Time series prediction using RNNs, with TensorFlow and Cloud ML Engine

This notebook illustrates:

1. Creating a Recurrent Neural Network in TensorFlow
2. Creating a Custom Estimator in tf.estimator
3. Training on Cloud ML Engine

Simulate some time-series data

Essentially a set of sinusoids with random amplitudes and frequencies.

In [23]:

```
import os
PROJECT = 'cloud-training-demos' # REPLACE WITH YOUR PROJECT ID
BUCKET = 'cloud-training-demos-ml' # REPLACE WITH YOUR BUCKET NAME
REGION = 'us-central1' # REPLACE WITH YOUR BUCKET REGION e.g. us-central1
os.environ['TFVERSION'] = '1.8'  # Tensorflow version
```

In [24]:

```
# for bash
os.environ['PROJECT'] = PROJECT
os.environ['BUCKET'] = BUCKET
os.environ['REGION'] = REGION
```

In [25]:

```
%%bash
gcloud config set project $PROJECT
gcloud config set compute/region $REGION
```

```
Updated property [core/project].
Updated property [compute/region].
```

In [26]:

```
import tensorflow as tf
print(tf.__version__)
```

```
1.8.0
```

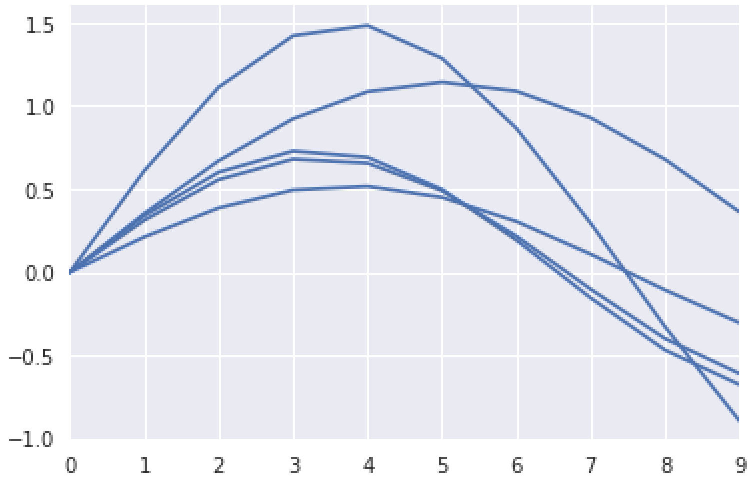
In [27]:

```
import numpy as np
import seaborn as sns
import pandas as pd

SEQ_LEN = 10
def create_time_series():
  freq = (np.random.random() * 0.5) + 0.1  # 0.1 to 0.6
  ampl = np.random.random() + 0.5  # 0.5 to 1.5
  x = np.sin(np.arange(0, SEQ_LEN) * freq) * ampl
  return x

for i in range(0, 5):
  sns.tsplot( create_time_series() );  # 5 series
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/matplotlib/font_m
anager.py:1320: UserWarning: findfont: Font family ['sans-serif'] no
t found. Falling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```



In [28]:

```
def to_csv(filename, N):
  with open(filename, 'w') as ofp:
    for lineno in range(0, N):
      seq = create_time_series()
      line = ",".join(map(str, seq))
      ofp.write(line + '\n')

to_csv('train.csv', 1000)  # 1000 sequences
to_csv('valid.csv',  50)
```

In [29]:

```
!head -5 train.csv valid.csv
```

```
==> train.csv <==
0.0,0.41494536271196236,0.734451551569238,0.8850308030101335,0.83204
92576193983,0.5876928930606661,0.20816469810142113,-0.21924225053918
68,-0.5962225619744843,-0.836069183382574
0.0,0.16050934484878304,0.3187737388366111,0.4725796299246527,0.6197
758245612592,0.7583035751751281,0.8862253746803034,1.001752055261012
6,1.1032678124214037,1.1893528043033446
0.0,0.2019768145665784,0.3895471390163l743,0.5493320613627289,0.6699
345313575295,0.7427522829645161,0.7625914120164823,0.728036844128641
7,0.6415532683388727,0.509309337136115
0.0,0.20513560202370504,0.38685799767444984,0.5244262531017174,0.602
1389789663988,0.6111264126796244,0.5503627711095774,0.42678332861145
07,0.2544928576485325,0.05315577684418399
0.0,0.20112443738388527,0.3910248758298022,0.559103685595879,0.69598
10200217685,0.794018269517251,0.8477443436542725,0.8541609921610173,
0.812910126000857 4,0.726293800975636

==> valid.csv <==
0.0,0.4518834761927216,0.8463432780938488,1.1332529187241658,1.27615
29862603307,1.256884270714006,1.077895368911241,0.7619315258995626,
0.3491442536614218,-0.10801097394423191
0.0,0.4790437748864075,0.8201581888176755,0.9251273884779285,0.76372
79675979914,0.38243108423806044,-0.108977813563 2325,-0.5690091192298
001,-0.8652076640769141,-0.9122900603467594
0.0,0.3825159065598339,0.6359357547959016,0.6747323732958791,0.48581
2213690283,0.13293431847834153,-0.2648078457410006,-0.57317948049446
85,-0.6881076260362654,-0.5708049526750818
0.0,0.09017440871015896,0.178585410505 6497,0.26350408280046406,0.343
2697973071416,0.4163226944765429,0.481234187353148,0.536734898327307
5,0.5817394824668976,0.6153678519953671
0.0,0.7084840320523366,1.200134233070997,1.3244794677975964,1.043463
4746979885,0.44309210797471377,-0.29288903247151693,-0.9392305365880
832,-1.2981174951417012,-1.2597113929215498
```

RNN

For more info, see:

1. http://colah.github.io/posts/2015-08-Understanding-LSTMs/ for the theory
2. https://www.tensorflow.org/tutorials/recurrent for explanations
3. https://github.com/tensorflow/models/tree/master/tutorials/rnn/ptb for sample code

Here, we are trying to predict from 9 values of a timeseries, the tenth value.

Imports

Several tensorflow packages and shutil

In [30]:

```
import tensorflow as tf
import shutil
import tensorflow.contrib.metrics as metrics
import tensorflow.contrib.rnn as rnn
```

Input Fn to read CSV

Our CSV file structure is quite simple -- a bunch of floating point numbers (note the type of DEFAULTS). We ask for the data to be read BATCH_SIZE sequences at a time. The Estimator API in tf.contrib.learn wants the features returned as a dict. We'll just call this timeseries column 'rawdata'.

Our CSV file sequences consist of 10 numbers. We'll assume that 9 of them are inputs and we need to predict the last one.

In [31]:

```
DEFAULTS = [[0.0] for x in range(0, SEQ_LEN)]
BATCH_SIZE = 20
TIMESERIES_COL = 'rawdata'
# In each sequence, column index 0 to N_INPUTS - 1 are features, and column inde
x N_INPUTS to SEQ_LEN are labels
N_OUTPUTS = 1
N_INPUTS = SEQ_LEN - N_OUTPUTS
```

Reading data using the Estimator API in tf.estimator requires an input_fn. This input_fn needs to return a dict of features and the corresponding labels.

So, we read the CSV file. The Tensor format here will be a scalar -- entire line. We then decode the CSV. At this point, all_data will contain a list of scalar Tensors. There will be SEQ_LEN of these tensors.

We split this list of SEQ_LEN tensors into a list of N_INPUTS Tensors and a list of N_OUTPUTS Tensors. We stack them along the first dimension to then get a vector Tensor for each. We then put the inputs into a dict and call it features. The other is the ground truth, so labels.

In [32]:

```
# Read data and convert to needed format
def read_dataset(filename, mode, batch_size = 512):
  def _input_fn():
    # Provide the ability to decode a CSV
    def decode_csv(line):
      # all_data is a list of scalar tensors
      all_data = tf.decode_csv(line, record_defaults = DEFAULTS)
      inputs = all_data[:len(all_data) - N_OUTPUTS]  # first N_INPUTS values
      labels = all_data[len(all_data) - N_OUTPUTS:] # last N_OUTPUTS values

      # Convert each list of rank R tensors to one rank R+1 tensor
      inputs = tf.stack(inputs, axis = 0)
      labels = tf.stack(labels, axis = 0)

      # Convert input R+1 tensor into a feature dictionary of one R+1 tensor
      features = {TIMESERIES_COL: inputs}

      return features, labels

    # Create list of files that match pattern
    file_list = tf.gfile.Glob(filename)

    # Create dataset from file list
    dataset = tf.data.TextLineDataset(file_list).map(decode_csv)

    if mode == tf.estimator.ModeKeys.TRAIN:
        num_epochs = None # indefinitely
        dataset = dataset.shuffle(buffer_size = 10 * batch_size)
    else:
        num_epochs = 1 # end-of-input after this

    dataset = dataset.repeat(num_epochs).batch(batch_size)

    iterator = dataset.make_one_shot_iterator()
    batch_features, batch_labels = iterator.get_next()
    return batch_features, batch_labels
  return _input_fn
```

Define RNN

A recursive neural network consists of possibly stacked LSTM cells.

The RNN has one output per input, so it will have 8 output cells. We use only the last output cell, but rather use it directly, we do a matrix multiplication of that cell by a set of weights to get the actual predictions. This allows for a degree of scaling between inputs and predictions if necessary (we don't really need it in this problem).

Finally, to supply a model function to the Estimator API, you need to return a EstimatorSpec. The rest of the function creates the necessary objects.

In [33]:

```
LSTM_SIZE = 3  # number of hidden layers in each of the LSTM cells

# Create the inference model
def simple_rnn(features, labels, mode):
  # 0. Reformat input shape to become a sequence
  x = tf.split(features[TIMESERIES_COL], N_INPUTS, 1)

  # 1. Configure the RNN
  lstm_cell = rnn.BasicLSTMCell(LSTM_SIZE, forget_bias = 1.0)
  outputs, _ = rnn.static_rnn(lstm_cell, x, dtype = tf.float32)

  # Slice to keep only the last cell of the RNN
  outputs = outputs[-1]

  # Output is result of linear activation of last layer of RNN
  weight = tf.get_variable("weight", initializer=tf.initializers.random_normal,
shape=[LSTM_SIZE, N_OUTPUTS])
  bias = tf.get_variable("bias", initializer=tf.initializers.random_normal, shap
e=[N_OUTPUTS])
  predictions = tf.matmul(outputs, weight) + bias

  # 2. Loss function, training/eval ops
  if mode == tf.estimator.ModeKeys.TRAIN or mode == tf.estimator.ModeKeys.EVAL:
    loss = tf.losses.mean_squared_error(labels, predictions)
    train_op = tf.contrib.layers.optimize_loss(
      loss = loss,
      global_step = tf.train.get_global_step(),
      learning_rate = 0.01,
      optimizer = "SGD")
    eval_metric_ops = {
      "rmse": tf.metrics.root_mean_squared_error(labels, predictions)
    }
  else:
    loss = None
    train_op = None
    eval_metric_ops = None

  # 3. Create predictions
  predictions_dict = {"predicted": predictions}

  # 4. Create export outputs
  export_outputs = {"predict_export_outputs": tf.estimator.export.PredictOutput(
outputs = predictions)}

  # 5. Return EstimatorSpec
  return tf.estimator.EstimatorSpec(
      mode = mode,
      predictions = predictions_dict,
      loss = loss,
      train_op = train_op,
      eval_metric_ops = eval_metric_ops,
      export_outputs = export_outputs)
```

Estimator

Distributed training is launched off using an Estimator. The key line here is that we use tf.estimator.Estimator rather than, say tf.estimator.DNNRegressor. This allows us to provide a model_fn, which will be our RNN defined above. Note also that we specify a serving_input_fn -- this is how we parse the input data provided to us at prediction time.

In [34]:

```
# Create functions to read in respective datasets
def get_train():
  return read_dataset(filename = 'train.csv', mode = tf.estimator.ModeKeys.TRAIN
, batch_size = 512)

def get_valid():
  return read_dataset(filename = 'valid.csv', mode = tf.estimator.ModeKeys.EVAL,
batch_size = 512)
```

In [35]:

```
# Create serving input function
def serving_input_fn():
  feature_placeholders = {
      TIMESERIES_COL: tf.placeholder(tf.float32, [None, N_INPUTS])
  }

  features = {
    key: tf.expand_dims(tensor, -1)
    for key, tensor in feature_placeholders.items()
  }
  features[TIMESERIES_COL] = tf.squeeze(features[TIMESERIES_COL], axis = [2])

  return tf.estimator.export.ServingInputReceiver(features, feature_placeholders
)
```

In [36]:

```
# Create custom estimator's train and evaluate function
def train_and_evaluate(output_dir):
  estimator = tf.estimator.Estimator(model_fn = simple_rnn,
                       model_dir = output_dir)
  train_spec = tf.estimator.TrainSpec(input_fn = get_train(),
                                max_steps = 1000)
  exporter = tf.estimator.LatestExporter('exporter', serving_input_fn)
  eval_spec = tf.estimator.EvalSpec(input_fn = get_valid(),
                              steps = None,
                              exporters = exporter)
  tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

In [37]:

```
# Run the model
shutil.rmtree('outputdir', ignore_errors = True) # start fresh each time
train_and_evaluate('outputdir')
```

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'outputdir', '_servic
e': None, '_keep_checkpoint_every_n_hours': 10000, '_task_type': 'wo
rker', '_num_worker_replicas': 1, '_keep_checkpoint_max': 5, '_num_p
s_replicas': 0, '_global_id_in_cluster': 0, '_log_step_count_steps':
100, '_save_checkpoints_secs': 600, '_evaluation_master': '', '_trai
n_distribute': None, '_save_summary_steps': 100, '_tf_random_seed':
None, '_cluster_spec': <tensorflow.python.training.server_lib.Cluste
rSpec object at 0x7f65a2bc30b8>, '_master': '', '_task_id': 0, '_ses
sion_config': None, '_save_checkpoints_steps': None, '_is_chief': Tr
ue}
INFO:tensorflow:Running training and evaluation locally (non-distrib
uted).
INFO:tensorflow:Start train and evaluate loop. The evaluate will hap
pen after 600 secs (eval_spec.throttle_secs) or training is finishe
d.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into outputdir/model.ckpt.
INFO:tensorflow:step = 1, loss = 2.1511927
INFO:tensorflow:global_step/sec: 13.8533
INFO:tensorflow:step = 101, loss = 0.5264278 (7.220 sec)
INFO:tensorflow:global_step/sec: 14.175
INFO:tensorflow:step = 201, loss = 0.42215365 (7.055 sec)
INFO:tensorflow:global_step/sec: 14.2523
INFO:tensorflow:step = 301, loss = 0.34791386 (7.017 sec)
INFO:tensorflow:global_step/sec: 14.7247
INFO:tensorflow:step = 401, loss = 0.26609486 (6.791 sec)
INFO:tensorflow:global_step/sec: 14.6274
INFO:tensorflow:step = 501, loss = 0.21945082 (6.836 sec)
INFO:tensorflow:global_step/sec: 14.4637
INFO:tensorflow:step = 601, loss = 0.1646782 (6.914 sec)
INFO:tensorflow:global_step/sec: 14.5217
INFO:tensorflow:step = 701, loss = 0.13758004 (6.887 sec)
INFO:tensorflow:global_step/sec: 14.123
INFO:tensorflow:step = 801, loss = 0.1219064 (7.081 sec)
INFO:tensorflow:global_step/sec: 15.7489
INFO:tensorflow:step = 901, loss = 0.10583858 (6.350 sec)
INFO:tensorflow:Saving checkpoints for 1000 into outputdir/model.ckp
t.
INFO:tensorflow:Loss for final step: 0.085023135.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-09-12-20:01:59
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from outputdir/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-09-12-20:01:59
INFO:tensorflow:Saving dict for global step 1000: global_step = 100
0, loss = 0.069563285, rmse = 0.26374853
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['serving
_default', 'predict_export_outputs']
```

```
INFO:tensorflow:Restoring parameters from outputdir/model.ckpt-1000
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: b"outputdir/export/exporter/t
emp-b'1536782520'/saved_model.pb"
```

Standalone Python module

To train this on Cloud ML Engine, we take the code in this notebook and make a standalone Python module.

In [38]:

```
%%bash
# Run module as-is
echo $PWD
rm -rf outputdir
export PYTHONPATH=${PYTHONPATH}:${PWD}/simplernn
python -m trainer.task \
  --train_data_paths="${PWD}/train.csv*" \
  --eval_data_paths="${PWD}/valid.csv*"  \
  --output_dir=outputdir \
  --job-dir=./tmp
```

```
/content/datalab/training-data-analyst/courses/machine_learning/deep
dive/05_artandscience
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/h5py/__init__.py:
36: FutureWarning: Conversion of the second argument of issubdtype f
rom `float` to `np.floating` is deprecated. In future, it will be tr
eated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'outputdir/', '_task_ty
pe': 'worker', '_service': None, '_num_ps_replicas': 0, '_tf_random_
seed': None, '_task_id': 0, '_num_worker_replicas': 1, '_master':
'', '_is_chief': True, '_keep_checkpoint_max': 5, '_evaluation_maste
r': '', '_log_step_count_steps': 100, '_global_id_in_cluster': 0, '_
train_distribute': None, '_save_checkpoints_steps': None, '_cluster_
spec': <tensorflow.python.training.server_lib.ClusterSpec object at
0x7fa28bd35780>, '_session_config': None, '_keep_checkpoint_every_n_
hours': 10000, '_save_checkpoints_secs': 600, '_save_summary_steps':
100}
INFO:tensorflow:Running training and evaluation locally (non-distrib
uted).
INFO:tensorflow:Start train and evaluate loop. The evaluate will hap
pen after 600 secs (eval_spec.throttle_secs) or training is finishe
d.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
2018-09-12 20:02:07.143754: I tensorflow/core/platform/cpu_feature_g
uard.cc:140] Your CPU supports instructions that this TensorFlow bin
ary was not compiled to use: AVX2 FMA
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into outputdir/model.ckpt.
INFO:tensorflow:step = 1, loss = 0.86221
INFO:tensorflow:global_step/sec: 15.0388
INFO:tensorflow:step = 101, loss = 0.30457026 (6.650 sec)
INFO:tensorflow:global_step/sec: 16.4018
INFO:tensorflow:step = 201, loss = 0.19842917 (6.097 sec)
INFO:tensorflow:global_step/sec: 19.017
INFO:tensorflow:step = 301, loss = 0.15593535 (5.258 sec)
INFO:tensorflow:global_step/sec: 19.325
INFO:tensorflow:step = 401, loss = 0.1427422 (5.175 sec)
INFO:tensorflow:global_step/sec: 19.4141
INFO:tensorflow:step = 501, loss = 0.11589954 (5.151 sec)
INFO:tensorflow:global_step/sec: 19.1968
INFO:tensorflow:step = 601, loss = 0.11280591 (5.209 sec)
INFO:tensorflow:global_step/sec: 19.0675
INFO:tensorflow:step = 701, loss = 0.099279635 (5.244 sec)
INFO:tensorflow:global_step/sec: 18.7077
INFO:tensorflow:step = 801, loss = 0.08796086 (5.345 sec)
INFO:tensorflow:global_step/sec: 19.2725
INFO:tensorflow:step = 901, loss = 0.078957975 (5.189 sec)
INFO:tensorflow:Saving checkpoints for 1000 into outputdir/model.ckp
t.
INFO:tensorflow:Loss for final step: 0.07356742.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-09-12-20:03:03
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from outputdir/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-09-12-20:03:04
```

```
INFO:tensorflow:Saving dict for global step 1000: global_step = 100
0, loss = 0.06301524, rmse = 0.25102836
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['predict
_export_outputs', 'serving_default']
INFO:tensorflow:Restoring parameters from outputdir/model.ckpt-1000
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: b"outputdir/export/exporter/t
emp-b'1536782584'/saved_model.pb"
```

Try out online prediction. This is how the REST API will work after you train on Cloud ML Engine

In [39]:

```
%%writefile test.json
{"rawdata_input": [0,0.214,0.406,0.558,0.655,0.687,0.65,0.549,0.393]}
```

```
Overwriting test.json
```

In [40]:

```
# local predict doesn't work with Python 3 yet.
# %%bash
# MODEL_DIR=$(ls ./outputdir/export/exporter/)
# gcloud ml-engine local predict --model-dir=./outputdir/export/exporter/$MODEL_
DIR --json-instances=test.json
```

Cloud ML Engine

Now to train on Cloud ML Engine.

In [41]:

```
%%bash
# Run module on Cloud ML Engine
OUTDIR=gs://${BUCKET}/simplernn/model_trained
JOBNAME=simplernn_$(date -u +%y%m%d_%H%M%S)
gsutil -m rm -rf $OUTDIR
gcloud ml-engine jobs submit training $JOBNAME \
   --region=$REGION \
   --module-name=trainer.task \
   --package-path=${PWD}/simplernn/trainer \
   --job-dir=$OUTDIR \
   --staging-bucket=gs://$BUCKET \
   --scale-tier=BASIC \
   --runtime-version=1.4 \
   -- \
   --train_data_paths="gs://${BUCKET}/train.csv*" \
   --eval_data_paths="gs://${BUCKET}/valid.csv*"  \
   --output_dir=$OUTDIR
```

```
jobId: simplernn_180912_200305
state: QUEUED

CommandException: 1 files/objects could not be removed.
Job [simplernn_180912_200305] submitted successfully.
Your job is still active. You may view the status of your job with t
he command

  $ gcloud ml-engine jobs describe simplernn_180912_200305

or continue streaming the logs with the command

  $ gcloud ml-engine jobs stream-logs simplernn_180912_200305
```

Variant: long sequence

To create short sequences from a very long sequence.

In [42]:

```
import tensorflow as tf
import numpy as np

def breakup(sess, x, lookback_len):
  N = sess.run(tf.size(x))
  windows = [tf.slice(x, [b], [lookback_len]) for b in range(0, N-lookback_len)]
  windows = tf.stack(windows)
  return windows

x = tf.constant(np.arange(1,11, dtype=np.float32))
with tf.Session() as sess:
    print('input=', x.eval())
    seqx = breakup(sess, x, 5)
    print('output=', seqx.eval())
```

```
input= [ 1.   2.   3.   4.   5.   6.   7.   8.   9. 10.]
output= [[1. 2. 3. 4. 5.]
 [2. 3. 4. 5. 6.]
 [3. 4. 5. 6. 7.]
 [4. 5. 6. 7. 8.]
 [5. 6. 7. 8. 9.]]
```

Variant: Keras

You can also invoke a Keras model from within the Estimator framework by creating an estimator from the compiled Keras model:

In [43]:

```
def make_keras_estimator(output_dir):
  from tensorflow import keras
  model = keras.models.Sequential()
  model.add(keras.layers.Dense(32, input_shape=(N_INPUTS,), name=TIMESERIES_INPU
T_LAYER))
  model.add(keras.layers.Activation('relu'))
  model.add(keras.layers.Dense(1))
  model.compile(loss = 'mean_squared_error',
                optimizer = 'adam',
                metrics = ['mae', 'mape']) # mean absolute [percentage] error
  return keras.estimator.model_to_estimator(model)
```

In [ ]:

```
%%bash
# Run module as-is
echo $PWD
rm -rf outputdir
export PYTHONPATH=${PYTHONPATH}:${PWD}/simplernn
python -m trainer.task \
  --train_data_paths="${PWD}/train.csv*" \
  --eval_data_paths="${PWD}/valid.csv*"  \
  --output_dir=${PWD}/outputdir \
  --job-dir=./tmp --keras
```