McGill **Artificial Intelligence** Society

Deploying Your Machine Learning Models

By Li Zhang and Diego Lopez

Agenda

- Background
- Saving your model
- Deploying with TensorFlow.js
- Deploying with Flask

Static vs Dynamic Websites

Some Terminology

- Client
  - Hardware or software that is accessing a service
  - For our intents and purposes, refers to the browser
- Server
  - Hardware or software that is providing a service
- Front end
  - Synonym for client-side, i.e. user-facing code
- Back end
  - Synonym for server-side

Static Websites

- Websites where the server provides already existing files (HTML, JS, CSS)
- All the server does in this case is serve the necessary files
- For our purposes, static websites can be hosted on github at no cost, even if there is high traffic

Dynamic Websites

- Websites where the server has to generate websites files (HTML, JS, CSS) on the go
- Usually, sites with logins are dynamic
- For our purposes, dynamic websites can be hosted on cloud services but may incur costs

Saving Your Weights

Keras

- Whether we wish to save model to train later or for inference, we use
  - `model.save('path/to/dir/model.h5')`
- We can later load the model using
  - `from keras.models import load_model`
  - `model = load_model('path/to/dir/model.h5')`
- Note: for TensorFlow.js, Keras models are (highly) recommended

PyTorch

- Since we care only about inference, save weights using
  - `torch.save(model.state_dict(), path)`
- Load weights using
  - `model = ModelClass()`
  - `model.load_state_dict(torch.load(path))`

Deploying with TensorFlow.js

TensorFlow.js?

- JavaScript Library for client-side machine learning
- Can do inference and training
- For our purposes, we will use it to import already trained models into the browser
- At the moment, it's easiest to use Keras models
- A possible alternative is ONNX.js

Usage

- First, we need to convert the .h5 file into TensorFlow.js's format
- Then, we include this file somewhere inside the website directory (models directory)
- Finally, we load the model and predict on the client side

Client-Side Inference

- Since inference is done on the client side, this means that some preprocessing code which had previously been written in Python has to be recreated in JavaScript
- Thankfully, the Tensor API is almost identical, modulo the fact that JavaScript uses camelCase everywhere

Async & Await

- On the browser, the main thread is responsible for rendering the user interface
- This means that long computations would result in slow UI for the user
- For this reason, some code can be run asynchronously, so that it does not ‘pause’ the main thread
- Async and Await are JavaScript keywords for this purpose

TensorFlow.js Demo

Deploying with Flask

Flask?

- Micro web framework written in Python
- Web frameworks such as it aim to limit overhead for creating common functionality such as web APIs
- A possible alternative is Django

Usage

- Since we’re running from python and we’ll have server code as well as model inference code, it’s best to abstract the model behind a class or a function in a separate file
- The app.py file will contain all the server logic

Front End

- Flask applications still need front-end code, for instance, code that sends image data back to the server for inference
- For this purpose, the web templating tool Jinja is used, which allows for dynamic generation of HTML which depends on variable values and other logic.

Flask Demo

bit.ly/32dUf6A

Further readings

- Pre-trained GPT2/BERT/TransformerXL
- Flask
- MNIST
- PyTorch

Thanks!

Any questions?

You can find us at
https://mcgillai.com