DEEP LEARNING INSTITUTE

Modelling Time Series Data with Theano

Charles Killam, LP.D.
Certified Instructor, NVIDIA Deep Learning Institute
NVIDIA Corporation

DEEP LEARNING INSTITUTE

DLI Mission

Helping people solve challenging problems using AI and deep learning.

- Developers, data scientists and engineers
- Self-driving cars, healthcare and robotics
- Training, optimizing, and deploying deep neural networks

TOPICS

- Lab Perspective
- RNNs / LSTMs
- Keras / Theano
- Pandas / Numpy / Matplotlib
- Lab
  - Discussion / Overview
  - Launching the Lab Environment
  - Lab Review

LAB PERSPECTIVE

PURPOSE / GOAL

- Predict severity of illness in patients based on information found in electronic health records (EHRs)
- Provide feedback to clinicians when trying to assess the impact of treatment decision or raise early warning signs to flag

WHAT THIS LAB IS

- Discussion on the tools, techniques and processes commonly used to build RNN / LSTM networks to evaluate EHRs
- Introduction to aspects of RNNs, LSTMs, Keras, Theano, Pandas, Numpy and Matplotlib
- Guided, hands-on exercise using the tools noted above to build a LSTM network to evaluate EHRs

WHAT THIS LAB IS NOT

- Introduction to machine learning from first principles
- Explanation of electronic health records
- Rigorous mathematical formalism of neural networks
- Survey of all the features and options of Keras / Theano

ASSUMPTIONS

- You are familiar with:
  - Concept of electronic health records
  - Basics of neural networks
  - Basics of Pandas, Numpy and Matplotlib
- Helpful to have:
  - Familiarity with recurrent neural network (RNNs)

TAKE AWAYS

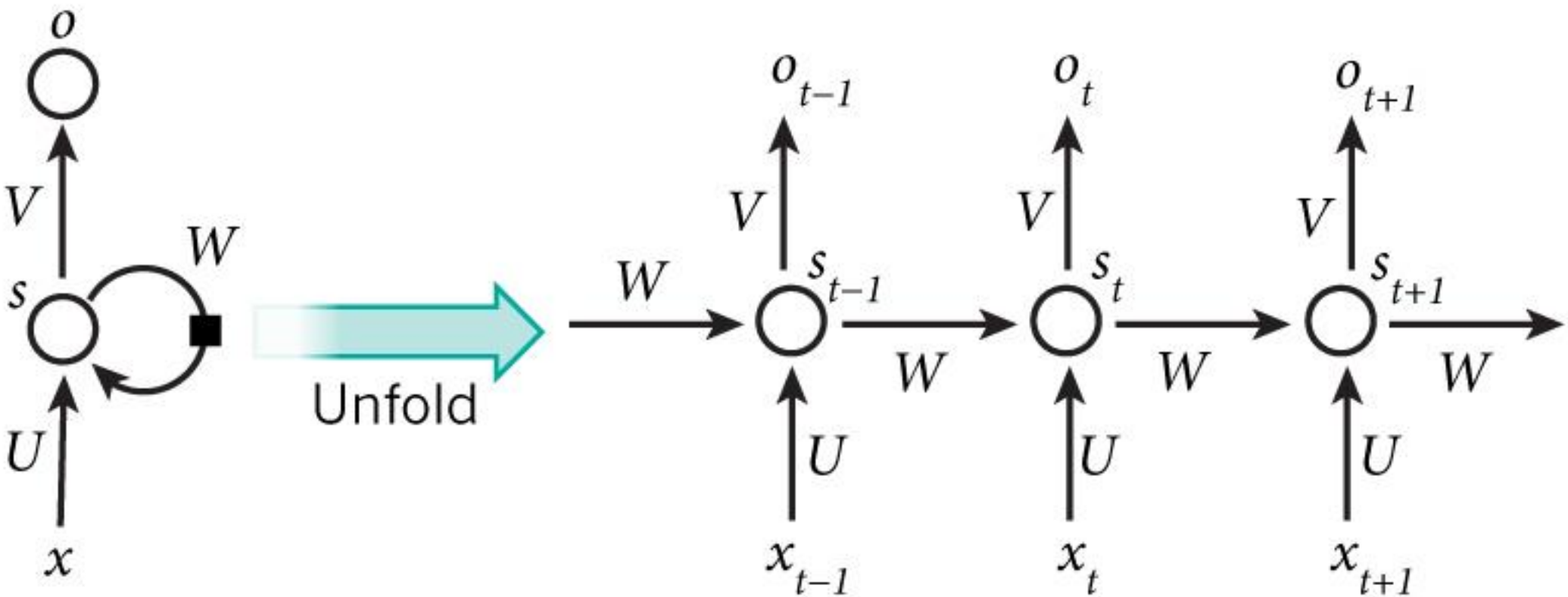
- Ability to setup your own recurrent neural network workflow using Keras / Theano and adapt it to your use case
- Know where to go for more info on RNNs, Keras and Theano
- Familiarity with data preparation process using Pandas, Numpy and Keras

RNN / LSTM

RECURRENT NEURAL NETWORK

- RNN = Recurrent Neural Network
  - Similar to traditional feed-forward network
  - RNNs include previous output state
  - Limited to looking back only a few steps due to vanishing gradient
    - Errors are backpropagated through time
    - Inputs from previous time steps get exponentially down weighted and are eventually driven to zero

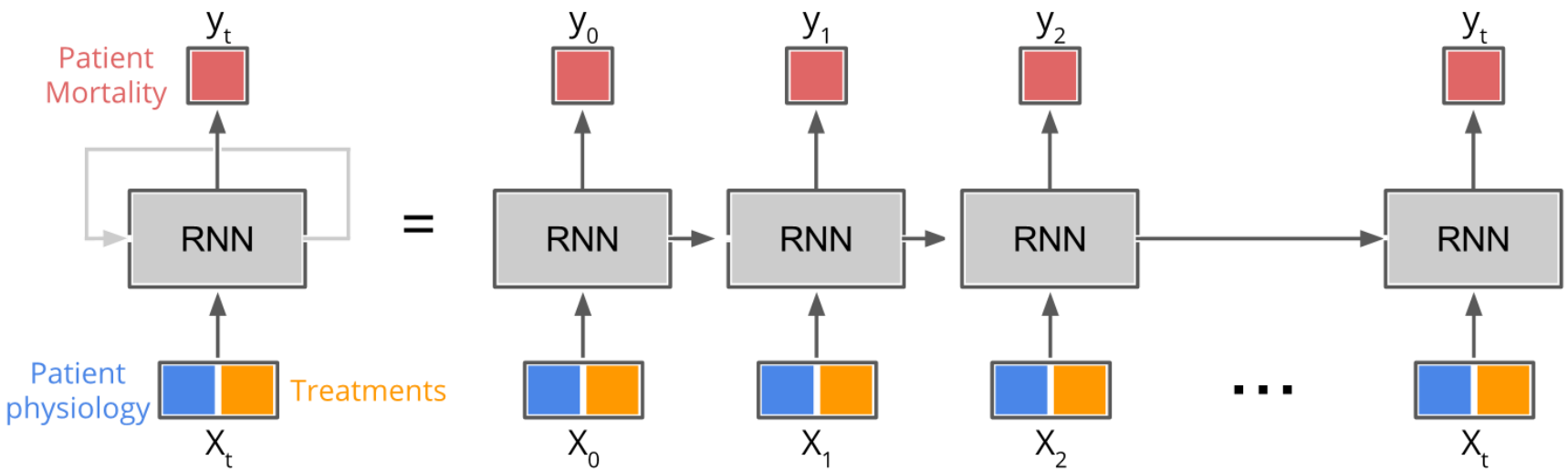
RNN



LONG SHORT TERM MEMORY

- LSTM = Long Short Term Memory
  - Variant of RNN
  - No vanishing gradient problem
  - LSTMs can learn “very deep” tasks that require memories of events that happened or millions of discrete time steps ago
- At each time step a measurement is recorded and used as input into the LSTM to yield a probability of survival prediction
  - Enables a real time monitoring of the patients probability of survival and insight into the patients trajectory

KERAS RNN


y_t
Patient Mortality
RNN
Patient physiology
Treatments
X_t
=
y_0
y_1
y_2
y_t
RNN
RNN
RNN
RNN
...
X_0
X_1
X_2
X_t

KERAS / THEANO

KERAS

- Modular neural network written in Python
- Runs on TensorFlow and Theano
  - Theano excels at RNNs / LSTMs
- Keras library allows for easy and fast prototyping
- Runs on GPUs and CPUs
- Compatible with Python 2.7 – 3.5

THEANO

- Theano excels at RNNs in general and LSTMs in particular
- "Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently" (http://deeplearning.net/software/theano/)
- Runs on either GPU or CPU architectures

PANDAS / NUMPY / MATPLOTLIB

PANDAS

- Used in academia and commercial domains
- Open-source, BSD-licensed project
- Fast and efficient DataFrame object for data manipulation with integrated indexing
- Contains tools for reading and writing data between in-memory data structures and different formats such as:
  - CSV and text files
  - Microsoft Excel
  - SQL databases
  - HDF5

NUMPY

- NumPy is a Python scientific computing package
- Open-source software
- Includes:
  - Support for large, N-dimensional arrays and matrices
  - Collection of high-level mathematical functions

MATPLOTLIB

- Matplotlib is a Python 2D plotting library producing publication quality figures
- Matplotlib can be used in:
  - Python scripts
  - Python and IPython shell
  - Jupyter notebook
  - Web application servers
- Supports Python version 2.7 – 3.5

LAB DISCUSSION / OVERVIEW

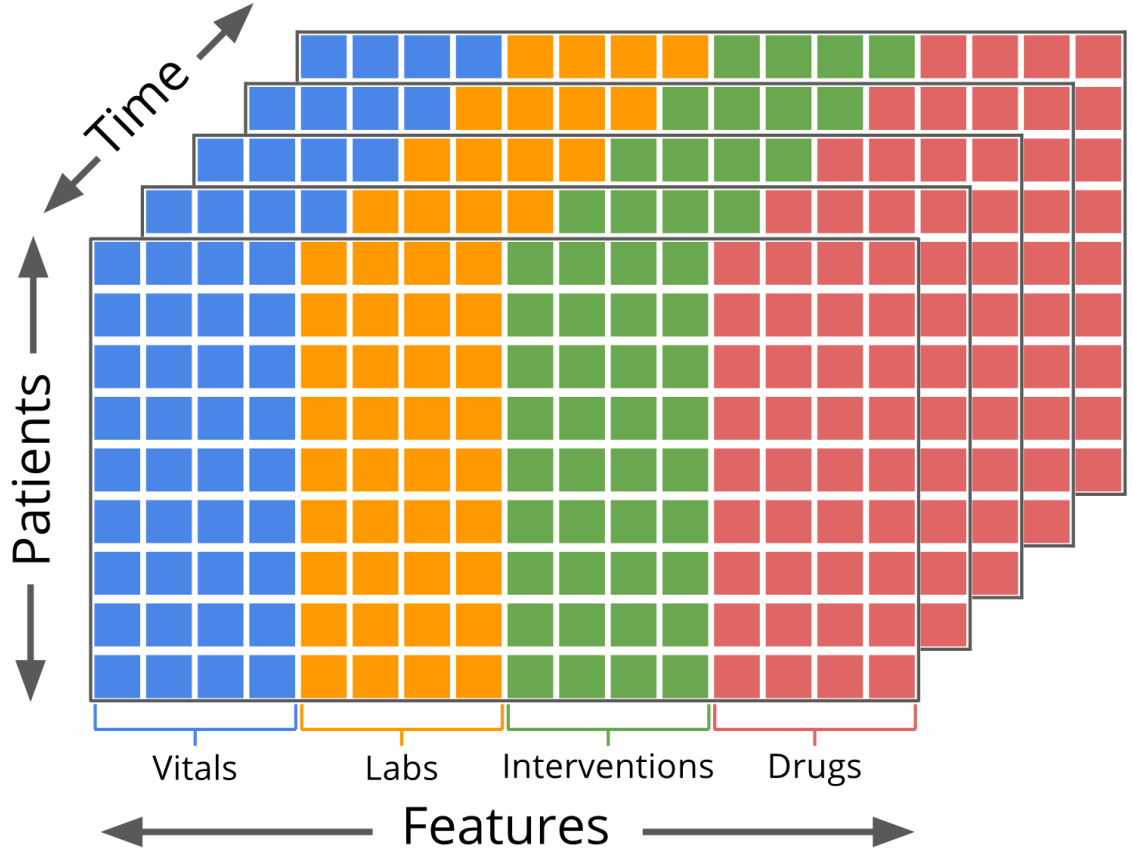
DATA

- Electronic health records (EHRs)
- Contains medical treatments and histories of patients over time
  - 15 years of data
- Data provided by PICU at Children's Hospital Los Angeles
  - 76,693 observations across 5,000+ unique patient encounters
- Data is an irregular time series of measurements taken over the course of a patient's stay in the ICU

DATA

Measurements include:

- Statistics - gender, age, weight
- Vitals - heart rate, respiratory rate
- Labs – glucose, creatinine
- Interventions – intubation, O2
- Drugs – dopamine, epinephrine



DATA

- Not all measurements were taken for all patients
- Dependent variable:
  - Alive – 1
  - Not alive – 0
- 1,113,529 rows containing 265 independent variables
- Mean observations per patient encounter = 223
- Median observations per patient encounter = 94

DATA

- Hierarchical Data Format (HDF) 5
  - Stores and organizes large amounts of scientific data
  - Designed by National Center for Supercomputing Applications
  - API supports most languages
  - Libraries compatible with Windows, OSX and Linux
  - Binary format
    - Not human readable
    - Efficient in storage size
    - Scales will to very large operational projects

LAB PROCESS

1. Setup
    a. Configure Theano options
    b. Import Numpy, Pandas and Matplotlib
    c. Define folders which contain training / testing datasets
    d. Load data using Pandas API

LAB PROCESS

2. Data Preparation
    a. Data review
    b. Data normalization
    c. Filling data gaps
    d. Data sequencing

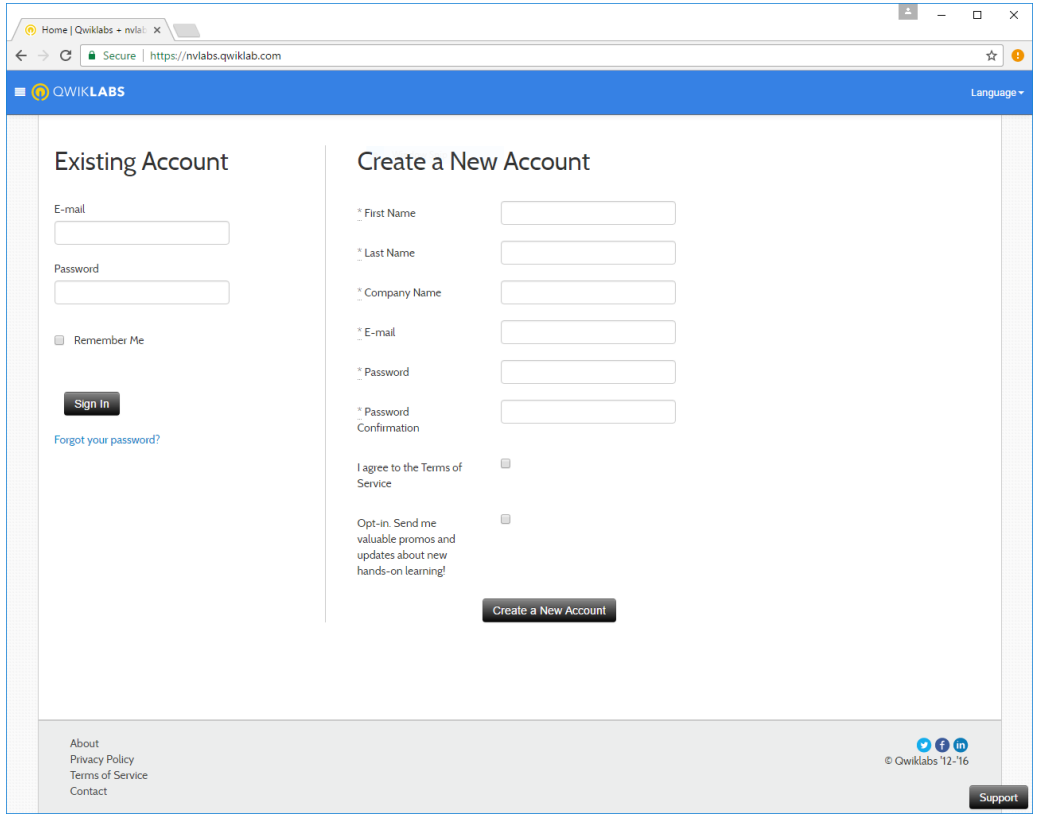
LAB PROCESS

3. Architect LSTM network using Keras and Theano
4. Build the model (feed data into network for training)
5. Evaluate model using validation (test) data
6. Visualize results
7. Compare baseline to PRISM3 and PIM2

LAB ENVIRONMENT

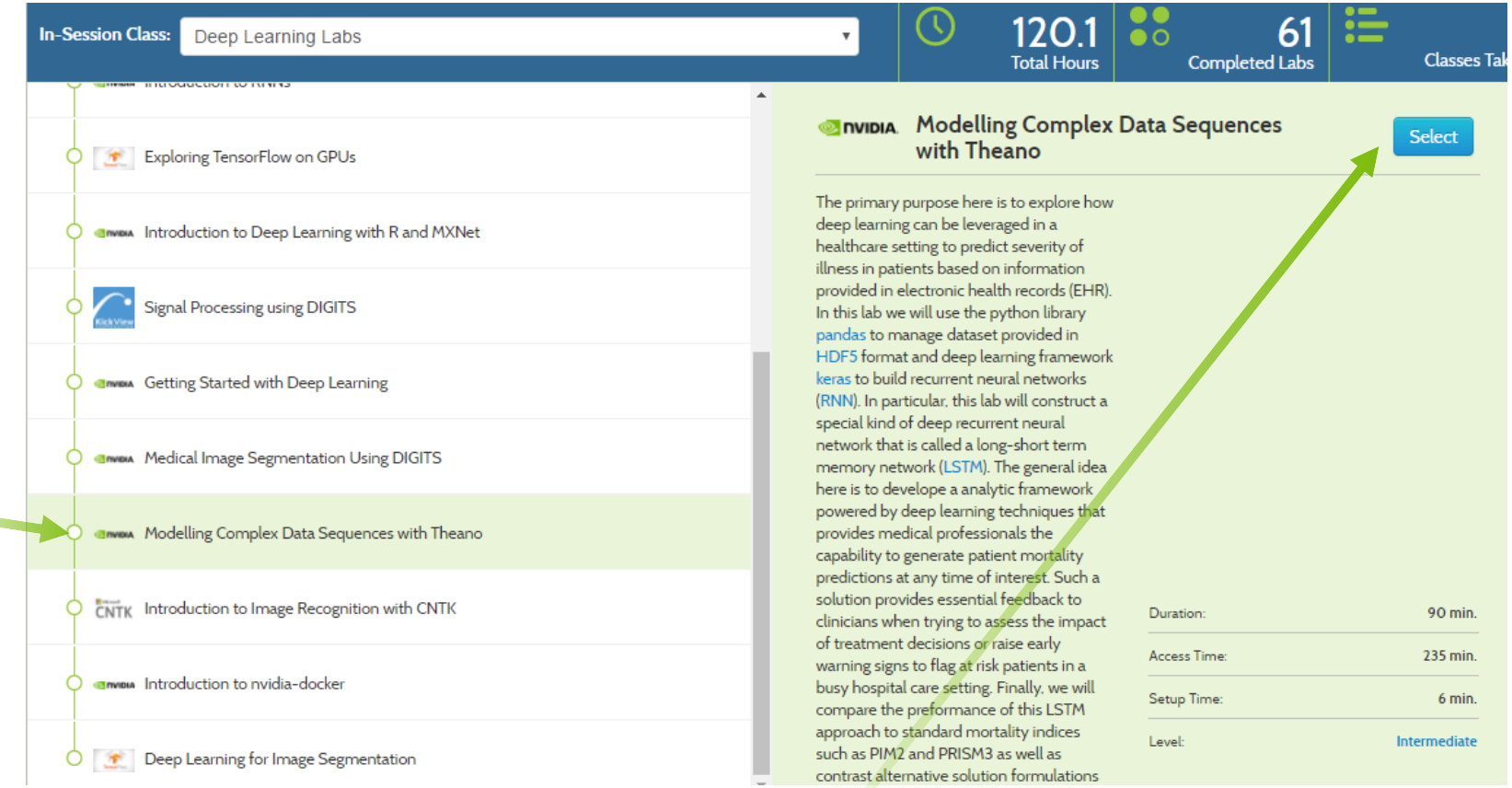
NAVIGATING TO QWIKLABS

1. Navigate to: https://nvlabs.qwiklab.com
2. Login or create a new account



ACCESSING LAB ENVIRONMENT

Click on Modelling Complex Data Sequences with Theano



Then click on Select

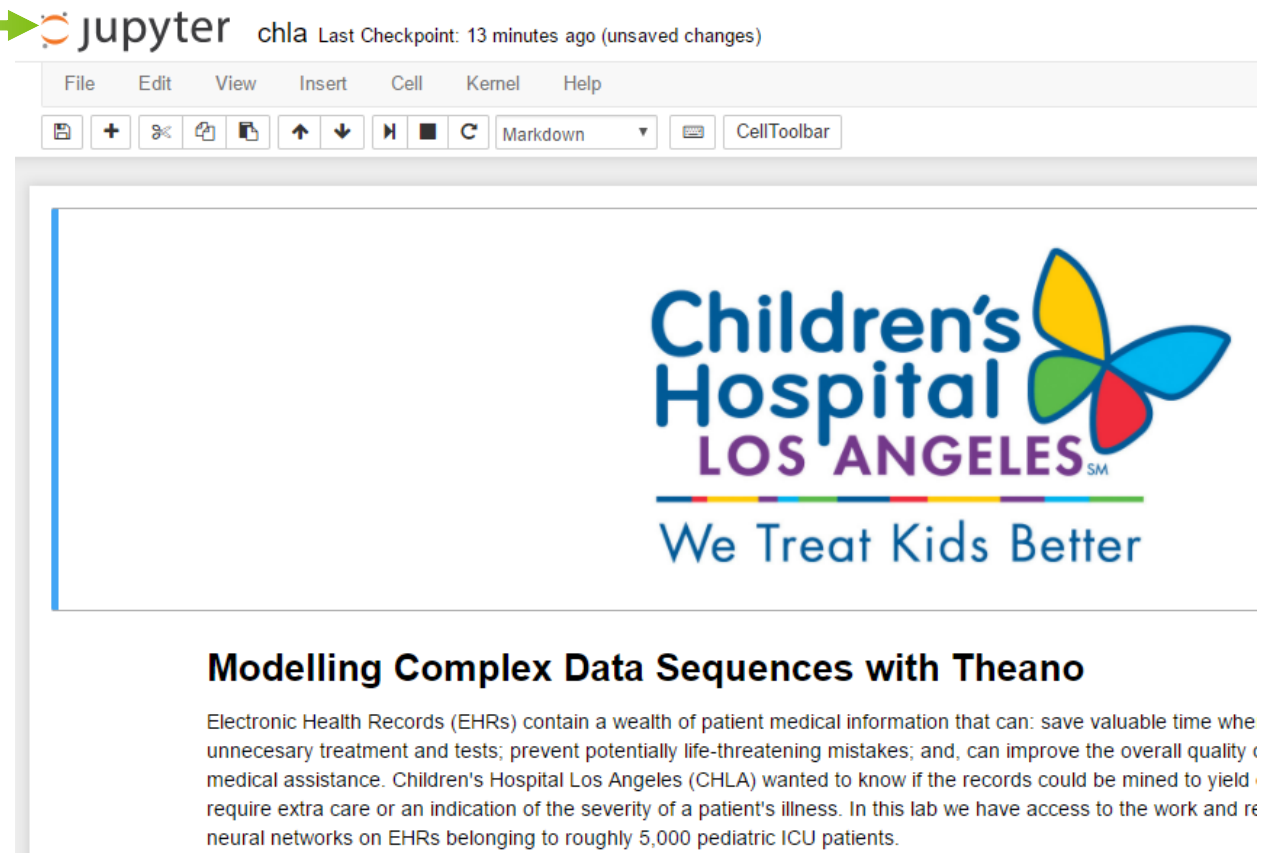
ACCESSING LAB INSTRUCTIONS

1. Click “Start Lab” to create an instance of the lab environment

2. Once the lab environment starts, click “here” to access lab instructions (Jupyter notebook)

ACCESSING LAB INSTRUCTIONS

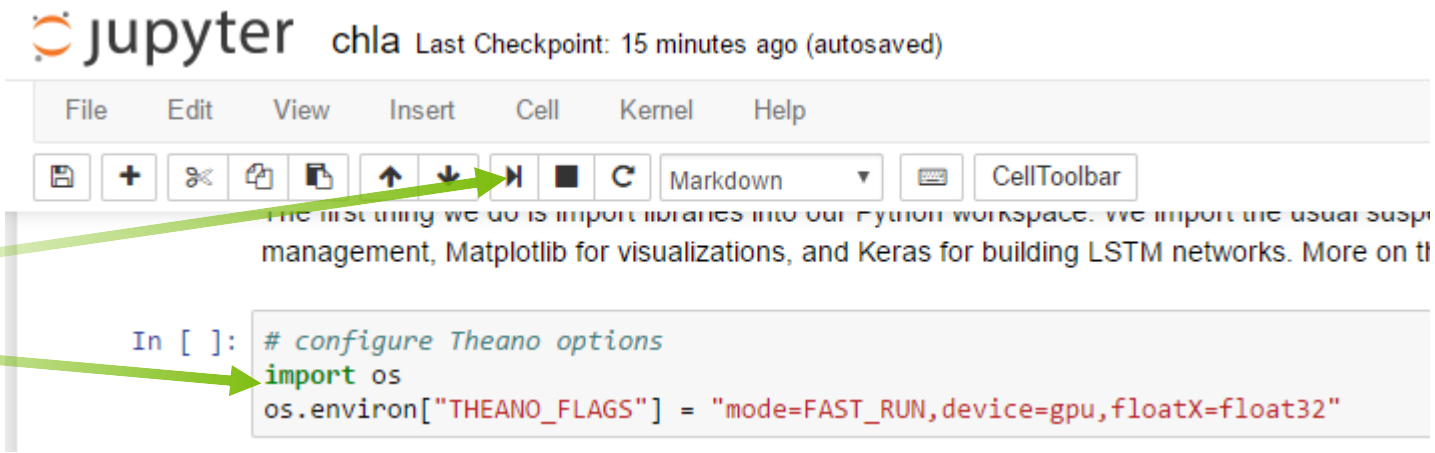
Should see Jupyter notebook



Place cursor in code block and click execute button

ACCESSING LAB INSTRUCTIONS

Place cursor in code block and click execute button



LAB REVIEW

LAB REVIEW

1. Setup
    a. Configure Theano options
    b. Import Numpy, Pandas and Matplotlib
    c. Define folders which contain training / testing datasets
    d. Load data using Pandas API

LAB REVIEW - IMPORT LIBRARIES #1B

In [ ]:
```
# configure theano options
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=gpu,floatX=float32"
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random

# configure notebook to display plots
%matplotlib inline
```

LAB REVIEW - DEFINE PATHS #1C

In [ ]:
```
# set up user paths
data_dir = './data'

# training data inputs: x and targets: y
x_train_path = os.path.join(data_dir, 'X_train.hdf')
y_train_path = os.path.join(data_dir, 'y_train.hdf')

# validation data inputs: x and targest: y
x_valid_path = os.path.join(data_dir, 'X_test.hdf')
y_valid_path = os.path.join(data_dir, 'y_test.hdf')
```

LAB REVIEW - LOAD DATA #1D

```
In [ ]: X_train = pd.read_hdf(x_train_path)
        y_train = pd.read_hdf(y_train_path)

        X_valid = pd.read_hdf(x_valid_path)
        y_valid = pd.read_hdf(y_valid_path)
```

LAB REVIEW

2. Data Preparation
    a. Data review
    b. Data normalization
    c. Filling data gaps
    d. Data sequencing

LAB REVIEW - DATA REVIEW #2A

In [5]: X_train

Out[5]:

		ABG Base excess (mEq/L)	ABG FiO2	ABG HCO3 (mEq/L)	ABG O2 sat (%)	ABG PCO2 (mmHg)	ABG PO2 (mmHg)	ABG TCO2 (mEq/L)	ABG pH	ALT (SGPT) (units/L)	AST (SGOT) (units/L)	...	Vasopressin	Vecuro
b'encounterID'	b'absoluteTime'													
	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	0.250000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	0.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	0.583333	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	0.750000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	1.383333	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	57.0	70.0	...	NaN	NaN

LAB REVIEW - DATA REVIEW #2A

In [7]:
```
# first select a random patient counter (encounter identifyer)
eIdx = random.choice(list(X_train.index.levels[0]))

# next specify a few variables to look at
variables = [
    'Age','Heart rate (bpm)','PulseOximetry','Weight',
    'SystolicBP','DiastolicBP','Respiratory rate (bpm)',
    'MotorResponse','Capillary refill rate (sec)'
]

# note that the full list of variables can be constructed using
#list(X_train.columns.values)

# have a look at the varibles for the patien
X_train.loc[eIdx, variables]
```

Out[7]:

	Age	Heart rate (bpm)	PulseOximetry	Weight	SystolicBP	DiastolicBP	Respiratory rate (bpm)	MotorResponse	Capillary refill rate (sec)
b'absoluteTime'									
0.000000	14.293174	118.0	100.0	53.0	113.000000	61.666667	23.0	6.0	3.0
0.166667	14.293193	97.0	100.0	NaN	108.000000	56.000000	27.0	NaN	NaN
0.683333	14.293252	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
0.916667	14.293279	93.0	99.0	NaN	109.000000	65.000000	27.0	6.0	2.0

LAB REVIEW - DATA REVIEW #2A

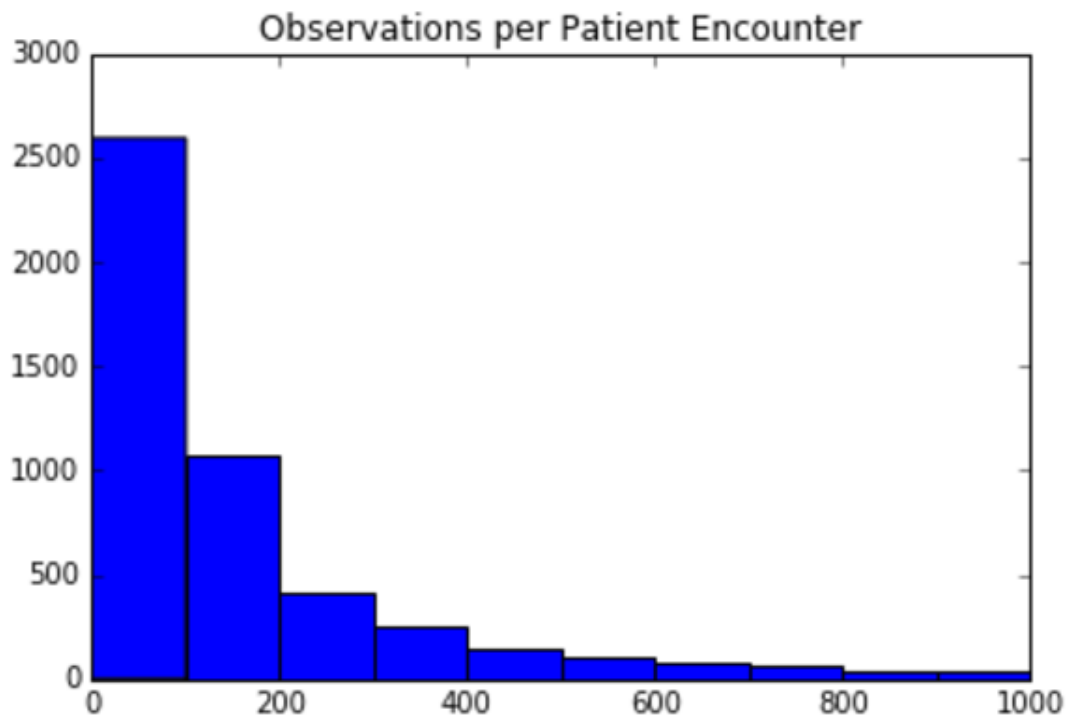
In [8]:
```
y_train
```

Out[8]:

		mortalityResponse
b'encounterID'	b'absoluteTime'	
	0.000000	1
	0.250000	1
	0.500000	1
	0.583333	1
	0.750000	1
	1.383333	1
	1.750000	1
	2.250000	1
	2.500000	1
	2.750000	1
	3.583333	1

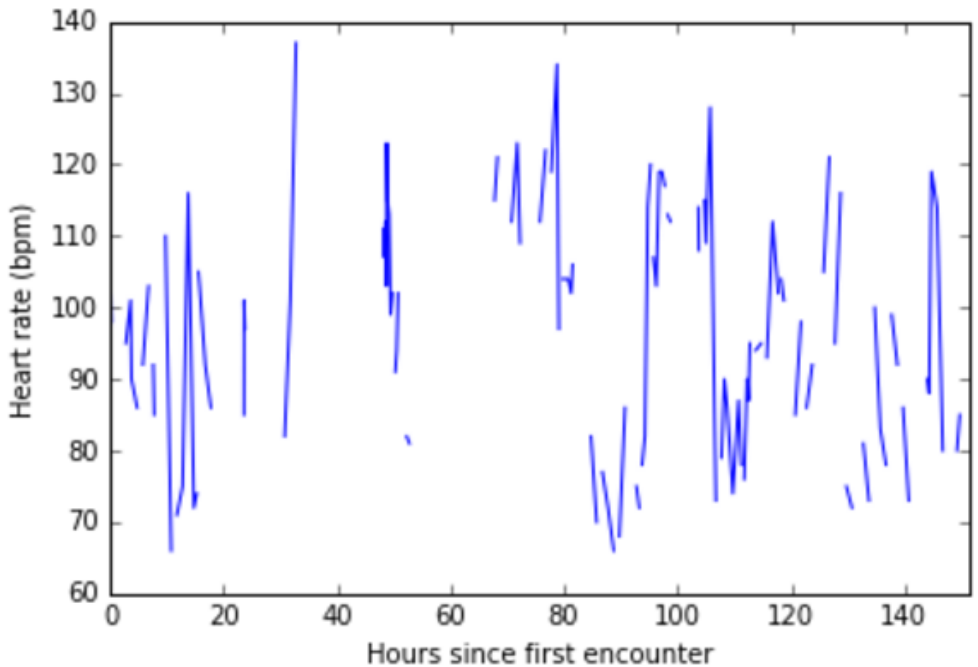
LAB REVIEW - DATA REVIEW #2A

```
In [11]: plt.hist(nobs,range=(0,1000))
         plt.title("Observations per Patient Encounter")
         plt.show()
```



LAB REVIEW - DATA REVIEW #2A

In [12]:
```
X_train.loc[8, "Heart rate (bpm)"].plot()
plt.ylabel("Heart rate (bpm)")
plt.xlabel("Hours since first encounter")
plt.show()
```



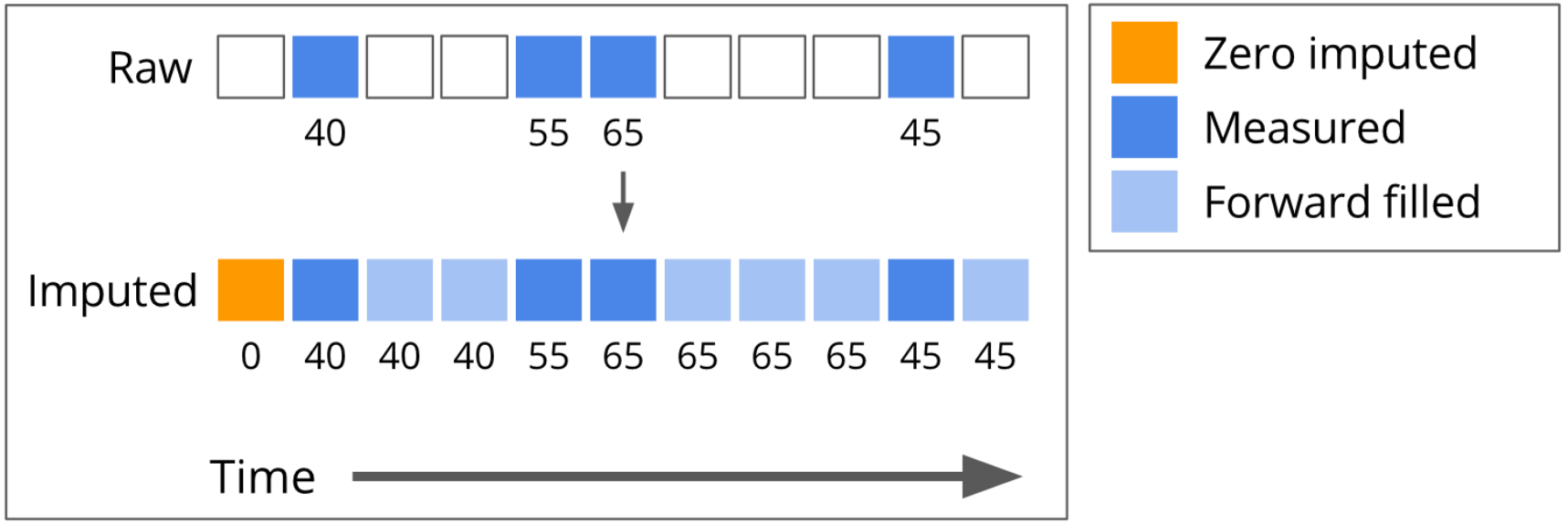
LAB REVIEW - DATA NORMALIZATION #2B

In [13]:
```
# create file path for csv file with metadata about variables
metadata = os.path.join(data_dir, 'ehr_features.csv')

# read in variables from csv file (using pandas) since each varable there is tagged with a category
variables = pd.read_csv(metadata, index_col=0)

# next, select only variables of a particular category for normalization
normvars = variables[variables['type'].isin(['Interventions', 'Labs', 'Vitals'])]

# finally, iterate over each variable in both training and validation data
for vId, dat in normvars.iterrows():

    X_train[vId] = X_train[vId] - dat['mean']
    X_valid[vId] = X_valid[vId] - dat['mean']
    X_train[vId] = X_train[vId] / (dat['std'] + 1e-12)
    X_valid[vId] = X_valid[vId] / (dat['std'] + 1e-12)
```

LAB REVIEW - DATA GAPS #2C

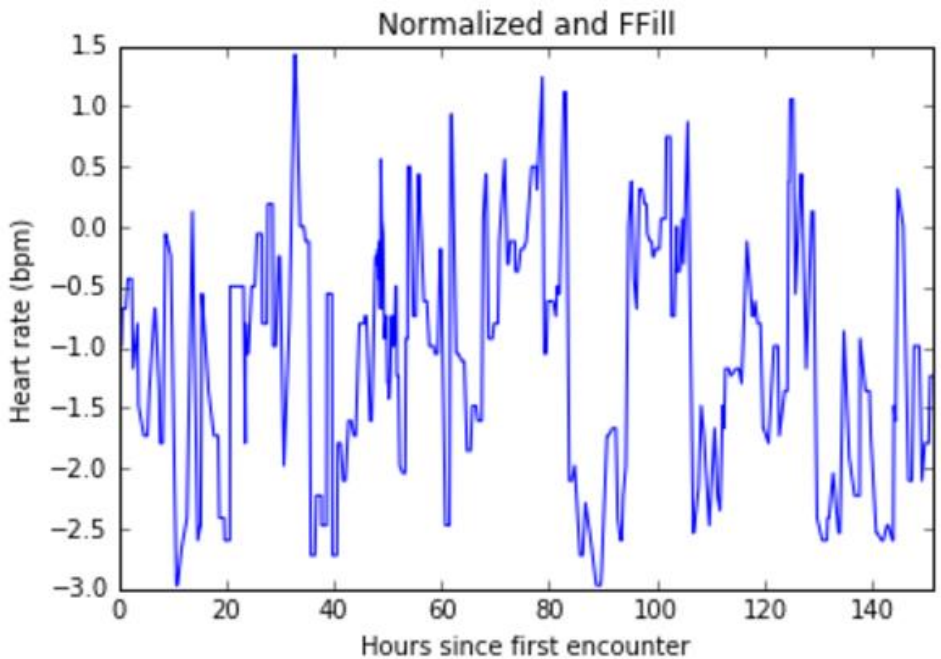
In [14]:
```
# first select variables which will be filled in
fillvars = variables[variables['type'].isin(['Vitals', 'Labs'])].index

# next forward fill any missing values with more recently observed value
X_train[fillvars] = X_train.groupby(level=0)[fillvars].ffill()
X_valid[fillvars] = X_valid.groupby(level=0)[fillvars].ffill()

# finally, fill in any still missing values with 0 (i.e. values that could not be filled forward)
X_train.fillna(value=0, inplace=True)
X_valid.fillna(value=0, inplace=True)
```

LAB REVIEW - DATA GAPS #2C



LAB REVIEW - DATA GAPS #2C

```
In [15]: X_train.loc[8, "Heart rate (bpm)"].plot()
         plt.title("Normalized and FFill")
         plt.ylabel("Heart rate (bpm)")
         plt.xlabel("Hours since first encounter")
         plt.show()
```



LAB REVIEW - DATA GAPS #2C

In [16]: X_train

Out[16]:

		ABG Base excess (mEq/L)	ABG FiO2	ABG HCO3 (mEq/L)	ABG O2 sat (%)	ABG PCO2 (mmHg)	ABG PO2 (mmHg)	ABG TCO2 (mEq/L)	ABG pH	ALT (SGPT) (units/L)	AST (SGOT) (units/L)	...	Vasop
b'encounterID'	**b'absoluteTime'**												
	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
	0.250000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
	0.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
	0.583333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
	0.750000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
	1.383333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.899637	-0.677127	...	0.0
	1.750000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.899637	-0.677127	...	0.0
	2.250000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.899637	-0.677127	...	0.0

LAB REVIEW - DATA SEQUENCING #2D

In [*]:
```
import keras
from keras.preprocessing import sequence

# max number of sequence length
maxlen = 500

# get a list of unique patient encounter IDs
teId = X_train.index.levels[0]
veId = X_valid.index.levels[0]

# pad every patient sequence with 0s to be the same length,
# then transforms the list of sequences to one numpy array
# this is for efficient minibatching and GPU computations
X_train = [X_train.loc[patient].values for patient in teId]
y_train = [y_train.loc[patient].values for patient in teId]

X_train = sequence.pad_sequences(X_train, dtype='float32', maxlen=maxlen, padding='post', truncating='post')
y_train = sequence.pad_sequences(y_train, dtype='float32', maxlen=maxlen, padding='post', truncating='post')

# repeat for the validation data

X_valid = [X_valid.loc[patient].values for patient in veId]
y_valid = [y_valid.loc[patient].values for patient in veId]

X_valid = sequence.pad_sequences(X_valid, dtype='float32', maxlen=maxlen, padding='post', truncating='post')
y_valid = sequence.pad_sequences(y_valid, dtype='float32', maxlen=maxlen, padding='post', truncating='post')
```

```
Using Theano backend.
```

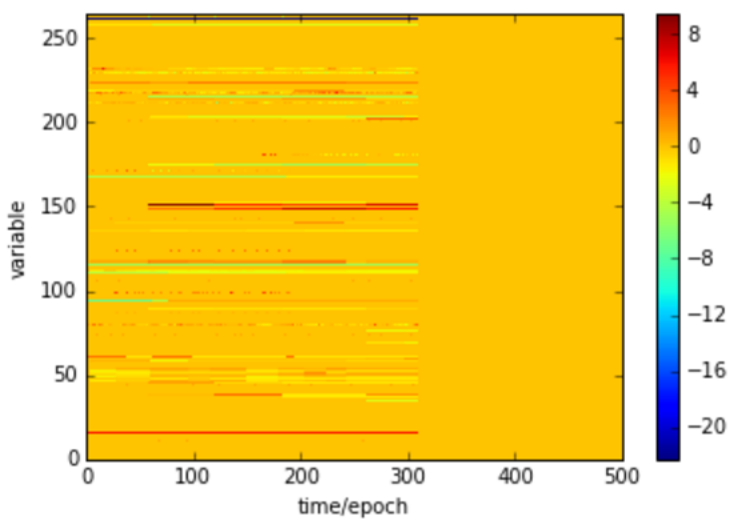
LAB REVIEW - DATA SEQUENCING #2D

In [18]:
```
# print the shape of the array which will be used by the network
# the shape is of the form (# of encounters, length of sequence, # of features)
print("X_train shape: %s | y_train shape: %s" % (str(X_train.shape), str(y_train.shape)))
print("X_valid shape: %s | y_valid shape: %s" % (str(X_valid.shape), str(y_valid.shape)))
```

```
X_train shape: (5000, 500, 265) | y_train shape: (5000, 500, 1)
X_valid shape: (2690, 500, 265) | y_valid shape: (2690, 500, 1)
```

LAB REVIEW - DATA SEQUENCING #2D

In [21]:
```
# figure out how many encounters we have
numencnt = X_train.shape[0]

# choose a random patient encounter to plot
ix = random.randint(0,numencnt-1)

# plot a matrix of observation values
plt.pcolor(np.transpose(X_train[ix,:,:]))
plt.ylabel("variable")
plt.xlabel("time/epoch")
plt.ylim(0,265)
plt.colorbar()
plt.show()
```



LAB REVIEW

3. Architect LSTM network using Keras and Theano
4. Build the model (feed data into network for training)
5. Evaluate model using validation (test) data
6. Visualize results
7. Compare baseline to PRISM3 and PIM2

LAB REVIEW - ARCHITECT LSTM #3

In [22]:
```
from keras.layers import LSTM, Dense, Input, TimeDistributed, Masking
from keras.models import Model
from keras.optimizers import RMSprop

# Note: building model using Keras Functional API (version > 1.0)

# construct inputs
x = Input((None, X_train.shape[-1]) , name='input')
mask = Masking(0, name='input_masked')(x)

# stack LSTMs
lstm_kwargs = {'dropout_W': 0.25, 'dropout_U': 0.1, 'return_sequences': True, 'consume_less': 'gpu'}
lstm1 = LSTM(128, name='lstm1', **lstm_kwargs)(mask)

# output: sigmoid layer
output = TimeDistributed(Dense(1, activation='sigmoid'), name='output')(lstm1)
model = Model(input=x, output=output)

# compile model
optimizer = RMSprop(lr=0.005)
model.compile(optimizer=optimizer, loss='binary_crossentropy')

# print layer shapes and model parameters
model.summary()
```

LAB REVIEW - ARCHITECT LSTM #3

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              (None, None, 265)    0
__________________________________________________________________________________________________
input_masked (Masking)          (None, None, 265)    0           input[0][0]
__________________________________________________________________________________________________
lstm1 (LSTM)                    (None, None, 128)    201728      input_masked[0][0]
__________________________________________________________________________________________________
output (TimeDistributed)        (None, None, 1)      129         lstm1[0][0]
==================================================================================================
Total params: 201857
__________________________________________________________________________________________________
```

LAB REVIEW – BUILD / TRAIN MODEL #4

In [23]:
```
# this will take a while...
history = model.fit(X_train, y_train, batch_size=128, nb_epoch=5, verbose=1)
```

```
Epoch 1/5
5000/5000 [==============================] - 23s - loss: 0.2631
Epoch 2/5
5000/5000 [==============================] - 24s - loss: 0.2200
Epoch 3/5
5000/5000 [==============================] - 24s - loss: 0.2017
Epoch 4/5
5000/5000 [==============================] - 23s - loss: 0.1974
Epoch 5/5
5000/5000 [==============================] - 24s - loss: 0.1873
```

LAB REVIEW – EVALUATE MODEL #5

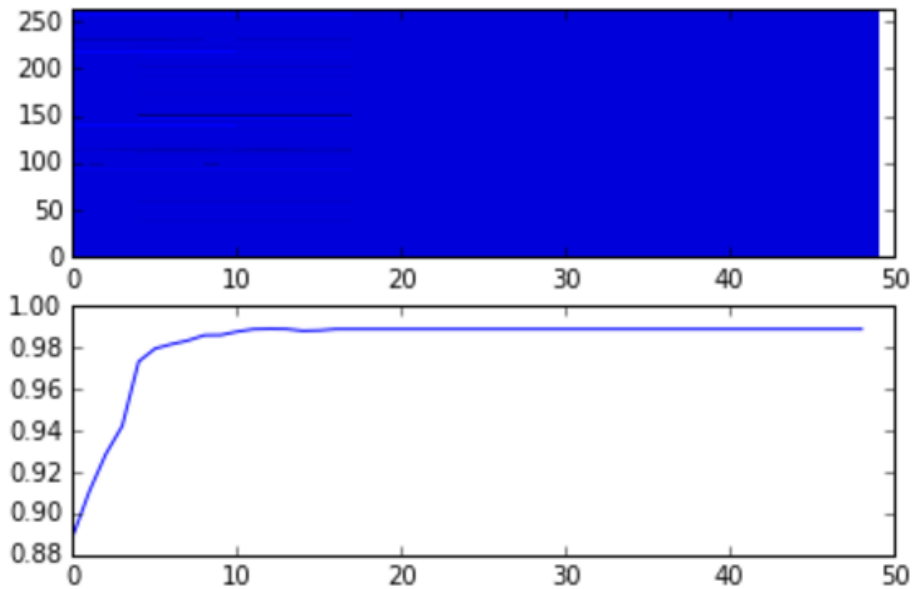
```
In [24]: # generate RNN results on holdout validation set
         preds = model.predict(X_valid)
```

```
In [25]: preds.shape

Out[25]: (2690, 500, 1)
```

That is, we have 2690 patient encounters for testing, and at each of the observations the model predicts survivability. Lets plot some predictions!

LAB REVIEW - VISUALIZE RESULTS #6

In [26]:
```
# figure out how many encounters we have
numencnt = X_valid.shape[0]

# choose a random patient encounter to plot
ix = random.randint(0,numencnt-1)

# create axis side by side
f, (ax1, ax2) = plt.subplots(2, 1)

# plot the obs chart for patient encounter
ax1.pcolor(np.transpose(X_valid[ix,1:50,:]))
ax1.set_ylim(0,265)

# plot the patient survivability prediction
ax2.plot(preds[ix,1:50]);
```



LAB REVIEW - COMPARE BASELINE #7

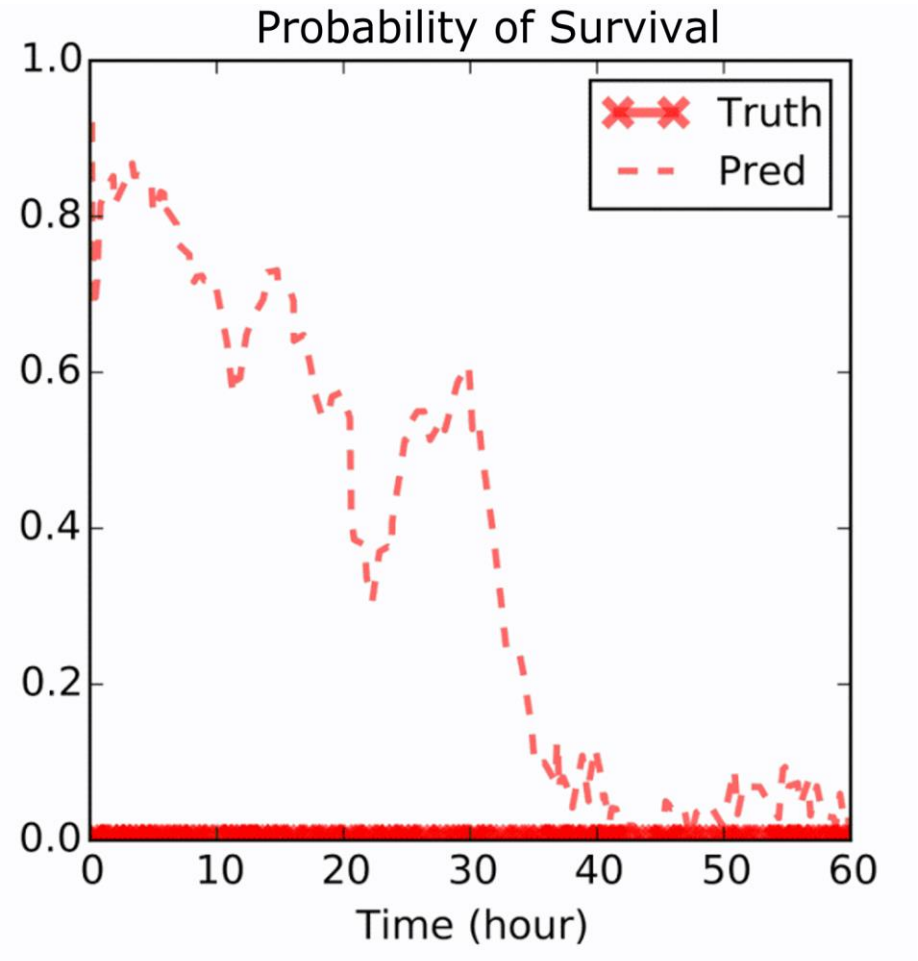
In [27]:
```
from sklearn.metrics import roc_curve, auc

# get 0/1 binary lable for each patient encounter
label = y_valid[:, 0, :].squeeze();

# get the last prediction in [0,1] for the patient
prediction = preds[:, -1, :].squeeze()

# compute ROC curve for predictions
rnn_roc = roc_curve(label,prediction)

# compute the area under the curve of prediction ROC
rnn_auc = auc(rnn_roc[0], rnn_roc[1])
```

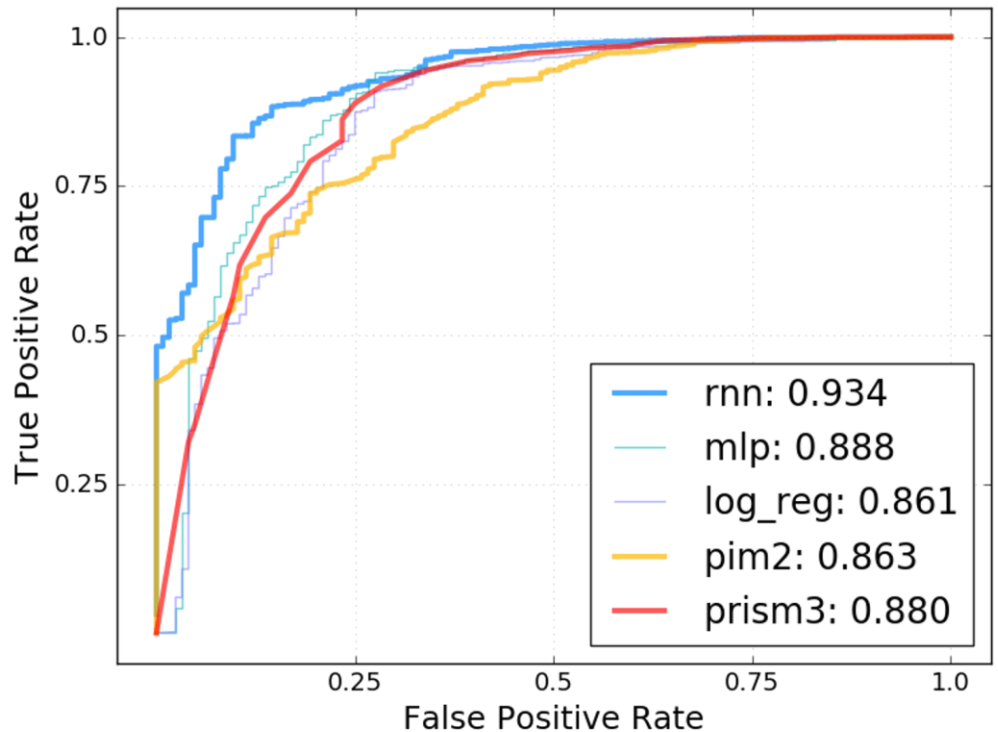
In [28]:
```
# scores for baselines PRISM3 and PIM2 were aggregated and stored in `data/pim2prism3.csv`.
# load the scores and then compute the ROC curves and AUC
index = pd.read_csv(os.path.join(data_dir, 'pim2prism3.csv'))

# get the mortality reponse for each patient
mortrep = index['mortalityResponse'];

# generage ROC curves for each index
pim2_roc   = roc_curve(mortrep, -index['PIM2'  ])
prism3_roc = roc_curve(mortrep, -index['PRISM3'])

# compute the area under the curve for each index
pim2_auc   = auc(  pim2_roc[0],   pim2_roc[1])
prism3_auc = auc(prism3_roc[0], prism3_roc[1])
```

LAB REVIEW - COMPARE BASELINE #7

In [29]:
```
# plot rocs & display AUCs
plt.figure(figsize=(7, 5))
line_kwargs = {'linewidth': 4, 'alpha': 0.8}
plt.plot(prism3_roc[0], prism3_roc[1], label='prism3: %0.3f' % prism3_auc, color='#4A86E8', **line_kwargs)
plt.plot(pim2_roc[0], pim2_roc[1], label='pim2: %0.3f' % pim2_auc, color='#FF9900', **line_kwargs)
plt.plot(rnn_roc[0], rnn_roc[1], label='rnn: %0.3f' % rnn_auc, color='#6AA84F', **line_kwargs)
plt.legend(loc='lower right', fontsize=20)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.xticks([0, 0.25, 0.5, 0.75, 1.0], fontsize=14)
plt.yticks([0, 0.25, 0.5, 0.75, 1.0], fontsize=14)
plt.xlabel("False Positive Rate", fontsize=18)
plt.ylabel("True Positive Rate", fontsize=18)
plt.title("Severity of Illness ROC Curves", fontsize=24)
plt.grid(alpha=0.25)
plt.tight_layout()
```

LAB REVIEW - COMPARE BASELINE #7





WHAT ELSE?

- Many ways to explore and improve model:
  - Add a second and third LSTM layer to the network
  - Change the number of layers and the number of neurons in those layers
  - Change some of the meta parameters in the network configuration like dropout or learning rate, etc.
  - Try using a CNN? Does it outperform the RNN / LSTM model?

WHAT’S NEXT

WHAT'S NEXT

- Use / practice what you learned
- Discuss with peers practical applications of DNN
- Reach out to NVIDIA and the Deep Learning Institute
- Attend local meetup groups
- Follow people like Andrej Karpathy and Andrew Ng

WHAT'S NEXT

TAKE SURVEY

…for the chance to win an NVIDIA SHIELD TV.

Check your email for a link.

ACCESS ONLINE LABS

Check your email for details to access more DLI training online.

ATTEND WORKSHOP

Visit www.nvidia.com/dli for workshops in your area.

JOIN DEVELOPER PROGRAM

Visit https://developer.nvidia.com/join for more.

DEEP
LEARNING
INSTITUTE

Children's Hospital LOS ANGELES

We Treat Kids Better

www.nvidia.com/dli