# EXPORTING MODELS AND INFERENCING IN KERAS

DR. MATTHEW SMITH

ADACS, SWINBURNE UNIVERSITY OF TECHNOLOGY

# PREREQUISITES

- WARNING: This is not a stand-alone example.
- The testing and training data required to run these examples is not provided with this GIT repository.
- You will need the data contained within ADACS_ML_A and copy the training and test data into the Train and Test directories before use.
- If you need help, ask me or one of the other supervisors / instructors / gurus.

# INFERENCING

- Inferencing is the process of:
  - Taking the learned "knowledge" of a previously trained machine, and
  - Application of this knowledge to new, previously unseen, data.
- As such, the process of inferencing might be considered as the goal of machine learning.
- Since there is no training (or heavy computation) the process of inferencing is usually quite fast.

# INFERENCING

- Up to now, the codes provided to you:
  - Create a Keras model,
  - Train it – perform computations on the training data set for computing the model weights (using model.fit())
  - Test it – use the test data set to check the computed weights against known data sets (using model.evaluate()).

# INFERENCING

- To move forward, we need to:
  - Modify our existing codes to export our model in a form which can be loaded later on, and
  - Create a new code – which we shall call infer.py – which will perform inferencing given a single data set as an input.

# EXPORTING YOUR KERAS MODEL

- Open your train.py file – the first few lines of code here should look familiar.
- The addition are the lines of code shown in the lower half.

```
# Fit the model using the training set
history = model.fit(X_train, Y_train, epochs=N_epochs, batch_size=32)

# Plot the history
plot_history(history)

# Final evaluation of the model using the Test Data
print("Evaluating Test Set")
scores = model.evaluate(X_test, Y_test, verbose=1)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Export the model to file
model_json = model.to_json()
with open("model.json", "w") as json_file:
        json_file.write(model_json)
# Save the weights as well, in HDF5 format
model.save_weights("model.h5")
```

Existing

New

# EXPORTING YOUR KERAS MODEL

- One method for exporting the model information to file is to employ the JSON format (pronounced JAY-SON)
- JSON: Javascript Object Notation
- This is simply a light-weight, text based format used for data exchange.
- The idea is that a JSON file is easily viewed by humans and interpretted by computer systems – it looks very similar to a C/C++ code.

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

# EXPORTING YOUR KERAS MODEL

- The object holding the json data is produced using the code:

model_json = model.to_json()

- This is then written to file for us to load later.

```
# Fit the model using the training set
history = model.fit(X_train, Y_train, epochs=N_epochs, batch_size=32)

# Plot the history
plot_history(history)

# Final evaluation of the model using the Test Data
print("Evaluating Test Set")
scores = model.evaluate(X_test, Y_test, verbose=1)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Export the model to file
model_json = model.to_json()
with open("model.json", "w") as json_file:
        json_file.write(model_json)
# Save the weights as well, in HDF5 format
model.save_weights("model.h5")
```

# EXPORTING YOUR KERAS MODEL

- This is a sample of the JSON file produced by Keras using the to_json() function: (at the terminal type: cat model.json <enter>)


```
{"class_name": "Sequential", "keras_version": "2.1.4", "config": [{"class_name": "Dense", "config": {
"kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale"
: 1.0, "seed": null, "mode": "fan_avg"}}, "name": "dense_1", "kernel_constraint": null, "bias_regular
izer": null, "bias_constraint": null, "dtype": "float32", "activation": "relu", "trainable": true, "k
ernel_regularizer": null, "bias_initializer": {"class_name": "Zeros", "config": {}}, "units": 16, "ba
tch_input_shape": [null, 128], "use_bias": true, "activity_regularizer": null}}, {"class_name": "Dens
e", "config": {"kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "un
iform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "dense_2", "kernel_constraint": null
, "bias_regularizer": null, "bias_constraint": null, "activation": "softmax", "trainable": true, "ker
nel_regularizer": null, "bias_initializer": {"class_name": "Zeros", "config": {}}, "units": 8, "use_b
ias": true, "activity_regularizer": null}}, {"class_name": "Dense", "config": {"kernel_initializer":
{"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "
```

# EXPORTING YOUR KERAS MODEL

- In addition to this, we need to export the weights computed by the model.

model.save_weights(INPUT)

Where INPUT here is the name of the HDF5 file we wish to save to. This file will also be used later in infer.py.

```
# Fit the model using the training set
history = model.fit(X_train, Y_train, epochs=N_epochs, batch_size=32)

# Plot the history
plot_history(history)

# Final evaluation of the model using the Test Data
print("Evaluating Test Set")
scores = model.evaluate(X_test, Y_test, verbose=1)
print("Accuracy: %.2f%%" % (scores[1]*100))

# Export the model to file
model_json = model.to_json()
with open("model.json", "w") as json_file:
        json_file.write(model_json)
# Save the weights as well, in HDF5 format
model.save_weights("model.h5")
```

# EXPORTING YOUR KERAS MODEL

- HDF5 File format = $5^{th}$ generation Hierarchical Data Format (HDF), which is designed to store large amounts of data.
- Originally developed at the US's NCSA (National Center for Supercomputing Applications), home of the Blue Waters supercomputer (originally IBM, later Cray)
- We can load these files in Python (by importing the h5py module), but we won't need to do that directly.

# INFER.PY – IMPORTING MODELS

# IMPORTING YOUR KERAS MODEL

- Importing a previously generated Keras model is almost a simple reversal of the export steps – we need to:
  - Open the model.json file for reading and load the data,
  - Create a model from the information contained within the loaded data, and
  - Compile the model, in the same way we compiled the model during training.

# IMPORTING YOUR KERAS MODEL

- Here are the key elements of this process.

```
# Load the JSON file
json_file = open('model.json','r')
loaded_model_json = json_file.read()
json_file.close()

# Set up the neural layer configuration in the model
model = model_from_json(loaded_model_json)
# Load the weights into the model
model.load_weights("model.h5")
# Compile it
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

- In this case, the model.json and model.h5 files are in the same directory as this python script.

# IMPORTING YOUR KERAS MODEL

- Another note: here we are required to compile the model.
- This gives us an opportunity to use different optimizers and loss computation methods than those used during training. I wouldn't.

```
# Load the JSON file
json_file = open('model.json','r')
loaded_model_json = json_file.read()
json_file.close()

# Set up the neural layer configuration in the model
model = model_from_json(loaded_model_json)
# Load the weights into the model
model.load_weights("model.h5")
# Compile it
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

# THE FINAL PRODUCT: INFER.PY

- Let's have a look at the entire code.
- The first new addition is the model_from_json module which needs to be imported.
- We can see this code parses the input provided from the command prompt – if the user does not provide an ID, we use ID = 2 as a default value.

```
# infer.py
# Written by Dr. Matthew Smith, Swinburne University of Technol
# Load a precomputed keras model and its weights for a single i
# USAGE: python infer.py ID where ID is the ID of the training
# we wish to load.

# Import modules
import numpy as np
from keras.models import Sequential
from keras.models import model_from_json
from utilities import *
import sys

# Parse the input
no_arg = len(sys.argv)
if (no_arg == 2):
        ID = int(sys.argv[1])
else:
        print("Usage: python view.py <Data_ID>")
        print("where Data_ID is a number.")
        print("Example: python infer.py 2")
        print("  -- will load X_2.dat and infer its class")
        ID = 2

print("Loading file = " + str(ID))
```

# THE FINAL PRODUCT: INFER.PY

- In this case, we are going to be lazy and load one of the test data sets for inferencing.
- Normally you would have the data you wished inferenced provided in another way – but since we are short on time, let's use the tools we have available.
- We use the read_test_data() function (from utilities.py) to load a single set of data in – if you wish to modify this later to load multiple sets, the tools are there for you.

```
# We still need to know how long the time series is
N_sequence = 128     # Length of each piece of data

# Create variables for use while inferencing.
# Keeping it in array form; you might want to inference
# multiple data sets later.
X_infer = np.empty([1,N_sequence])
Y_infer = np.empty(1)

# We can take this data from anywhere - let's load one of the training sets
X_infer[0,], Y_infer[0]  = read_test_data(ID, N_sequence)
```

# THE FINAL PRODUCT: INFER.PY

- After this, we are free to load our model (we could have done this first, but no matter).
- We use two functions to perform our inference:

```
# Load the JSON file
json_file = open('model.json','r')
loaded_model_json = json_file.read()
json_file.close()

# Set up the neural layer configuration in the model
model = model_from_json(loaded_model_json)
# Load the weights into the model
model.load_weights("model.h5")
# Compile it
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Now try classifying the single data file we loaded
Class_infer = model.predict_classes(X_infer)

# Compute the class predictions - shouldn't be used as certainties.
Class_prob = model.predict(X_infer)

print("The predicted class is %d" % Class_infer[0])
print("Class Predictions: Class 0 = %f, Class 1 = %f" % ((1.0-Class_prob[0]), Class_prob[0]))
print("The actual loaded class is %d" % Y_infer[0])
```

# THE FINAL PRODUCT: INFER.PY

- The purpose of our ML engine was to predict classes – i.e. perform a classification task.
- Hence we use the predict_classes() function – in this case, the function will return either a [0] or [1] – if we load more data sets, it will be an array of 0's or 1's.

```
# Now try classifying the single data file we loaded
Class_infer = model.predict_classes(X_infer)
```

- There are additional inputs for the predict_classes function: feel free to browse the Keras documentation for these.

# THE FINAL PRODUCT: INFER.PY

- The previous function returned the predicted classes, which is very convenient for us.
- If we wish the raw output of the NN to be provided, we use the predict() function:

```
# Compute the class predictions - shouldn't be used as certainties.
Class_prob = model.predict(X_infer)
```

- This function replaces the predict_proba() function from earlier versions of Keras, though predict_proba should still work if used here (and provide the same result)

# THE FINAL PRODUCT: INFER.PY

## predict

```
predict(x, batch_size=None, verbose=0, steps=None)
```

Generates output predictions for the input samples.

Computation is done in batches.

**Arguments**

- **x**: The input data, as a Numpy array (or list of Numpy arrays if the model has multiple inputs).
- **batch_size**: Integer. If unspecified, it will default to 32.
- **verbose**: Verbosity mode, 0 or 1.
- **steps**: Total number of steps (batches of samples) before declaring the prediction round finished. Ignored with the default value of `None`.

# THE FINAL PRODUCT: INFER.PY

- Since we have a binary classification problem with a single output in our NN, we will have a single value returned.
- It's not really a probability – but just between you and me, let's pretend it is.

```
# Load the JSON file
json_file = open('model.json','r')
loaded_model_json = json_file.read()
json_file.close()

# Set up the neural layer configuration in the model
model = model_from_json(loaded_model_json)
# Load the weights into the model
model.load_weights("model.h5")
# Compile it
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Now try classifying the single data file we loaded
Class_infer = model.predict_classes(X_infer)

# Compute the class predictions - shouldn't be used as certainties.
Class_prob = model.predict(X_infer)

print("The predicted class is %d" % Class_infer[0])
print("Class Predictions: Class 0 = %f, Class 1 = %f" % ((1.0-Class_prob[0]), Class_prob[0]))
print("The actual loaded class is %d" % Y_infer[0])
```

# THE FINAL PRODUCT: INFER.PY

- Since we are using the functions contained within utilities.py, and loading the test data for inferencing, we should place infer.py in the same directories as train.py.
- When we call this script (python infer.py 24), this is what we get:

```
Loading file = 24
Loading file ./Test/X_24.dat
The predicted class is 0
Class Predictions: Class 0 = 0.931161, Class 1 = 0.068839
The actual loaded class is 0
```

- You can see that we are quite sure that the class is not class 1 (accurate), and we have correctly picked the class.

# ACTIVITY

- Make sure the codes provided by GIT are working correctly. Don't forget to reload the modules required by ozstar if you haven't already (. script.sh)
- Make a copy of the previous directory (ADACS_ML_A) (cp –r ADACS_ML_A FOLDER_NAME) and attempt:
  - Modify train.py yourself to export the JSON information and weights data,
  - Write your own inference code which is called by:

    **python infer.py FILENAME**

    where FILENAME is the name of the file containing the data we wish to classify.