TENSORFLOW AND DEEP LEARNING

Lili Jiang, Xuan-Son Vu , Michele Persiani

Department of Computing Science

Organized by HPC2N

8-9 May, 2019

UMEÅ UNIVERSITY

DAY 1-MORNING: TENSORFLOW

- 9:00-10:00 Lecture: Introduction to Tensorflow.
- 10:00-10:30 Exercise: Play with basic Tensorflow and TensorBoard.

10:30 Coffee break

- 11:00-12:00 Exercise: Play with Tensorflow on real dataset.

12:00-13:00 Lunch

DAY 1-AFTERNOON: DEEP LEARNING MODELS IN NLP-1

- 13:00-14:00 Lecture: Introduction of Deep Learning on Natural Language Processing (NLP).

- 14:00-14:30 Exercise: Run data representation in basic.

14:30 Coffee break

- 15:00-16:00 Exercise: Play with NLP-based deep learning models (RNN, Transformer).

DAY 2-MORNING: DEEP LEARNING MODELS IN NLP-2

- 9:00-10:00 Lecture: Introduction of different state-of-the-art deep learning models on NLP.

  10:15 Coffee break

- 10:30-11:45 Exercise: Play with NLP-based deep learning models (Bert and ELMo).

- 11:45-12:00 Lecture: Wrap-up

  12:00-13:00 Lunch

DAY 2-AFTERNOON: GAN (GENERATIVE ADVERSARIAL NETWORKS)

- 13:00-14:00 Lecture: Introduction of GAN
- 14:00-15:15 Exercise: Play with GAN.

  15:15 Coffee break
- 15:45-16:15 Lecture: Wrap-up

COURSE WEBSITES

- Course github

https://github.com/ddmatumu/TFnDeepLearning

- HPC2N website

https://www.hpc2n.umu.se/events/courses/deep-learning-spring-2019

- Feedback:

www.menti.com (Code: 21 40 80)

- Finished the installation by following the instructions on github?
- Access internet (for those from other Uni)?
- How many Windows users?

INTRODUCTION TO TENSORFLOW

Lili Jiang, Xuan-Son Vu , Michele Persiani

Organised by HPC2N

08 May, 2019

UMEÅ UNIVERSITY

TENSORFLOW

- An open source library for numerical computation.
- Launched by Google in 2015, November
- Tensorflow 2.0 Alpha released in 2019, March

WHY TENSORFLOW

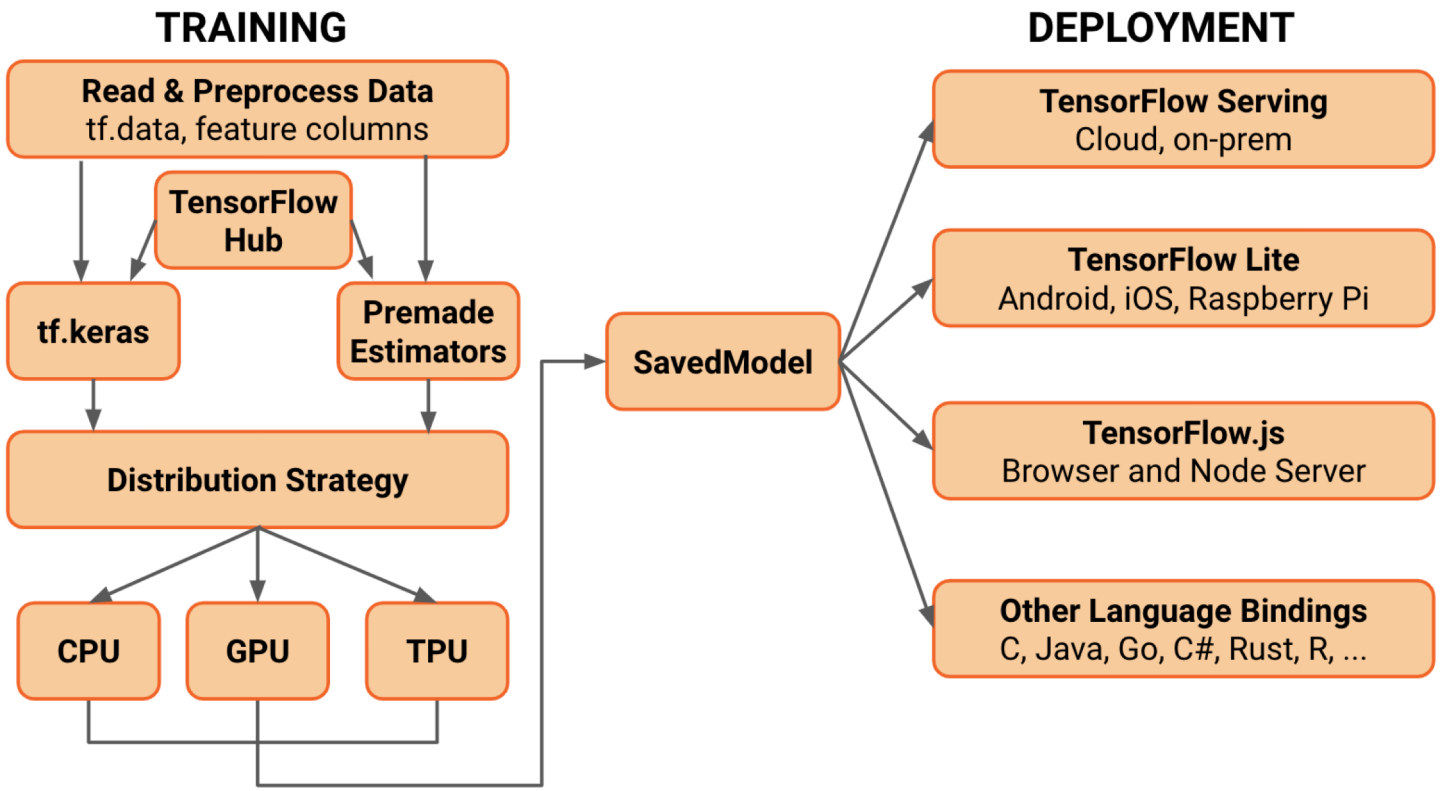
- Flexibility + Scalability

WHY TENSORFLOW

- Chainer and PyTorch
  - Flexible enough for research but less scalable
- Caffe and MXNet
  - Scalable but less flexible
- Keras
  - Simpler/beginner, but difficult for customization.
- Theano
  - High performance, but difficult syntax

WHY TENSORFLOW

- And last but not least:
  - Tensorflow meets industry's requirements



WHAT IS TENSOR

- **An n-dimensional array**
- 0-d tensor: scalar (number)
- 1-d tensor: vector
- 2-d tensor: matrix
- .....

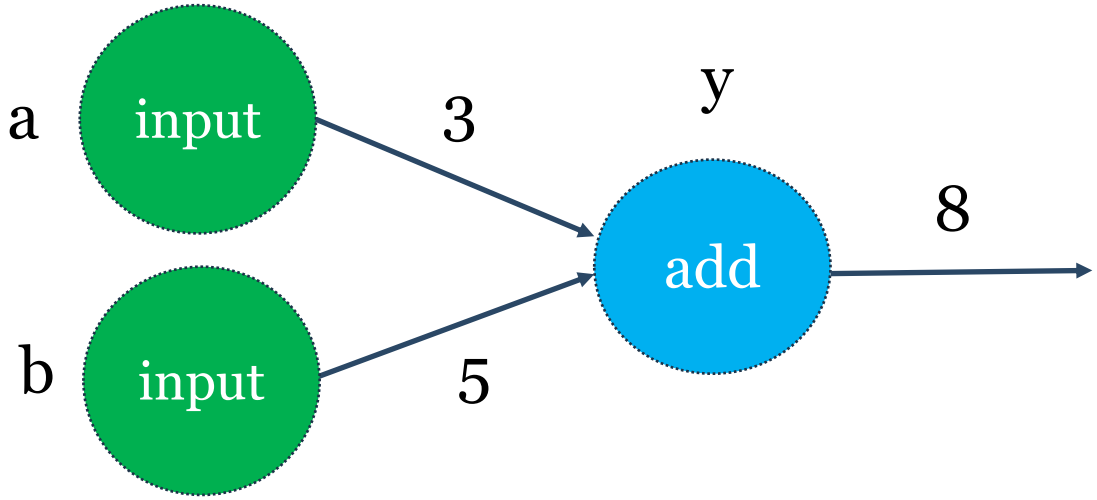
WHAT IS TENSOR

- t_0 = 3 #scalar is 0-d tensor
- t_1 = [“cat”, “dog”,] # 1-d tensors
- t_1 = [[1, 2], [2, 3]] # 2-d tensors
- ……

Tensorflow = Tensor + Flow = Data + Flow

DATA FLOW GRAPH

- Use a session to execute operations in the graph



Nodes: operators, variables, and constants
Edges: tensors

TENSORFLOW

- Each node is an operation (op).
- Data is represented as Tensors.
  Op takes Tensors and returns Tensors.
- Variables maintain state across executions of the graph.

TENSORFLOW ”HELLO WORLD”

```
import tensorflow as tf

a = tf.constant(3)
b = tf.constant(5)
y = tf.add(a, b)

with tf.Session() as sess:
        print(sess.run(x))
```

Graph and sessions

TWO PHASES IN THE PROGRAM

- Construct the computation graph.
- Executes a graph in the context of a Session.

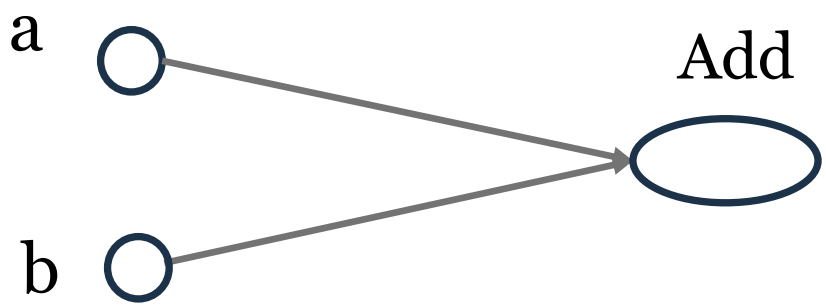
TF.SESSION()

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

- Session will also allocate memory to store the current values of variables.

VISUALIZE IN TENSORBOARD

- Take **Add** as an example:



TENSORBOARD

- Visualize the computation graph
- Present the vector space, to see how they related with each other.
- Help tuning / training parameters
- Debug your computational graphs

TENSORFLOW DATA TYPES

- Constant
  - a = tf.constant(2, name='a')
- Variable
  - tf.Variable(3, name="scalar")
- Placeholder
  - E.g., function f(x, y) = 2 * x + y without knowing value of x or y.
  - x, y are placeholders for the actual values.

CONSTANT

```
import tensorflow as tf
a = tf. constant (2, name = ’a’)
b = tf. constant ([2, 3], name = ’b’)
y = tf. add(a, b, name = ’add’)
with tf. Session() as sess:
        print (sess.run(y))
```

RANDOM CONSTANT

- tf.random_normal
- tf.truncated_normal
- tf.random_uniform
- tf.random_shuffle
- tf.random_crop
- tf.multinomial
- tf.random_gamma

TIP OF USING CONSTANT

- Only use constants for primitive types.
- Use variables or readers for more data that requires more memory

VARIABLE

- # create variables with tf.Variable
- s = tf.Variable(2, name="scalar")
- m = tf.Variable([[0, 1], [2, 3]], name="matrix")
- W = tf.Variable(tf.zeros([784,10]))

VARIABLE

create variables with tf.get_variable

- s = tf.get_variable("scalar", initializer=tf.constant(2))
- m = tf.get_variable("matrix", initializer=tf.constant([[1, 2], [2, 3]]))
- W = tf.get_variable("matrix", shape=(10, 10), initializer=tf.zeros_initializer())

(**recommended!**)

VARIABLE

- Use variables for more data requiring memory.
- tf.variable or tf. Variable?

- Tf. constant is an op
- tf. Variable is a class.

ARITHMETIC OPS

- tf.abs
- tf.negative
- tf.square
- tf.round
- tf.sqrt
- tf.exp

TIP OF USING VARIABLE

```
#initialization
W = tf.Variable(tf.zeros([784,10]))
with tf.Session() as sess:
    sess.run(W.initializer)
```

TIP OF USING VARIABLE

```
#initialize all variables at once:
with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

# Initialize only a subset of variables:
with tf.Session() as sess:
        sess.run(tf.variables_initializer([a, b]))
```

PLACEHOLDER

- For example, a function f(x, y) = 2 * x + y without knowing value of x or y.
- x, y are placeholders for the actual values.

PLACEHOLDER (EXAMPLES)

tf.placeholder(dtype, shape=None, name=None)

- a = tf.placeholder(tf.float32, shape=[3])
- b = tf.constant([5, 5, 5], tf.float32)

PLACEHOLDER VS. VARIABLE

- Variable defines: shape + its values + data type
- Placeholder defines: shape + data type

- Variable: hold variables to be trained
- Placeholder: hold data to feed into the model

- b = tf.Variable(tf.zeros([1]))
- x = tf.placeholder(tf.float64, [None, 25])

TENSORFLOW AND NUMPY

tensor = tf.constant(np_array)

Example:

```
sess = tf.Session()
a = tf.zeros([2, 3], np.int32)
a_out = sess.run(a)
print(type(a_output)) # ⇒ <class 'numpy.ndarray'>
```

CONTROL DEPENDENCIES

```
tf.Graph.control_dependencies(control_inputs)
# defines which ops should be run first
# your graph g have 5 ops: a, b, c, d, e

g = tf.get_default_graph()
with g.control_dependencies([a, b, c]):
```

THE TRAP OF LAZY LOADING

```
x = tf.Variable(10, name='x’)
y = tf.Variable(20, name='y’)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for _ in range(10):
        sess.run(tf.add(x, y))
     writer.close()
```

NORMAL LOADING

```
x = tf.Variable(10, name='x')
y = tf.Variable(20, name='y')
z = tf.add(x, y)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for _ in range(10):
        sess.run(z)
    writer.close()
```

TIP / WARNING

- Separate definition of ops from computing/running ops
- Ensure that function is also loaded once the first time it is called.

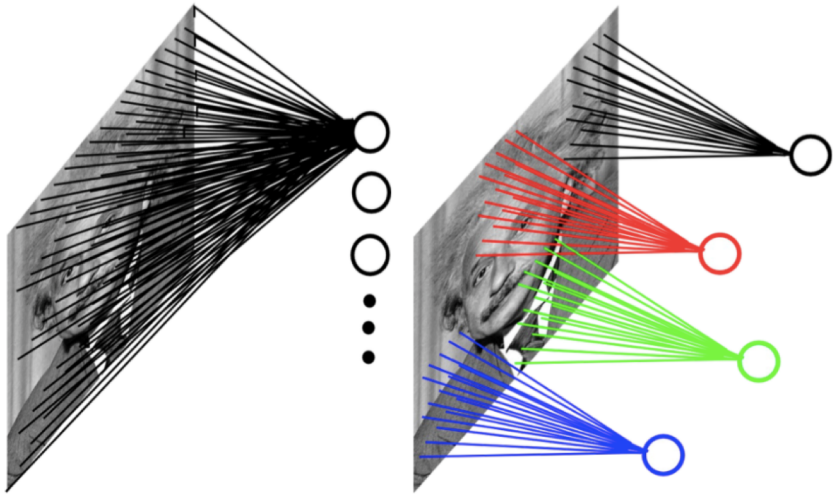
Instead of a+b,

build neural networks in tensorflow?

CNN? RNN?

CONVOLUTIONAL NEURAL NETWORK

- CNNs are a special type of neural network whose hidden units are only connected to local receptive field. The number of parameters needed by CNNs is much smaller.

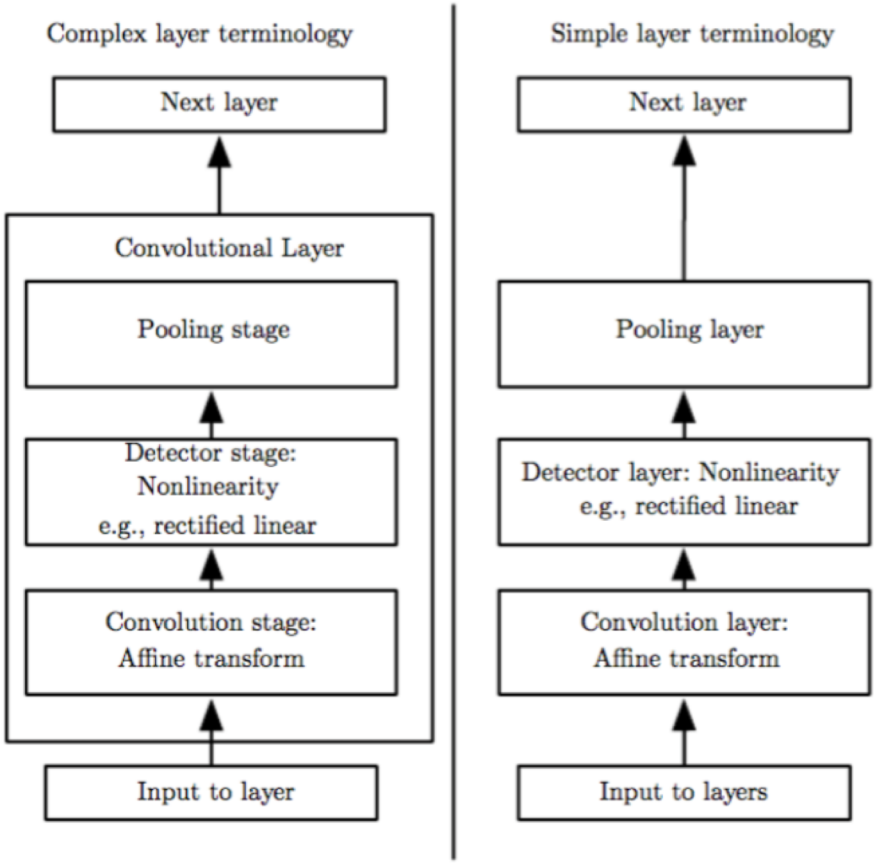


Example: 200x200 image

a) fully connected: 40,000 hidden units => 1.6 billion parameters

b) CNN: 5x5 kernel, 100 feature maps => 2,500 parameters

THREE STAGES OF A CONVOLUTIONAL LAYER

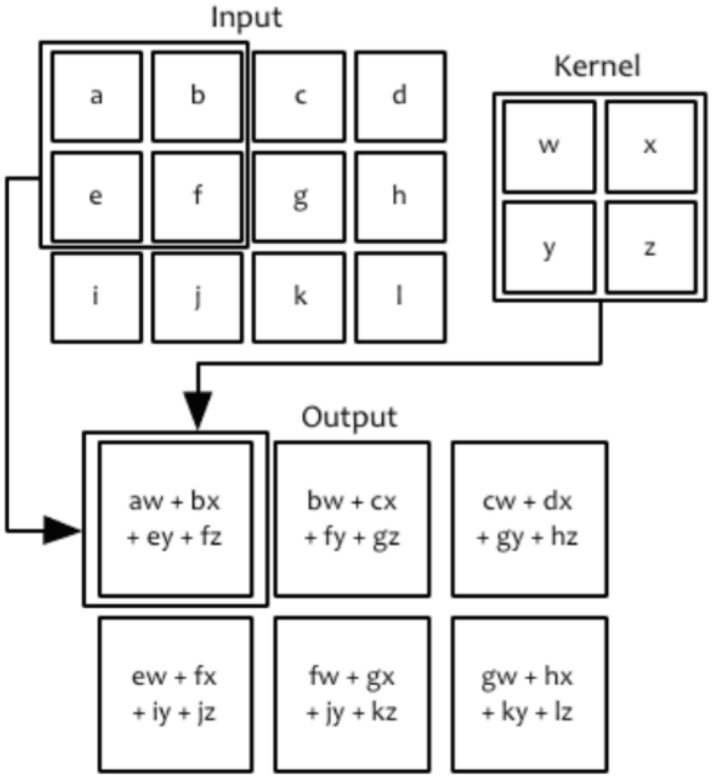


Convolution stage

Nonlinearity: a nonlinear transform such as rectified linear or tanh

Pooling: output a summary statistics of local input, such as max pooling and average pooling

CONVOLUTION OPERATION IN CNN

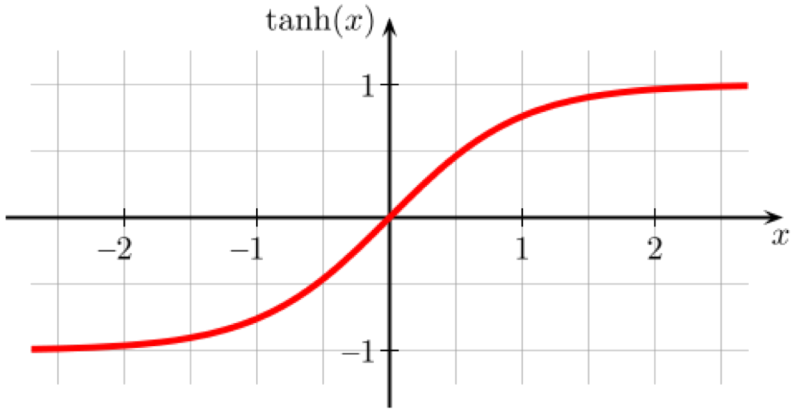


- Input: an image (2-D array) x
- Convolution kernel/operator(2-D array of learnable parameters): $\boldsymbol{w}$
- Feature map (2-D array of processed data): $\boldsymbol{s}$
- Convolution operation in 2-D domains

$$s[i,j] = (x * w)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} x[i+m, j+n]\, w[m,n]$$

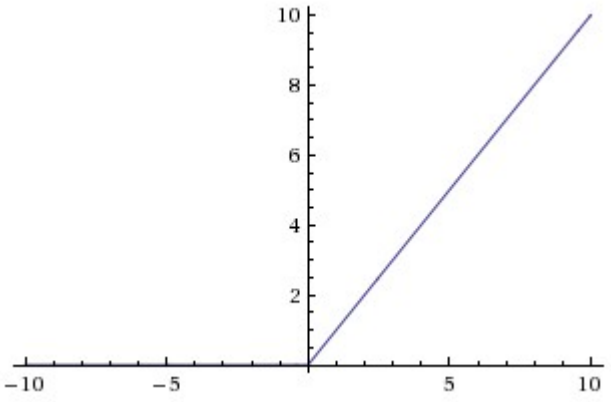
NON-LINEARITY

Tanh(x)



$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

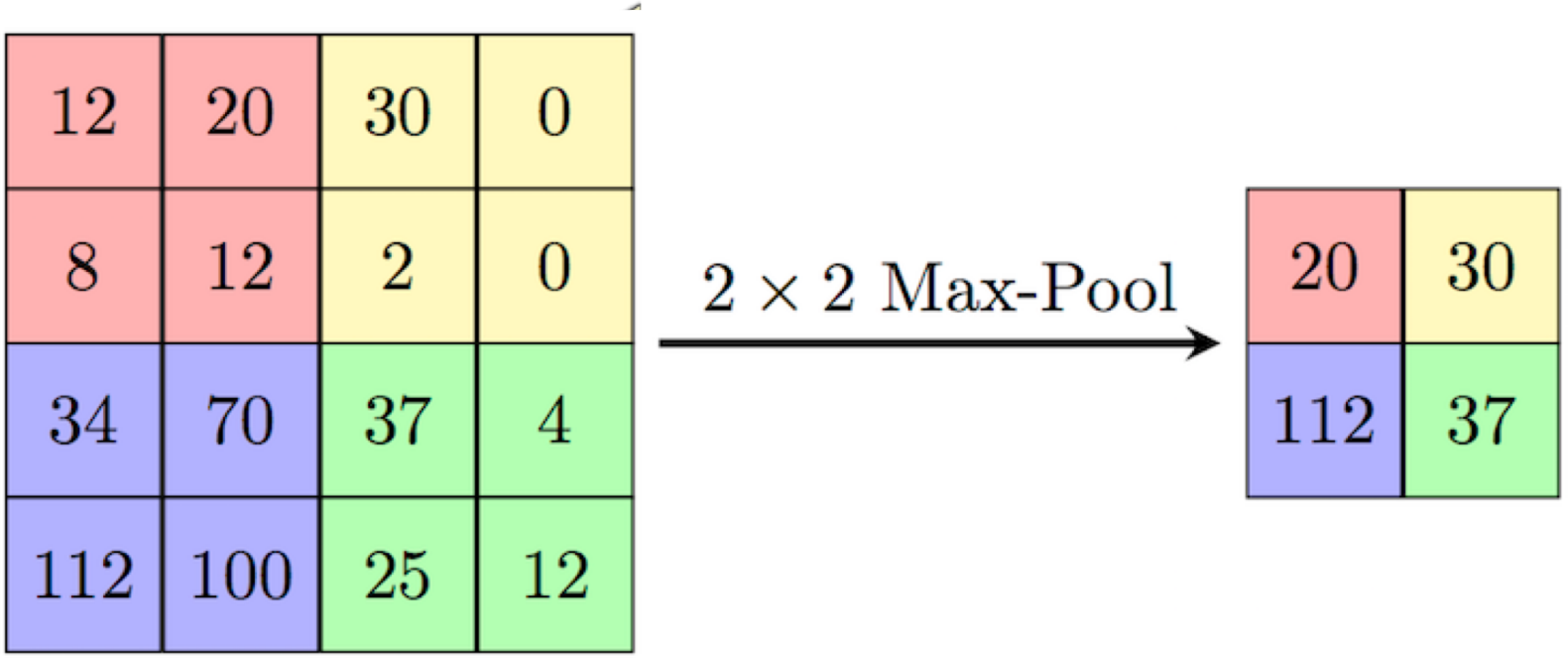
ReLU



$$f(x) = \max(0, x)$$

POOLING

- Common pooling operations:
  - Max pooling: reports the maximum output within a rectangular neighborhood.
  - Average pooling: reports the average output of a rectangular neighborhood (possibly weighted by the distance from the central pixel).

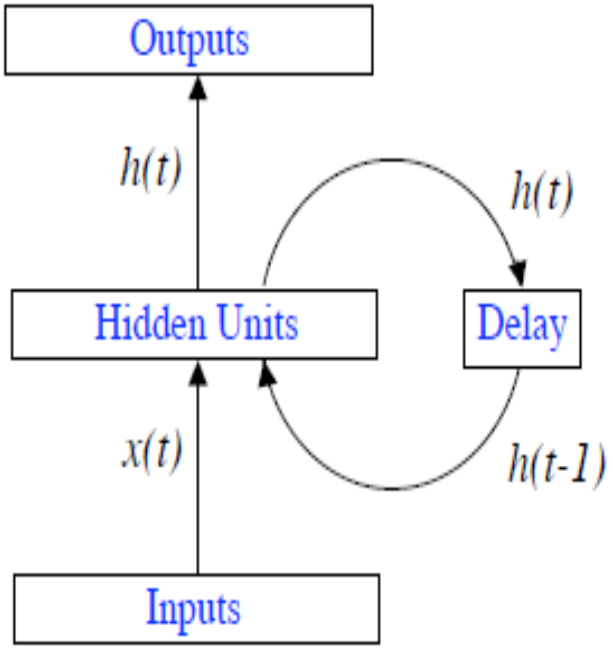


CNN IN TENSORFLOW

- tf.layers module -> construct a neural network
  - conv2d(): arguments - number of filters, filter kernel size, padding, and activation function.
  - max_pooling2d(): arguments - pooling filter size and stride.
  - dense(): arguments - number of neurons and activation function.

WHAT ARE RNNS?

The simplest form of *fully recurrent neural network* with the previous set of hidden unit activations feeding back into the network along with the inputs



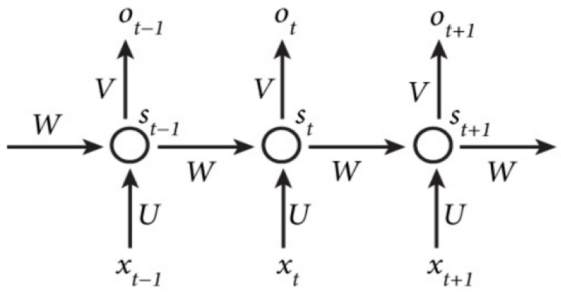
$$h(t) = f_H(W_{IH}x(t) + W_{HH}h(t-1))$$

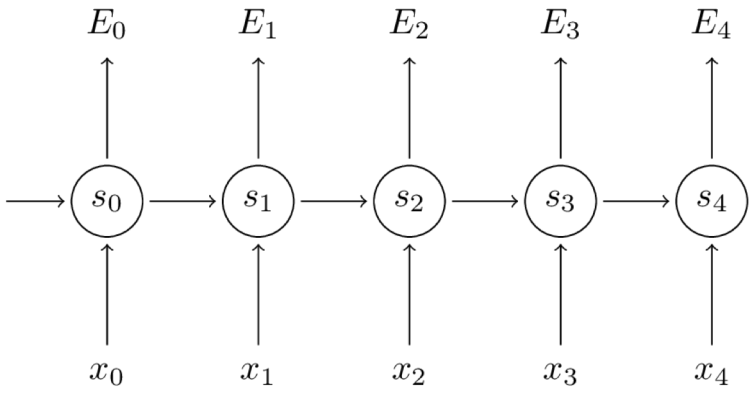
$$y(t) = f_O(W_{HO}h(t))$$

f_H and f_O are the activation function for hidden and output unit; W_{IH}, W_{HH}, and W_{HO} are connection weight matrices which are learnt by training.

TRAINING RNNS

Back Propagation Through Time (BPTT), an extension of the back-propagation (BP), is often used to learn RNNs.





- The output of this RNN is $\hat{y}_t$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = softmax(Vs_t)$$

- The loss/error function:

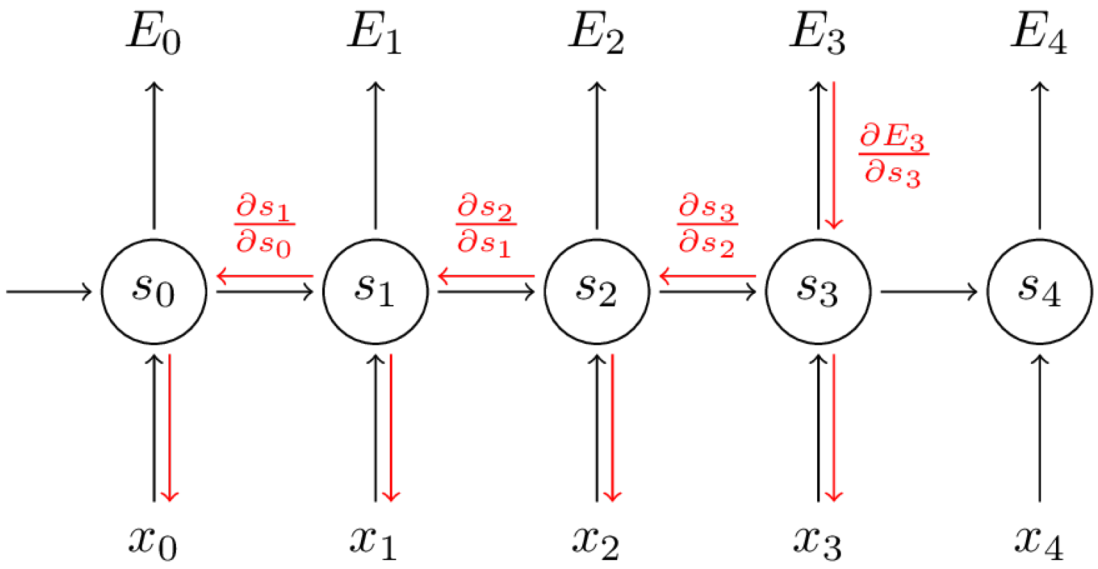
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

TRAINING RNNS

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Because W is used in every step up to the output we care about, we need to back-propagate gradients from $t = 3$ through the network all the way to $t = 0$



RNN IN TENSORFLOW

tf.Graph() + tf.Session()

- Initialize placeholders which are filled with batches of training data.
- Define the RNN model and to calculate the output values (logits) -> loss value
- Use an Optimizer to optimize the weights of the RNN.

RNN FLAVOR IN TENSORFLOW

Cells

- tf.nn.rnn_cell.BasicRNNCell(num_hidden)
- tf.nn.rnn_cell.BasicLSTMCell(num_hidden)
- GruCell, MultiRNNCell

Container

- tf.nn.static_rnn(cell, data, dtype=tf.float32)
- dynamic_rnn (cell, data, ...)
- static_bidirectional_rnn(.....)

RNN FLAVOR IN TENSORFLOW

Cells

tf.keras.layers.SimpleRNNCell
tf.keras.layers.LSTMCell

- tf.nn.rnn_cell.BasicRNNCell(num_hidden)
- tf.nn.rnn_cell.BasicLSTMCell(num_hidden)
- GruCell, MultiRNNCell

Container

- tf.nn.static_rnn(cell, data, dtype=tf.float32)
- dynamic_rnn (cell, data, ….)
- static_bidirectional_rnn(…..)

keras.layers.RNN(cell)
keras.layers.RNN(cell, unroll=True)
keras.layers.Bidirectional(keras.layers
RNN(cell, unroll=True))

EXCERCISE 1

- Basic operations in Tensorflow
- Linear Regression using tensorflow

EXCERCISE 2

- Predicting HPC total energy consumption using tensorflow

FEEDBACK ALWAYS WELCOME !

- Go to www.menti.com
- Code: 21 40 80

Thanks

UMEÅ UNIVERSITY