2019 I/O Summer Workshop

by Tsung-Ren (Tren) Huang (trhuang@g.ntu.edu.tw (mailto:trhuang@g.ntu.edu.tw))

2.1 Model training & testing

2.1.0 Model construction

In [25]:

```
# DNN network
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X.shape[1],)))
model.add(Dense(128, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               1024
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 17,794
Trainable params: 17,794
Non-trainable params: 0
_________________________________________________________________
```

2.1.1 Model training

In [26]:

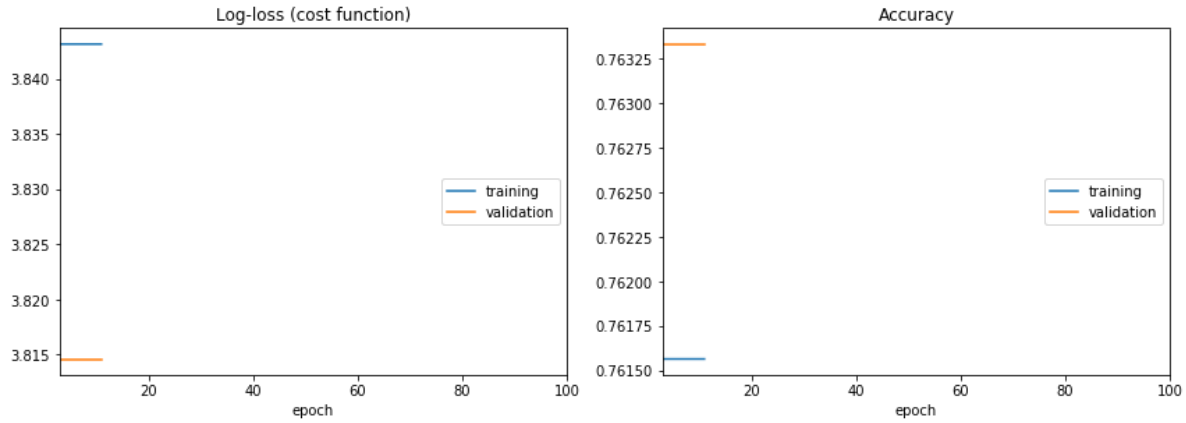
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
```

In [33]:

```
#! pip install livelossplot
from livelossplot import PlotLossesKeras
from keras.callbacks import EarlyStopping, ModelCheckpoint
es = EarlyStopping(monitor='val_loss', patience=10, verbose=1) # Stop when training
ck = ModelCheckpoint(filepath='dnn.h5', verbose=1, save_best_only=True, monitor='val
```

In [34]:

```
model.fit(train_x, train_y, epochs=100, batch_size=1000, validation_data=(test_x, te
```




```
Log-loss (cost function):
training   (min:    3.843, max:    3.843, cur:    3.843)
validation (min:    3.815, max:    3.815, cur:    3.815)

Accuracy:
training   (min:    0.762, max:    0.762, cur:    0.762)
validation (min:    0.763, max:    0.763, cur:    0.763)

Epoch 00011: val_loss did not improve from 3.81462
Epoch 00011: early stopping
```

Out[34]:

```
<keras.callbacks.History at 0x2ae363b350f0>
```

2.1.2 Model testing

In [35]:

```
from keras.models import load_model
model=load_model('dnn.h5')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               1024
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 17,794
Trainable params: 17,794
Non-trainable params: 0
_________________________________________________________________
```

In [36]:

```
predicted_responses=model.predict(test_x)
print(predicted_responses)
```

```
[[  1.00000000e+00   1.85397982e-26]
 [  1.00000000e+00   3.82774343e-34]
 [  1.00000000e+00   2.40379376e-23]
 ...,
 [  1.00000000e+00   5.54625995e-29]
 [  1.00000000e+00   3.54454233e-27]
 [  1.00000000e+00   4.12801033e-33]]
```

In [37]:

```
predicted_y=argmax(predicted_responses,axis=1)
print(predicted_y)
```

```
[0 0 0 ..., 0 0 0]
```

In [38]:

```
print(sum(predicted_y==argmax(test_y))/len(test_y))
```

```
1.0
```

2 Convolutional Neural Networks

The following example is adapted from:

Huang, T.-R. (in press). Understanding potentially biased artificial agents powered by supervised learning: Perspectives from cognitive psychology and cognitive neuroscience. *Chinese Journal of Psychology*. http://mil.psy.ntu.edu.tw/machine-bias (http://mil.psy.ntu.edu.tw/machine-bias)

2.1 Model training & testing

2.1.0 Model construction

In [43]:

```
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Flatten, Activation
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=(128, 128, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 62, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 62, 62, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30752)             0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               3936384
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 3,946,657
Trainable params: 3,946,657
Non-trainable params: 0
_________________________________________________________________
```

2.1.1 Model training

In [44]:

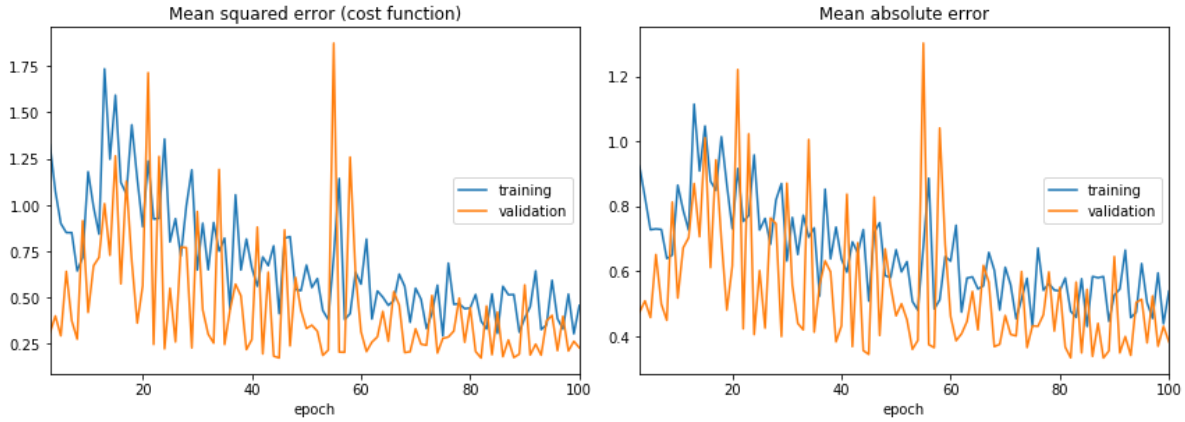
```
model.compile(loss='mean_squared_error', optimizer='RMSprop', metrics=['mae'])
```

In [45]:

```
from livelossplot import PlotLossesKeras
from keras.callbacks import EarlyStopping, ModelCheckpoint
es = EarlyStopping(monitor='val_loss', patience=100, verbose=1) # Stop when training
ck = ModelCheckpoint(filepath='cnn.h5', verbose=1, save_best_only=True, monitor='val
```

In [46]:

```
model.fit(train_x, train_y, batch_size=100, epochs=100, validation_data=(test_x, tes
```




```
Mean squared error (cost function):
training   (min:    0.294, max:  198.533, cur:    0.457)
validation (min:    0.174, max:    1.874, cur:    0.230)

Mean absolute error:
training   (min:    0.430, max:    8.634, cur:    0.538)
validation (min:    0.333, max:    1.302, cur:    0.383)

Epoch 00100: val_loss did not improve from 0.17416
```

Out[46]:

```
<keras.callbacks.History at 0x2ae29d4b66d8>
```

2.1.2 Model testing

In [47]:

```
from keras.models import load_model
model=load_model('cnn.h5')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 62, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 62, 62, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30752)             0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               3936384
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 3,946,657
Trainable params: 3,946,657
Non-trainable params: 0
_________________________________________________________________
```

In [48]:

```
predicted_y=model.predict(test_x)
print(predicted_y)
```

```
[[ 2.29196763]
 [ 2.12851262]
 [ 2.19235158]
 [ 1.86390662]
 [ 1.93450367]
 [ 2.31301093]
 [ 1.8808887 ]
 [ 2.23275185]
 [ 2.66196847]
 [ 2.46302772]
 [ 2.27051091]
 [ 2.60738397]
 [ 2.46897507]
 [ 2.12155414]
 [ 3.28036237]
 [ 4.04533911]
 [ 3.57508421]
 [ 4.3025794 ]
 [ 2.47654867]
 [ 3.19339609]
 [ 2.93638706]
 [ 2.67830801]
 [ 3.18565011]
 [ 2.84134555]
 [ 2.0826757 ]
 [ 2.30775714]
 [ 3.11743116]
 [ 2.9734323 ]
 [ 2.46967959]
 [ 1.89477682]
 [ 2.49114299]
 [ 2.05923748]
 [ 2.40301561]
 [ 2.86217237]
 [ 1.97661746]
 [ 2.40444541]
 [ 2.16838956]
 [ 3.39055371]
 [ 2.51928234]
 [ 2.41086245]
 [ 2.22713923]
 [ 2.92198634]
 [ 2.73719597]
 [ 2.26920795]
 [ 2.55093837]
 [ 2.85629392]
 [ 2.54644823]
 [ 2.34416747]
 [ 2.04403329]
 [ 3.22449017]]
```

In [49]:

```
# Pearson's correlation between human and machine ratings:
np.corrcoef(test_y, np.squeeze(predicted_y))
```

Out[49]:

```
array([[ 1.        ,  0.70350253],
       [ 0.70350253,  1.        ]])
```