Using a Neural Network to Predict Drug-Kinase Affinities

By Marcus Schubert

Introduction

- DREAM Challenge
- Convolutional Neural Network
  - Image recognition
- Kinases cause diseases

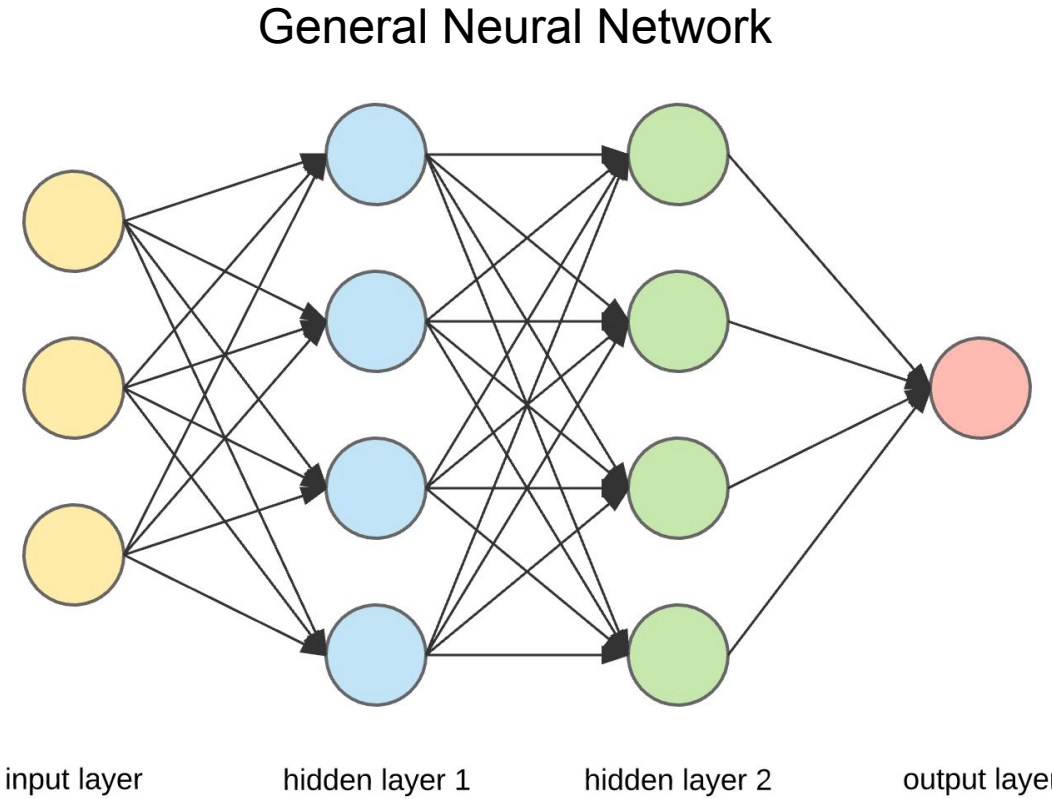
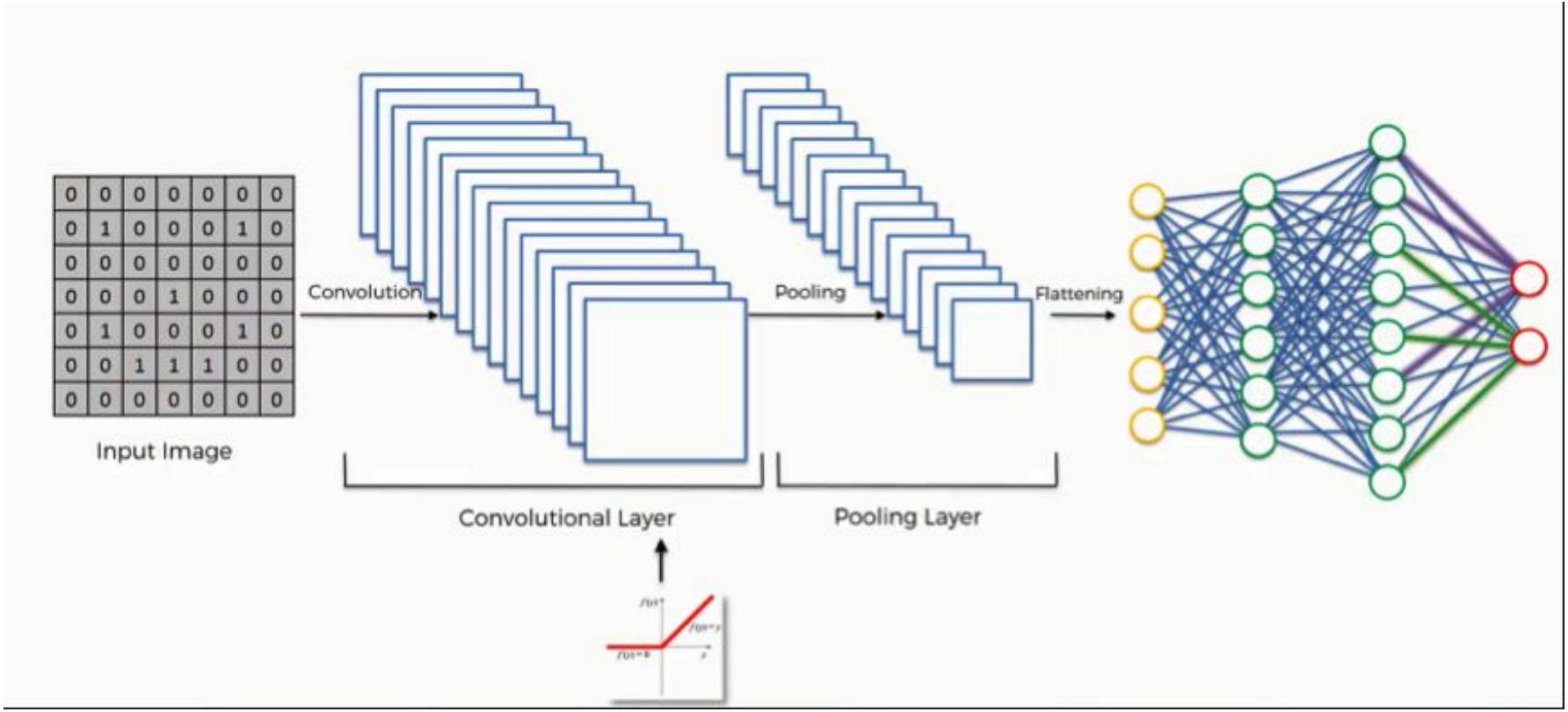


Figure 1: A visual representation of the three layers of a neural network. In the hidden layers, calculations are performed on input from a database or user, which influences an output value.

Convolutional Neural Network



Review of Literature

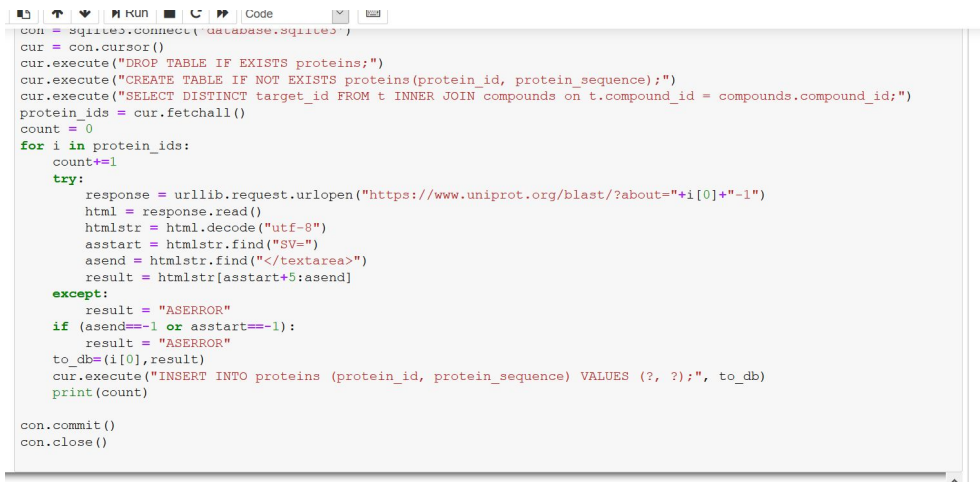
- https://www.synapse.org/#!Synapse:syn15667962/wiki/
  - DREAM Challenge
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5959832/
  - How to use keras for medical image analysis
- https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005678
  - Pilot project
- https://www.ebi.ac.uk/chembl/
  - Source for inchls
- https://www.uniprot.org/
  - Source for protein sequences
- https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/
  - Keras tutorial
- https://drugtargetcommons.fimm.fi/
  - Drug-Kinase interaction dataset

Problem Statement

Do neural networks offer practical benefits in predicting drug-kinase interactions?

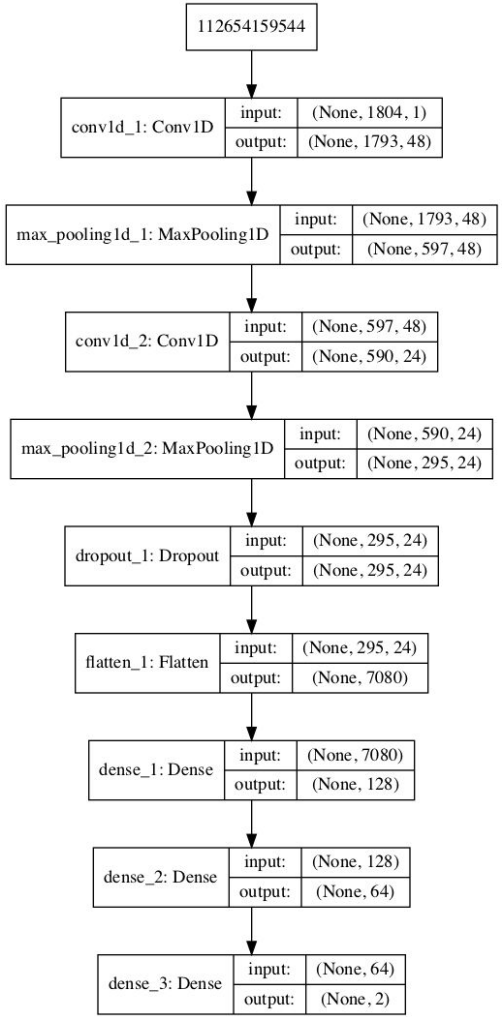
Methods

- Dataset (Drug Target Commons)
  - ~6,000,000 interactions
- Python
- SQLite
- IC50 values
  - Concentration to inhibit 50%
- ChEMBL
  - Standard InChI
- Uniprot
  - Amino acid sequence



Methods

- Change dataset size
  - Amino acid sequence length
  - InChI length
  - IC50 value
- Number of CNN layers (default = 1)
- Number of kernels & length (default = 32 kernels, 8 long)
- Dropout (default = 0.2)
- IC50 cutoff (default = 200 M)



Results

Using 60,000 Interactions

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 23s 486us/step - loss: 0.4755 - acc: 0.7760 - val_loss: 0.4295 - val
_acc: 0.7985
Epoch 2/3
47990/47990 [==============================] - 24s 491us/step - loss: 0.4133 - acc: 0.8040 - val_loss: 0.3983 - val
_acc: 0.8081
Epoch 3/3
47990/47990 [==============================] - 25s 520us/step - loss: 0.3878 - acc: 0.8154 - val_loss: 0.3829 - val
_acc: 0.8157

<keras.callbacks.History at 0x19edd279550>
```

Using 600,000 Interactions

```
batch_size = 128
epochs = 1
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 565575 samples, validate on 141394 samples
Epoch 1/1
565575/565575 [==============================] - 1087s 2ms/step - loss: 0.4468 - acc: 0.7838 - val_loss: 0.4239 - v
al_acc: 0.7938
```

Results

Using 1 Convolutional Layer

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 23s 480us/step - loss: 0.4503 - acc: 0.7872 - val_loss: 0.4060 - val
_acc: 0.8023
Epoch 2/3
47990/47990 [==============================] - 24s 492us/step - loss: 0.3898 - acc: 0.8127 - val_loss: 0.3795 - val
_acc: 0.8156
Epoch 3/3
47990/47990 [==============================] - 27s 556us/step - loss: 0.3734 - acc: 0.8215 - val_loss: 0.3744 - val
_acc: 0.8157

<keras.callbacks.History at 0x2c1a38ed400>
```

Using 2 Convolutional Layers

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 23s 476us/step - loss: 0.4622 - acc: 0.7806 - val_loss: 0.4135 - val
_acc: 0.8016
Epoch 2/3
47990/47990 [==============================] - 24s 494us/step - loss: 0.4008 - acc: 0.8090 - val_loss: 0.3872 - val
_acc: 0.8110
Epoch 3/3
47990/47990 [==============================] - 25s 522us/step - loss: 0.3773 - acc: 0.8190 - val_loss: 0.3725 - val
_acc: 0.8190

<keras.callbacks.History at 0x2c1a2fc4b00>
```

Results

Using 8 kernels

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 7s 148us/step - loss: 0.4854 - acc: 0.7736 - val_loss: 0.4384 - val_
acc: 0.7940
Epoch 2/3
47990/47990 [==============================] - 7s 150us/step - loss: 0.4228 - acc: 0.7989 - val_loss: 0.3966 - val_
acc: 0.8091
Epoch 3/3
47990/47990 [==============================] - 8s 157us/step - loss: 0.3886 - acc: 0.8138 - val_loss: 0.3776 - val_
acc: 0.8177

<keras.callbacks.History at 0x172cc323ba8>
```

Using 32 kernels

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 23s 480us/step - loss: 0.4503 - acc: 0.7872 - val_loss: 0.4060 - val
_acc: 0.8023
Epoch 2/3
47990/47990 [==============================] - 24s 492us/step - loss: 0.3898 - acc: 0.8127 - val_loss: 0.3795 - val
_acc: 0.8156
Epoch 3/3
47990/47990 [==============================] - 27s 556us/step - loss: 0.3734 - acc: 0.8215 - val_loss: 0.3744 - val
_acc: 0.8157

<keras.callbacks.History at 0x2c1a38ed400>
```

Results

Dropout .4

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 24s 490us/step - loss: 0.4738 - acc: 0.7792 - val_loss: 0.4297 - val
_acc: 0.7942
Epoch 2/3
47990/47990 [==============================] - 26s 551us/step - loss: 0.4158 - acc: 0.8022 - val_loss: 0.3923 - val
_acc: 0.8094
Epoch 3/3
47990/47990 [==============================] - 28s 580us/step - loss: 0.3947 - acc: 0.8113 - val_loss: 0.3819 - val
_acc: 0.8183

<keras.callbacks.History at 0x19aa02439b0>
```

Dropout .2

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 23s 480us/step - loss: 0.4503 - acc: 0.7872 - val_loss: 0.4060 - val
_acc: 0.8023
Epoch 2/3
47990/47990 [==============================] - 24s 492us/step - loss: 0.3898 - acc: 0.8127 - val_loss: 0.3795 - val
_acc: 0.8156
Epoch 3/3
47990/47990 [==============================] - 27s 556us/step - loss: 0.3734 - acc: 0.8215 - val_loss: 0.3744 - val
_acc: 0.8157

<keras.callbacks.History at 0x2c1a38ed400>
```

Results

IC50 Cutoff = 50

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 24s 497us/step - loss: 0.3667 - acc: 0.8362 - val_loss: 0.3422 - val
_acc: 0.8444
Epoch 2/3
47990/47990 [==============================] - 24s 506us/step - loss: 0.3267 - acc: 0.8473 - val_loss: 0.3293 - val
_acc: 0.8482
Epoch 3/3
47990/47990 [==============================] - 25s 514us/step - loss: 0.3109 - acc: 0.8563 - val_loss: 0.3166 - val
_acc: 0.8533
```

IC50 Cutoff = 200

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 23s 480us/step - loss: 0.4503 - acc: 0.7872 - val_loss: 0.4060 - val
_acc: 0.8023
Epoch 2/3
47990/47990 [==============================] - 24s 492us/step - loss: 0.3898 - acc: 0.8127 - val_loss: 0.3795 - val
_acc: 0.8156
Epoch 3/3
47990/47990 [==============================] - 27s 556us/step - loss: 0.3734 - acc: 0.8215 - val_loss: 0.3744 - val
_acc: 0.8157

<keras.callbacks.History at 0x2c1a38ed400>
```

Results

- IC50 cutoff = 800
- 2 kernels, each 1 long
- 2 nodes on Dense layer
- 607 Parameters

```
batch_size = 128
epochs = 3
model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

```
Train on 47990 samples, validate on 11998 samples
Epoch 1/3
47990/47990 [==============================] - 4s 75us/step - loss: 0.6506 - acc: 0.6728 - val_loss: 0.6098 - val_a
cc: 0.7055
Epoch 2/3
47990/47990 [==============================] - 3s 70us/step - loss: 0.5988 - acc: 0.7076 - val_loss: 0.5856 - val_a
cc: 0.7152
Epoch 3/3
47990/47990 [==============================] - 3s 70us/step - loss: 0.5855 - acc: 0.7119 - val_loss: 0.5777 - val_a
cc: 0.7216
```

Discussion

- More data means less accuracy, but more practical benefits
- Amount of layers / kernels no significant impact
- Only changing many parameters made significant impact

Conclusion

Conclusion

- Problem statement is confirmed
- Accuracy of about .8 means neural net is practical to use

Applications

- Can be used to save time + money on research
- Protein inhibitors used to treat a variety of diseases
  - Kinases control cell & protein activity
  - Kinases cause diseases such as cancer

Acknowledgements

- Thank you to Mr. Simon for being my mentor and for the rest of the SRP staff for

Questions?