TensorFlow (Part 2)

AI Workshop, 2-4 December 2019

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


meme
A meme (/ˈ mi: m/ meem), a neologism coined by Richard Dawkins, is "an idea, behavior, or style that spreads from person to person within a culture". A meme acts as a unit for carrying cultural ideas, symbols, or practices that can be transmitted from one mind to another through writing, speech, gestures, rituals, or other imitable phenomena with a mimicked theme.

Disclaimer

This lecture is compiled from my lectures as well as materials gathering from lectures found in public domains.

Outline

- Machine Learning Practical
  - TensorFlow, Keras and DL
  - Practical: Linear Classifier, MLP, and CNN

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

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.

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.

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.

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.

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.

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.

Building a Model in Keras using Sequential

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

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

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

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

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')
```

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'])
```

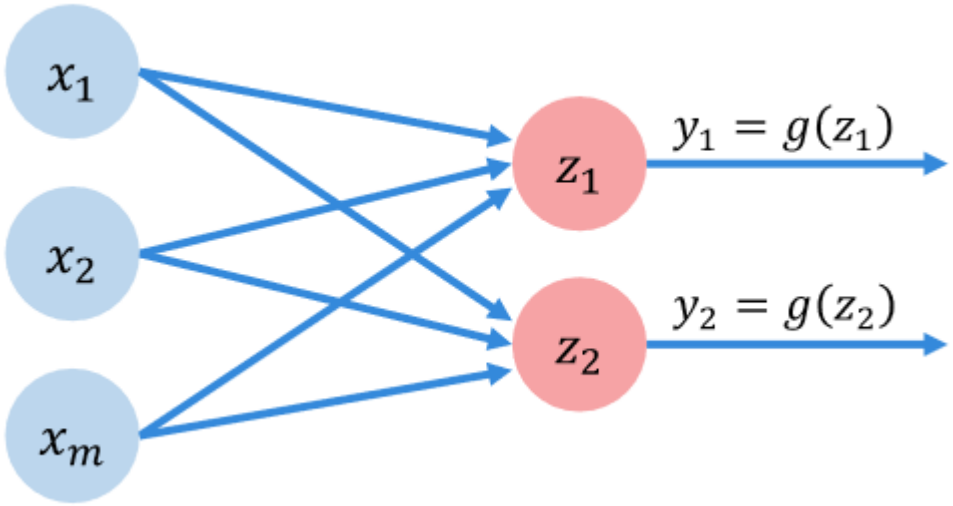
Train, Save and Load Model

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

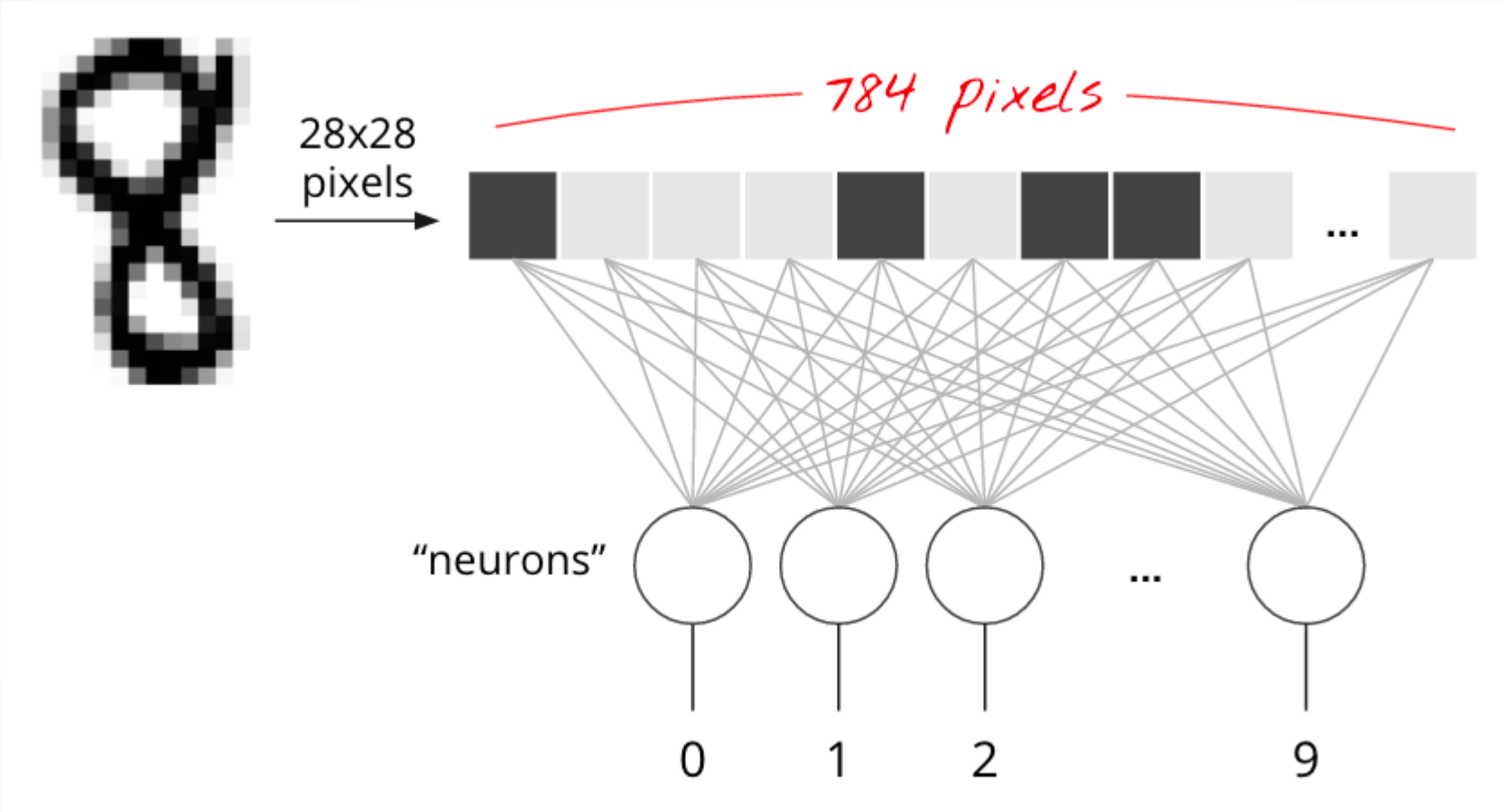
```
# Save entire model to a HDF5 file
model.save('my_model.h5')
```

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

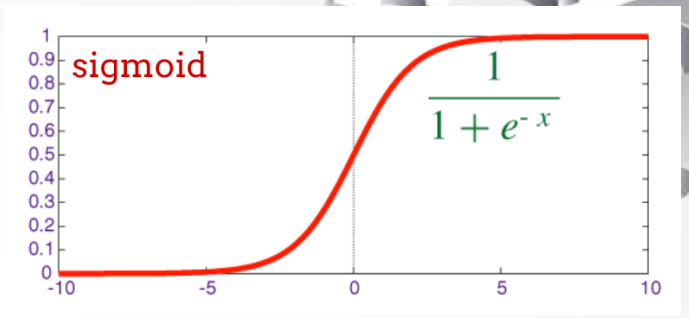
Practical 3: Linear Softmax Classifier



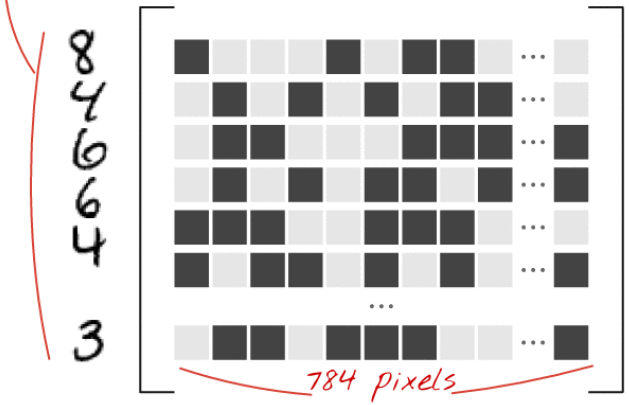
Practical 3: Linear Softmax Classifier

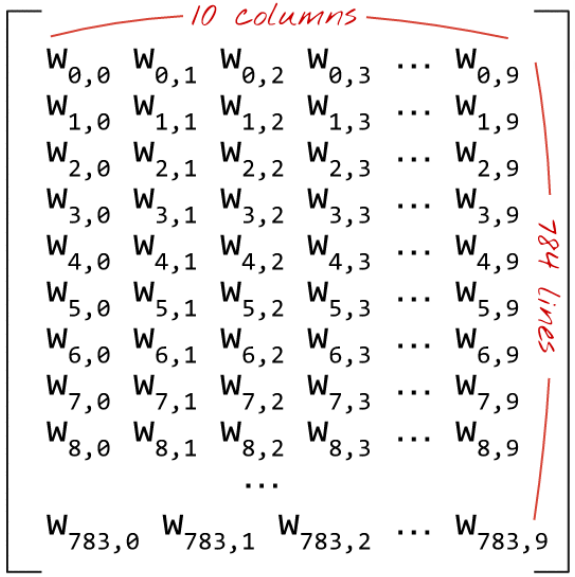


Practical 3: Linear Softmax Classifier

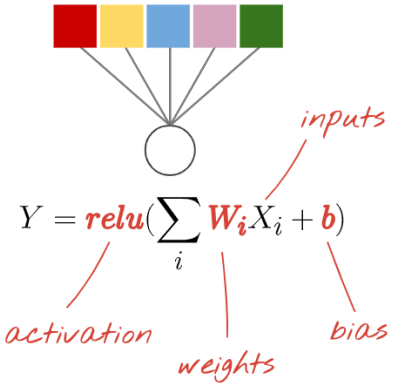


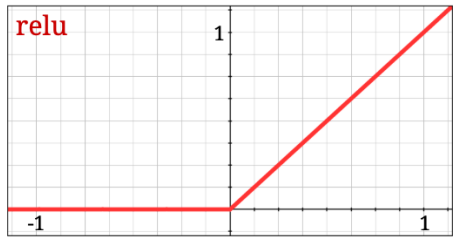
X: 100 images, one per line, flattened





$L_{0,0}$





Predictions Y[100, 10]

Images X[100, 784]

Weights W[784,10]

Biases b[10]

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

applied line by line

matrix multiply

broadcast on all lines

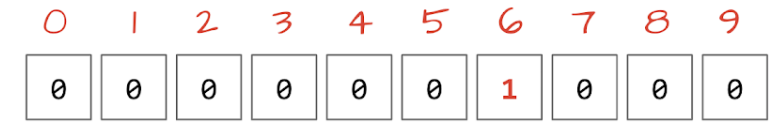
tensor shapes in [ ]

Practical 3: Linear Softmax Classifier

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

weighted sum+bias

L1 norm

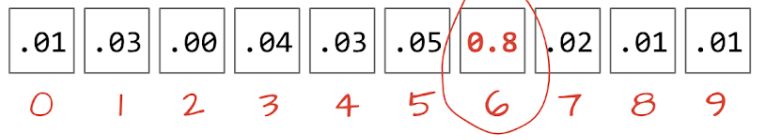


actual probabilities, "one-hot" encoded

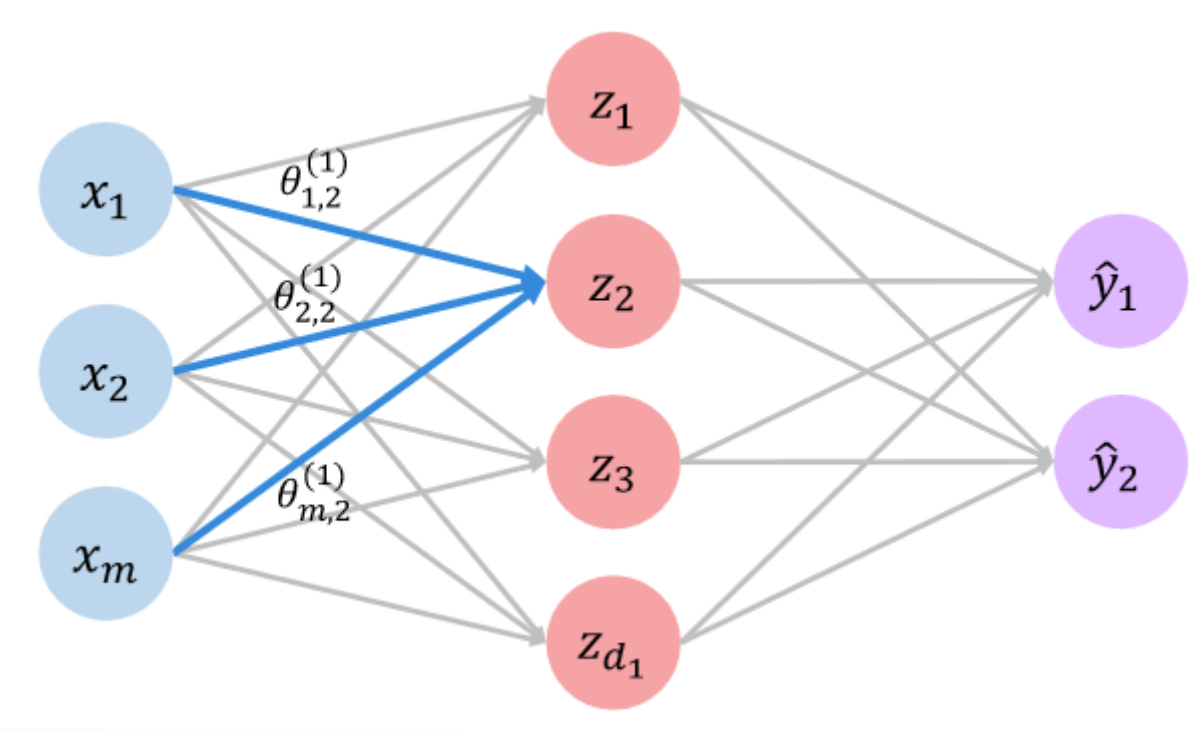
Cross entropy: $-\sum Y'_i . log(Y_i)$

computed probabilities

this is a "6"



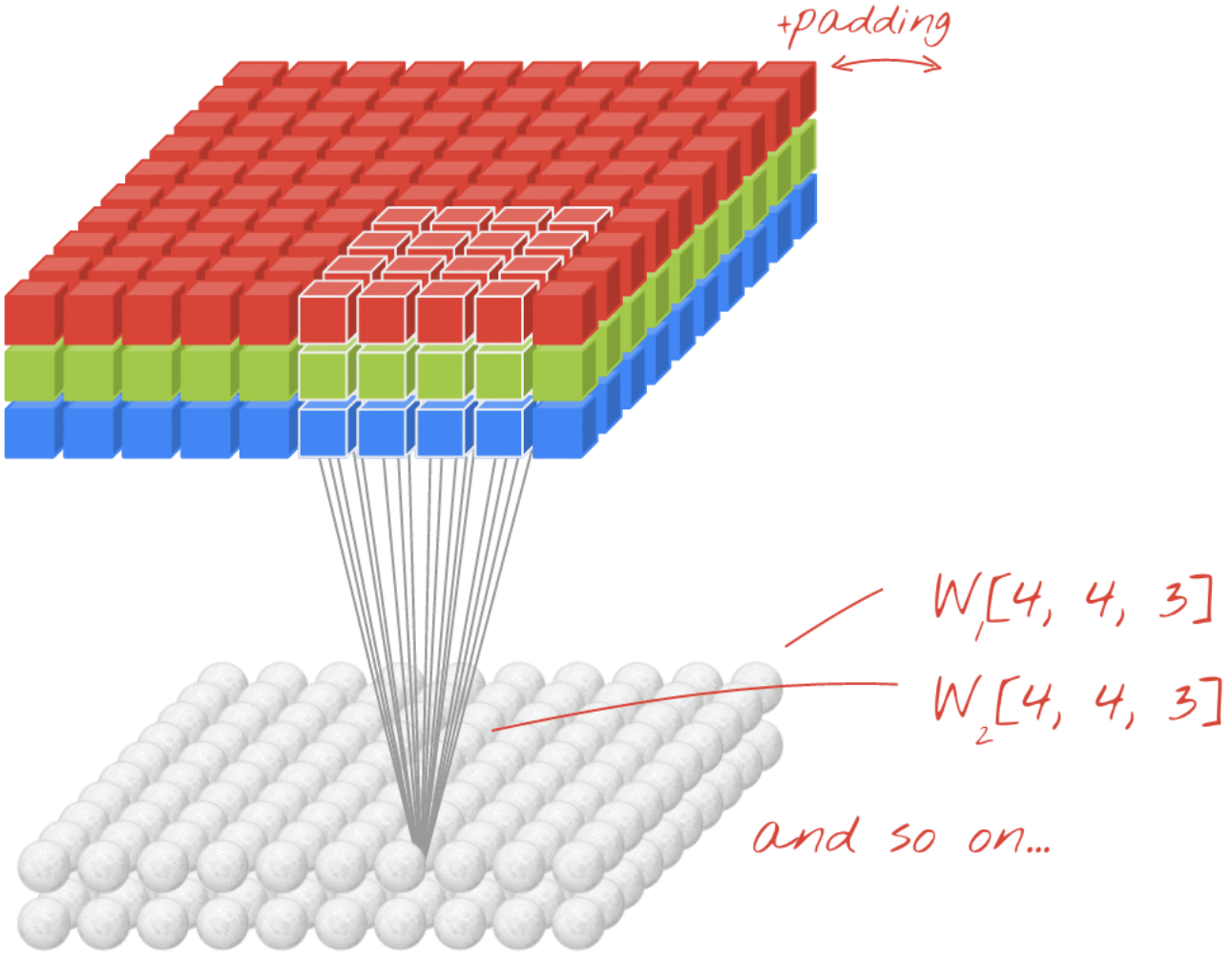
Practical 4: Multi-Layer Perceptron

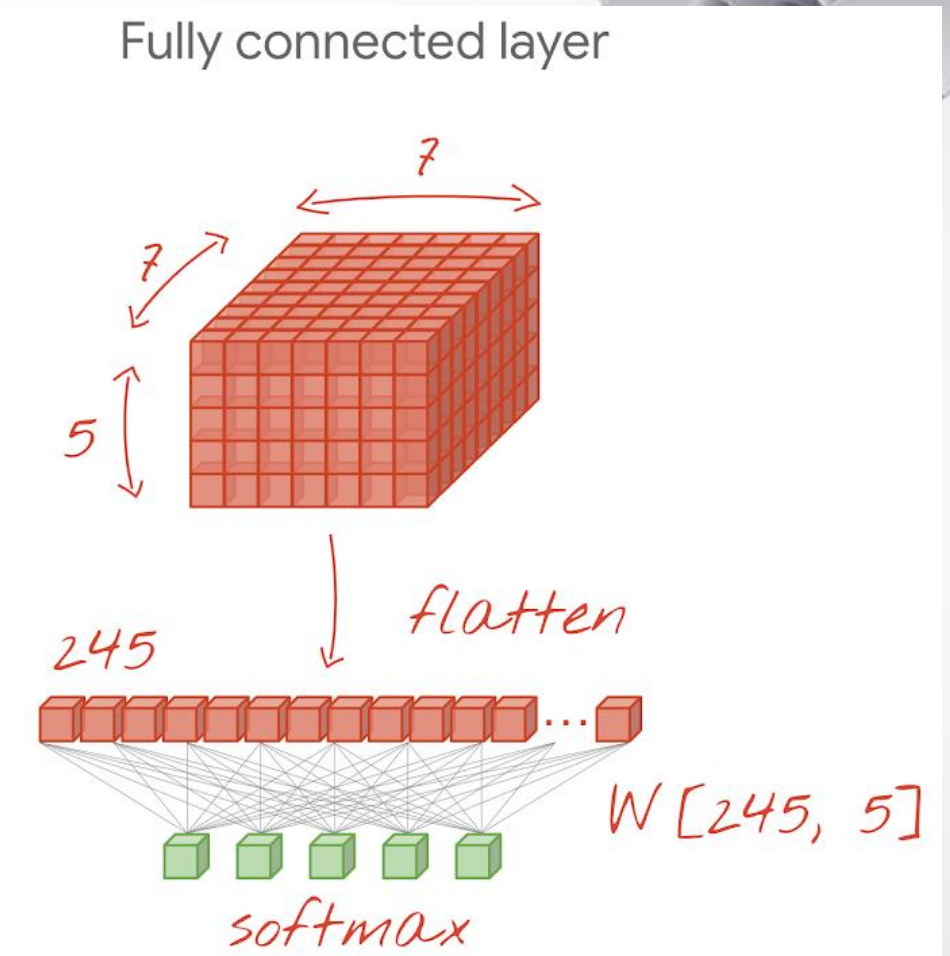


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

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

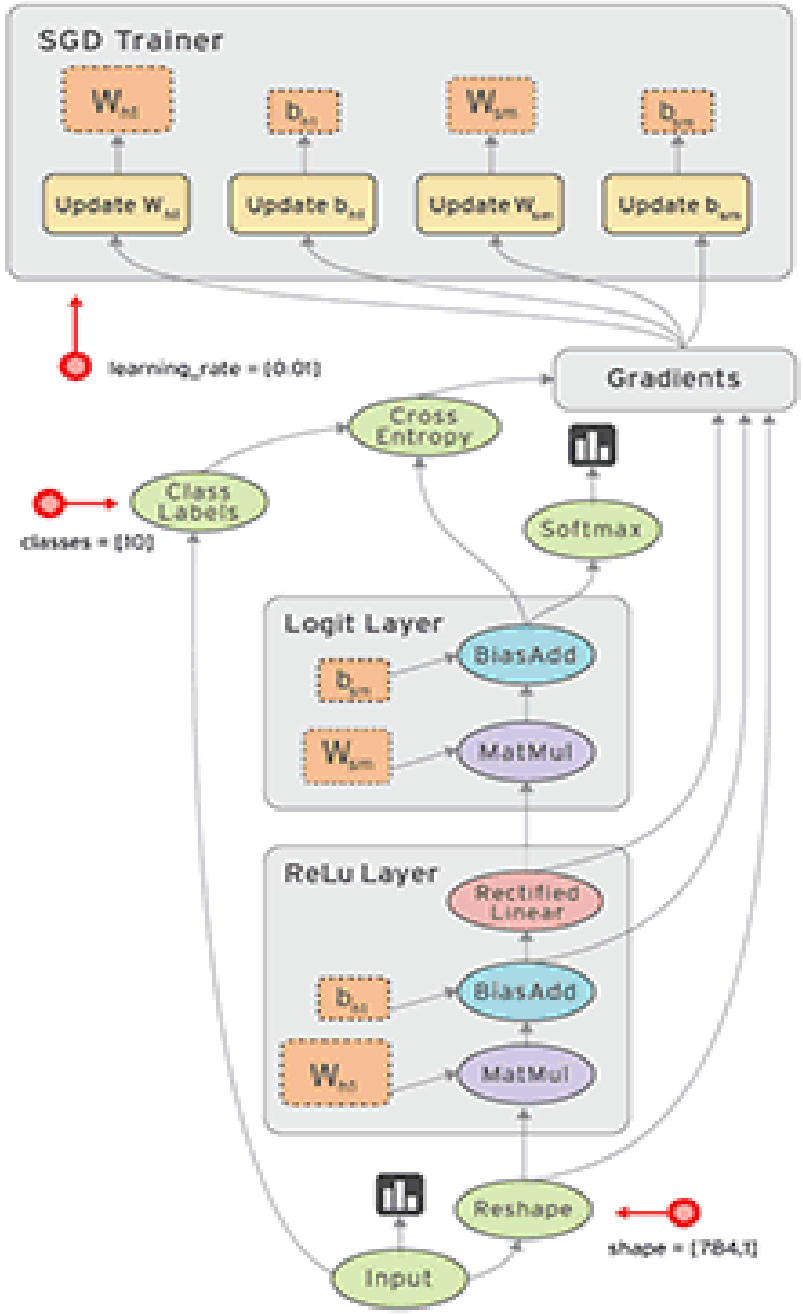
Practical 5: Convolutional Neural Network



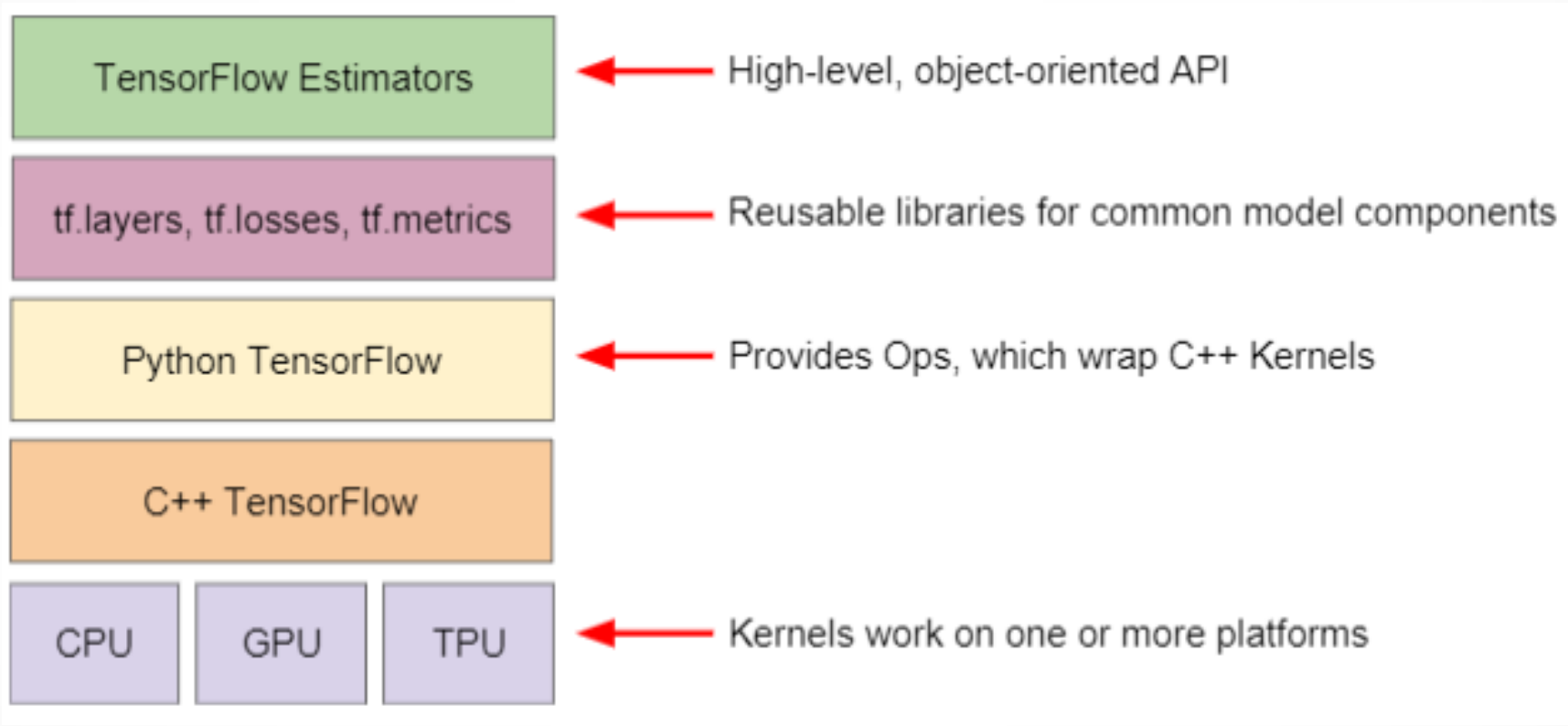


Recap: TensorFlow

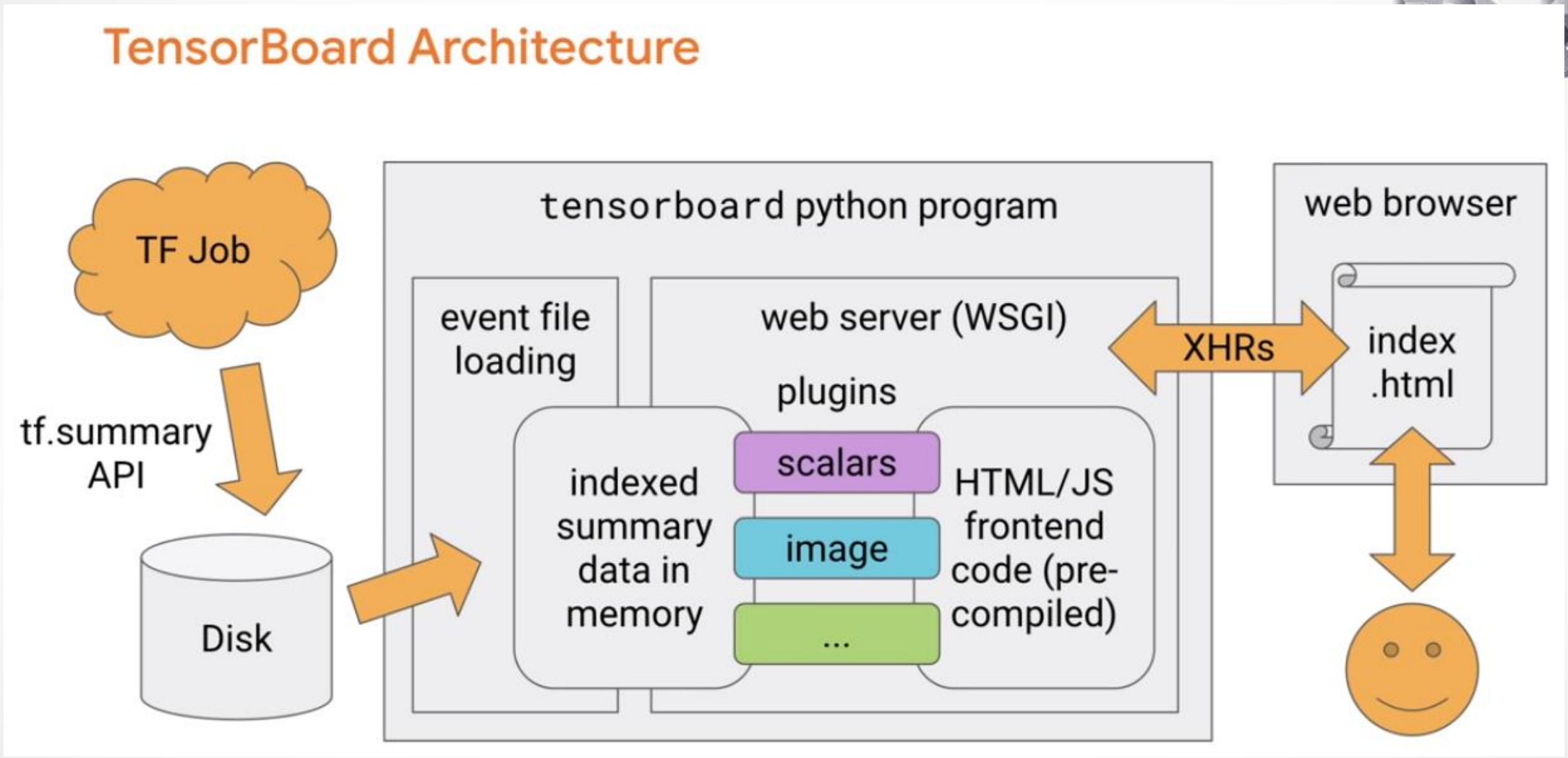
*** Credit Google site, slideshare site and Stanford University



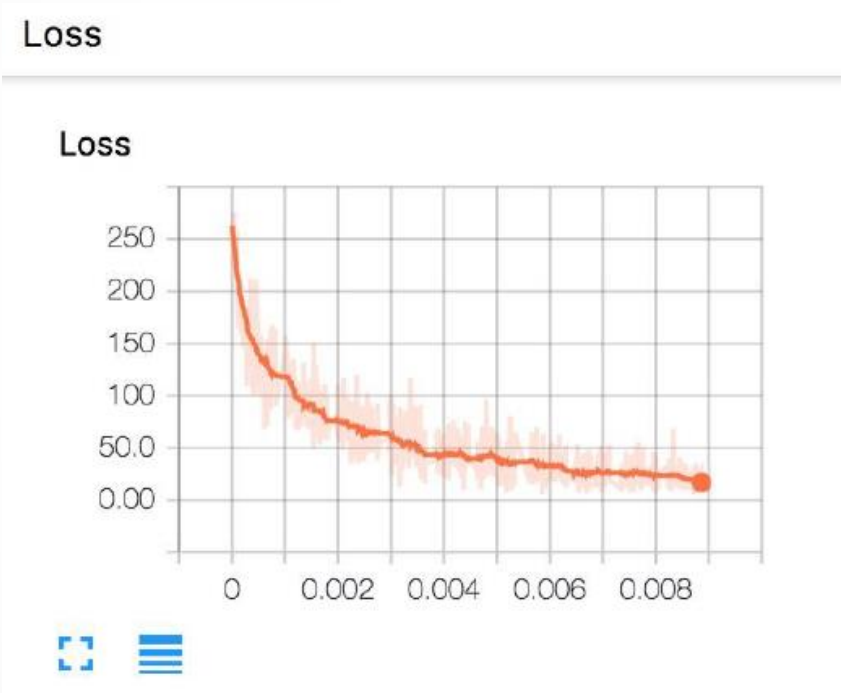
TensorFlow Toolkit

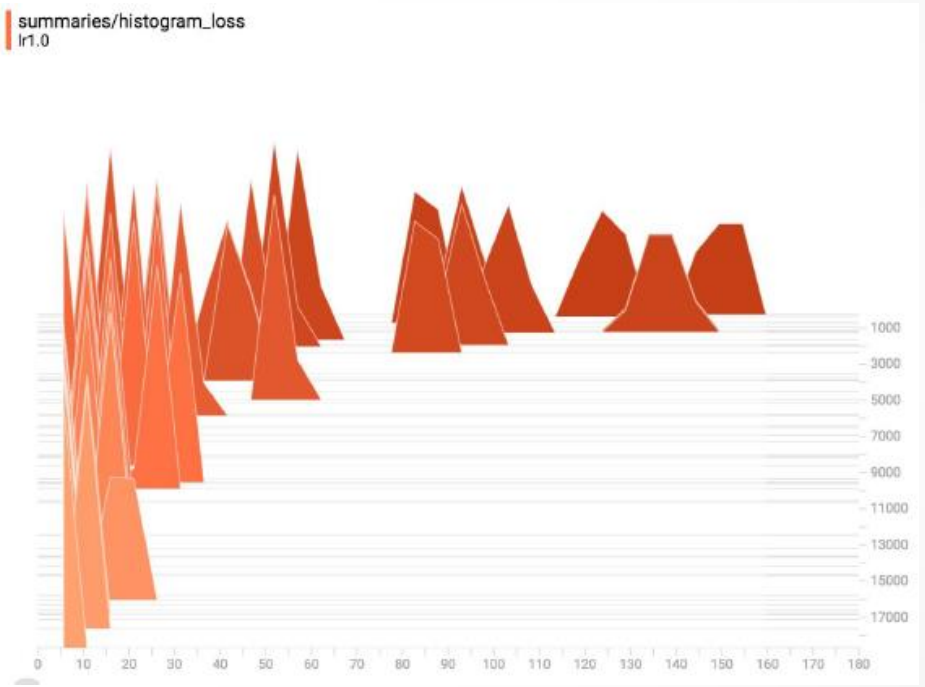


TensorBoard

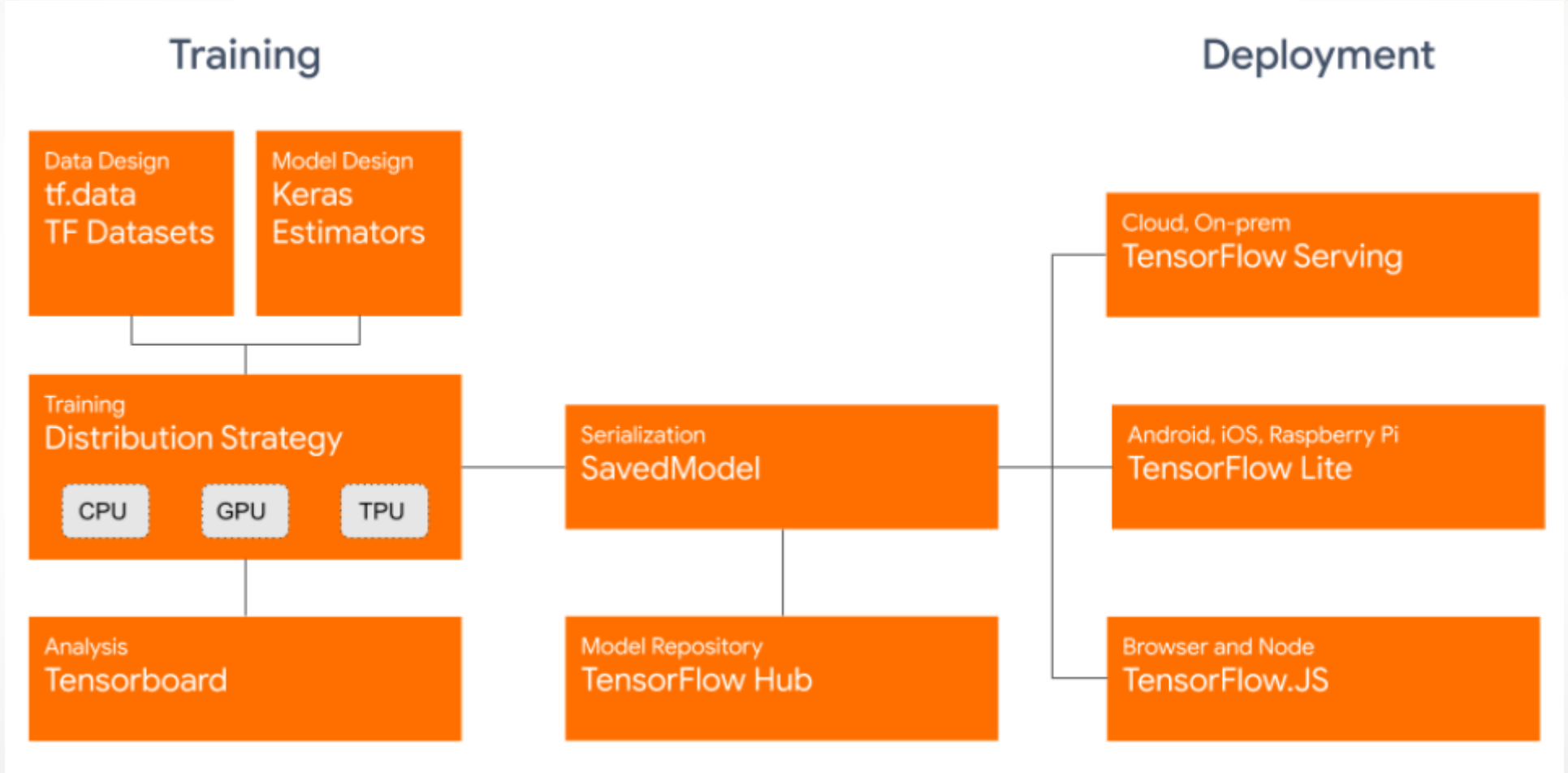


Visualize on TensorBoard



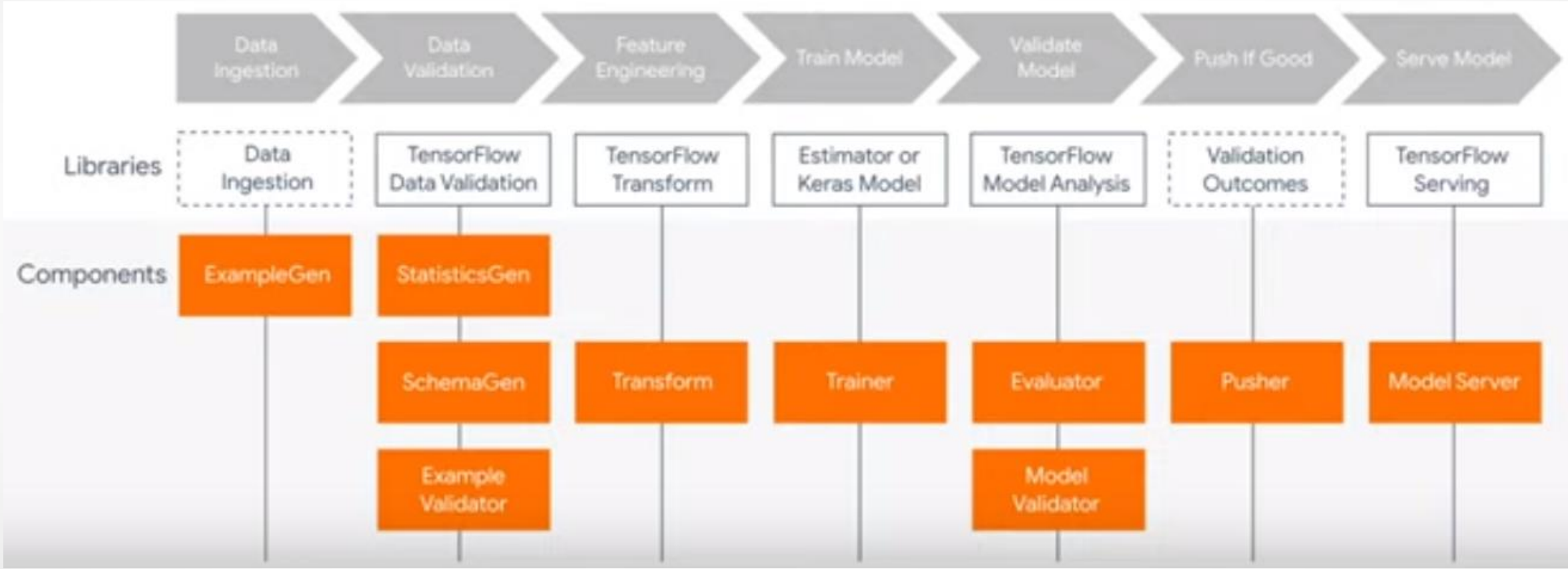


Production Pipeline

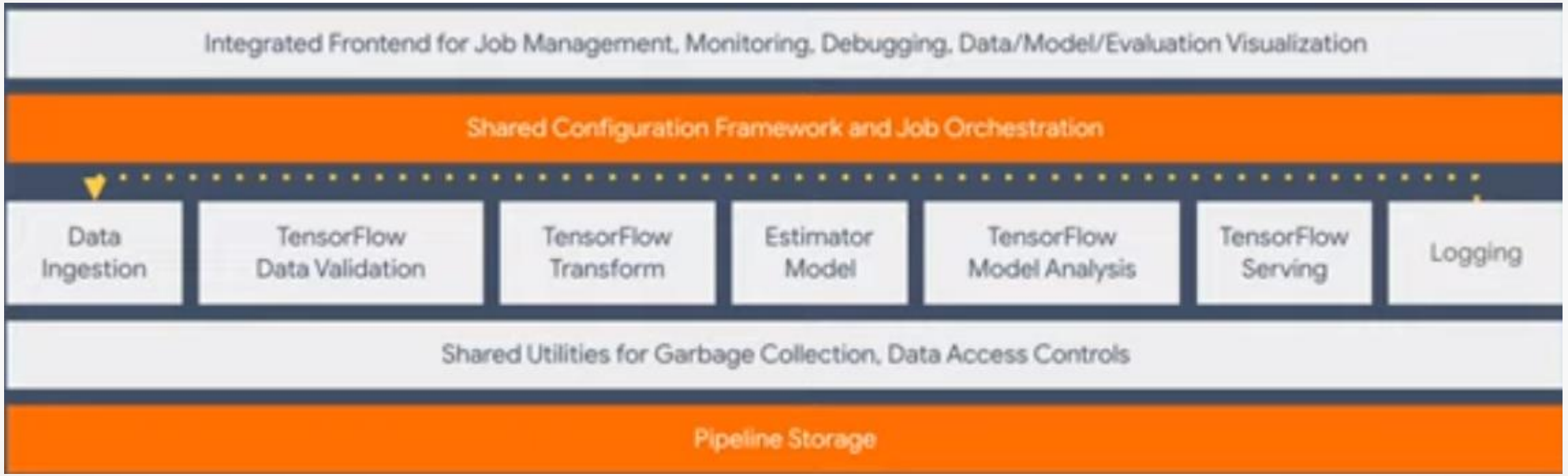


Production Pipeline

Production Pipeline



Production Pipeline



Q & A