RNN

Trong bài này chúng ta sẽ làm quen với RNN và làm việc với Estimator trên Tensorflow.

Sinh viên sinh viên phải cập nhật Tensorflow lên bản mới nhất 1.4 để thực thi được một số đoạn code mẫu.

Bài tập được biên soạn từ có tham khảo ở các nguồn sau đây có chỉnh sửa:

aymericdamien (https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/recurrent_network.py)

tensorflow example (https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/learn/mnist.py)

Dữ liệu dữ liệu được sử dụng trong bài tập này là dữ liệu MNIST đã quen thuộc ở các bài trước.

In [ ]:
```
from __future__ import print_function

import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
```

Các tham số cần thiết

In [ ]:
```
# Training Parameters
params = {
    'learning_rate': 0.001,
    'training_steps': 20000,
    'batch_size': 128,
    'display_step': 100,

    # Network Parameters
    'num_input': 28,  # MNIST data input (img shape: 28*28)
    'timesteps': 28,  # timesteps
    'num_hidden': 128,  # hidden layer num of features
    'num_classes': 10  # MNIST total classes (0-9 digits)
}
```

Ở đây, chúng ta làm quen với Estimator "High level tools for working with models".

Chúng ta sẽ tách rời estimator spec và model, với mỗi bài toán xác định thì spec gần như không đổi, do đó chúng ta có thể **tái sử dụng cho nhiều model khác nhau**.

Spec này được sử dụng cho bài toán MNIST ở bên dưới, tùy theo mode (training, evaluation, predict) mà một số tham số sẽ bị bỏ qua.

Chúng ta có thể tận dụng nó ở các bước training, evaluation và test.

In [ ]:
```
def estimator_spec(logits, labels, mode, params):
    loss, train_op, predictions = None, None, None
    if mode == tf.estimator.ModeKeys.TRAIN or mode == tf.estimator.ModeKeys.EVAL:
        onehot_labels = tf.one_hot(labels, params['num_classes'], 1, 0)
        loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits
, labels=onehot_labels))

    if mode == tf.estimator.ModeKeys.TRAIN:
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=params['learnin
g_rate'])
        train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())

    if mode == tf.estimator.ModeKeys.PREDICT:
        predictions = {'prediction': tf.argmax(logits, 1),
                       'prob': tf.nn.softmax(logits)}
    eval_metric_ops = {'accuracy': tf.metrics.accuracy(labels=labels, predictions=t
f.argmax(logits, 1))}
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=predictions,
        loss=loss,
        train_op=train_op,
        eval_metric_ops=eval_metric_ops)
```

Phía phía dưới là model, chúng ta định nghĩa một hàm 1 model_fn để chuyển vào estimator sẽ gọi lại sau.

Hàm này nhận các tham số theo đặc tả **chú ý thứ tự** tham số:

- *features* sẽ nhớ các thông tin về input.
- *labels* các nhãn (0..9) tương ứng với giá trị đúng của bức hình đó.
- mode tham khảo thêm
- *params* các tham số chúng ta đưa thêm vào theo cấu hình ở trên

Dữ liệu đưa vào đây sẽ ở dạng batch

Hàm sẽ trả về 1 estimator spec như đã được định nghĩa ở phần trước trên.

In [ ]:
```
def model_fn(features, labels, mode, params):
    '''
    To classify images using a recurrent neural network, we consider every image
    row as a sequence of pixels. Because MNIST image shape is 28*28px, we will then
    handle 28 sequences of 28 steps for every sample.
    '''

    x = tf.reshape(features['x'], [-1, params['timesteps'], params['num_input']])
    # Unstack to get a list of 'timesteps' tensors of shape (batch_size, n_input)
    x = tf.unstack(x, params['timesteps'], 1)

    # Define a lstm cell with tensorflow
    lstm_cell = rnn.BasicLSTMCell(params['num_hidden'], forget_bias=1.0)

    # Get lstm cell output
    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)
    # outputs, states = tf.nn.dynamic_rnn(lstm_cell, x, dtype=tf.float32)

    # Linear activation, using rnn inner loop last output
    logits = tf.layers.dense(states[-1], units=params['num_classes'])
    return estimator_spec(logits=logits, labels=labels, mode=mode, params=params)
```

In [ ]:
```
def main(args=None):
    run_config = tf.estimator.RunConfig()
    # run_config = run_config.replace(**{'save_checkpoints_steps': 100, 'keep_check
point_max': 20})
    classifier = tf.estimator.Estimator(model_fn=model_fn,
                                        model_dir='./checkpoint',
                                        params=params,
                                        config=run_config)

    # make data
    mnist = tf.contrib.learn.datasets.DATASETS['mnist']('/tmp/mnist')

    train_input_fn = tf.estimator.inputs.numpy_input_fn(
        x={'x': mnist.train.images},
        y=mnist.train.labels.astype(np.int32),
        batch_size=params['batch_size'],
        num_epochs=None,
        shuffle=True)

    eval_input_fn = tf.estimator.inputs.numpy_input_fn(
        x={'x': mnist.train.images},
        y=mnist.train.labels.astype(np.int32),
        num_epochs=1,
        shuffle=False)

    # classifier.train(input_fn=train_input_fn, steps=params['training_steps'], hoo
ks=[])

    train_spec = tf.estimator.TrainSpec(input_fn=train_input_fn, max_steps=params['
training_steps'])
    eval_spec = tf.estimator.EvalSpec(input_fn=eval_input_fn)
    tf.estimator.train_and_evaluate(classifier, train_spec, eval_spec)
    score = classifier.evaluate(eval_input_fn, steps=1)
    print(score)
```

In [ ]:
```
if __name__ == '__main__':
    main(None)
```

Bài tập

Bài nộp của sinh viên là chính là **file này** sau khi được đổi tên thành **MSSV.E12_RNN.ipynb** và đừng quên ghi thông tin sinh viên vào các ô ở dưới.

Địa chỉ nộp bài: https://www.dropbox.com/request/TsxZLj4KbIvtvjYy3M25 (https://www.dropbox.com/request/TsxZLj4KbIvtvjYy3M25)

Deadline nộp bài: **10:00 thứ 3 tuần tiếp theo**

Quy tắc chấm điểm (áp dụng từ bài này tới các bài sau nếu không có chú thích gì thêm)

- *Điểm bài này sẽ được tổng hợp với điểm chấm trên lớp (nếu có) để ra điểm cuối cùng, do đó sinh viên vắng học trên lớp dù làm bài có toàn vẹn cũng có thể nhận được điểm thấp hơn điểm tối đa*

Thông tin sinh viên:

In [ ]:
```
sid = 'MSSV'
name = 'Tên Bạn'

import tensorflow as tf
import numpy as np

# sinh viên import các thư viện cần thiết ở đây
```

Hãy vào thư mục code hiện tại, bạn sẽ thấy một thư mục có tên là *checkpoint*, dùng **tensorboard** để mở và xem những thông tin summary hữu ích trong quá trình training. Hãy chụp một ảnh và chèn vào đây.

Chèn hình tensorboard

Điều chỉnh tham số phù hợp để evaluate sau mỗi 1000 steps (hiện tại là sau mỗi 600 giây)

Thay vì dùng giá trị learning rate cố định như code mẫu, hãy sử dụng tf.train.exponential_decay để điều chỉnh learning rate theo giá trị khởi đầu mong muốn và giảm phù hợp theo get_global_step (https://www.tensorflow.org/api_docs/python/tf/train/get_global_step)

Thêm learning rate đã điều chỉnh ở trên vào tensorboard summary để dễ dàng theo dõi (điều chỉnh code ở trên)

Một trong những vấn đề của DL là overfitting, để phần nào giải quyết vấn đề đó, người ta thường dùng regularization và/hoặc dropout tham khảo thêm (https://www.quora.com/How-does-the-dropout-method-work-in-deep-learning-And-why-is-it-claimed-to-be-an-effective-trick-to-improve-your-network). Hãy thêm dropout cho RNN cell sử dụng ở trên và l2_regularizer cho densely-connected layer.

Hiện model chúng ta sử dụng BasicLSTMCell, và cho kết quả chưa đủ tốt (so với bài trước các em đã làm dùng CNN), hãy thử một số loại RNN cell khác nhau xem có cải thiện gì không. Tham khảo RNN (https://www.tensorflow.org/versions/master/api_docs/python/tf/contrib/rnn) **Mỗi cách sử dụng, model khác nhau, sinh viên hãy tạo một hàm khác và truyền hàm đó vào ở hàm main, KHÔNG ghi đè model_fn cũ đã có**

Điều chỉnh model theo ý muốn để lấy được kết quả tốt nhất có thể, một số gợi ý có thể tham khảo:

- sử dụng bidirectional recurrent neural network (https://www.tensorflow.org/api_docs/python/tf/nn/bidirectional_dynamic_rnn)
- thay đổi các lớp (nhiều lớp hơn chẳng hạn)

Hãy report (lưu **output** kết quả cuối cùng) em có thể đạt được. *Hai lần chạy khác nhau của cùng một model có thể ra kết quả sai khác nhau đôi chút, tuy nhiên, ở đây, chỉ xem xét kết quả cuối cùng mà các em lưu lại ở output*