Additional assignment: BERT finetuning.

Practical part

1. **Setup Environment.** Please, use exactly this environment and don't use any additional libraries! Otherwise, your results will not be reproducible on our server!

```
conda create -n a-bert anaconda=2019.10 pytorch=1.3  python=3.7
pip install transformers==2.1.1 tensorboard==1.9
source activate a-bert
```

2. Load *bert-base-uncased* model and corresponding tokenizer. Please, don't use other models, for competition all implementations of finetuning should use identical pretrained weights! Other models are not available on reproduction server.
    a. Check that tokenizer correctly preprocesses text for the selected model (remember, the selected model is trained on lowercased texts!)
3. Write *vectorize_ex* function that receives text and label for one example and required sequence length. It should return input representations for BERT (sequence of subword ids, segment ids, input mask and the label id).
    a. Which special tokens were always attached to the input text during BERT pretraining? It may be a good idea to attach them during finetuning also, so BERT will see what it used to see.
    b. Maximum length of an input text is limited by the number of positional embeddings in BERT. How much is it? If the text is longer, the simplest option is to truncate it (in Research part you can try other options).
    c. You should pad each example to the required sequence length, because all examples in the batch should be the same length!
    d. Build input mask to mask attention to the paddings - we don't want the number of paddings to effect predictions!
4. Write main training loop. Plot learning curves: at least plot loss on train and dev (on one plot) and accuracy on train and dev (on another). For plotting Matplotlib or TensorflowX libraries will be installed on the server.
    a. Use Adam optimizer. Select learning rate between 1e-6 and 1e-3.
    b. Try using learning rate warmup (usually, 5-10% of the whole training time is enough) and learning rate decay. Carefully select the number of epochs (if your learning rate decays to small values too early, you will underfit).
    c. What is the size of the memory in your GPU? What is the largest size you can use for BERT finetuning? Try using larger batches (32,128) via gradient accumulation trick.

Research part.

1. Truncating long texts can result in loss of important information at the end. Try taking k words at the start or k words at the end for both training and inference and plot BERT accuracy w.r.t k (use logscale).
2. Further training of MLM on texts from a target task dataset improved classification accuracy. Train BERT on texts from the filimdb dataset with MLM objective (implement *pretrain(texts)* function which is called by evaluate.py) and only then finetune classifier.
3. What is the difference between Adam and AdamW (Adam with weight decay) optimizers? Build plots for both of them and compare.
4. Try using gradient clipping. Does it help?
5. Play with regularization. Select L2-regularization/weight decay strengths, dropout and attention dropout probabilities.