Fourth Industrial Summer School

Advanced Machine Learning

Neural Networks and Deep learning-Part5

Session Objectives

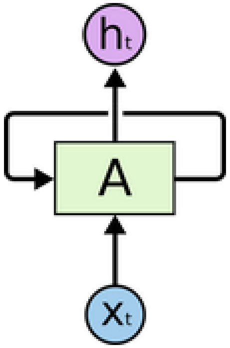
- ✓ Introduction
- ✓ Fundamentals
- ✓ Neural Network Intuitions
- ✓ 2-Layer Neural Network
- ✓ Deep Neural Networks
- ✓ CNNs
- ✓ RNNS
- ✓ Keras with Tensorflow

Recurrent Neural Networks

RNNs

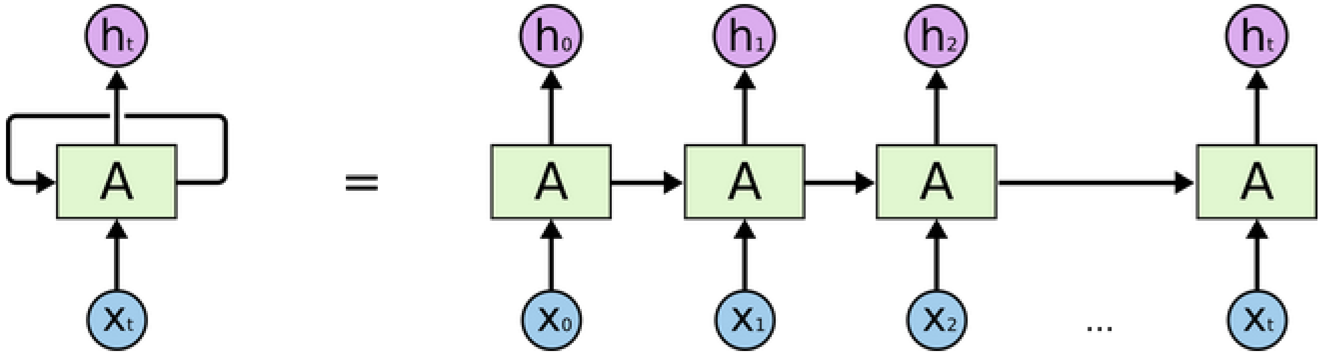
Recurrent Neural Networks (RNNs)

- Use for sequence modeling, e.g., speech recognition, handwriting recognition, language modeling, and translation.
- A recurrent cell has loop allowing the information to persist



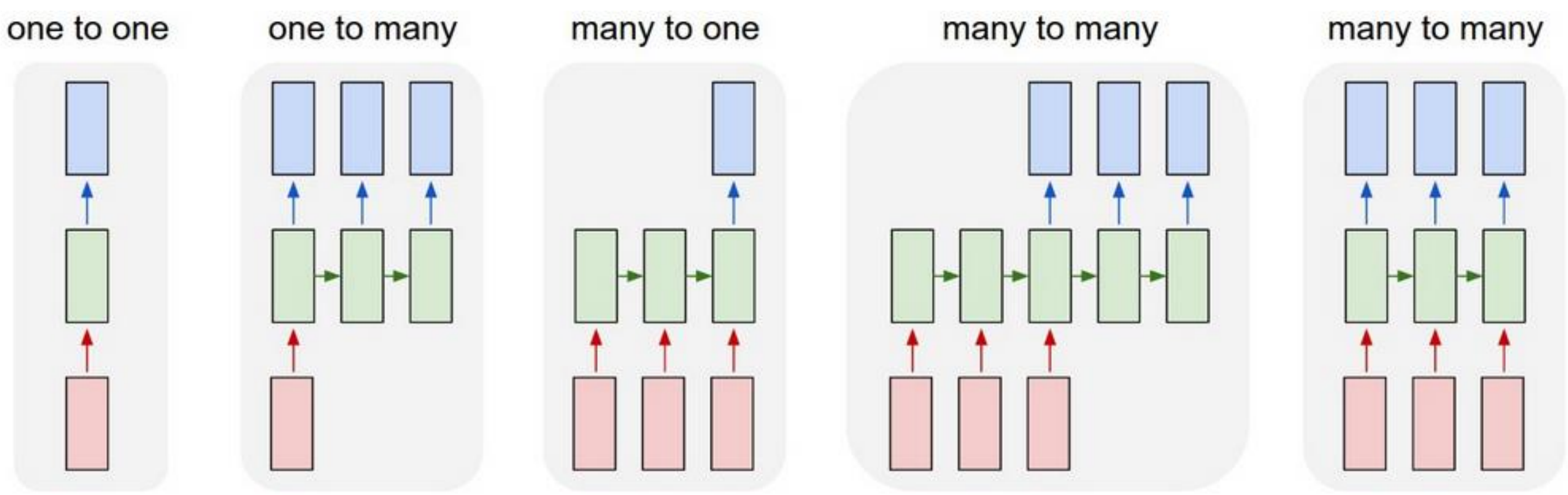
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolling



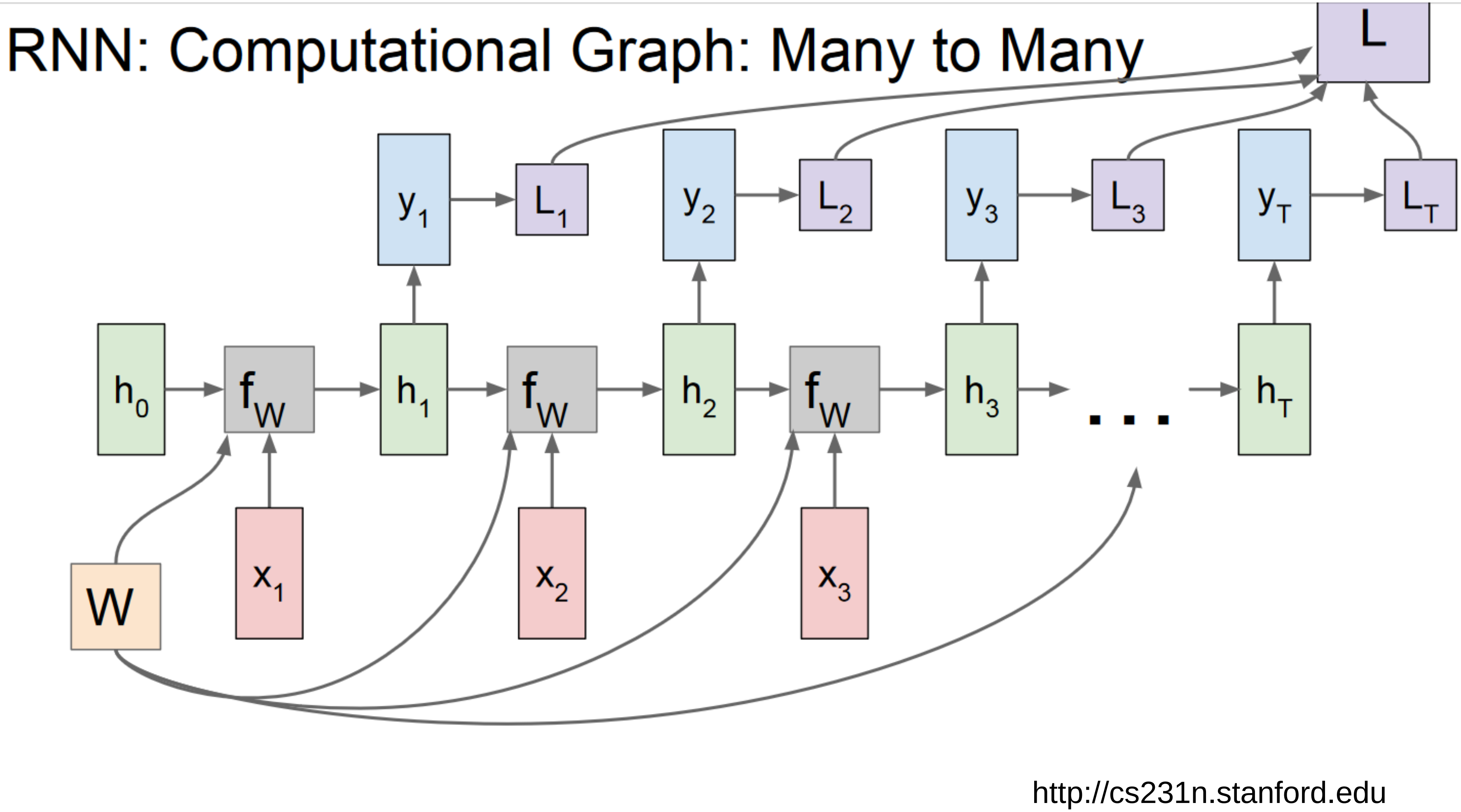
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Different architectures



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Computation

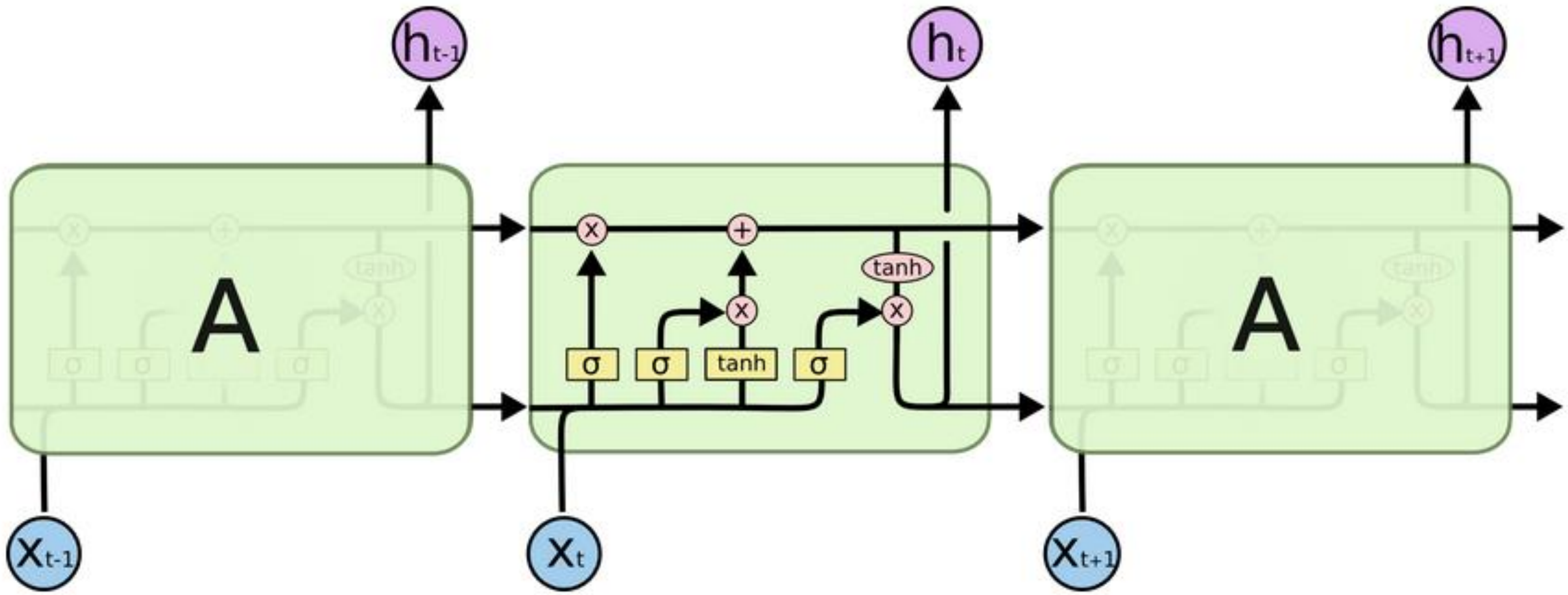


- Truncated backpropagation through time.

Long-Term Dependencies

- Need to model dependencies of varied length ranging from few frames to a large number of frames in the history
- Vanilla RNNs not that good for long-term dependencies
- Solution: LSTMs

Long Short Term Memory (LSTM)



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- "Cell state"

Working of LSTM

Gates:

- Forget(F): What do we need to throwaway (forget) from the cell state (0⟶1)
- Input (I): What data to allow in to from the input (0⟶1)
- Output (O): Decide what to output from the cell state (0⟶1)

- What new information to store in cell state:

  So, the new cell state is:

$$Cell_t = F * Cell_{t-1} + I * \tanh(new\ information)$$

- What to output from the cell

$$output_t = O * \tanh(Cell_t)$$

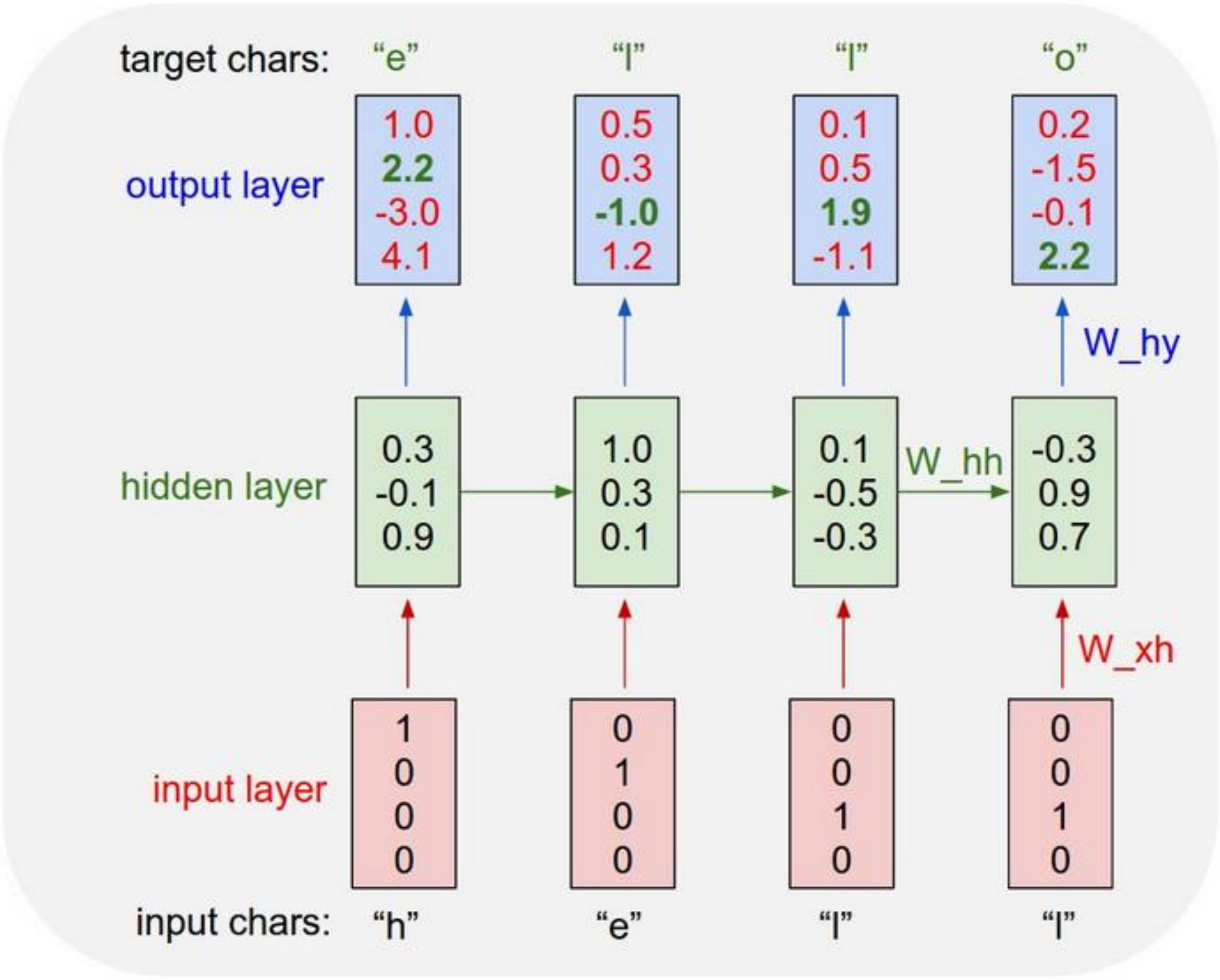
Other variation of LSTM

- Multilayer RNNs (deep RNNs)
- BLSTM
- Gated Recurrent Unit (GRUs)

Applications

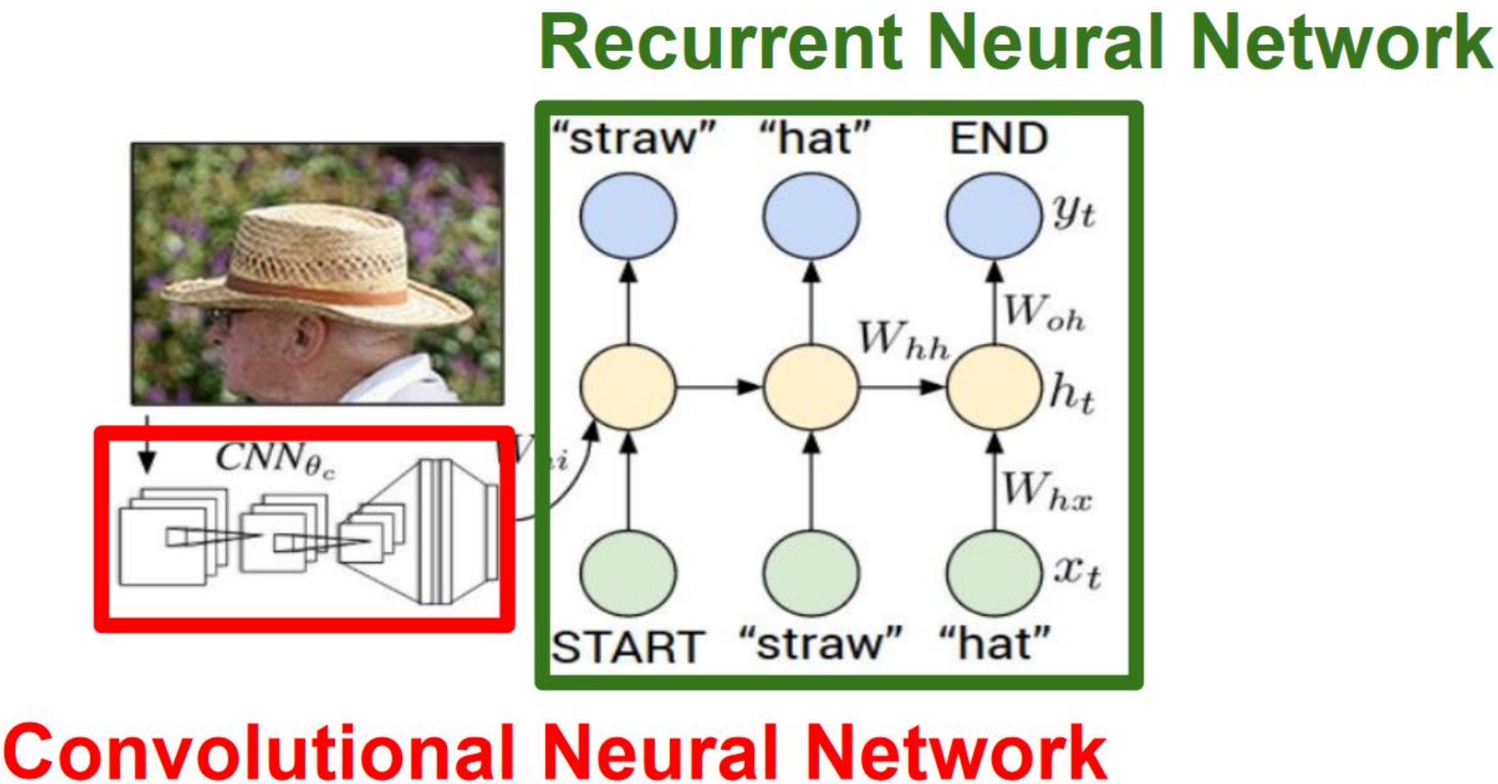
- Character/word language models (also text generation)
- Image captioning
- Sentiment analysis
- Poem-meter classification
- Handwriting recognition

Character language model



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Image captioning



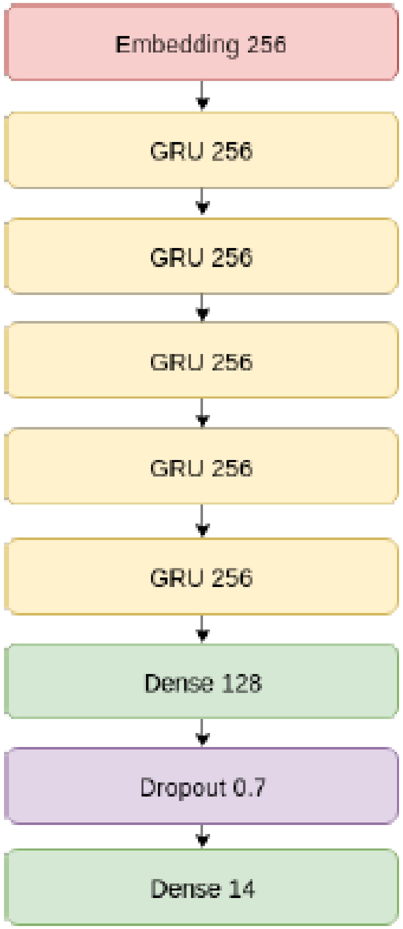
http://cs231n.stanford.edu

Visual question answering model

- Output: Select the correct one-word answer
- Input: Natural-language question + image.

https://keras.io/getting-started/functional-api-guide/#visual-question-answering-model

Arabic poem-meter classification



Name	Description	# parameters in million	Accuracy on Validation
2-NN	Neural Network with two layers	0.1m	24.02 %
3-NN	Neural Network with three layers	0.2m	30.02 %
4-NN	Neural Network with four layers	0.2m	30.39 %
1-GRU	Simple Model with 1 GRU layer	0.4m	9.280 %
1-BiGRU	1 Bidirectional GRU layer	0.8m	89.08 %
2-BiGRU	2 Bidirectional GRU layers	2.0m	92.91 %
3-BiGRU	3 Bidirectional GRU layers	3.2m	93.54 %
4-BiGRU	4 Bidirectional GRU layers	4.4m	94.30 %
5-BiGRU	5 Bidirectional GRU layers	5.6m	**94.50** %
6-BiGRU	6 Bidirectional GRU layers	6.8m	92.69 %

Keras

with TensorFlow

From https://keras.io

Keras

A high-level library which can work over tensorflow

- Quick model development
- User friendliness
- Modularity
- Easy extensibility
- Work with Python

Keras Models

- Sequential: A linear stack of layers
- Model (Functional API): Any arbitrary setup

Keras Sequential

- A linear stack of layers

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

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

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

Note!!! With tensorflow implementation of keras, we need to use tensorflow.keras instead of keras:
e.g., from tensorflow.keras.model import Sequential

model.add()

- Stacking layers in sequence

```
from keras.layers import Dense

model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

Core Layers:

- Dense: Fully connected layers
- Activation
- Dropout
- Flatten

Layers- Conv2D

- 2D convolution layer (e.g. spatial convolution over images).

Important Arguments

- Filters: No of filters (the dimensionality of the output space)
- kernel_size
- Strides
- Activation
- kernel_initializer

Layers-MaxPooling2D

- Max pooling operation for spatial data.

Important Arguments:

- pool_size
- strides

RNN-Layers

- SimpleRNN
- LSTM
- GRU

https://keras.io/layers/recurrent/

model.compile()

- Once the model structure is ready, configure its learning process with:

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')
```

- https://keras.io/optimizers/
- https://keras.io/losses/

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
```

model.fit()

Perform the actual training, it has following main arguments:

- Xs and Ys
- batch_size
- Epochs
- validation_split

```
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

model.evaluate()

- Evaluate the performance

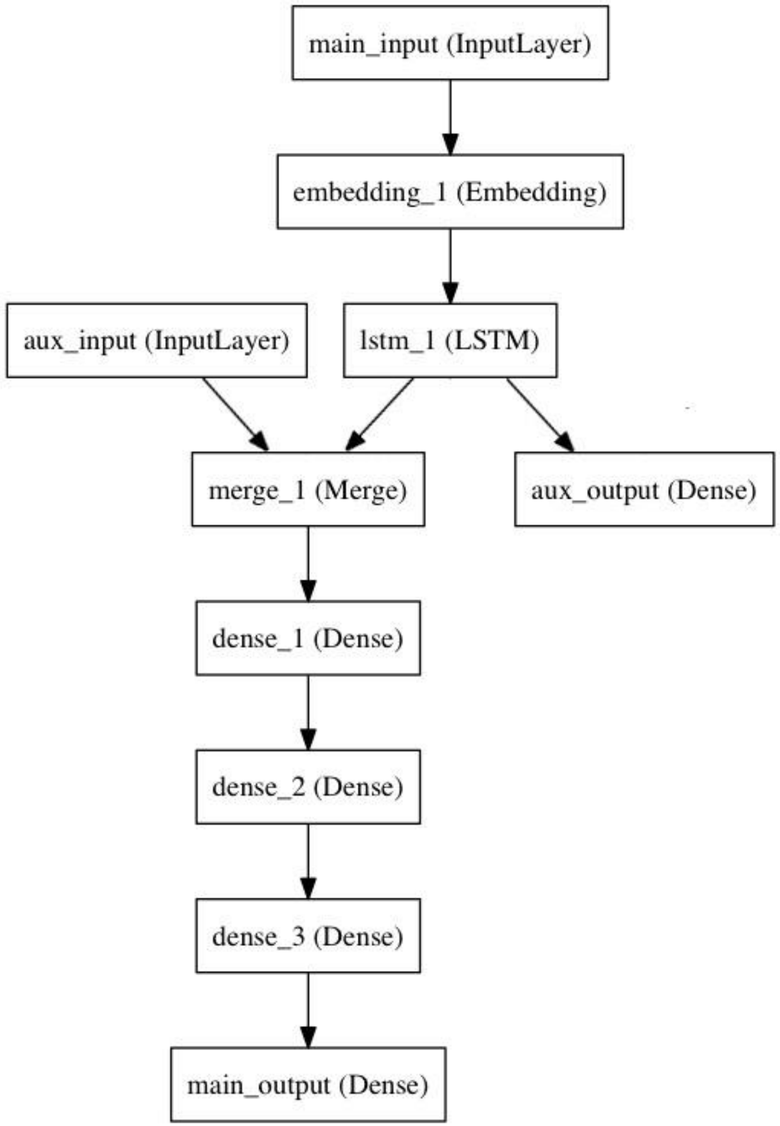
```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

Keras Models-Functional API

- The Keras functional API is the way to go for defining complex models:
  - Multi-output models
  - Models with shared layers

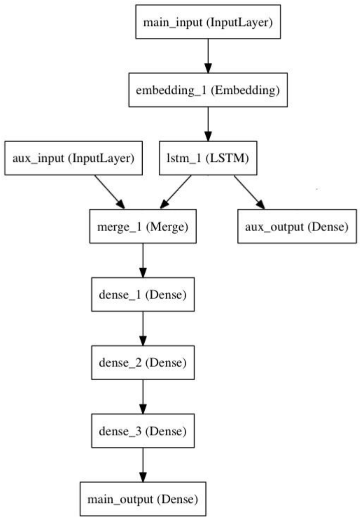
```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, 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
```

Multi-input and multi-output models



Multi-input and multi-output models

- main_input = Input(shape=(100,), dtype='int32', name='main_input')
- x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)
- lstm_out = LSTM(32)(x)
- auxiliary_output = Dense(1, activation='sigmoid', name='aux_output')(lstm_out)
- auxiliary_input = Input(shape=(5,), name='aux_input')
- x = keras.layers.concatenate([lstm_out, auxiliary_input])
- # We stack a deep densely-connected network on top
- x = Dense(64, activation='relu')(x)
- x = Dense(64, activation='relu')(x)
- x = Dense(64, activation='relu')(x)
- # And finally we add the main logistic regression layer
- main_output = Dense(1, activation='sigmoid', name='main_output')(x)
- model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output, auxiliary_output])
- model.compile(optimizer='rmsprop', loss='binary_crossentropy',loss_weights=[1., 0.2])
- model.fit([headline_data, additional_data], [labels, labels],epochs=50, batch_size=32)



Multi-input and multi-output models-2

```
model.compile(optimizer='rmsprop',
              loss={'main_output': 'binary_crossentropy', 'aux_output': 'binary_crossentropy'},
              loss_weights={'main_output': 1., 'aux_output': 0.2})

# And trained it via:
model.fit({'main_input': headline_data, 'aux_input': additional_data},
          {'main_output': labels, 'aux_output': labels},
          epochs=50, batch_size=32)
```

Shared layers

```
import keras
from keras.layers import Input, LSTM, Dense
from keras.models import Model

tweet_a = Input(shape=(280, 256))
tweet_b = Input(shape=(280, 256))


# This layer can take as input a matrix
# and will return a vector of size 64
shared_lstm = LSTM(64)

# When we reuse the same layer instance
# multiple times, the weights of the layer
# are also being reused
# (it is effectively *the same* layer)
encoded_a = shared_lstm(tweet_a)
encoded_b = shared_lstm(tweet_b)

# We can then concatenate the two vectors:
merged_vector = keras.layers.concatenate([encoded_a, encoded_b], axis=-1)

# And add a logistic regression on top
predictions = Dense(1, activation='sigmoid')(merged_vector)

# We define a trainable model linking the
# tweet inputs to the predictions
model = Model(inputs=[tweet_a, tweet_b], outputs=predictions)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit([data_a, data_b], labels, epochs=10)
```

References-1

- https://keras.io
- Introduction to Deep Learning, National Research University Higher School of Economics
- Fei-Fei Li Convolutional Neural Networks for Visual Recognition, Stanford University (http://cs231n.stanford.edu/)
- Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015
- Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Andrej Karpathy, http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Andrew Ng, Neural Networks and Deep Learning, Stanford University

References-2

- http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf
- Andrew Ng, Machine Learning Yearning, deeplearning.ai