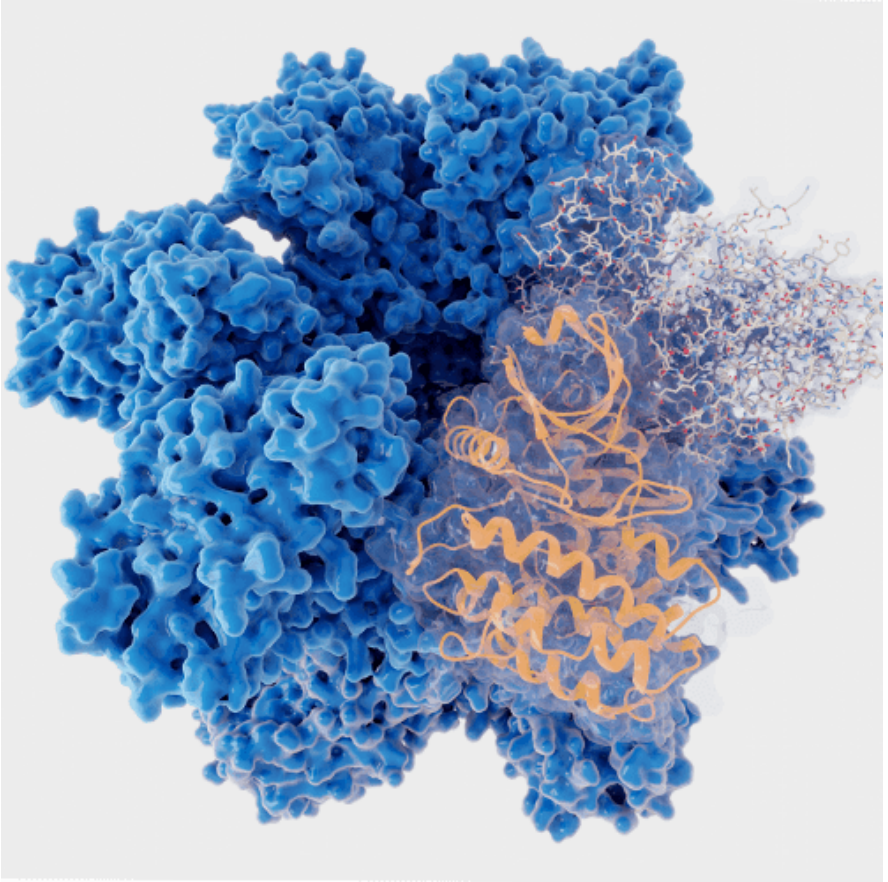
Sequence Modeling Homework #3

Build a Character-based multitask RNN/Transformer for bioactivity prediction

Science

In this homework, you will build a multitask recurrent neural network to predict the biological activity of small molecules for the proteins from the Janus Kinase family -- JAK1, JAK2, JAK3, and TYK2.

Janus kinases (Jaks) are critical signaling elements for a large subset of cytokines. As a consequence, they play pivotal roles in the pathophysiology of many diseases including neoplastic and autoimmune diseases.



Data

Biological activity is expressed as a binary value -- a molecule can be "active" or "not active" for a protein of interest.

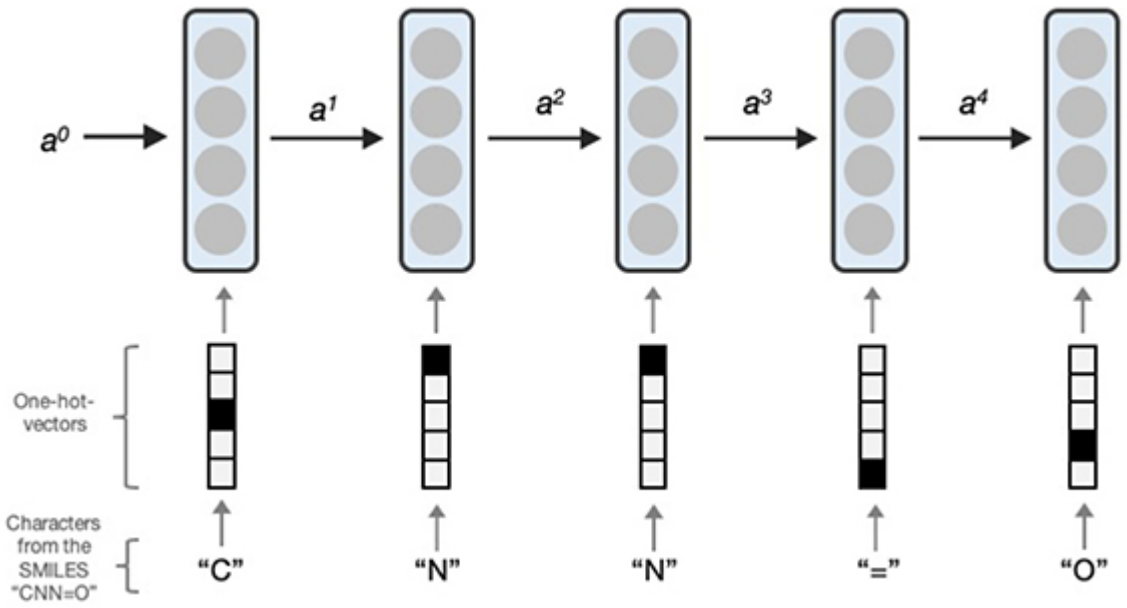
The training data also contains missing endpoints since not all molecules were tested against all four proteins.

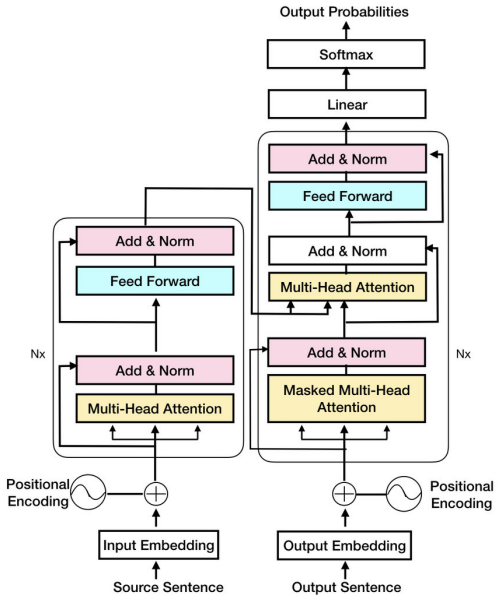
However, test data does not contain missing endpoints.

Data

StdSMILES	P23458	O60674	P52333	P29597
C[C@H](NC(=O)c1c[nH]c2ncc(-c3ccc(C(=O)NCc4ccccc4)s3)nc12)C(C)(C)C	999	999	1	999
CC(=CNC(=O)Nc1cc(C)ccc1O)C(C)C	0	999	999	999
Oc1ccc2c(c1)OC(c1ccc(OCCN3CCCCCC3)cc1)C1=C2CCOc2cc(O)ccc21	0	999	0	999
CC1(C)CC1C(=O)N/C(=C/CCCCSCC(N)C(=O)O)C(=O)O	0	999	999	999
Cc1ccc(NC(=O)C2=NNC(=O)CC2)cc1F	999	0	999	999
COc1ccc(CCNC(=O)c2nonc2N)cc1	999	0	999	999
O=C(OC1CN2CCC1CC2)C(CO)(c1ccccc1)c1ccccc1	999	0	999	999
Cc1cc(OCC(=O)c2ccc(F)cc2)ccc1N1C(=O)CCC1=O	999	0	999	999
Cn1cc(C2CN(c3ncnc4[nH]ccc34)CC2CO)cn1	0	999	0	999
O=C(Cc1noc2ccccc12)Nc1cccc(Cn2ccnc2)c1	0	999	0	999
CN(C)S(=O)(=O)c1ccc(Nc2nn(C3(CC#N)CCN(C(=O)Cc4ccc(C(F)(F)F)cc4)CC3)c3cc[nH]c(=O)c23)cc1	1	1	999	999
N#CCC1(n2cc(-c3ncnc4[nH]ccc34)cn2)CN(C2CCN(C(=O)Nc3cccc(Cl)c3)CC2)C1	1	1	999	999
COCCCN=C(S)Nc1ccc(C)cc1	999	0	999	999
CCn1cc(C2=NOC(C(=O)Nc3ccccc3C(=O)OC)C2)c(C)n1	999	0	999	999
Cc1cc(C(F)(F)F)nc(SCC(=O)Nc2ccccc2)n1	999	0	999	999
Cc1cccc(C(=O)c2c(N)sc3c2CCC(C)C3)c1	999	0	999	999
O=C(CS(=O)(=O)c1ccc(Cl)cc1)Nc1ccc(Cl)c(C(F)(F)F)c1	0	999	0	999
COc1cc(C(=O)N2CCc3ccccc32)ccc1OCc1c(C)noc1C	999	0	999	999
c1ccc(Nc2ncccn2)cc1	999	999	0	999
O=C(CCl)c1ccc2nc(C(=O)O)oc2c1	0	999	999	999
COC(=O)C(C)Sc1nnc2n(-c3cccc(C)c3)c(=O)c3c4c(sc3n12)CCCC4	999	0	999	999

Model





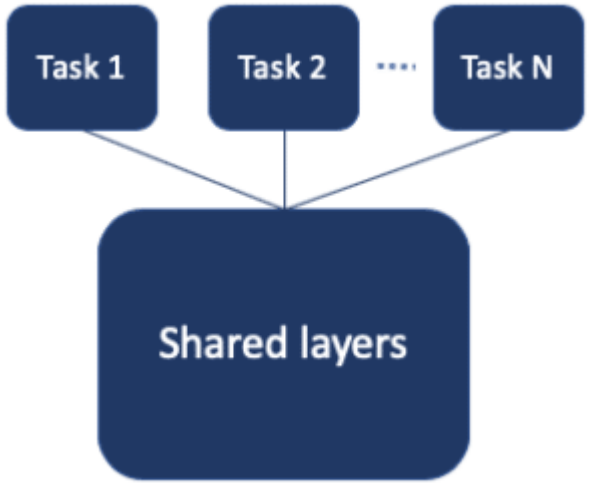
SMILES string

Characters	o	1	c	(	c	c	c	1	)	C
c	0	0	1	0	1	1	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
(	0	0	0	1	0	0	0	0	0	0
)	0	0	0	0	0	0	0	0	1	0
o	1	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	1

Hints:
SMILES strings in the training set are of variable lengths. To deal with that, you need to add a special padding symbol to your SMILES. Consider adding the padding symbols to the beginning of SMILES or flipping the SMILES backward if you decide to add padding to the end. This trick will help RNN to train better!

An alternative solution to deal with variable input lengths is to use PyTorch utilities pad_packed_sequence and pack_padded_sequence.

Multitask Loss



Your model's performance will be evaluated using mean multi-column AUC metrics for 4 predicted biological activities.

A hint for implementing multitask loss:

PyTorch's standard loss functions (such as CrossEntropyLoss) are inherited from parent classes such as _WeightedLoss or _Loss. You can use the same trick for your multitask loss implementations.

See also:
https://ruder.io/multi-task/

Bonus Points: Data Augmentation

If you want to improve your model's performance further, consider using data augmentation with the SMILES enumerator.

This .py file contains a class that will allow you to enable data augmentation in your training pipeline with a few lines of code (after you copy the .py file from the link to your working directory).

For example, if your SMILES string is stored in a variable sm, the following code will return you a different SMILES string that encodes the same molecule:

```
from SmileEnumerator import SmilesEnumerator
smiles_augmentation = SmilesEnumerator()
augmented_sm = smiles_augmentation.randomize_smiles(sm)
```