**Minutes**

3/11/22

* Discussed batching issues.
  + Dataloader takes input batches where they are all the same size (e.g. 28x28 like MNIST uses).
    - Same size advantage – efficient indexing.
    - Would have to crop sequences so they were a uniform size, and treat each crop individually.
  + Therefore, protein sequences of different sizes cannot be batched together.
  + Can batch with size 1, so the protein sequence is a single input each time.
    - Some limitations: training with batches can improve learnability, large batch size will keep the GPU busy (meaning, all 1000 cores may spin up but not all of them are utilised). Also if the GPU is not busy, then training time will take longer because data is being passed between CPU and GPU and GPU is doing nothing.
  + Also looked at padding the sequences, but this can cause unnecessary computations to be made. As the longest sequence may be much longer than others.
    - Padding would mean kernels applied to a padded image could return the same image as only the padding was removed during convolution.
  + Looked at one long sequence of all the protein sequences combined. Sequences have padding in between so the sequences were not combined together when the kernel was applied to the sequence.
    - This could work as the kernel only moves horizontally through the sequence – the kernel operations are independent. Therefore, the computations would be the same over certain tensor sections as these tensor sections would be the separate tensors if they were computed one at a time.
* Discussed kernels:
  + Looked at 3x3 and 5x5 kernel examples.
  + Multiple kernels used for each sequence. Different kernels extract different features.
  + For example, in an image of a cat one of the kernels may extract a shape feature for that of a cat, so the model can label this as a cat.
  + In PyTorch – Conv1D, Conv2D, Conv3D for different input sizes.
  + With a padded sequence – our output could be the same size as the input sequence after kernel was applied.
  + One kernel over a section of the sequence – will return values dependent on contents of kernel matrix. We could threshold the weights produced by the kernel convolution to 10, for example, the kernel applied produces [20,0,…,4,0,…,20,6,0...,20] for amino acids. With the threshold we just have a vector with the 20 values as non-zero - (vector is same length as amino acid vector).
  + Different kernels applied – each one will have their own weights. This means our output will have a larger depth than our input.
* Looked at dataset class on google colab.
  + To change – have the encoded ordered/disordered sequence as the label.
  + Can replace the list of disordered regions with this label.
* Discussed PyTorch:
  + That tensors and numpy are very similar as they can perform vectorised calculations efficiently and often use the same method names (e.g. like\_zeros).
  + Array slicing to map disordered 1 labels.
  + PyTorch internally labels classes (0,1,2,3) instead of (1,0,0,0) for example.
* Discussed PSIPRED and DISOPREDs window approach.
  + They treated each 15 wide windows as separate inputs and didn’t consider the entire sequence.
  + A limitation to the approach was only utilising standard loops and CPUs for training.
  + They didn’t batch, but this was possible as all of the inputs were the same size.
* Discussed Fully Convolutional Networks (FCN).
  + Paper to read this week.
  + Fully connected layers (in the CNN) expect input to be of the same size, FCN does not. Good for differing sequence sizes.
  + Only performs the convolution – can be faster to train.

Goals for this week:

* Complete dataset class.
* Use PyTorch convolution kernels with sequences in the dataset.
* Read FCN paper.