlation is at -1.%. The model therefore did not capture either the future prediction returns nor the tail probability distribution.



hyp	ref	real
0.157559	0.183904	-0.127599
0.230358	0.276473	-0.021063
0.245069	0.284212	-0.012157
0.254937	0.288576	-0.007135
0.263489	0.291400	-0.003884
0.271313	0.294069	-0.000812
0.279059	0.296190	0.001628
0.287271	0.299210	0.005104
0.297345	0.303837	0.010429
0.310696	0.313008	0.020984
0.399657	0.655972	0.415695

5 Conclusion

To conclude, the model could be improved by including the validation loss to iterate over the training dataset to obtain better latent space representation of variables.

We should also remove outliers in the data pre-processing step to avoid skewness in the output distribution.

In addition we should use another distribution than Gaussian for the latent variables in this experiment aiming to replicate the heavy tail (high kurtosis) often found in the financial world. We advise skew Student's t distribution.

References

[1] Geoffrey I. Webb Angus Dempster, François Petitjean. Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. 2019.

[2] Olga Vechtomova Pascal Poupart Hareesh Bahuleyan, Lili Mou. Variational attention for sequence-to-sequence models. 2018.

[3] Justin Chiu Demi Guo Alexander M. Rush Yuntian Deng, Yoon Kim. Latent alignment and variational attention. 2018.