Laboratory 14: TensorFlow DNA sequence regression

In this laboratory exercise, you will use TensorFlow to train an efficient machine learning model to predict protein stability. The dataset is a mixture of naturally occurring miniproteins (43, 46, or 50 amino acid residues) and artificially induced point mutations that have been characterized using a high–throughput assay (Rocklin et al. 2017). The dataset has been partitioned into 11,905 (85.33%) train, 1024 (7.33%) test, and 1024 (7.33%) validate samples.

You will convert the DNA sequences to integer tokens with a 10–bit unigram language model (SentencePiece; Kudo 2018), convert each token into a vector of floating–point numbers with two small embeddings using the quotient–remainder trick (Shi et al. 2020), and then infer protein stability with a small–scale Compact Convolutional Transfromer (CCT; Hassani et al. 2021) modified to use the more computationally efficient AveragePooling attention (Yu et al. 2021) in place of conventional attention.

Tasks

(1) Create a TensorFlow environment to build and train the CCT model.

- (a) Make a working project directory by typing `mkdir stability && cd stability` in the terminal.
- (b) Start a Docker build file by typing `echo 'FROM tensorflow/tensorflow:2.12.0' > Dockerfile` in the terminal.
- (c) Add a RUN statement to the build file by typing `echo 'RUN python3 -m pip install --upgrade pip && python3 -m pip install --upgrade setuptools && python3 -m pip install tensorflow-addons && python3 -m pip install tensorflow-text' >> Dockerfile` in the terminal.
- (d) Build the Docker image by typing `docker build -t 'tensorflow:2.12.0' .` in the terminal. This process may take several minutes to complete depending on your network speed.
- (e) Download the segmented data by typing `PARTITIONS=('test' 'train' 'validation'); for PARTITION in "${PARTITIONS[@]}"; do wget 'https://github.com/dpl10/phytoinformatics2023/raw/main/stability-'$PARTITION'.tfr'; done` in the terminal. Answer question (1).
- (f) Download the model script by typing `wget https://raw.github.com/dpl10/phytoinformatics2023/main/cct.py` in the terminal.
- (g) Download the training and testing scripts by typing `TYPES=('finetune' 'predict' 'test' 'train'); for TYPE in "${TYPES[@]}"; do wget 'https://github.com/dpl10/phytoinformatics2023/raw/main/'$TYPE'Sequences.py'; done` in the terminal.
- (h) Make the scripts executable by typing `chmod +x *.py` in the terminal.
- (i) Download the SentencePeice 10–bit unigram model by typing `wget https://raw.github.com/dpl10/phytoinformatics2023/main/unigram-10.pb` in the terminal.
- (j) Download a pretrained CCT model by typing `wget https://github.com/dpl10/phytoinformatics2023/raw/main/cct-pretrain.h5` in the terminal.

(2) Start the Docker instance by typing `docker run -u $(id -u):$(id -g) --rm -it -v "$PWD:/tmp" -w /tmp tensorflow:2.12.0` in the terminal.'

(3) Construct a randomly initialized CCT model by typing `./cct.py -a 1 -b -d 32 -f selu -l 2 -m 4 -o cct.h5 -p 0,1,2,3 -r 123456789 -u 48 -v 10` in the terminal. Answer question (2).

(4) View the untrained model in netron using the graphical interface or a web browser (https://netron.app). Answer question (3).

(5) Make a directory to save the training result by typing `mkdir denovo` in the terminal.

(6) Train the CCT model by typing `time ./trainSequences.py -a 1 -b 64 -c -e 64 -f mse+clr+a -l 0.05 -m cct.h5 -o denovo -r 123456789 -s unigram-10.pb -t stability-train.tfr -u 48 -v stability-validation.tfr` in the terminal. Depending upon your computer, this step will take between 2 minutes and 2 hours. Answer question (4).

(7) Evaluate the best–trained model by typing `./testSequences.py -a 1 -b 64 -c -m $(ls -ltr denovo/*/best-model.h5 | awk '{print $9}' | tail -1) -s unigram-10.pb -t stability-test.tfr -u 48` in the terminal. Answer question (5).

(8) Create a directory to save finetuned models to by typing `mkdir finetune` in the terminal.

(9) Fine–tune the pretrained CCT model by typing `time ./finetuneSequences.py -a 1 -b 64 -c -e 64 -f mse+clr+a -l 0.001 -m cct-pretrain.h5 -o finetune -r 123456789 -s unigram-10.pb -t stability-train.tfr -u 48 -v stability-validation.tfr` in the terminal. Answer question (6).

(10) Evaluate the best model from the fine–tuning by typing `./testSequences.py -a 1 -b 64 -c -m $(ls -ltr finetune/*/best-model.h5 | awk '{print $9}' | tail -1) -s unigram-10.pb -t stability-test.tfr -u 48` in the terminal. Answer question (7).

(11) Output a table of [-1,+1] rescaled known values and predicted values by typing `./predictSequences.py -a 1 -c -m $(ls -ltr denovo/*/best-model.h5 | awk '{print $9}' | tail -1) -s unigram-10.pb -t stability-test.tfr -u 48 > denovo-predictions.tsv` in the terminal.

(12) Output a similar table for the finetuned model by typing `./predictSequences.py -a 1 -c -m $(ls -ltr finetune/*/best-model.h5 | awk '{print $9}' | tail -1) -s unigram-10.pb -t stability-test.tfr -u 48 > finetune-predictions.tsv` in the terminal.

(13) Close the Docker image by typing `exit` in the terminal.

(14) To evaluate the *de novo* prediction calculations, type `tail -n +2 denovo-predictions.tsv | datamash ppearson 1:2` in the terminal.

(15) To evaluate the finetune prediction calculations, type `tail -n +2 finetune-predictions.tsv | datamash ppearson 1:2` in the terminal. Answer question (8).

Questions (https://forms.gle/QwQ4syDwEFZFxQUR6)

(1) For task (1)(e), explain what each part of the command does.

(2) For task (3), explain what each argument of the command does.

(3) For task (4):

(a) How many parameters does this CCT model have?
(b) How many parameters are trainable?
(c) What is the size of each transformer unit?
(d) How many sequence tokens per sample can be processed by the model?
(e) What is the size of the output layer?

(4) For task (6):

(a) Explain what each part of the command does.
(b) How long (real time == wall clock time) did it take to train the model?
(c) What was the model's MSE on the validation dataset?
(d) Did the model overfit during training?

(5) For task (7):

(a) What was the model's MSE on the test dataset?
(b) Is this very different from the corresponding values on the validation dataset?
(c) Would the model benefit from additional epochs of training?

(6) For task (9):

(a) How many parameters does this CCT model have?
(b) How many parameters are trainable?
(c) Compare and contrast the number of trainable parameters and their location in this model and in 'cct.h5'.
(d) How long (real time == wall clock time) did it take to fine–tune the model?
(e) How does this compare to the *de novo* training time?
(f) What was the model's MSE on the validation dataset?
(g) Did the model overfit during training?

(7) For task (10):

(a) What was the model's MSE on the test dataset?
(b) Which training worked better? Explain.

(8) For task (15):

(a) Which model is better correlated with the test dataset?
(b) Is there any pattern to the errors made by the models?
(c) How could you (perhaps) overcome this bias?

Literature cited

Hassani, A., S. Walton, N. Shah, A. Abuduweili, J. Li & H. Shi. 2021. Escaping the big data paradigm with compact transformers. arXiv 2104.05704.

Kudo, T. 2018. Subword regularization: improving neural network translation models with multiple subword candidates. arXiv 1804.10959.

Rocklin, G., T. Chidyausiku, I. Goreshnik, A. Ford, S. Houliston, A. Lemak, L. Carter, R. Ravichandran, V. K. Mulligan, A. Chevalier, C. H. Arrowsmith & D. Baker. 2017. Global analysis of protein folding using massively parallel design, synthesis, and testing. Science 357: 168–175.

Shi, H.-J., D. Mudigere, M. Naumov & J. Yang. 2020. Compositional embeddings using complementary partitions for memory–efficient recommendation systems. *In:* Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 165–175. Association for Computing Machinery.

Yu, M., M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng & S. Yan. 2021. MetaFormer is actually what you need for vision. arXiv 2111.11418.

Due at the start of class May 9.