docs https://keras.io

Deep Nets with

http://vem.quantumunlimited.org/the-gates-of-horn/

Professor Marie Roch

SAN DIEGO STATE UNIVERSITY

These slides only cover enough to get started with feed-forward networks and do not cover regularization which is very important. We'll get there later.

Keras

- Front end API to several deep learning libraries as backends:
  - Theano (Université de Montréal)
  - TensorFlow (Google)
  - Cognitive Toolkit (CNTK, Microsoft)

SAN DIEGO STATE UNIVERSITY

Keras

- Advantages
  - High-level specification of neural nets and other computation.
  - Transparent GPU vs non-GPU programming
  - Rapid specification

Keras concepts : Models

Models can be:

- Specified: Functionality is specified by invoking model methods, e.g. add a new layer of N nodes.
- Compiled: A compile method writes the back-end code to generate the model
- Fitted: Optimization step where weights are learned
- Evaluated: Tested on new data

Keras concepts : Models

We can use a Sequential model for a feed-forward network

```
from keras.models import Sequential
model = Sequential()
```


Keras concepts: Layers

- Layers can be added to a model
- Dense layers
  - compute $f(W^T x+b)$
  - user specifies
    - number of units
    - input/output tensor shapes (tensors are N-dimensional arrays)
    - activation functions
    - other options…

A Keras model

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

# Three category prediction with 2 hidden layers
# and 30 features, categorical output (3 categories)

model.add(Dense(10, activation='relu', input_dim = 30))
model.add(Dense(10, activation='relu', input_dim = 10))
# Output probability of each category
model.add(Dense(3, activation='softmax', input_dim = 10))
```



```
from keras.utils import np_utils

# model definition from previous slide…

# Specify type of gradient descent, loss metric, and
# measurement metric
model.compile(optimizer = "Adam",
              loss = "categorical_crossentropy",
              metrics = [metrics.categorical_accuracy])

# Not needed; prints architecture summary
model.summary()

# We need examples and labels for supervised learning
# examples:  NxM numpy.array where N=# samples, M=# features
examples = get_features()  # you write this
```

```
# Nx1 vector of our 3 categories
labels = get_labels()  # you write this

# Our network uses a Multinoulli distribution to
# output one of three choices.  Our labels are scalars,
# we need to convert these to vectors:
# 0 -> [1 0 0], 1 -> [0 1 0], 2 -> [0 0 1]
# this is sometimes called a “one-hot” vector
from keras.utils import np_utils
onehotlabels = np_utils.to_categorical(labels)

# train the model
# 10 passes (epochs) over data, mini-batch size 100
model.fit(examples, labels, batch_size=100, epochs=10)
```


Using a trained model

- To predict outputs

```
results = model.predict(examples)
```

  - results is Nx3 probabilities
  - What are the following?
    - `np.sum(results, axis=1)`
    - `np.argmax(results, axis=1)`

Using a trained model

- To evaluate performance

```
# Returns list of metrics
results = model.evaluate(test_examples, test_labels)

# model.metrics_names tells us what was measured
# here:  ['loss', 'categorical_accuracy']

print(results[1])   # accuracy
# In some fields, it is common to report error: 1 - accuracy
```


To install SciKit: conda install scikit-learn

N-fold cross validation

```
# Create a plan for k-fold testing with shuffling
# of examples using SciKit's StratifiedKFold
# http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

# Randmize examples within each fold
kfold = StratifiedKFold(n_folds, shuffle=True)

# Generate indices that can be used to split into training
# and test data, e.g. examples[train_idx]
for (train_idx, test_idx) in kfold.split(Examples, Labels):
    # normally, we would gather results about each fold
    train_and_evaluate(examples, one_hot_labels,
              train_idx, test_idx)
```

An architecture for building networks

- Data-driven network construction
- Store constructors and their arguments in a list of tuples e.g.

```
network = [
   (Dense, [in_N], {'activation': 'relu'})
   (Dense, [out_N], {'activation': 'softmax'})]
```

- Tuple:
  - layer name
  - list of positional arguments
  - dictionary of named arguments

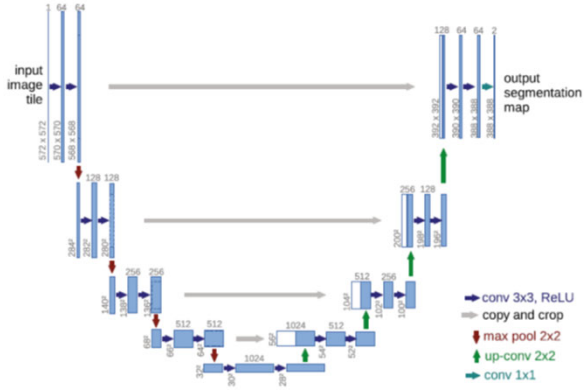
An architecture for building networks

Model construction is easy:

- Create a sequential model
- Loop over tuples
  - Call the layer type to construct a layer
    - Use * to pass in positional args: *tuple[1]
    - Use ** to treat dictionary as named args: **tuple[2]
  - Add the layer to the model

Other types of models

- Not all models are sequential:

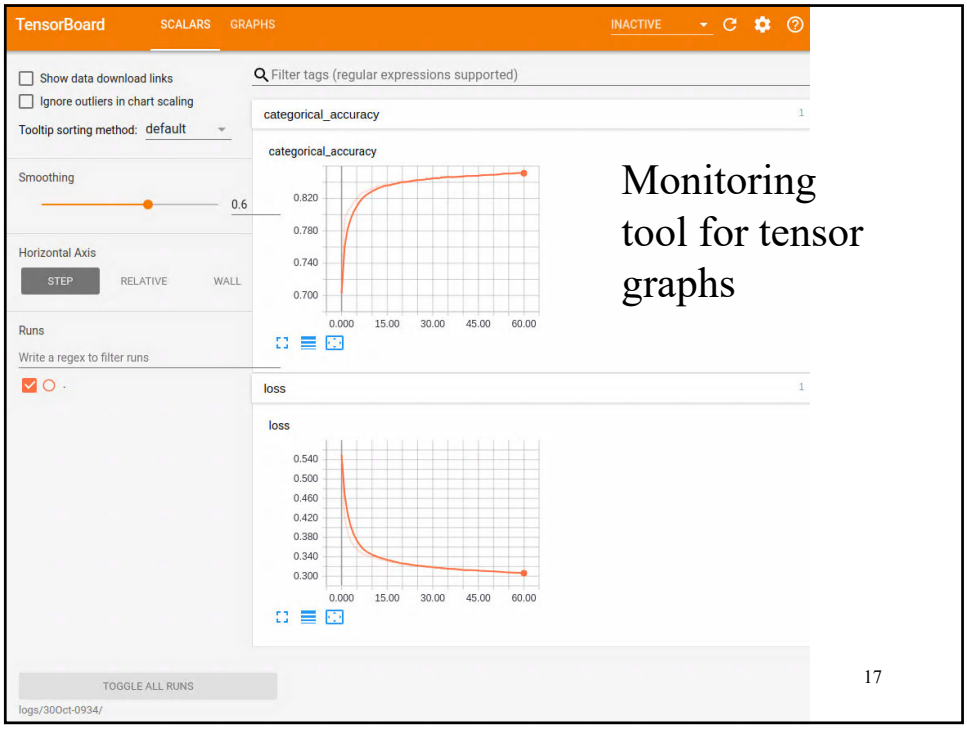


Ronneberger et al.'s U-Net MICCAI 2015

Complicated architectures

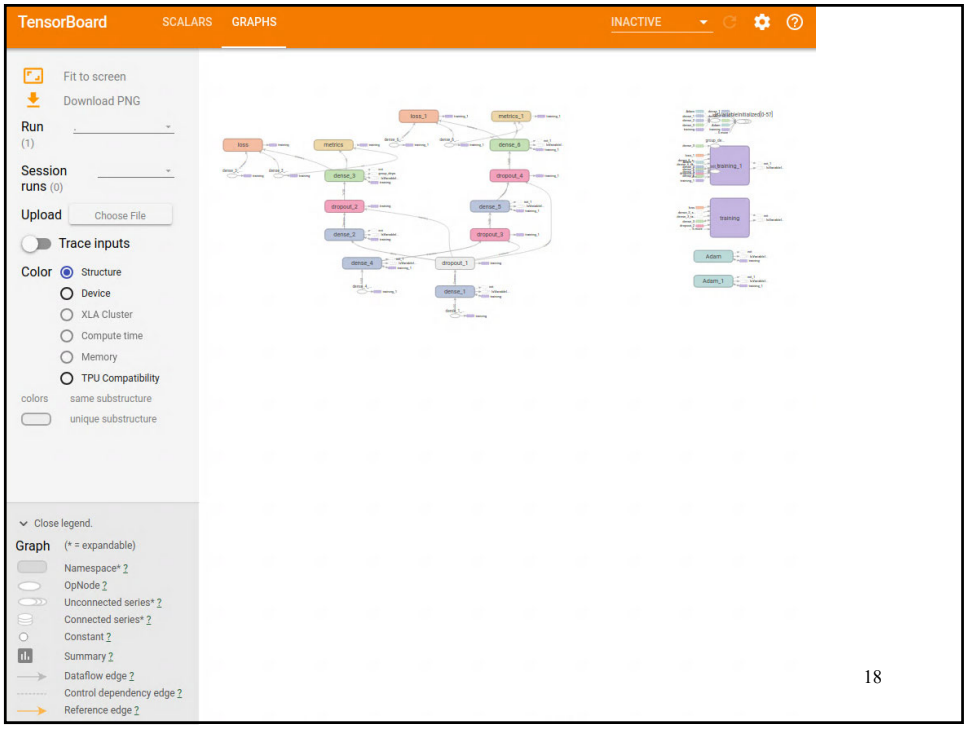
Use the functional API

```
# This returns a tensor
inputs = Input(shape=(N_inputs,))
# a layer instance is callable on a tensor, and returns a tensor
x = Dense(N_width, activation='relu')(inputs)
x = Dense(N_width, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
# This creates a model that includes the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels) # starts training
```



Monitoring tool for tensor graphs

17



18

TensorBoard

```
from keras.callbacks import TensorBoard
…
tensorboard = TensorBoard(
         # Write to logs directory, e.g. logs/30Oct-05:00
        log_dir="logs/{}".format(time.strftime('%d%b-%H%M')),
        histogram_freq=0,
        write_graph=True,  # Show the network
        write_grads=True   # Show gradients
        )
    # train the net
    model.fit(examples, onehotlabels,
              epochs=epochs, callbacks=[loss, tensorboard])
```

Then start tensorboard from the command prompt:

- `tensorboard –logdir logs/30Oct-05:00`

```
TensorBoard 1.5.1 at http://localhost:6006 (Press CTRL+C to
quit)
```

Point chrome at the URL and off you go…