1 Code explanation

The motivation behind pooling is that the activations in the pooled map s are less sensitive to the precise locations of features inside the original feature map, while also reducing its dimensions in a way that the output features still represent the original ones.

For such purposes, several methods had been developed, for instance, the average and max pooling. However, both of these mechanisms have important drawbacks. For instance, in the average pooling case, all elements in a pooling region are considered, even if many have low magnitude. When combined with linear rectification non-linearities, this has the effect of down-weighting strong activations since many zero elements are included in the average. Additionally, in the max pooling case, although it does not suffer from such drawbacks, it increases overfitting, making it more difficult to generalize to unseen data.

Therefore, this model is a modified version of the stochastic pooling mechanism described by Hossein, G. and Hossein, K. [1]. Inspired by the reduced capacity of deep neural networks to generalize to unseen data and, in contrast, to overfit, the authors of such method tried to define a mechanism able to make the pooling process a stochastic one, while also considering some of the advantages of the deterministic, widely used ones, such as average and max pooling.

In stochastic pooling, the pooled output is produced by sampling from a multinomial distribution obtained from each of the pooled regions. As a consequence, stochastic pooling forbids overfitting due to the stochastic component, while also considering the advantages of max pooling as higher probabilities are assigned to larger elements and lower ones are suppressed through lower probabilities.

Specifically, the probability of each region is calculated through the normalization of each of the features within the region:

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

Where ai is the activated value of the i-th feature inside the pooled region Rj. Then, the pooled sample sj of such region is obtained by:

$$S_j = a_l \text{ where } l \sim P_{(p_1,\ldots,p_{|Rj|})}$$

In our model, however, such probabilities are obtained from the Softmax function, specifically:

$$P_i = \frac{e^{x_i}}{\sum_{k \in R_j} e^{x_k}}$$

Where xi is the value of the i-th feature inside the pooled region Rj. At test time, however, the model replaces the original probabilistic sampling by a probabilistic form of averaging. This modification is intended to reduce noisy predictions where each region's pooled sample is obtained from a weighted sum of the pooled features:

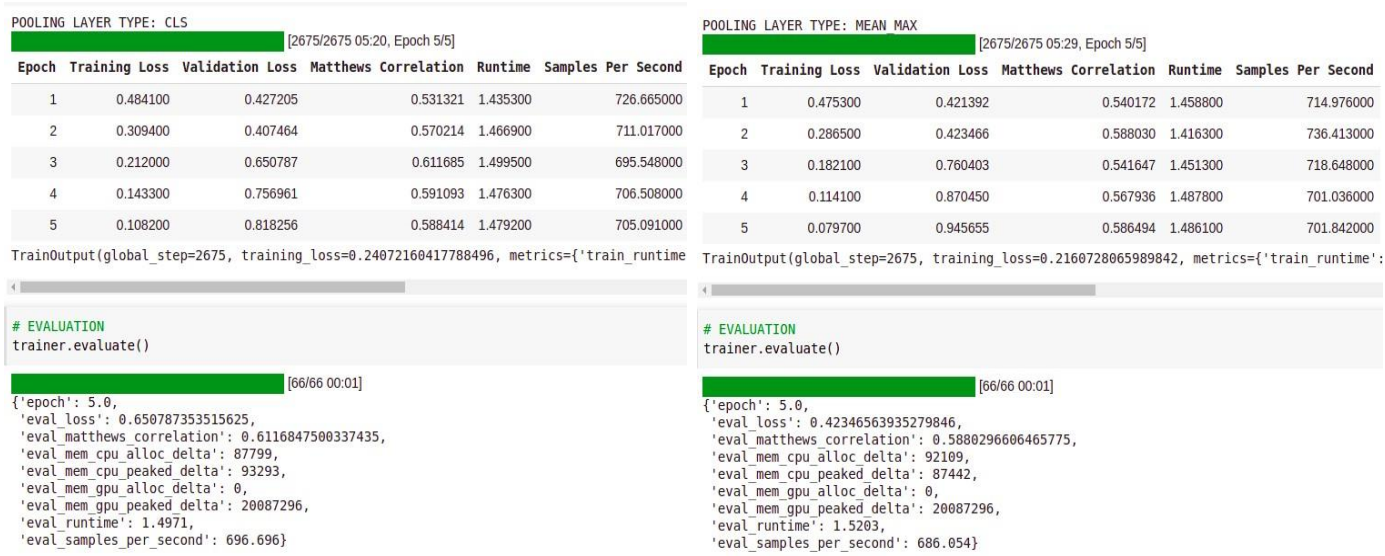
$$S_j = \mathrm{X}\, p_i a_{i,\, i \in R_j}$$

(In our model, ai is also replaced by xi). As our model so far does not consider the activation part, in contrast with the original mechanism, at this point we add a linear transformation given by:

$$W \in \mathrm{R}_{H \times H}$$

Which is followed by a tanh activation.

2 Results



POOLING LAYER TYPE: CLS

[2675/2675 05:20, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Matthews Correlation	Runtime	Samples Per Second
1	0.484100	0.427205	0.531321	1.435300	726.665000
2	0.309400	0.407464	0.570214	1.466900	711.017000
3	0.212000	0.650787	0.611685	1.499500	695.548000
4	0.143300	0.756961	0.591093	1.476300	706.508000
5	0.108200	0.818256	0.588414	1.479200	705.091000

```
TrainOutput(global_step=2675, training_loss=0.24072160417788496, metrics={'train_runtime
```

```
# EVALUATION
trainer.evaluate()
```

[66/66 00:01]

```
{'epoch': 5.0,
 'eval_loss': 0.650787353515625,
 'eval_matthews_correlation': 0.6116847500337435,
 'eval_mem_cpu_alloc_delta': 87799,
 'eval_mem_cpu_peaked_delta': 93293,
 'eval_mem_gpu_alloc_delta': 0,
 'eval_mem_gpu_peaked_delta': 20087296,
 'eval_runtime': 1.4971,
 'eval_samples_per_second': 696.696}
```

POOLING LAYER TYPE: MEAN MAX

[2675/2675 05:29, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Matthews Correlation	Runtime	Samples Per Second
1	0.475300	0.421392	0.540172	1.458800	714.976000
2	0.286500	0.423466	0.588030	1.416300	736.413000
3	0.182100	0.760403	0.541647	1.451300	718.648000
4	0.114100	0.870450	0.567936	1.487800	701.036000
5	0.079700	0.945655	0.586494	1.486100	701.842000

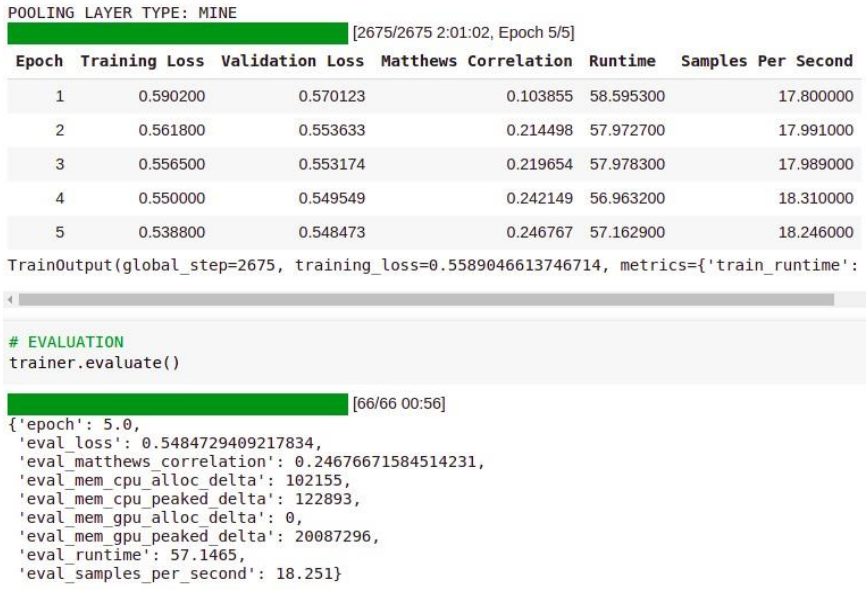
```
TrainOutput(global_step=2675, training_loss=0.2160728065989842, metrics={'train_runtime':
```

```
# EVALUATION
trainer.evaluate()
```

[66/66 00:01]

```
{'epoch': 5.0,
 'eval_loss': 0.42346563935279846,
 'eval_matthews_correlation': 0.5880296606465775,
 'eval_mem_cpu_alloc_delta': 92109,
 'eval_mem_cpu_peaked_delta': 87442,
 'eval_mem_gpu_alloc_delta': 0,
 'eval_mem_gpu_peaked_delta': 20087296,
 'eval_runtime': 1.5203,
 'eval_samples_per_second': 686.054}
```

Figure 1: 5epochs result of CLS and MEANMAX



POOLING LAYER TYPE: MINE

[2675/2675 2:01:02, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Matthews Correlation	Runtime	Samples Per Second
1	0.590200	0.570123	0.103855	58.595300	17.800000
2	0.561800	0.553633	0.214498	57.972700	17.991000
3	0.556500	0.553174	0.219654	57.978300	17.989000
4	0.550000	0.549549	0.242149	56.963200	18.310000
5	0.538800	0.548473	0.246767	57.162900	18.246000

```
TrainOutput(global_step=2675, training_loss=0.5589046613746714, metrics={'train_runtime':
```

```
# EVALUATION
trainer.evaluate()
```

[66/66 00:56]

```
{'epoch': 5.0,
 'eval_loss': 0.5484729409217834,
 'eval_matthews_correlation': 0.24676671584514231,
 'eval_mem_cpu_alloc_delta': 102155,
 'eval_mem_cpu_peaked_delta': 122893,
 'eval_mem_gpu_alloc_delta': 0,
 'eval_mem_gpu_peaked_delta': 20087296,
 'eval_runtime': 57.1465,
 'eval_samples_per_second': 18.251}
```

Figure 2: 5epochs result of STOCHASTIC Pooling

While our model performed poorly than CLS and MeanMax in terms of training loss (its convergence was much slower in comparison, although still decreased monotonically), it had a better performance in terms of validation loss (while it increased in both CLS and MeanMax, it actually decreased in our model) and demonstrated an evaluation loss that was between that of CLS and MeanMax.

Besides, while our model took about two hours for five epochs, CLS and MeanMax took around 10-20 minutes.

To sum up, our model showed much better results regarding generalization, as expected, since it was actually able to reduce validation loss while the other two mechanisms only increased it, however, this improvement was detrimental to its convergence, which slowed down. Additionally, although it did not achieve the evaluation results of the MeanMax pooling, it did surpass that of the CLS.

3 References

[1] Hossein Gholamalinezhad and Hossein Khosravi. Pooling Methods in Deep Neural Networks, a Review. arXiv.org, 2020.

[2] Matthew D. Zeiler and Rob Fergus. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. arXiv.org, 2013.