Language Modelling (LM) - Lab. 4

Luca Cazzola (248716)

University of Trento

luca.cazzola-1@studenti.unitn.it

1. Introduction

The goal is to achieve **Language Modeling** (LM) using neural sequence models as backbone. Starting from a vanilla RNN trained with SGD we're going to be experimenting with other type of recurrent cells, such as LSTM and GRU, as well as with optimizers (SGD, AdamW), and hyper-parameters. Also some regularization techniques taken from [1] and shortly discussed in (2.1, 2.2, 2.3) will be implemented on the architecture. Code available at [2]

2. Implementation details

The baseline model is structured as follows :

1. Embedding layer : projects word ids into a vector space
2. Recurrent cell : transforms the embedded input sequence into an hidden representation
3. Linear layer : projects the hidden layer to our output space (vocabulary)

All the above components have been built using PyTorch modules, to the exception of multi-layer recurrent cells, which are manually stacked in order to apply a custom dropout layer (2.2) in between stacked layers. Dropout layers are also inserted to both the embedding and final hidden state outputs.

2.1. Weight Tying

Given the fact that such models use the vocabulary length as both input and output sizes, an effective [3] and theoretically driven [4] approach is to share the weights between the embedding layer and the output linear layer. Such an approach is both easy to implement and works as regularizer by reducing the parameter count (as the vocabulary length is generally large).

```
if size(embedding) == size(hidden) :
    output.bias.zero_()
    output.weights = embedding.weights
```

2.2. Variational Dropout

In standard dropout the mask is randomly set according to the given probability at each forward pass to the layer. On sequences it means each input X, hidden state H, as well as the last hidden representations Y are assigned to different masks. An alternative approach [5] is to generate a single **shared mask between elements at the same stage**, such that $\forall.x_i \in X$ share the same mask (same for $\forall.h_i \in H$ and $\forall.y_i \in Y$).

$$dropMask(X) \neq dropMask(H) \neq dropMask(Y)$$

A dropout mask can be generated from a Bernoulli distribution with $P(1) = (1 - dropoutProb)$ as big as the hidden size and copied over the sequence length dimension. In the original paper they actually generate a single mask across the entire batch, but I decided to generate one mask per batch element to promote diversity in dropped out elements [1].

2.3. Non-monotonically Triggered AvSGD

SGD with no momentum has found a lot of success in neural language models [1], but during training it will eventually reach a stationary point. It's proposed to use Average-SGD (ASGD [6]), which computes the gradient step based on the running average of weights from time step T (hyper-parameter) to the current one t. Instead of manually setting T training begins with SGD and switches to ASGD with $T = 0$ when a "non-monotonic flag" is triggered :

$$t > L \wedge ppl > \min\left(ppls[:-L]\right)$$

If at least L iterations have passed and the current perplexity (ppl) hasn't been improving over iterations 0 to $t - L$ then it triggers. From this step onward at inference time the weights will be substituted to their running average [7].

3. Results

The dataset of reference is **PennTreeBank**, the evaluation metric of choice is model's **perplexity**. All tests train for 100 epochs, with an early stop patience of 3 epochs.

3.1. Test A

Baseline model consists in single layer RNN with equal embedding & hidden size of 300. Dropout is 0.1 for embedding & output layers (zero for hidden layers as there's no RNN stacking). Data (Table 1) shows as expected immediate improvements changing the vanilla RNN cell with LSTM or GRU. Adding dropout also helps making training much smoother reaching better performances. LSTM and GRU cells follow the same underlyig principle of being more robust to forgetting: GRU cells are lighter, which makes them less prone to overfit, but LSTM generally produces better results, which is also expressed in (Table 1). Switching the optimizer to AdamW further pushes down a bit the perplexity.

3.2. Test B

Baseline model consists in 2 layer LSTM with equal embedding & hidden size of 300. The choice of stacking recurrent layers is in order to test (2.2) also at hidden level. Dropout is 0.1 for embedding & output layers, 0.25 for hidden layers (inspired by [1]). Results reported in (Table 2) shows that models respond well to the incremental addition of more and more regularization techniques, leading to a decreasing trend in preplexity. On LSTM model + all regularizers + (2.3) we're able to reach < 100 perplexity. It's interesting to see especially how weight-tying is able to both increase performances while reducing the number of parameters by a 0.6 factor.

Model	Size	Learning Rates			
(SGD optimizer)		**5.0**	**3.0**	**1.0**	**0.1**
RNN (baseline)	6.2M	unstable*	224.34	**159.70**	164.96
LSTM	6.7M	141.52	**140.73**	145.53	161.08
GRU	6.6M	161.27	150.53	**143.36**	148.85
LSTM + dropout	6.7M	130.73	125.68	**122.67**	158.32
GRU + dropout	6.6M	162.09	141.38	**129.94**	141.94
(AdamW optimizer)		**5e-3**	**1e-3**	**5e-4**	**1e-4**
LSTM + dropout	6.7M	131.27	120.98	120.12	**<u>118.66</u>**
GRU + dropout	6.6M	155.37	134.82	130.01	**121.83**

Table 1: *Task (**A**) model perplexities*

Bold values are the highest for that specific model configuration. The underlined value is the highest in the entire table.

** Indicates unstable convergence, the learning rate was set too high*

Model	Size	Learning Rates			
(SGD optimizer)		**5.0**	**2.5**	**1.0**	**0.5**
LSTM (baseline)	7.5M	**107.45**	108.86	110.88	112.08
GRU (baseline)	7.1M	149.42	129.85	120.01	**115.79**
LSTM (tied)	4.5M	**102.97**	104.22	105.75	113.63
GRU (tied)	4.1M	122.52	109.94	112.03	**109.43**
LSTM (tied) + var.drop.	4.5M	**101.75**	103.30	106.94	111.77
GRU (tied) + var.drop.	4.1M	117.06	110.79	110.08	**109.94**
LSTM (tied) + var.drop. + nmt-ASGD	4.5M	**<u>96.99</u>**	99.39	102.65	110.74
GRU (tied) + var.drop. + nmt-ASGD	4.1M	107.85	106.18	**104.84**	105.57

Table 2: *Task (**B**) model perplexities*

Bold values are the highest for that specific model configuration. The underlined value is the highest in the entire table.

4. References

[1] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," 2017. [Online]. Available: https://arxiv.org/abs/1708.02182

[2] L. Cazzola, "Nlu exam project," 2024. [Online]. Available: https://github.com/LuCazzola/NLU-exam

[3] O. Press and L. Wolf, "Using the output embedding to improve language models," 2017. [Online]. Available: https://arxiv.org/abs/1608.05859

[4] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," 2017. [Online]. Available: https://arxiv.org/abs/1611.01462

[5] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," 2016. [Online]. Available: https://arxiv.org/abs/1512.05287

[6] PyTorch.org, "Asgd module," https://pytorch.org/docs/stable/generated/torch.optim.ASGD.html, accessed: 12/08/2024.

[7] Salesforce-GitHub, "Non-monotonically triggered avsgd implementation," https://github.com/salesforce/awd-lstm-lm/blob/1d466ec5875675ca6b6f0caacbb741240a29da9e/main.py#L225#L285, accessed: 12/08/2024.