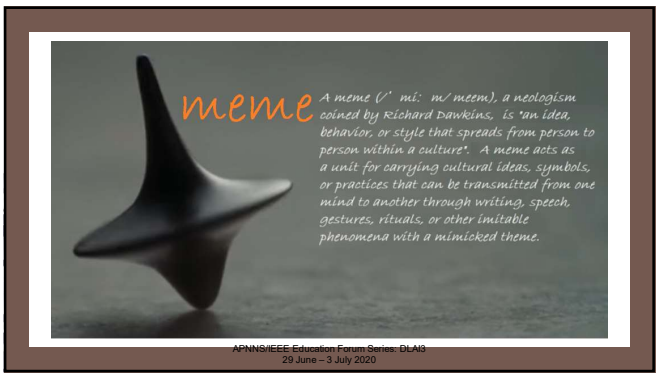
TensorFlow - Keras

DLAI3 Workshop, 1 July 2020

Somnuk Phon-Amnuaisuk
School of Computing and Informatics
Centre for Innovative Engineering
Universiti Teknologi Brunei

1



2

Outline

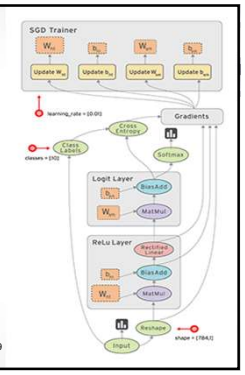
- Creating DL models
  - TensorFlow, and Keras
  - Sequential
  - Loss function and Training
  - Practical: Linear Classifier, MLP, and CNN

Acknowledgement: images and materials are from
- tensorflow.org
- https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0 (by Martin Gorner)

3

TensorFlow
TF 0.x →TF 1.x → TF 2.x

*** Credit Google site, slideshare site and Stanford University



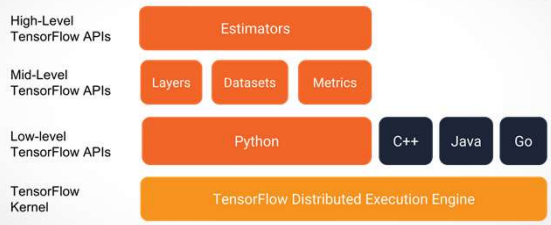
4

Tensorflow

- Starting in 2011, Google Brain built **DistBelief** as a proprietary machine learning system based on deep learning neural networks.
- **TensorFlow** is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017 (version 2.0 was released in October 2019).
- TensorFlow computations are expressed as stateful **dataflow graphs**. The name TensorFlow derives from the operations that such neural networks perform on **multidimensional data arrays**, which are referred to as **tensors**.
- TensorFlow provides stable Python (for version 3.7 across all platforms) and C APIs; and without API backwards compatibility guarantee: C++, Go, Java, JavaScript and Swift (early release).

5

TensorFlow

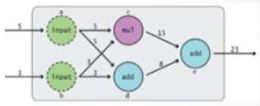


6

Graphs and Computations

TensorFlow Graph Execution separates definition of computations from their execution

1. Phase 1: assemble a graph
2. Phase 2: use a session (TF1.x) to execute operations in the graph.



7

Benefits of Graphs

1. Save computation. Only run subgraphs that lead to the values you want to fetch.
2. Break computation into small, differential pieces to facilitate auto-differentiation
3. Facilitate distributed computation, spread the work across multiple CPUs, GPUs, TPUs, or other devices
4. Many common machine learning models are taught and visualized as directed graphs

8

Tensors

- An n-dimensional array
  - 0-d tensor: scalar (number)
  - 1-d tensor: vector
  - 2-d tensor: matrix
  - and so on

9

Data Flow Graph

```
import tensorflow as tf

a = tf.add(3, 5)
```

TF automatically names the nodes when you don't explicitly name them.

x = 3

y = 5

10

Data Flow Graph

```
import tensorflow as tf
a = tf.add(3, 5)
print(a)

>> tf.Tensor(8, shape=(), dtype=int32)
```

11

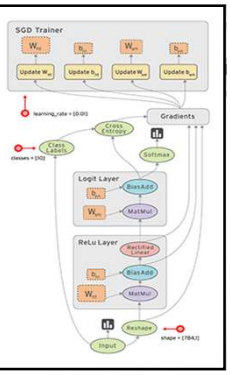
Tensor Constructors

```
tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]

# input_tensor is [[0, 1], [2, 3], [4, 5]]
tf.zeros_like(input_tensor) ==> [[0, 0], [0, 0], [0, 0]]

tf.fill([2, 3], 8) ==> [[8, 8, 8], [8, 8, 8]]

tf.linspace(start, stop, num, name=None)
tf.linspace(10.0, 13.0, 4) ==> [10. 11. 12. 13.]

tf.range(start, limit=None, delta=1, dtype=None, name='range')
tf.range(3, 18, 3) ==> [3 6 9 12 15]
tf.range(5) ==> [0 1 2 3 4]
```


12

Practical: TensorFlow

*** Credit Google site, slideshare site and Stanford University



13

Example 1: Quadratic Equation

Creating a Simple TensorFlow Program

Let's write a tensorflow graph to compute roots of a given quadratic equation

$$y = ax^2 + bx + c$$

$$e.g., y = 2x^2 - 3x - 1$$

```
a = tf.constant(2)
b = tf.constant(-3)
c = tf.constant(-1)
```


14

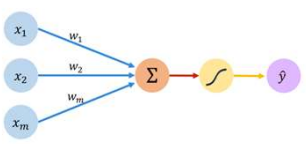
Example 2: Perceptron

Output

Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i\ w_i\right)$$

Non-linear activation function



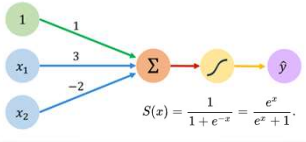
15

We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T\boldsymbol{W})$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

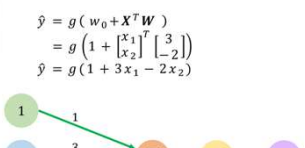
16

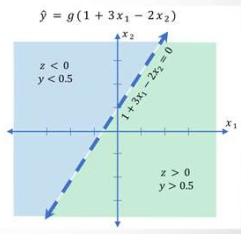
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T\boldsymbol{W})$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$





17

Keras

- Keras is an open-source neural-network library written in Python.
- It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML.
- It is designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Its primary author and maintainer is François Chollet, a Google engineer.

18

keras & tensorflow.keras

- TensorFlow's implementation of the Keras API is accessed via tf.keras.
- This is a high-level API to build and train DL models.
- `tf.keras` makes TensorFlow easier to use without sacrificing flexibility and performance.

19

tensorflow.keras.Sequential

tf.keras.Sequential(layers=None, name=None)

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, input_shape=(16,)))
…
model.add(tf.keras.layers.Dense(4))
```


20

tensorflow.keras.Sequential

```
compile(
    optimizer='rmsprop', loss=None, metrics=None,  loss_weights=None,
    sample_weight_mode=None, weighted_metrics=None, **kwargs
)

fit(
    x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None,
    validation_split=0.0, validation_data=None, shuffle=True, class_weight=None,
    sample_weight=None, initial_epoch=0, steps_per_epoch=None,
    validation_steps=None, validation_batch_size=None, validation_freq=1,
    max_queue_size=10, workers=1, use_multiprocessing=False
)
```


21

tensorflow.keras.Sequential

```
evaluate(
    x=None, y=None, batch_size=None, verbose=1, sample_weight=None,
    steps=None, callbacks=None, max_queue_size=10, workers=1,
    use_multiprocessing=False, return_dict=False
)

predict(
    x, batch_size=None, verbose=0, steps=None, callbacks=None,
    max_queue_size=10, workers=1, use_multiprocessing=False
)
```


22

Building a Model in Keras using Sequential

```
model = tf.keras.Sequential()

# Adds a densely-connected layer with 64 units to the model:
model.add(tf.keras.layers.Flatten(input_shape=[28,28,1]))

# Add another:
model.add(tf.keras.layers.Dense(64, activation='relu'))

# Add a softmax layer with 10 output units:
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```


23

Example of Layers in tf.keras.layers

```
# Create a sigmoid layer:
layers.Dense(64, activation='tf.sigmoid')

# A linear layer with L2 regularization of factor 0.01 applied to the bias vector:
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# A 2D convolutional layer
layers.Conv2D(kernel_size=6, filters=24, strides=2, activation='relu')

# A polling layer
layers.MaxPolling2D( pool_size=(2,2), stride=2, padding='same')
```


24

Build and Compile a Model

```
model = tf.keras.Sequential([
layers.Dense(64, activation='relu', input_shape=(32,)),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.train.AdamOptimizer(0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

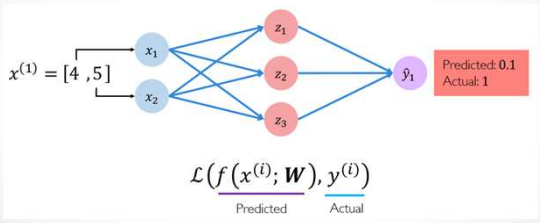

25

Train, Save and Load Model

```
# Train model
model.fit(data, labels, batch_size=32, epochs=5)


# Save entire model to a HDF5 file
model.save('my_model.h5')

# Recreate the exact same model, including weights and optimizer.
model = tf.keras.models.load_model('my_model.h5')
```

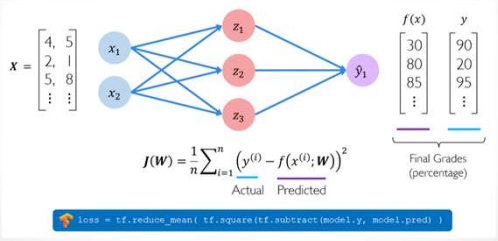

26

Loss Function



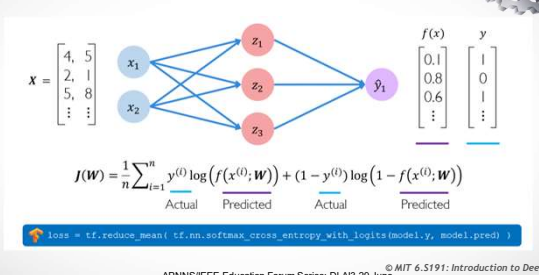
27

Mean-Square Error Loss



28

Binary Cross Entropy Loss



29

Gradient Descent

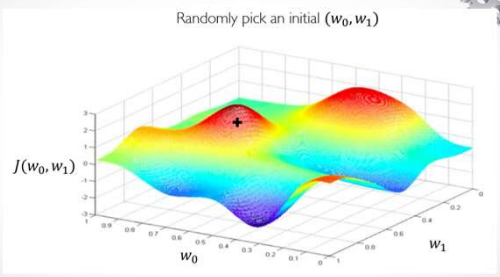
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$ — `weights = tf.random_normal(shape, stddev=sigma)`
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$ — `grads = tf.gradients(ys=loss, xs=weights)`
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$ — `weights_new = weights.assign(weights - lr * grads)`
5. Return weights

30

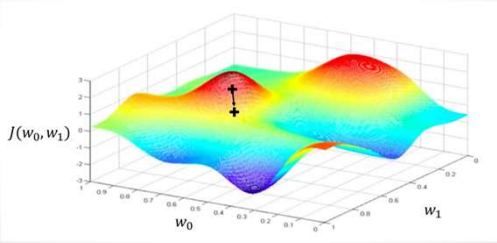
Gradient Descent

Randomly pick an initial (w_0, w_1)



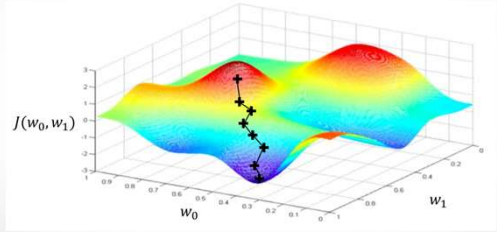
31

Gradient Descent



32

Gradient Descent

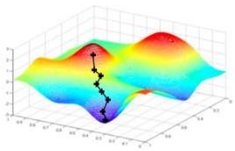


33

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



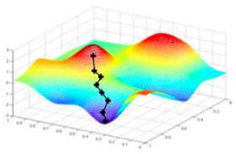
34

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

Easy to compute but **very noisy** (stochastic)!

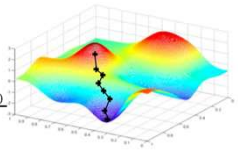


35

Minibatch Gradient Descent

Algorithm

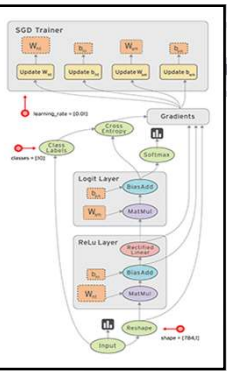
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(W)}{\partial W} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights



36

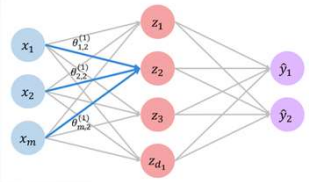
Practical: TensorFlow

*** Credit Google site, slideshare site and Stanford University



37

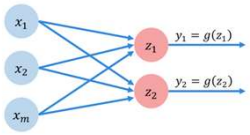
Example 3: Multi-Layer Perceptron



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,2}^{(1)}$$
$$= w_{0,2}^{(1)} + x_1\, w_{1,2}^{(1)} + x_2\, w_{2,2}^{(1)} + x_m\, w_{m,2}^{(1)}$$

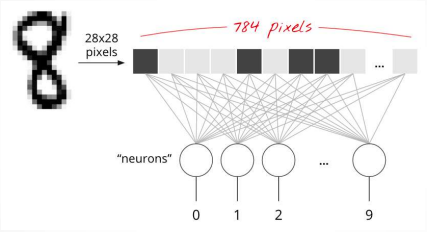
38

Example 4: Linear Softmax Classifier



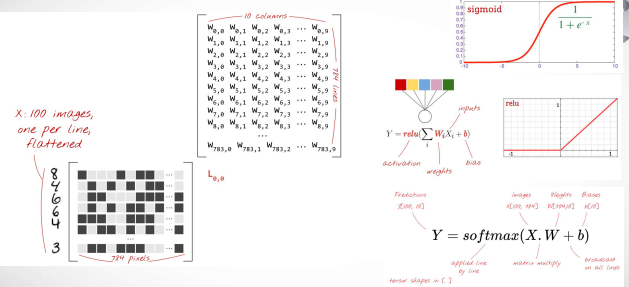
39

Example 4: Linear Softmax Classifier



40

Example 4: Linear Softmax Classifier



$$Y = softmax(X.W + b)$$

41

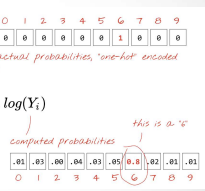
Example 4: Linear Softmax Classifier

$$softmax(L_n) = \frac{e^{L_n}}{||e^{L}||}$$

weighted sum+bias

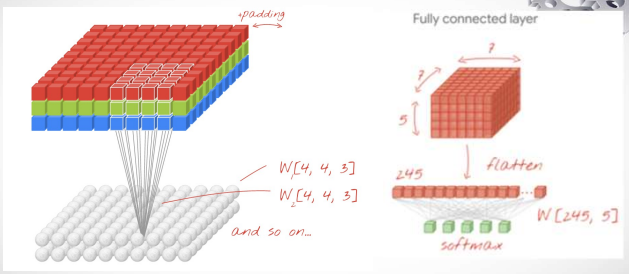
L1 norm

Cross entropy: $-\sum Y_i'.log(Y_i)$



42

Example 5: Convolutional Neural Network



43

Terminology Covered So Far (1)

- batch or mini-batch: training is always performed on batches of training data and labels. Doing so helps the algorithm converge. The "batch" dimension is typically the first dimension of data tensors. For example a tensor of shape [100, 192, 192, 3] contains 100 images of 192x192 pixels with three values per pixel (RGB).
- cross-entropy loss: a special loss function often used in classifiers.
- dense layer: a layer of neurons where each neuron is connected to all the neurons in the previous layer.

44

Terminology Covered So Far (2)

- features: the inputs of a neural network are sometimes called "features". The art of figuring out which parts of a dataset (or combinations of parts) to feed into a neural network to get good predictions is called "feature engineering".
- labels: another name for "classes" or correct answers in a supervised classification problem
- learning rate: fraction of the gradient by which weights and biases are updated at each iteration of the training loop.

45

Terminology Covered So Far (3)

- logits: the outputs of a layer of neurons before the activation function is applied are called "logits". The term comes from the "logistic function" a.k.a. the "sigmoid function" which used to be the most popular activation function. "Neuron outputs before logistic function" was shortened to "logits".
- loss: the error function comparing neural network outputs to the correct answers
- neuron: computes the weighted sum of its inputs, adds a bias and feeds the result through an activation function.
- one-hot encoding: class 3 out of 5 is encoded as a vector of 5 elements, all zeros except the 3rd one which is 1.

46

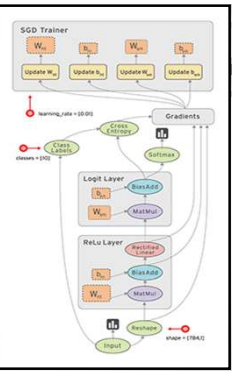
Terminology Covered So Far (4)

- relu: rectified linear unit. A popular activation function for neurons.
- sigmoid: another activation function that used to be popular and is still useful in special cases.
- softmax: a special activation function that acts on a vector, increases the difference between the largest component and all others, and also normalizes the vector to have a sum of 1 so that it can be interpreted as a vector of probabilities. Used as the last step in classifiers.
- tensor: A "tensor" is like a matrix but with an arbitrary number of dimensions. A 1-dimensional tensor is a vector. A 2-dimensions tensor is a matrix. And then you can have tensors with 3, 4, 5 or more dimensions.

47

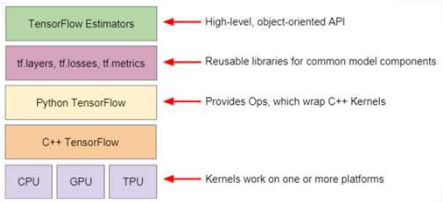
Recap: TensorFlow

*** Credit Google site, slideshare site and Stanford University



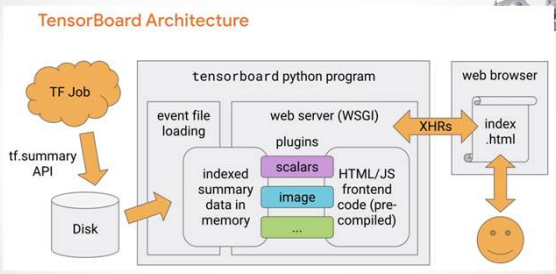
48

TensorFlow Toolkit



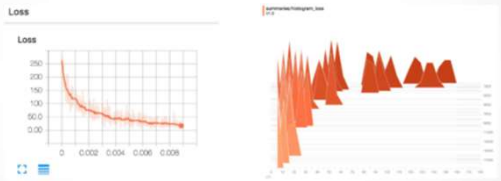
49

TensorBoard



50

Visualize on TensorBoard



51

Q & A

52