VLMMLR Packing Project

Jack Maguire – 10/16/2019

Data For Each AA At Each Position

	1	2	3	4	5	6	7	8	9	10	11	12	…
Temp	100	30	10	3	0.1	0.03	0.01	100	30	10	3	0.1	
acc rate to AA	1	0.8	…		0.1	…	0						
acc rate from AA	0.9	…											
acc rate intra AA	0.95	…											

Plan

	1	2	3	4	5	6	7	8	9	10	11	12	…
Temp	100	30	10	3	0.1	0.03	0.01	100	30	10	3	0.1	
acc rate to AA	1	0.8	…		0.1	…	0						
acc rate from AA	0.9	…											
acc rate intra AA	0.95	…											

Model input is 21 values:
- 20 values (above)
- # of columns remaining

Model output is a classification:
- 1 if the AA ends up being the final designed AA
- 0 if a different AA is chosen for this position

Motivation

- When I design ~100 amino acid proteins, I never get the same sequence twice
  - This is good for sampling diversity
  - Might be a red flag that we can do better

- If we can identify unfruitful amino acids early in the packing process we can either
  - Skip them to save runtime
  - Divert their resources to sample in better AA space

Setup

- Install tensorflow
  - pip3 install tensorflow-gpu==2.0.0-alpha
- git clone git@github.com:JackMaguire/VLMMLR_packer_contest.git

run.py

If you have a GPU, you can install tensorflow-gpu and disable GPU usage with these os.environ commands

I never import keras directly, but import it through tensorflow

```
import os
#Comment this out to use your gpu
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = ""

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
#...

import numpy
import pandas as pd

numpy.random.seed( 0 )
```

run.py: PARAMETERS

```
window_size = 5 # consider 5 loops of data (columns) at once

channels = 4 # temp + 3 rates


extra_values = 1 # number of loops remainint


num_input_values = (window_size * channels) + extra_values
```

This logic defines the number of input parameters for the network

run.py: CREATE MODEL

Make a neural network with 5 layers:

1 input

3 Dense

1 output

We tell TF to score the model with binary crossentropy, but to also measure binary accuracy

This will make more sense later

```
# Layers

input = Input(shape=(num_input_values,), name="in",
dtype="float32" )

dense1 = Dense( name="dense1", units=100, activation="relu"
)( input )

dense2 = Dense( name="dense2", units=100, activation="relu"
)( dense1 )

dense3 = Dense( name="dense3", units=100, activation="relu"
)( dense2 )

output = Dense( name="output", units=1, activation='sigmoid'
)( dense3 ) # final value is between 0 and 1


model = Model(inputs=input, outputs=output )


metrics_to_output=[ 'binary_accuracy' ]

model.compile( loss='binary_crossentropy', optimizer='adam',
metrics=metrics_to_output )

model.summary()
```

run.py: LOAD DATA

You can mostly skip this section

The only thing to know is that I lazily hardcoded the data paths

```
def read_from_file_raw( filename ):
    #ignore this


def read_from_file( filename ):
    #don’t worry about this
    #...


input,output = read_from_file( "data/training_data.100000.csv" )
test_input,test_output = read_from_file(
"data/validation_data.10000.csv" )
```

run.py: TRAIN

Interesting Callbacks:

- ModelCheckpoint
- ReduceLROnPlateau
- LearningRateScheduler
- TensorBoard (advanced)

Other Things to play with:

- batch_size
- class_weight
- epochs
- optimizer (in model.compile)

```
csv_logger = tensorflow.keras.callbacks.CSVLogger( "training_log.csv", separator=',', append=False )

# Many fun options: https://keras.io/callbacks/

callbacks=[csv_logger]
```

```
class_weight = {0: 1., 1: 20.}
```

```
model.fit( x=input, y=output, batch_size=64, epochs=1, verbose=1, callbacks=callbacks, validation_data=(test_input,test_output), shuffle=True, class_weight=class_weight )
```

```
model.save( "model.h5" )
```

run.py

Output tells you:

- training loss
- training accuracy (0.5)
- validation loss
- validation accuracy (0.5)

```
...

Epoch 4/10

1500000/1500000 [==============================] - 38s 25us/sample -
loss: 1.2698 - binary_accuracy: 0.6487 - val_loss: 0.7122 -
val_binary_accuracy: 0.6926

Epoch 5/10

1500000/1500000 [==============================] - 38s 25us/sample -
loss: 1.2469 - binary_accuracy: 0.6541 - val_loss: 0.7165 -
val_binary_accuracy: 0.6924

Epoch 6/10

1500000/1500000 [==============================] - 38s 25us/sample -
loss: 1.2343 - binary_accuracy: 0.6565 - val_loss: 0.6852 -
val_binary_accuracy: 0.6999

Epoch 7/10

 918912/1500000 [=================>............] - ETA: 13s - loss:
1.2205 - binary_accuracy: 0.6625
```

analyze.py

```
10000/10000 [==============================] - 0s 40us/sample - loss:
0.3835 - binary_accuracy: 0.8440
```

ncols+,	cutoff,	true_pos,	true_neg,	false_pos,	false_neg
1	0.5	397	8043	1441	119
1	0.4	427	7398	2086	89
1	0.3	456	6497	2987	60
1	0.2	488	5307	4177	28
1	0.1	506	3951	5533	10

```
prediction = model_output > cutoff
```

cutoff can be decreased to enforce strictness

jack.py: {1:200}

```
10000/10000 [==============================] - 0s 34us/sample - loss:
1.5462 - binary_accuracy: 0.2488
```

ncols+,	cutoff,	true_pos,	true_neg,	false_pos,	false_neg
15	0.5	504	1984	7500	12
15	0.4	511	1459	8025	5
15	0.3	516	965	8519	0
15	0.2	516	626	8858	0
15	0.1	516	459	9025	0

up-weighted penalty for false negatives

even the first round could eliminate ~10% of amino acids