**NER Using Continual Learning**

**Data Preprocessing**

First, we split the data into train (80), test (10) and validation (10) sets and apply lowercasing for tags/text.

Now, we convert the data to a form which can be used an input for the neural network that we’ll develop. Details of preprocessing can also be found in the markdown cells of the notebook. All the preprocessing code is present in the function **preprocess.**

We currently have the tags in index form, for eg: l:r:tag, where l, r are indices in text. Hence, we need to process this to a standard format in order to feed this to our neural network model.

As a first step in preprocessing, we convert the tags, for each entry in the data, to a list of tuples with 2nd element as tag and 1st element as the corresponding word(s) in text. eg: [("pneumonia","chronic\_disease"),("skin\_allergy","allergy\_name"),....]

Note: For a few cases, we get tags with overlapping indices. Here, we assume that the tag which we come across first is the correct one.

In the next step, we tokenize the text into tokens/subwords using the tokenizer extracted at the start from biobert Model. Tokens are integers, we convert them to string tokens using **tokenizer.convert\_ids\_to\_tokens** function. Now we iterate over tokens, for each entry in the data, and create words out of them by merging the tokens of a single word. This is done based on the logic that sequential tokens of a continuous word start with "##" when BERT tokenizer (WordPiece) is used.

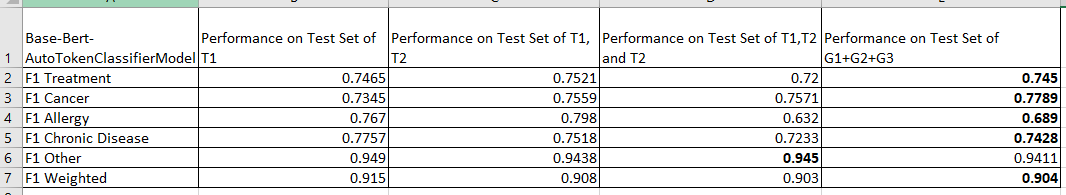
Now we have the list of words and tags for each entry. So, we iterate over these and a create a list of word labels which is dictionary of labels corresponding to each word id (index of word in the text). If word has no label in the tags list, then we assign it as "other". Now for each token, we can access the word id, i.e. we know what index word each token belongs to, hence we can create a list of labels for the tokens of text for each entry. Hence, the tokens (input\_ids and attention\_mask) and the label list of tokens forms our main input and ground truth respectively.

For task T2 and T3, we also need to augment the dataset using 100 entries from previous datasets. Here we try to create the train subset from previous dataset by keeping the distribution of tags same. We do this by sorting the datasets in descending order of count of each tag. So, now we have 4 sorted datasets, one for each tag. Then we choose top x\_i entries from each such that sum is 100 and x\_i is in line with distribution of tag I in that complete dataset. Hence, in some sense, we’ve used stratified sampling with a minor tweak of getting largest count of corresponding tags as well.

**Model Training**

We’ve used several models for our training purposes.

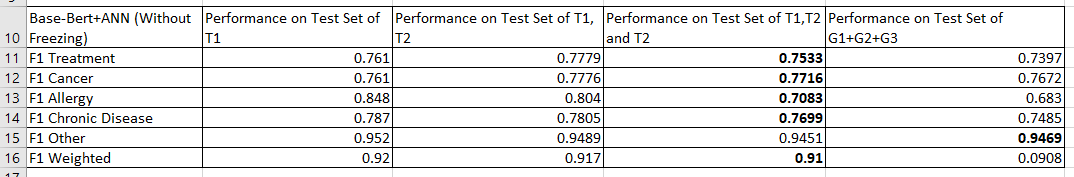
1. We used the **AutoModelForTokenClassifcation** from hugging face with **bert-base-uncased** as the base transformer. This model is simply Bert model plus a dense layer of size (last hidden state) to 5 (number of tags in our dataset).



This did not give very good results. Further the combined model outperformed the T3 model.

In the next models, we’ve ditched the AutoModelForTokenClassification models, and instead created custom dense network on top of the base Transformer model.

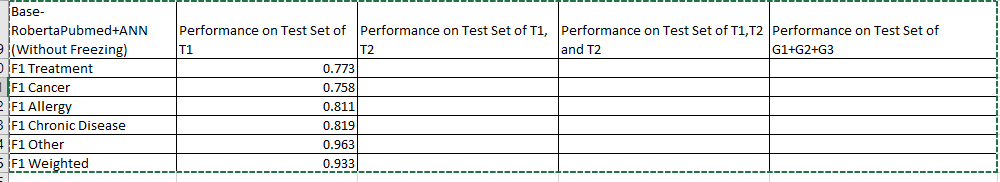
1. Here, we’ve again used a **bert-base-uncased** as the base transformer followed by an DNN of 3 layers (last hidden state size->512), (512->128), (128->5). We saw an improvement in the results using this model.



**c)** We then thought of using transformer models which have been already trained on biomedical datasets to ensure accurate and quick learning on our current dataset. This is the first Roberta based model we trained, hence we needed to change the preprocessing code as well to suit the Roberta tokenizer (BPE).

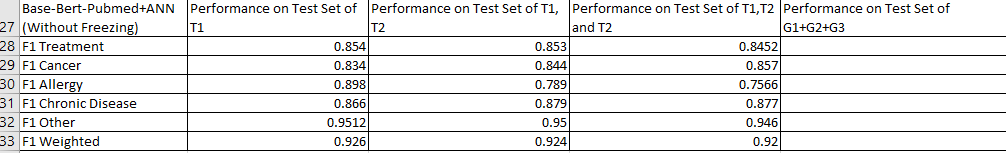
Base Transformer: <https://huggingface.co/allenai/biomed_roberta_base>

We used a similar DNN post the transformer as used in b)



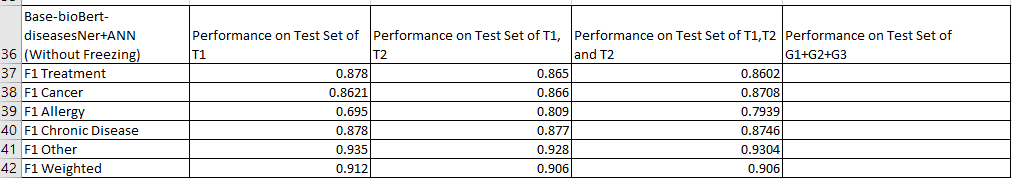
**d)** Base Transformer: <https://huggingface.co/microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext>

We used a similar DNN post the transformer as used in b)



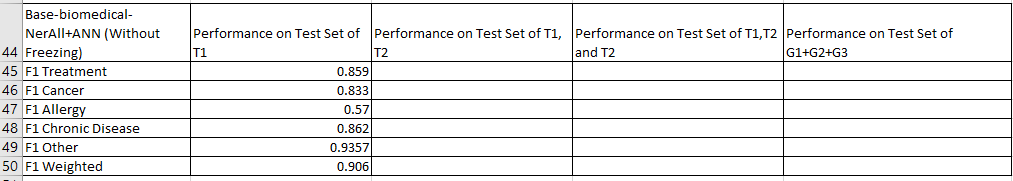
**e)** Base Transformer: <https://huggingface.co/alvaroalon2/biobert_diseases_ner>

We used a similar DNN post the transformer as used in b)



**f)** Base Transformer: <https://huggingface.co/d4data/biomedical-ner-all>

We used a similar DNN post the transformer as used in b)

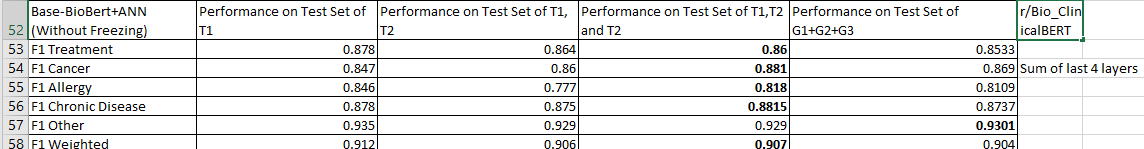


**Note: For models, from c to f, we’ve not executed complete training till T3 and combined models. We’ve only went ahead with training in models which gave better performances in T1 itself. We know that models performing subpar in T1 can potentially outperform other models in T3. However due to restricted GPU timings in colab, we had to follow this strategy.**

**Best Models:**

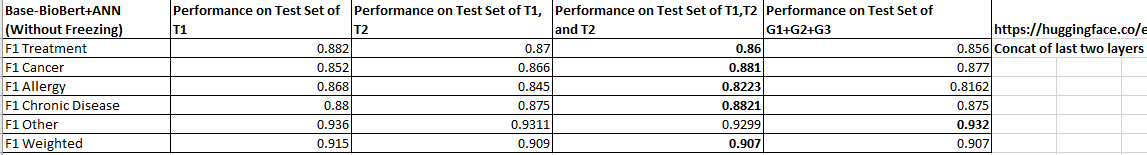
g) Base Transformer: <https://huggingface.co/emilyalsentzer/Bio_ClinicalBERT>

Here we’ve introduced a different DNN post the base transformer. Here we take the sum of last 4 hidden states of the transformer as our input to the Dense Network. Further in the dense network, we’ve added another layer and also introduced batch normalization and dropout layers. We’ve noticed a significant improvement in results.



h) Base Transformer: <https://huggingface.co/emilyalsentzer/Bio_ClinicalBERT>

Here, we’ve taken concatenation of last 2 hidden states as the input for our dense network. The rest of the neural network is same. This gave slightly better results and is our current best model.



We notice that our T3 model outperforms combined models for the g and h.

NOTE: All the training code for T1-T3 is present in the notebook **NER.ipynb.** All the result are present in **Results.xlsx.**

**Training Params:**

We’ve used similar training params for all the models.

Optimizer: AdamW

Learning Rate: 3e-4

Patience for Early Stopping: 2

Validation metric: For models a-c, we’ve used f1-weighted as the validation metric for early stopping. For rest of the models, we used summation of f1 scores of chronic disease, treatment, cancer and allergies. (Excluding others).

Batch Size: 8

Num Epochs: 20

Loss Function: Cross Entropy

**Training for T4**

All the code for training the current best model on task T4 is present in the notebook **TrainingT4.ipynb.** It’s just required to put correct inputs for dataset path, current model path and path to save T4 model in the notebook.

Please note that I was facing a few issues in uploading my model to huggingface hub since I had used a pytorch model (base Transformer+DNN) along with pytorch training instead of Hugging Face wrapper object **Trainer.** Hence, I’ve uploaded the state\_dicts (weights) of all the 4 models on the google drive link: [**https://drive.google.com/drive/folders/1Q5a9RX-qLwqk8Lxta7XWzaqqYn1OaSz3?usp=sharing**](https://drive.google.com/drive/folders/1Q5a9RX-qLwqk8Lxta7XWzaqqYn1OaSz3?usp=sharing)

|  |  |
| --- | --- |
| **Task** | **Model Name** |
| T1 | nerModelG1.zip |
| T2 | nerModelG2.zip |
| T3 | nerModelG3.zip |
| Combined Task | nerModelCombined.zip |

**Other Approaches**

Due to limited time, there are lots of thing that have been left to try.

1. Due to limited GPU Memory, we could only train with a batch size of 8. Higher batch size can be used to improve the results.
2. Hyperparameter Tuning: We can experiment and tune the optimizer, learning rate and patience for ES.
3. We can experiment with other models as base transfomrers like biogpt or even larger models. Here, due to memory constraints, we could only train with base sized models.
4. We can play around with the DNN that we use post transformer. Use a denser network, concatenate more hidden states, add skip connections etc.
5. Currently we fine tune all the layers of the base transformer in all tasks. We can play around with different freezing patterns. Freeze x layers after each task to avoid catastrophic forgetting.
6. Use different strategies to sample subset form previous datasets. Currently we’ve used stratified sampling. We can try taking a higher proportion of entries with tags for which performance was poorer in the previous task. Or simply take all entries with tags having least count in current dataset
7. We can play around with the validation metric. Currently its sum of f1 scores of 4 main tags (excluding others). We can use a weighted f1 of just these 4 tags or other metrics.
8. We can use a different continual learning strategy. In the preprocessing we can also add a tag to identify which dataset it belongs to. Finally when we’re back propagating, we can use a weighted loss to give a bit preference to the loss of previous dataset to avoid catastrophic forgetting. For eg:

Loss\_T2 = 1.2\*LossG1 + LossG2

1.2 is a tunable hyperparameter here.

9) We can use ensemble models of multiple (usually 3-5) models with decent performance. We can simply use bagging ensembling and take average or weighted average (tunable weights) of the softmax probabilities of our constituent models. Then report argmax prediction using these probabilities. This usually gives very good results.

10) Add some heuristics for common diseases/treatments/cancers/allergies.