Mobvoi 出门问问

Keras Tutorial

Yu Wang
Yale University

Introduction

- Installation:

  - Requirement: numpy, scipy, python (2.7+ or 3.3+)

  - Under Theano: Anaconda—> Theano —>Keras

    1. Go to https://www.continuum.io
       Download Anaconda, install using graphical installer
    2. Terminal: pip install Theano
    3. sudo pip install Keras

  - Under Tensorflow: Anaconda—> Tensorflow —>Keras

Keras

- Why Keras?
  - Easy Implementation, Modularity
  - Support CNN/RNN, and combination of two
  - MIMO training as well
  - Changing backend from Theano to Tensorflow or vice versa

Tensorflow in Keras

- By default, Keras use Theano on Backend.
- From Theano to Tensorflow:
- Find Keras Configuration file： ~/.keras/keras.json：

```
{"epsilon": 1e-07, "floatx": "float32", "backend": “theano”}
↕
{"epsilon": 1e-07, "floatx": "float32", "backend": “tensorflow”}
```

Modules

- Models : Sequential models, Group models
- Layers:
  - Core layers
  - Convolutional layers
  - Recurrent layers
  - Embedding layers
- Objectives
- Optimizers
- Activations

Sequential Model

- Sequential model is a linear stack of models

Initialization:

```
model = Sequential()
```

Add one layer to model:

```
model.add(Dense(32, input_dim=784))
```

```
model.add(Dense(32, input_shape=(784,)))
```

Add an activation layer

```
model.add(Activation('relu')
```

Sequential Model

Compilation:
similar to **session** in Tensorflow, or **theano.function** the Theano

```
model.compile(optimizer=‘rmsprop', loss=‘binary_crossentropy', metrics=['accuracy'])
```

Training:
Using fit command

```
model.fit(data, labels, nb_epoch=10, batch_size=32)
```

Core Layers

- Dense: fully connected layer

```
model.add(Dense(32, input_dim=16))
```

- Activation: Applies activation function to output

```
model.add(Activation('sigmoid'))
```

- Dropout: Applies dropout to the input

```
model.add(Dropout(0.5))
```

- Flattern: flattern the input

```
model.add(Flatten())
```

Convolutional layer(1-3 dimension)

- Convolutional CNN:

```
model.add(Convolution1D(64, 3, input_shape=(10, 32)))
```

- Max-Pooling layer

```
model.add(MaxPooling1D((pool_length=2, stride=None))
```

- Zero padding

```
model.add (ZeroPadding1D(padding=1))
```

Convolutional layer(1-3 dimension)

- Convolutional CNN:

```
model.add(Convolution2D(64, 3, 3, input_shape=(3,256, 256)))
```

- Max-Pooling layer

```
model.add(MaxPooling2D((pool_size=(2,2), stride=None))
```

- Zero padding

```
model.add (ZeroPadding2D(padding=(1,1)))
```

Example: MNIST : 99.25%

Recurrent layer

- Simple RNN

```
model.add(SimpleRNN(32, activation='tanh', dropout_W=0, dropout_U=0))
```

- Long-Short term memory

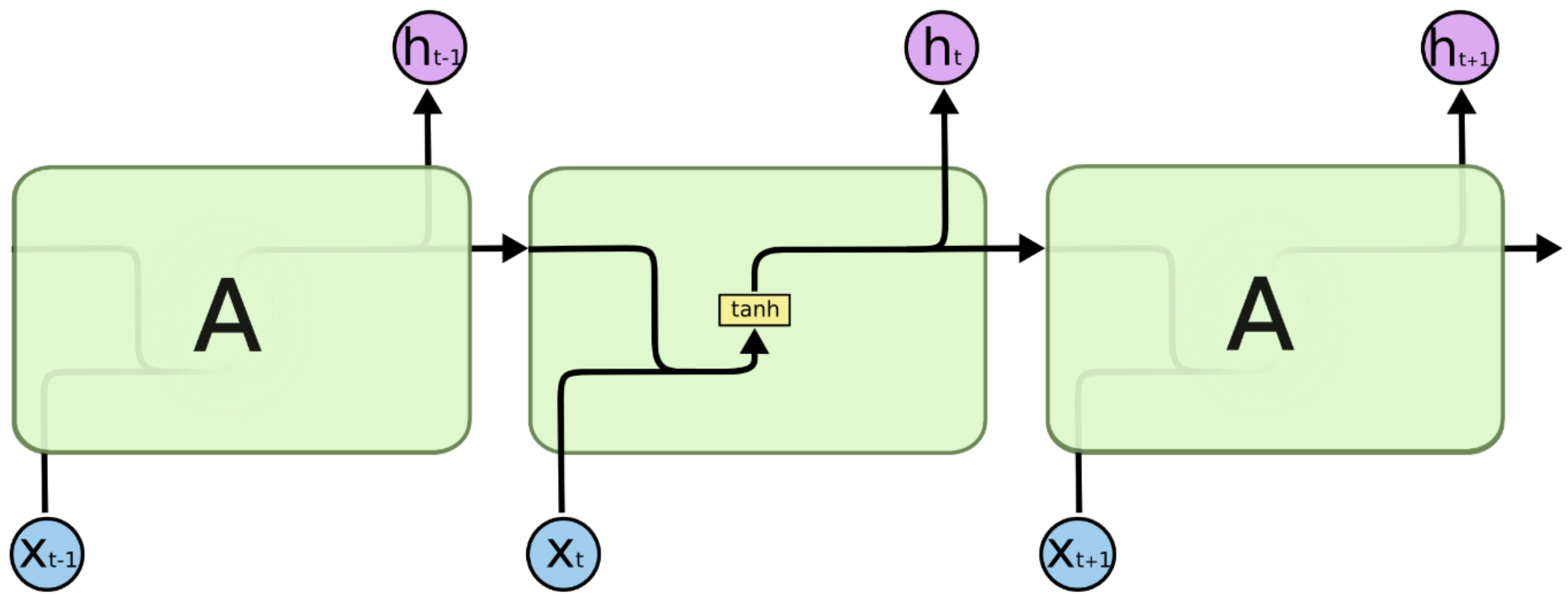
```
model.add(LSTM(32, input_shape=(10, 64),acticvation='tanh'))
```

- GRU

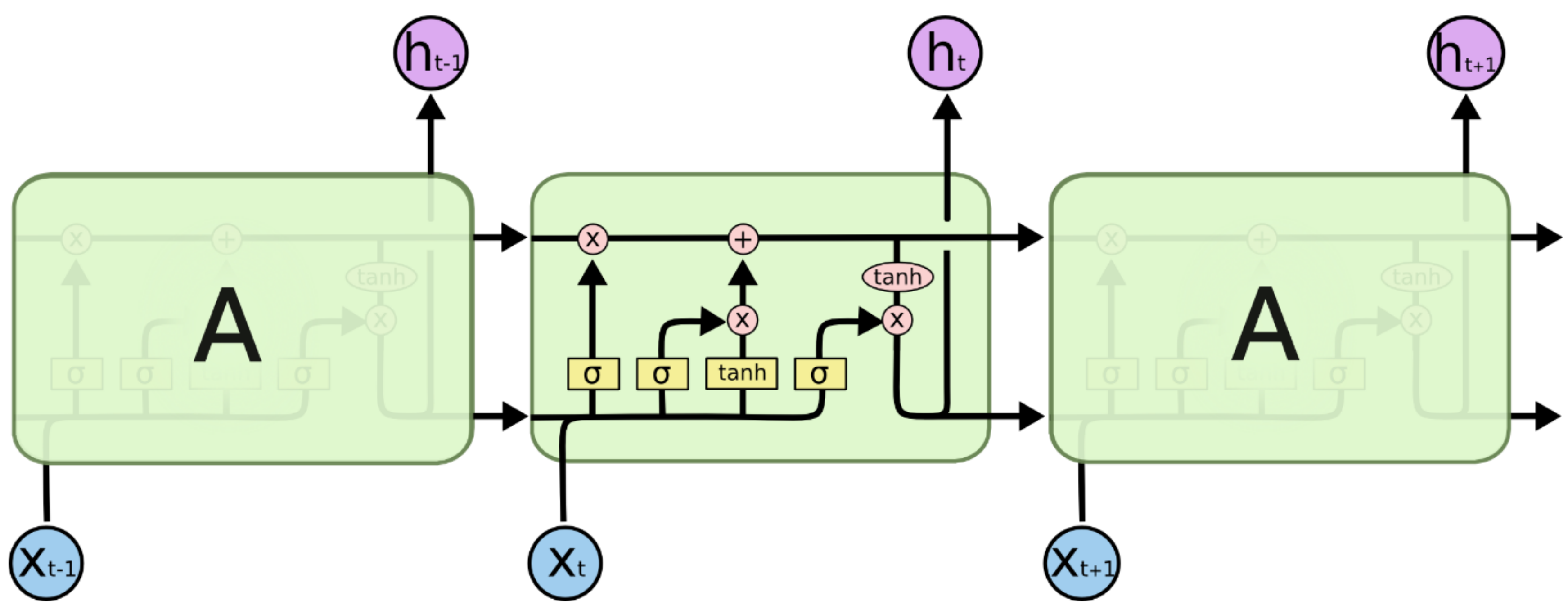
```
model.add(GRU(32, input_shape=(10, 64),acticvation='tanh'))
```

Mobvoi 出门问问

Comparison of RNN and LSTM



The repeating module in a standard RNN contains a single layer.

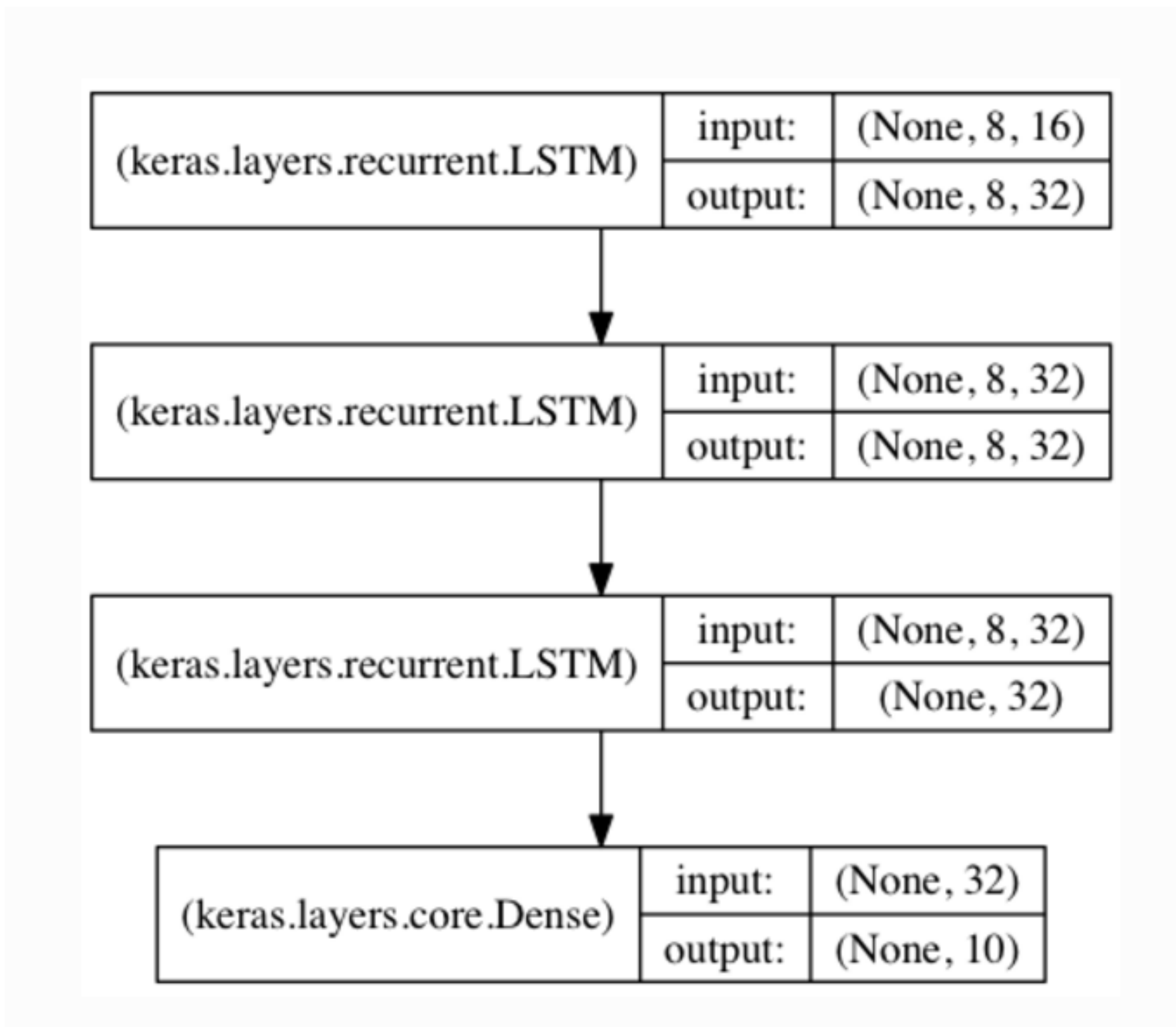


The repeating module in an LSTM contains four interacting layers.

Mobvoi 出门问问

Example- Stacked LSTM

Input dim: 16, LSTM output dim:32, Final output dim:10, time step: 8



Example- Stacked LSTM

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

data_dim = 16
timesteps = 8
nb_classes = 10

model = Sequential()
model.add(LSTM(32, return_sequences=True, input_shape=(timesteps, data_dim)))
model.add(LSTM(32, return_sequences=True))
model.add(LSTM(32))
model.add(Dense(10, activation='softmax'))

model.compile(loss=‘categorical_crossentropy', optimizer=‘rmsprop', metrics=[‘accuracy'])

model.fit(x_train, y_train, batch_size=64, nb_epoch=5, validation_data=(x_val, y_val))
```

Embedding layer

- Idea coming from word2vec: space matching

```
model.add(Embedding(1000, 64, input_length=10))
```

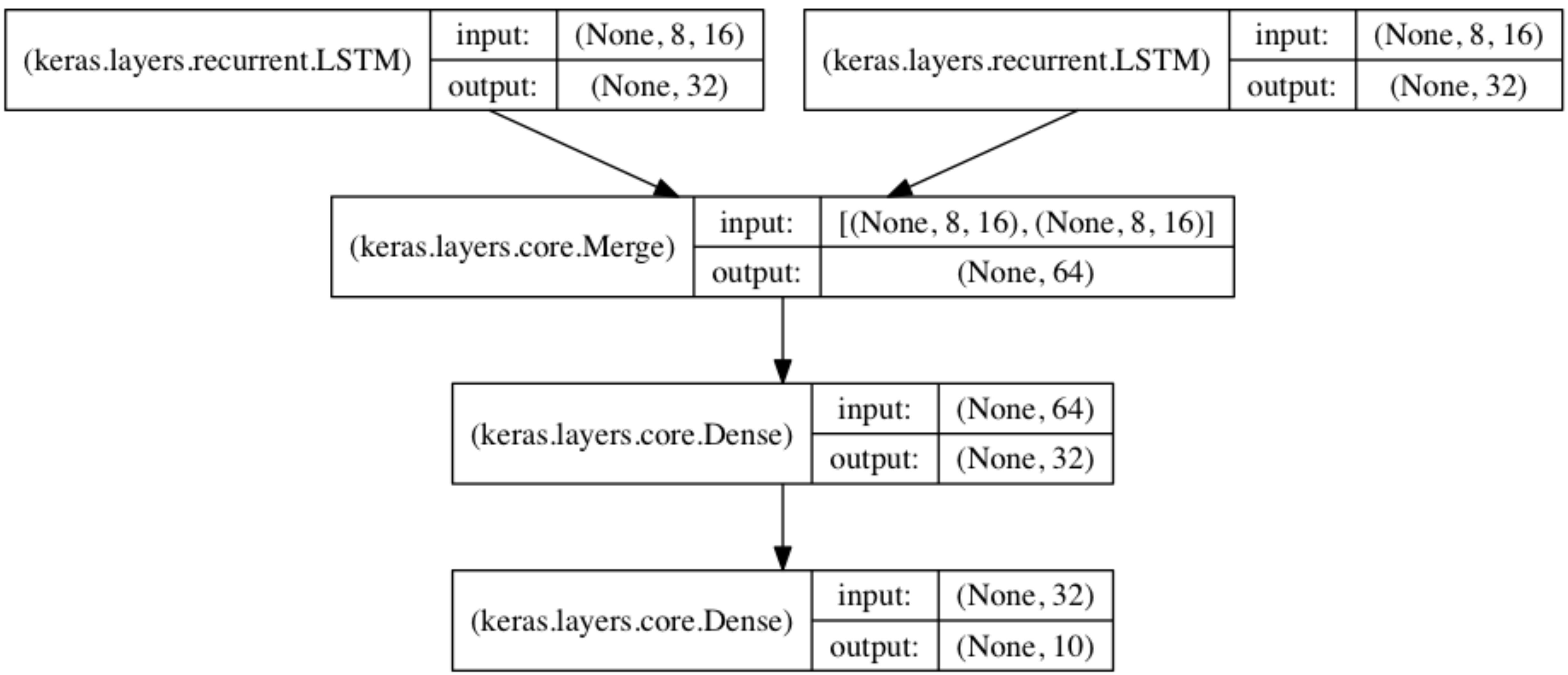
- Take an integer matrix of size (batch, input_length), the largest integer should be smaller than 1000.

- Notice: only can be used in first layer

```
model = Sequential()
model.add(Embedding(1000, 64, input_length=10))
```

Merge Layers

Merger multiple sequential models as a single input layer



Merge Layers

```
data_dim = 16
timesteps = 8
nb_classes = 10

encoder_a = Sequential()
encoder_a.add(LSTM(32, input_shape=(timesteps, data_dim)))

encoder_b = Sequential()
encoder_b.add(LSTM(32, input_shape=(timesteps, data_dim)))

decoder = Sequential()
decoder.add(Merge([encoder_a, encoder_b], mode='concat'))
decoder.add(Dense(32, activation='relu'))
decoder.add(Dense(nb_classes, activation='softmax'))

decoder.compile(loss='categorical_crossentropy',
            optimizer='rmsprop',
            metrics=['accuracy'])
```

Down to earth Questions

- How to obtain weights in each layer?

```
for layer in model.layers:
    weights = layer.get_weights()
```

- How to get configuration of each layer?

```
for layer in model.layers:
    config= layer.get_config()
```

- How can I use backend (like tensorflow) function?

```
from keras import backend as K
    K.matmul(W, x_in)+b
```

```
from keras import backend as K
    K.dot(W, x_in)+b
```

- How do I run on GPU? If using Tensorflow, code is compatible on GPU automatically.

Objectives

- One of the two parameters for compiling the model

```
model.compile(loss='mean_squared_error', optimizer='sgd')
```

mean_squared_error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_y)^2$$

mean_absolute_error

mean_squared_logarithmic_error

categorical_crossentropy

$$Cross\_Entro = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

kullback_leibler_divergence/kld

Optimizer

- The second parameter for compiling the model

```
model.compile(loss='mean_squared_error', optimizer='sgd')
```

Sgd

keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)

Adagrad

keras.optimizers.Adagrad(lr=0.01, epsilon=1e-08)

RMSprop

keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08)

Mobvoi 出门问问

Optimizer Comparison

- Sgd $$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t)$$
  - with Momentum $$v_{t+1} = \mu v_t - \alpha \nabla L(\theta_t) \qquad \theta_{t+1} = \theta_t + v_{t+1}$$
- Adagrad $$g_{t+1} = g_t + \nabla L(\theta_t)^2$$
$$\theta_{t+1} = \theta_t - \frac{\alpha \nabla L(\theta)^2}{\sqrt{g_{t+1}} + \epsilon}$$
- RMSprop

First order Momentum: $m_{t+1} = \gamma m_t + (1-\gamma)\nabla L$

Second order Momentum: $g_{t+1} = \gamma g_t + (1-\gamma)\nabla L^2$

$$v_{t+1} = \mu v_t - \frac{\alpha \nabla L(\theta)}{\sqrt{g_{t+1} - m_{t+1}^2 + \epsilon}}$$

Activation Layer

- Two approaches

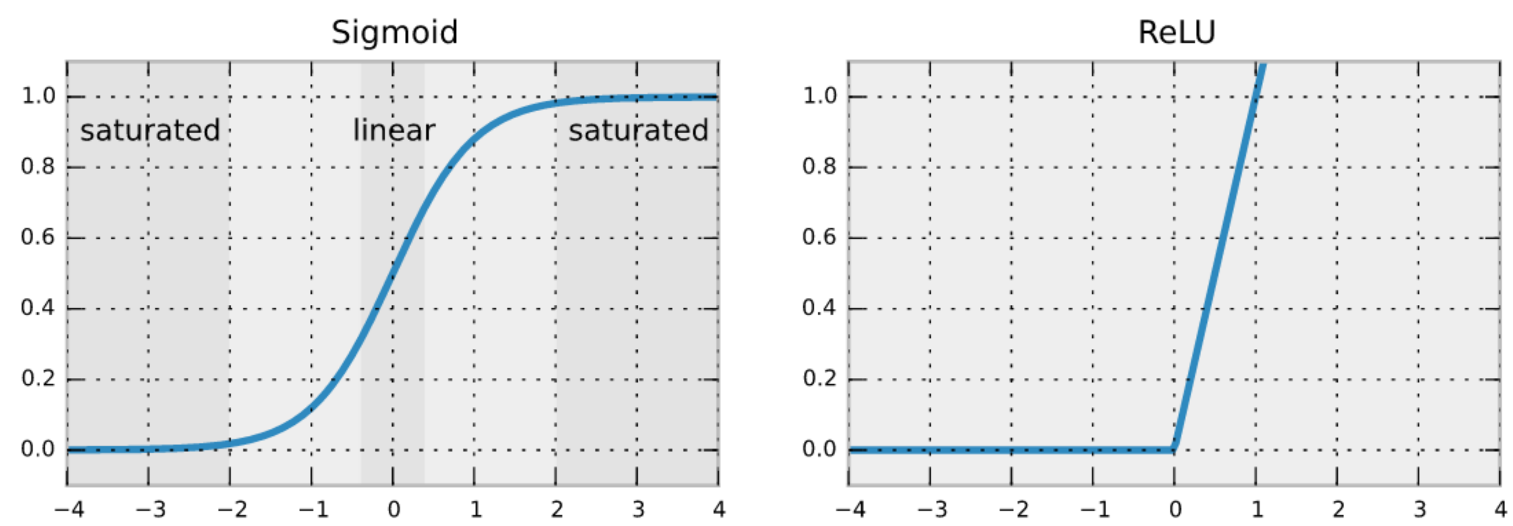
```
model.add(Dense(64))
model.add(Activation('tanh'))
```

↔

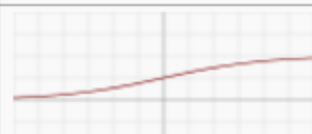
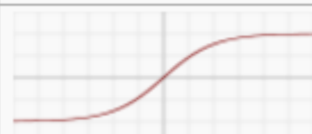
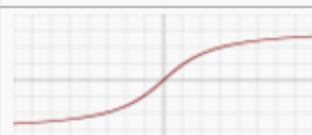
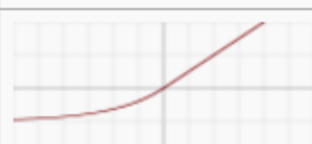
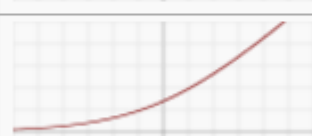
```
model.add(Dense(64), Activation='tanh')
```

- Different activation layers

```
relu
tanh
sigmoid
linear
```



Activation Layer

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

RNN example

- Time series data prediction: S&P 500 ETF 1990-1991 data (60,000+). (Data Provider: Cubist Systematic Strategist)

  Parameters:
  Memory size: 10, Epoch: 100, Batch size:64,
  Structure: 1 RNN+ 2 Dense+ 2 Dropout(0.3)

  Result: accuracy 97%, compared with ARIMA: 85%

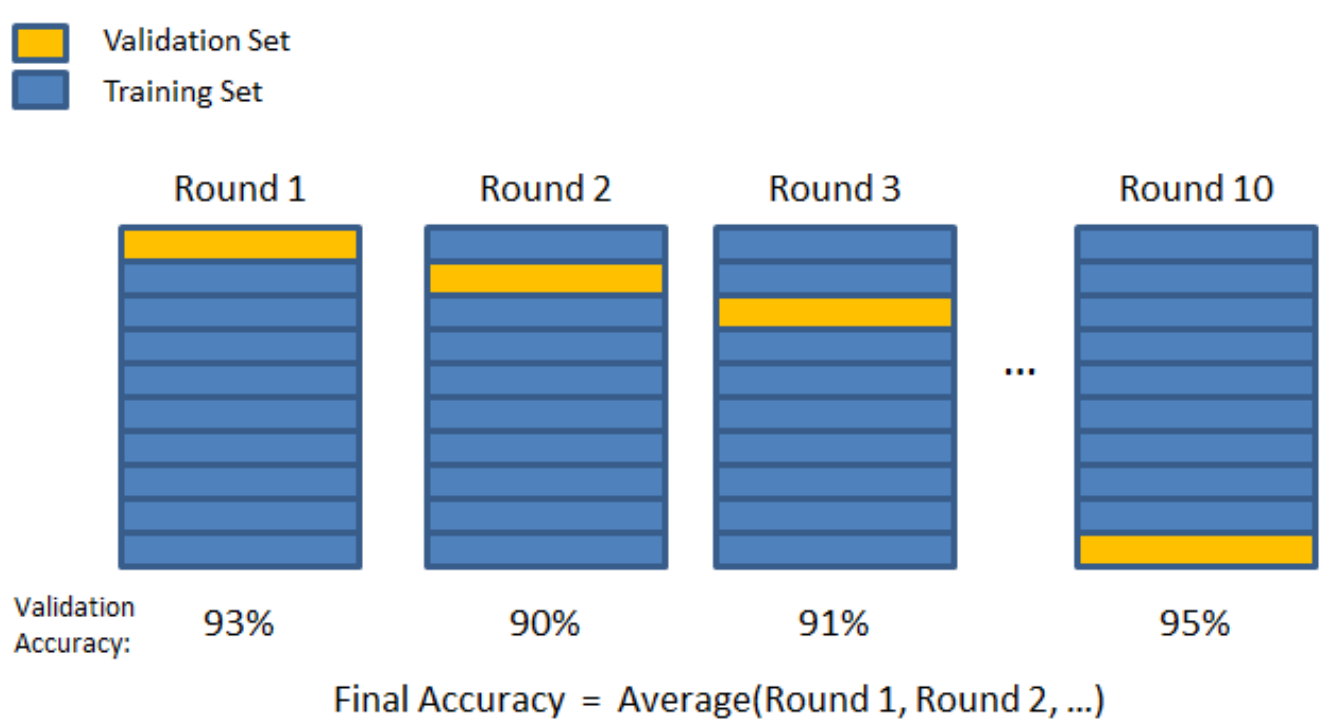
- DEMO TIME

Overfitting

- Dropout layer

```
model.add(Dropout(0.5))
```

- Early stopping & (cross) -validation
  1. Split the training data into training and validation data set
  2. more sophisticated is using cross-validation to tackle the issue.



Related Materials

- Official Doc: http://keras.io/
- Git: https://github.com/fchollet/keras/tree/master/keras
- Tutorial Video (U of Waterloo): https://www.youtube.com/playlist?list=PLFxrZqbLojdKuK7Lm6uamegEFGW2wki6P
- Code Comparison between Theano and Tensorflow (Link)