TensorLayer 2.0

Hao Dong

Peking University

PEKING UNIVERSITY

Background
- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information

How to Use Better
- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

PEKING UNIVERSITY

Background
- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information

How to Use Better
- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

History of Deep Learning Tools

北京大學 PEKING UNIVERSITY

- Automatic Differentiation

TensorFlow™

Key reasons for TensorFlow

- Largest user base
- Widest production adoption
- Well-maintained documents
- Battlefield-proof quality
- TPU !

PYTORCH

Microsoft CNTK

Caffe2

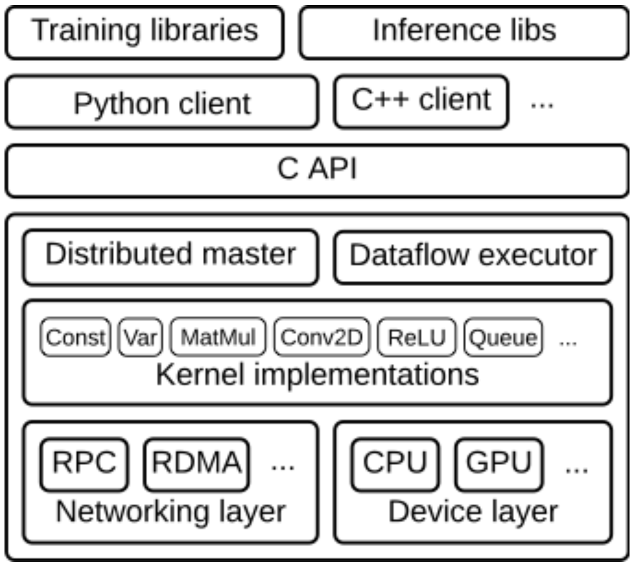
mxnet

PaddlePaddle

History of Deep Learning Tools

- Beyond Automatic Differentiation

Low-level interface: dataflow graph, placeholder, session, queue runner, devices …

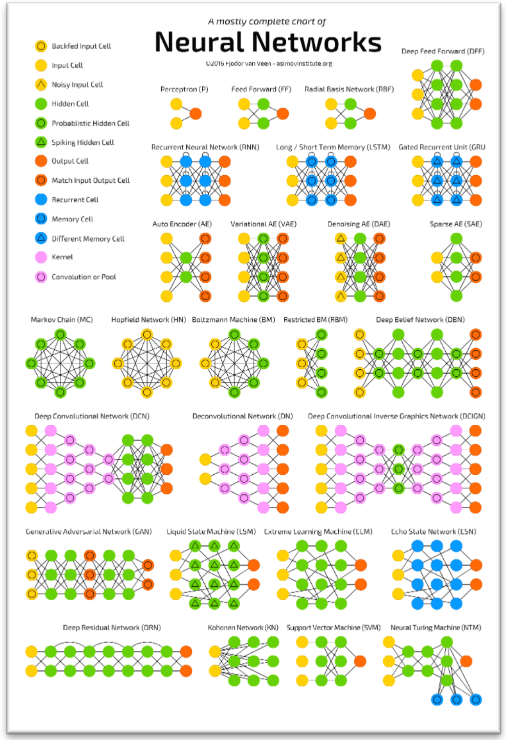


https://www.tensorflow.org/extend/architecture

Abstraction gap

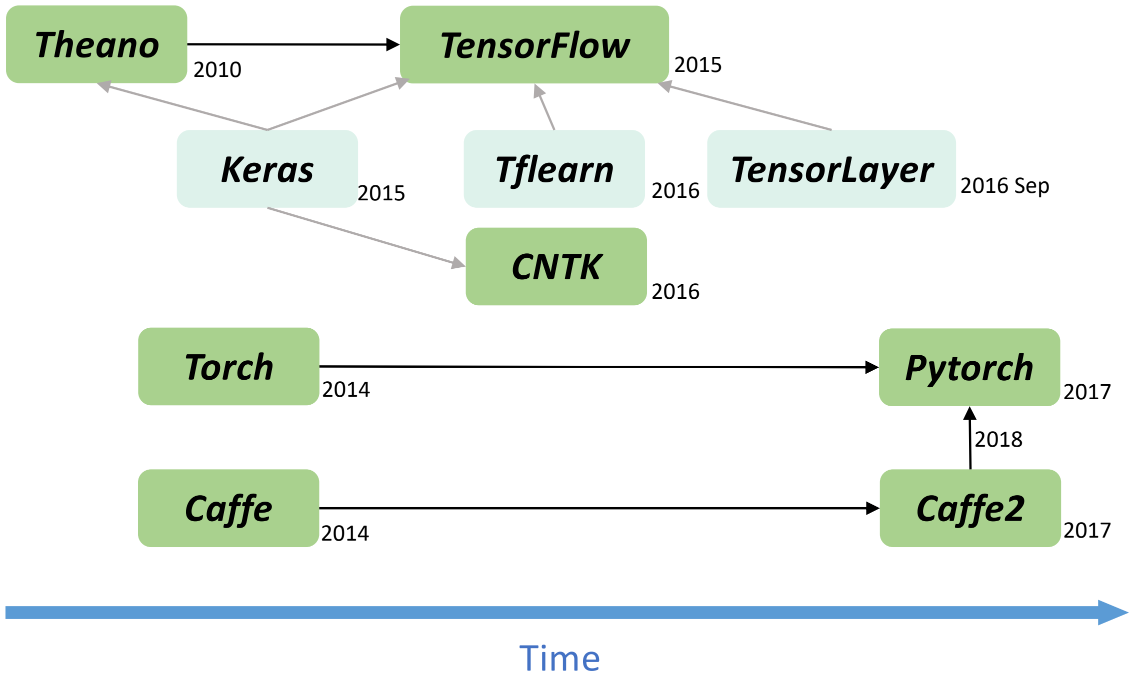
Bridged by wrappers: TensorLayer, Keras and TFLearn

Deep learning high-level elements: neural networks, layers and tensors

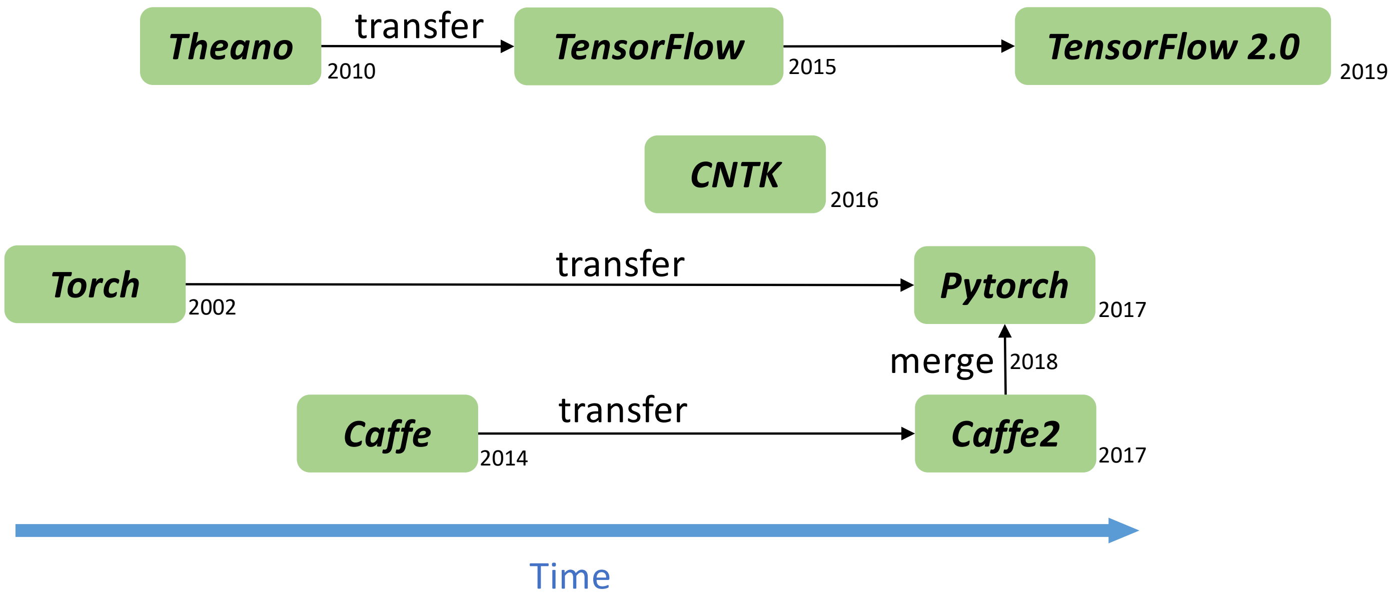


http://www.asimovinstitute.org/neural-network-zoo/

History of Deep Learning Tools



History of Deep Learning Tools



北京大学
PEKING UNIVERSITY

Background
- History of Deep Learning Tools
- **History of TensorLayer**
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information

- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

History of TensorLayer

2016

2019

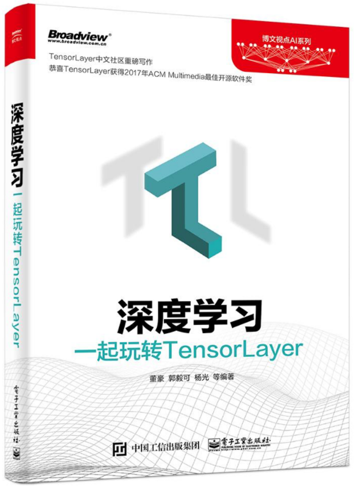
Time

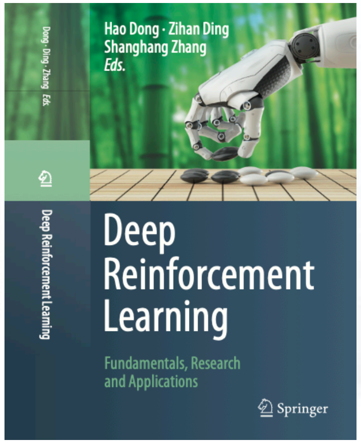
2015

2019

History of TensorLayer

- TensorLayer 2.0





ACM Special Interest Group on Multimedia

presents to

Hao Dong, Akara Supratak, Luo Mai, Fangde Liu, Axel Oehmichen, Simiao Yu, Yike Guo

Best Open Source Software Award 2017

"TensorLayer: A Versatile Library for Efficient Deep Learning Development"

October, 2017

TensorLayer: A Versatile Library for Efficient Deep Learning Development. *H. Dong, A. Supratak et al. ACM MM 2017.*

History of TensorLayer

Documentation (English)

Github

Background

- History of Deep Learning Tools
- History of TensorLayer
- **Future of TensorLayer**

How to Use

- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information

- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

PEKING UNIVERSITY

Future of TensorLayer

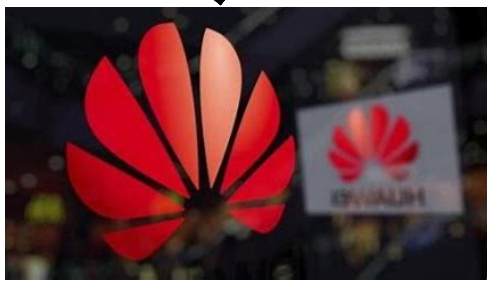
- Platform-agnostic
- Research oriented

TensorLayer

TensorFlow

MindSpore

...



GPU

TPU

Huawei 晟腾 ShengTeng ...

Future of TensorLayer

北京大學 PEKING UNIVERSITY

For TensorLayer 2.x, it is now actively developing and maintaining by the following people who has more than 50 contributions:

- **Hao Dong** (@zsdonghao) - https://zsdonghao.github.io
- **Jingqing Zhang** (@JingqingZ) - https://jingqingz.github.io
- **Rundi Wu** (@ChrisWu1997) - http://chriswu1997.github.io
- **Ruihai Wu** (@warshallrho) - https://warshallrho.github.io/

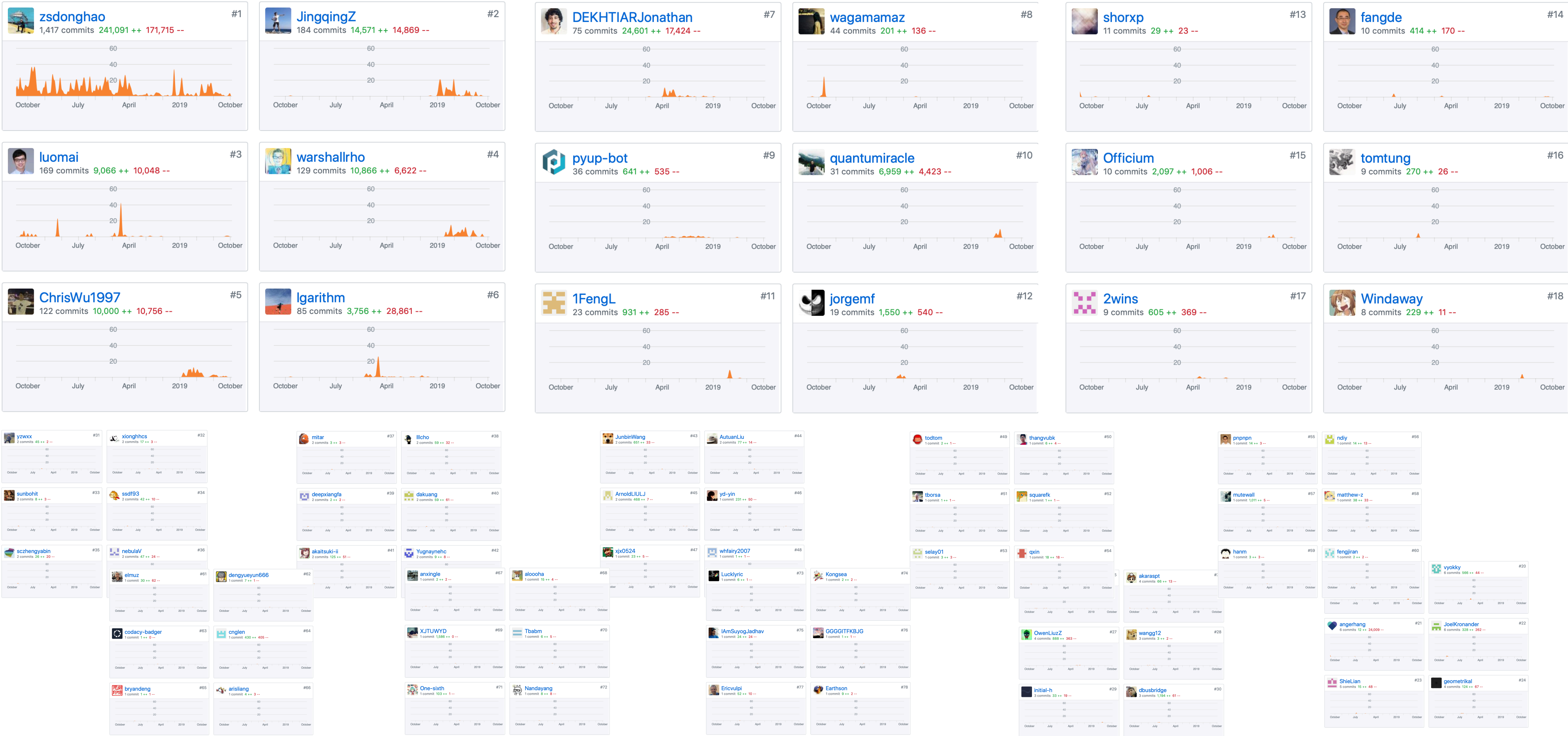
For TensorLayer 1.x, it was actively developed and maintained by the following people *(in alphabetical order)*:

- **Akara Supratak** (@akaraspt) - https://akaraspt.github.io
- **Fangde Liu** (@fangde) - http://fangde.github.io/
- **Guo Li** (@lgarithm) - https://lgarithm.github.io
- **Hao Dong** (@zsdonghao) - https://zsdonghao.github.io
- **Jonathan Dekhtiar** (@DEKHTIARJonathan) - https://www.jonathandekhtiar.eu
- **Luo Mai** (@luomai) - http://www.doc.ic.ac.uk/~lm111/
- **Simiao Yu** (@nebulaV) - https://nebulav.github.io

Numerous other contributors can be found in the Github Contribution Graph.

Contributors

Future of TensorLayer



Background

- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use

- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information
- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

Static vs. Dynamic Models

PEKING UNIVERSITY

Static Model

```
import tensorflow as tf
from tensorlayer.layers import Input, Dropout, Dense
from tensorlayer.models import Model

def get_model(inputs_shape):
    ni = Input(inputs_shape)
    nn = Dropout(keep=0.8)(ni)
    nn = Dense(n_units=800, act=tf.nn.relu, name="dense1")(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=800, act=tf.nn.relu)(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=10, act=tf.nn.relu)(nn)
    M = Model(inputs=ni, outputs=nn, name="mlp")
    return M

MLP = get_model([None, 784])
MLP.eval()
outputs = MLP(data)
```

Lasagne Fashion

Static vs. Dynamic Models

北京大學 PEKING UNIVERSITY

Dynamic Model

```
class CustomModel(Model):

    def __init__(self):
        super(CustomModel, self).__init__()

        self.dropout1 = Dropout(keep=0.8)
        self.dense1 = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
        self.dropout2 = Dropout(keep=0.8)#(self.dense1)
        self.dense2 = Dense(n_units=800, act=tf.nn.relu, in_channels=800)
        self.dropout3 = Dropout(keep=0.8)#(self.dense2)
        self.dense3 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)

    def forward(self, x, foo=False):
        z = self.dropout1(x)
        z = self.dense1(z)
        z = self.dropout2(z)
        z = self.dense2(z)
        z = self.dropout3(z)
        out = self.dense3(z)
        if foo:
            out = tf.nn.relu(out)
        return out

MLP = CustomModel()
MLP.eval()
outputs = MLP(data, foo=True) # controls the forward here
outputs = MLP(data, foo=False)
```

Chainer Fashion

Static vs. Dynamic Models

PEKING UNIVERSITY

Static Model

```python
import tensorflow as tf
from tensorlayer.layers import Input, Dropout, Dense
from tensorlayer.models import Model

def get_model(inputs_shape):
    ni = Input(inputs_shape)
    nn = Dropout(keep=0.8)(ni)
    nn = Dense(n_units=800, act=tf.nn.relu, name="dense1")(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=800, act=tf.nn.relu)(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=10, act=tf.nn.relu)(nn)
    M = Model(inputs=ni, outputs=nn, name="mlp")
    return M

MLP = get_model([None, 784])
MLP.eval()
outputs = MLP(data)
```

Lasagne Fashion

Dynamic Model

```python
class CustomModel(Model):

    def __init__(self):
        super(CustomModel, self).__init__()

        self.dropout1 = Dropout(keep=0.8)
        self.dense1 = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
        self.dropout2 = Dropout(keep=0.8)#(self.dense1)
        self.dense2 = Dense(n_units=800, act=tf.nn.relu, in_channels=800)
        self.dropout3 = Dropout(keep=0.8)#(self.dense2)
        self.dense3 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)

    def forward(self, x, foo=False):
        z = self.dropout1(x)
        z = self.dense1(z)
        z = self.dropout2(z)
        z = self.dense2(z)
        z = self.dropout3(z)
        out = self.dense3(z)
        if foo:
            out = tf.nn.relu(out)
        return out

MLP = CustomModel()
MLP.eval()
outputs = MLP(data, foo=True) # controls the forward here
outputs = MLP(data, foo=False)
```

Chainer Fashion

北京大學 PEKING UNIVERSITY

Background
- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- **Switching Train/Test Modes**
- Reuse Weights
- Model Information
- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

Switching Train/Test Models

PEKING UNIVERSITY

```
# method 1: switch before forward
Model.train() # enable dropout, batch norm moving avg ...
output = Model(train_data)
... # training code here
Model.eval()  # disable dropout, batch norm moving avg ...
output = Model(test_data)
... # testing code here

# method 2: switch while forward
output = Model(train_data, is_train=True)
output = Model(test_data, is_train=False)
```

北京大學 PEKING UNIVERSITY

Background
- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- Switching Train/Test Modes
- **Reuse Weights**
- Model Information
- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

Reuse Weights

PEKING UNIVERSITY 北京大學

Reuse Weights in Static Model

```
def create_base_network(input_shape):
        '''''Base network to be shared (eq. to feature extraction).
        '''
        input = Input(shape=input_shape)
        x = Flatten()(input)
        x = Dense(128, act=tf.nn.relu)(x)
        x = Dropout(0.9)(x)
        x = Dense(128, act=tf.nn.relu)(x)
        x = Dropout(0.9)(x)
        x = Dense(128, act=tf.nn.relu)(x)
        return Model(input, x)


def get_siamese_network(input_shape):
        """Create siamese network with shared base network as layer
        """
        base_layer = create_base_network(input_shape).as_layer() # convert model as layer

        ni_1 = Input(input_shape)
        ni_2 = Input(input_shape)
        nn_1 = base_layer(ni_1) # call base_layer twice
        nn_2 = base_layer(ni_2)
        return Model(inputs=[ni_1, ni_2], outputs=[nn_1, nn_2])

siamese_net = get_siamese_network([None, 784])
```

Reuse Weights

Reuse Weights in Dynamic Model

```
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense_shared = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
        self.dense1 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)
        self.dense2 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)
        self.cat = Concat()

    def forward(self, x):
        x1 = self.dense_shared(x) # call dense_shared twice
        x2 = self.dense_shared(x)
        x1 = self.dense1(x1)
        x2 = self.dense2(x2)
        out = self.cat([x1, x2])
        return out

model = MyModel()
```

Reuse Weights

Reuse Weights in Static Model

```python
def create_base_network(input_shape):
    '''Base network to be shared (eq. to feature extraction).
    '''
    input = Input(shape=input_shape)
    x = Flatten()(input)
    x = Dense(128, act=tf.nn.relu)(x)
    x = Dropout(0.9)(x)
    x = Dense(128, act=tf.nn.relu)(x)
    x = Dropout(0.9)(x)
    x = Dense(128, act=tf.nn.relu)(x)
    return Model(input, x)


def get_siamese_network(input_shape):
    """Create siamese network with shared base network as layer
    """
    base_layer = create_base_network(input_shape).as_layer() # convert model as layer

    ni_1 = Input(input_shape)
    ni_2 = Input(input_shape)
    nn_1 = base_layer(ni_1) # call base_layer twice
    nn_2 = base_layer(ni_2)
    return Model(inputs=[ni_1, ni_2], outputs=[nn_1, nn_2])

siamese_net = get_siamese_network([None, 784])
```

Reuse Weights in Dynamic Model

```python
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense_shared = Dense(n_units=800, act=tf.nn.relu, in_channels=784)
        self.dense1 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)
        self.dense2 = Dense(n_units=10, act=tf.nn.relu, in_channels=800)
        self.cat = Concat()

    def forward(self, x):
        x1 = self.dense_shared(x) # call dense_shared twice
        x2 = self.dense_shared(x)
        x1 = self.dense1(x1)
        x2 = self.dense2(x2)
        out = self.cat([x1, x2])
        return out

model = MyModel()
```

Background

- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use

- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- **Model Information**

- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

Model Information

Print Model Architecture

```python
import pprint

def get_model(inputs_shape):
    ni = Input(inputs_shape)
    nn = Dropout(keep=0.8)(ni)
    nn = Dense(n_units=800, act=tf.nn.relu)(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=800, act=tf.nn.relu)(nn)
    nn = Dropout(keep=0.8)(nn)
    nn = Dense(n_units=10, act=tf.nn.relu)(nn)
    M = Model(inputs=ni, outputs=nn, name="mlp")
    return M

MLP = get_model([None, 784])
pprint.pprint(MLP.config)
```


```
[{'args': {'dtype': tf.float32,
     'layer_type': 'normal',
     'name': '_inputlayer_1',
     'shape': [None, 784]},
 'class': '_InputLayer',
 'prev_layer': None},
{'args': {'keep': 0.8, 'layer_type': 'normal', 'name': 'dropout_1'},
 'class': 'Dropout',
 'prev_layer': ['_inputlayer_1_node_0']},
{'args': {'act’: ‘relu',
     'layer_type': 'normal',
     'n_units': 800,
     'name': 'dense_1'},
 'class': 'Dense',
 'prev_layer': ['dropout_1_node_0']},
{'args': {'keep': 0.8, 'layer_type': 'normal', 'name': 'dropout_2'},
 'class': 'Dropout',
 'prev_layer': ['dense_1_node_0']},
{'args': {'act': ‘relu',
     'layer_type': 'normal',
     'n_units': 800,
     'name': 'dense_2'},
 'class': 'Dense',
 'prev_layer': ['dropout_2_node_0']},
{'args': {'keep': 0.8, 'layer_type': 'normal', 'name': 'dropout_3'},
 'class': 'Dropout',
 'prev_layer': ['dense_2_node_0']},
{'args': {'act’: ‘relu',
     'layer_type': 'normal',
     'n_units': 10,
     'name': 'dense_3'},
 'class': 'Dense',
 'prev_layer': ['dropout_3_node_0']}]
```

Model Information

北京大學 PEKING UNIVERSITY

Print Model Information

```
print(MLP) # simply call print function

# Model(
#    (_inputlayer): Input(shape=[None, 784], name='_inputlayer')
#    (dropout): Dropout(keep=0.8, name='dropout')
#    (dense): Dense(n_units=800, relu, in_channels='784', name='dense')
#    (dropout_1): Dropout(keep=0.8, name='dropout_1')
#    (dense_1): Dense(n_units=800, relu, in_channels='800', name='dense_1')
#    (dropout_2): Dropout(keep=0.8, name='dropout_2')
#    (dense_2): Dense(n_units=10, relu, in_channels='800', name='dense_2')
# )
```

Save Weights Only

```
MLP.save_weights('./model_weights.h5')
MLP.load_weights('./model_weights.h5')
```

Get Specific Weights

```
# indexing
all_weights = MLP.all_weights
some_weights = MLP.all_weights[1:3]

# naming
some_weights = MLP.get_layer('dense1').all_weights
```

.trainable_weights
.nontrainable_weights

Save Weights + Architecture

```
MLP.save('./model.h5', save_weights=True)
MLP = Model.load('./model.h5', load_weights=True)
```

北京大學 PEKING UNIVERSITY

Background
- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information
- **Customize Layer without Weights**
- Customize Layer with Weights
- Dataflow
- Distributed Training

Customize Layer without Weights

```python
class Dropout(Layer):
    """
    The :class:`Dropout` class is a noise layer which randomly set some
    activations to zero according to a keeping probability.
    Parameters
    ----------
    keep : float
        The keeping probability.
        The lower the probability it is, the more activations are set to zero.
    name : None or str
        A unique layer name.
    """

    def __init__(self, keep, name=None):
        super(Dropout, self).__init__(name)
        self.keep = keep

        self.build()
        self._built = True

        logging.info("Dropout %s: keep: %f " % (self.name, self.keep))

    def build(self, inputs_shape=None):
        pass    # no weights in dropout layer

    def forward(self, inputs):
        if self.is_train:  # this attribute is changed by Model.train() and Model.eval()
            outputs = tf.nn.dropout(inputs, rate=1 - (self.keep), name=self.name)
        else:
            outputs = inputs
        return outputs
```

Background

- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use

- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information
- Customize Layer without Weights
- **Customize Layer with Weights**
- Dataflow
- Distributed Training

Customize Layer with Weights

```python
class Dense(Layer):
  """The :class:`Dense` class is a fully connected layer.

  Parameters
  ----------
  n_units : int
      The number of units of this layer.
  act : activation function
      The activation function of this layer.
  name : None or str
      A unique layer name. If None, a unique name will be automatically generated.
  """

  def __init__(
          self,
          n_units,   # the number of units/channels of this layer
          act=None,  # None: no activation, tf.nn.relu: ReLU ...
          name=None, # the name of this layer (optional)
  ):
      super(Dense, self).__init__(name) # auto naming, dense_1, dense_2 ...
      self.n_units = n_units
      self.act = act

  def build(self, inputs_shape): # initialize the model weights here
      shape = [inputs_shape[1], self.n_units]
      self.W = self._get_weights("weights", shape=tuple(shape), init=self.W_init)
      self.b = self._get_weights("biases", shape=(self.n_units, ), init=self.b_init)

  def forward(self, inputs): # call function
      z = tf.matmul(inputs, self.W) + self.b
      if self.act: # is not None
          z = self.act(z)
      return z
```

Customize Layer with Weights

```python
class Dense(Layer):
    """The :class:`Dense` class is a fully connected layer.

    Parameters
    ----------
    n_units : int
        The number of units of this layer.
    act : activation function
        The activation function of this layer.
    W_init : initializer
        The initializer for the weight matrix.
    b_init : initializer or None
        The initializer for the bias vector. If None, skip biases.
    in_channels: int
        The number of channels of the previous layer.
        If None, it will be automatically detected when the layer is forwarded for the first time.
    name : None or str
        A unique layer name. If None, a unique name will be automatically generated.
    """

    def __init__(
            self,
            n_units,
            act=None,
            W_init=tl.initializers.truncated_normal(stddev=0.1),
            b_init=tl.initializers.constant(value=0.0),
            in_channels=None,  # the number of units/channels of the previous layer
            name=None,
    ):
        # we feed activation function to the base layer, `None` denotes identity function
        # string (e.g., relu, sigmoid) will be converted into function.
        super(Dense, self).__init__(name, act=act)

        self.n_units = n_units
        self.W_init = W_init
        self.b_init = b_init
        self.in_channels = in_channels

        # in dynamic model, the number of input channel is given, we initialize the weights here
        if self.in_channels is not None:
            self.build(self.in_channels)
            self._built = True

        logging.info(
            "Dense  %s: %d %s" %
            (self.name, self.n_units, self.act.__name__ if self.act is not None else 'No Activation')
        )
    def build(self, inputs_shape): # initialize the model weights here
        if self.in_channels: # if the number of input channel is given, use it
            shape = [self.in_channels, self.n_units]
        else:                # otherwise, get it from static model
            self.in_channels = inputs_shape[1]
            shape = [inputs_shape[1], self.n_units]
        self.W = self._get_weights("weights", shape=tuple(shape), init=self.W_init)
        if self.b_init:      # if b_init is None, no bias is applied
            self.b = self._get_weights("biases", shape=(self.n_units, ), init=self.b_init)

    def forward(self, inputs):
        z = tf.matmul(inputs, self.W)
        if self.b_init:
            z = tf.add(z, self.b)
        if self.act:
            z = self.act(z)
        return z
```


```python
        # we feed activation function to the base layer, `None` denotes identity function
        # string (e.g., relu, sigmoid) will be converted into function.
        super(Dense, self).__init__(name, act=act)

        self.n_units = n_units
        self.W_init = W_init
        self.b_init = b_init
        self.in_channels = in_channels

        # in dynamic model, the number of input channel is given, we initialize the weights here
        if self.in_channels is not None:
            self.build(self.in_channels)
            self._built = True

        logging.info(
            "Dense  %s: %d %s" %
            (self.name, self.n_units, self.act.__name__ if self.act is not None else 'No Activation')
        )
    def build(self, inputs_shape): # initialize the model weights here
        if self.in_channels: # if the number of input channel is given, use it
            shape = [self.in_channels, self.n_units]
        else:                # otherwise, get it from static model
            self.in_channels = inputs_shape[1]
            shape = [inputs_shape[1], self.n_units]
        self.W = self._get_weights("weights", shape=tuple(shape), init=self.W_init)
        if self.b_init:      # if b_init is None, no bias is applied
            self.b = self._get_weights("biases", shape=(self.n_units, ), init=self.b_init)

    def forward(self, inputs):
        z = tf.matmul(inputs, self.W)
        if self.b_init:
            z = tf.add(z, self.b)
        if self.act:
            z = self.act(z)
        return z
```

北京大學 PEKING UNIVERSITY

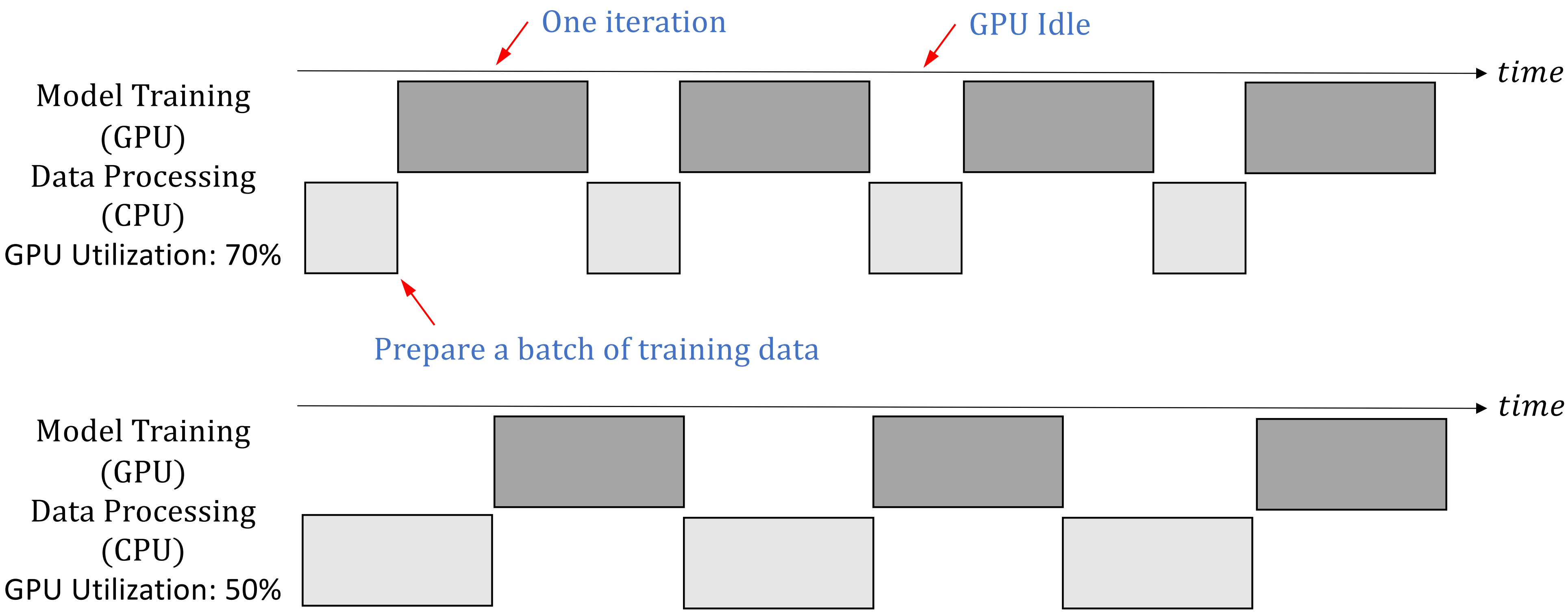
Background
- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

How to Use
- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information

- Customize Layer without Weights
- Customize Layer with Weights
- **Dataflow**
- Distributed Training

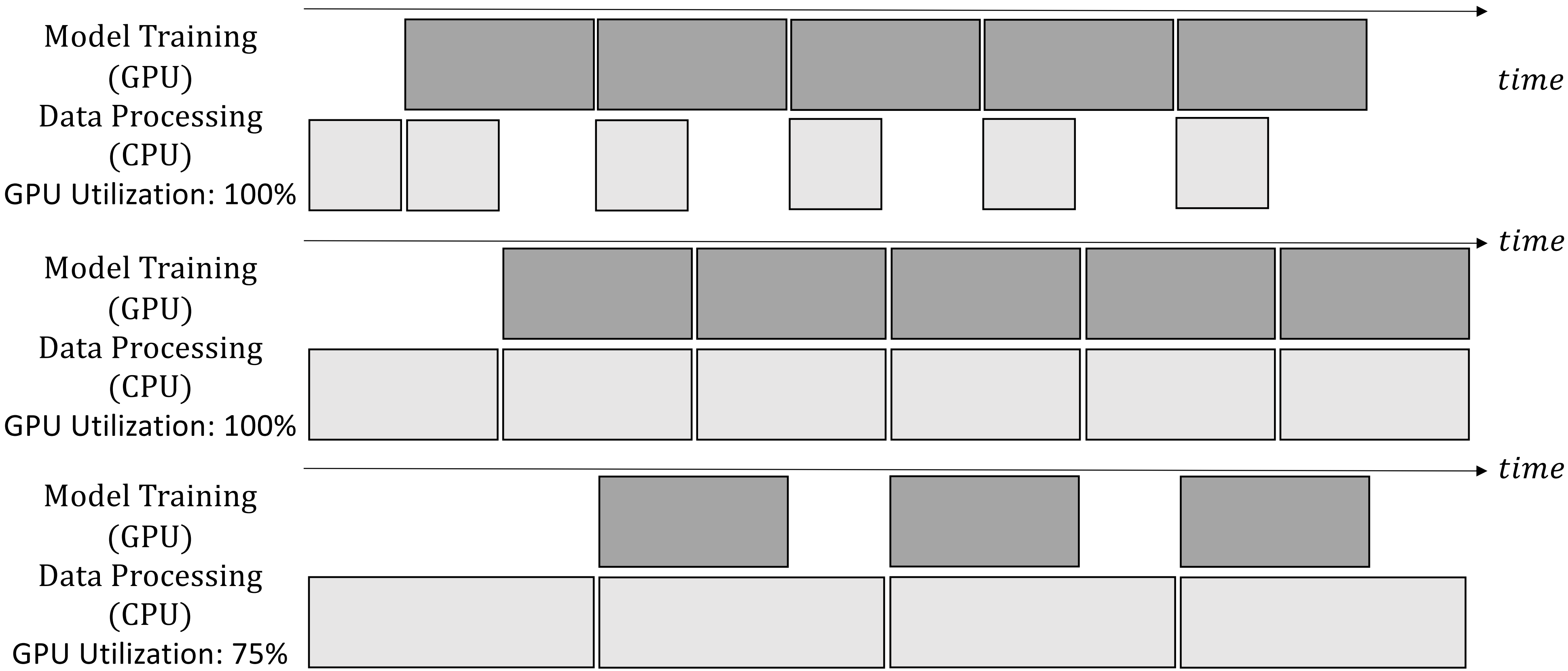
Dataflow

- Training without Dataflow



Dataflow

- Training with Dataflow



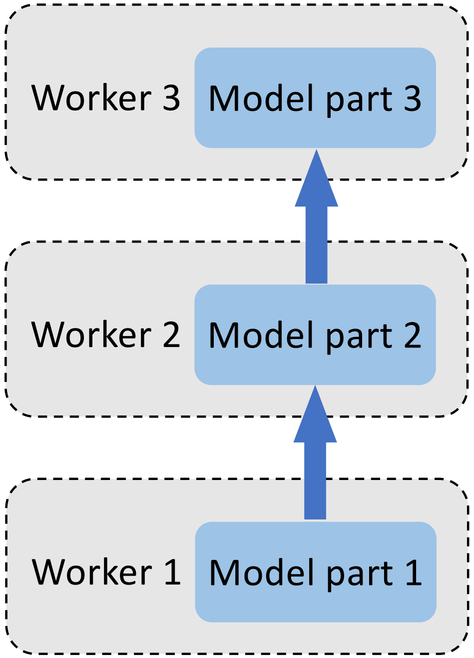
Background

- History of Deep Learning Tools
- History of TensorLayer
- Future of TensorLayer

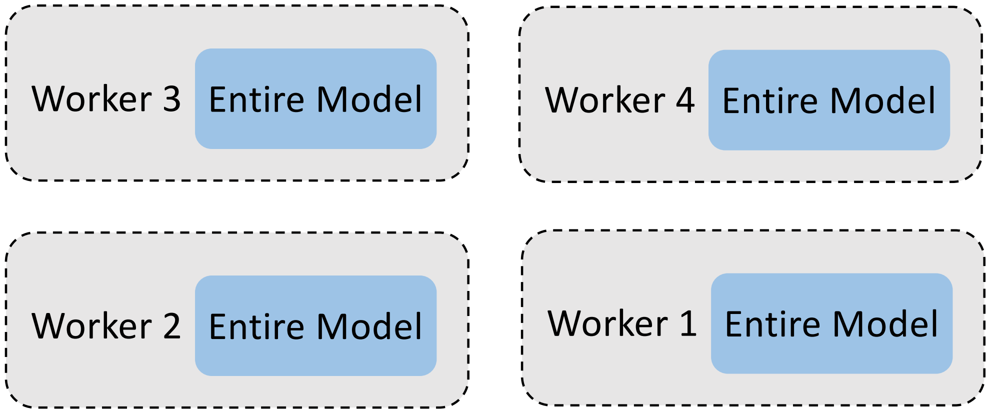
How to Use

- Static vs. Dynamic Models
- Switching Train/Test Modes
- Reuse Weights
- Model Information
- Customize Layer without Weights
- Customize Layer with Weights
- Dataflow
- Distributed Training

Distributed Training



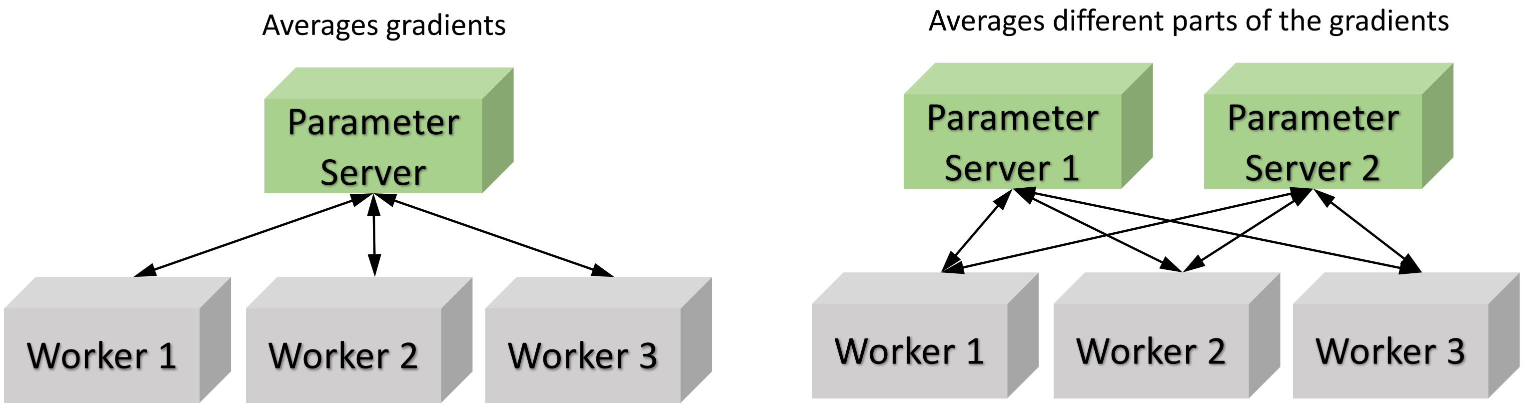
Model Parallelism



Data Parallelism

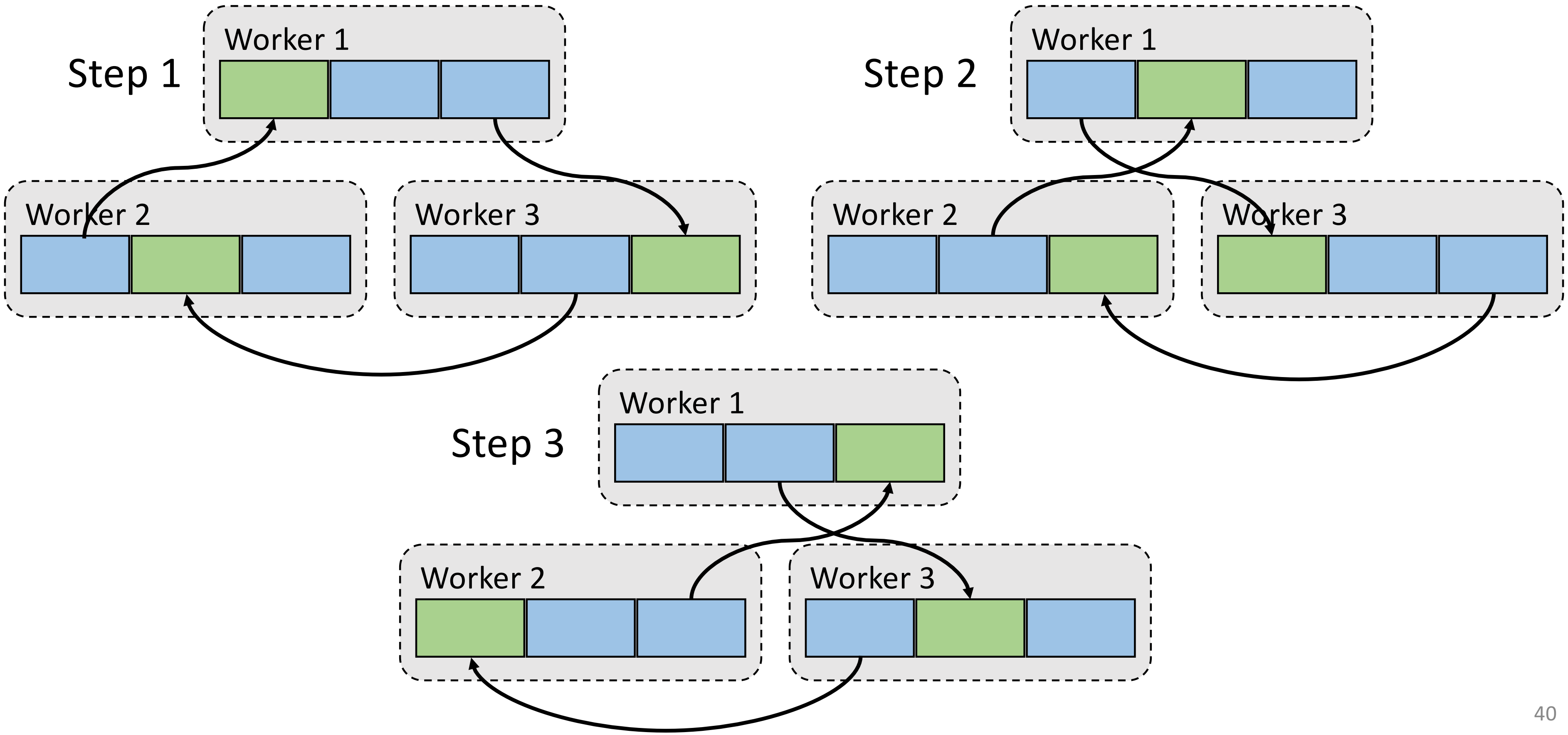
Distributed Training

- Distributed Training: Data Parallelism - Parameter Server



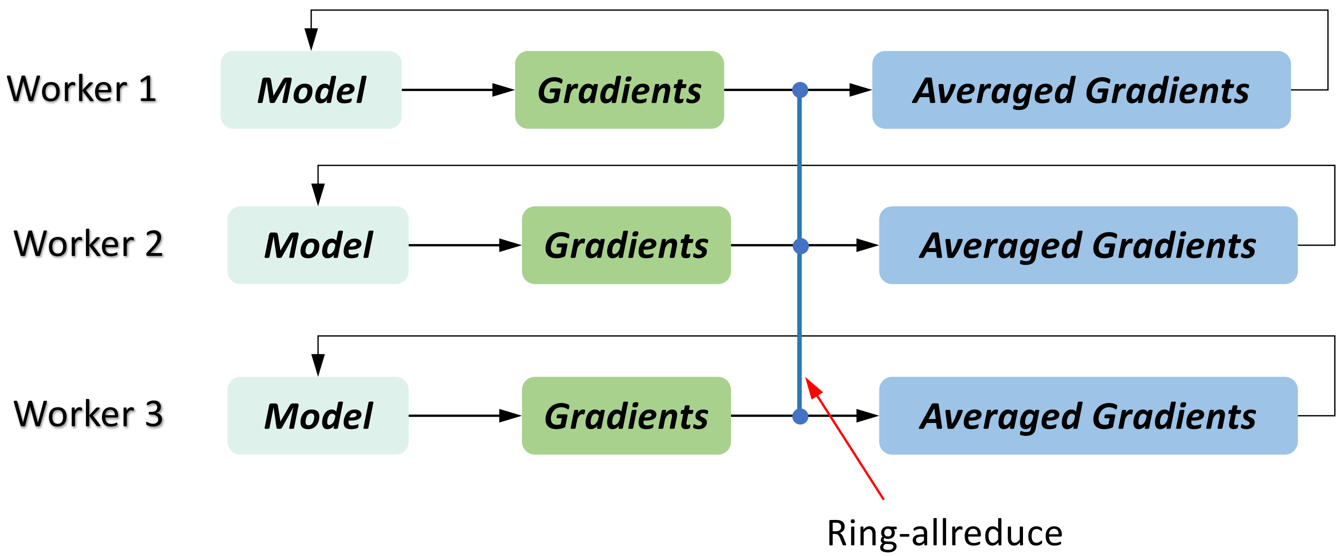
Distributed Training

- Distributed Training: Data Parallelism - Horovod - All ringreduce



Distributed Training

- Distributed Training: Data Parallelism - Horovod - All ringreduce



Please install TensorFlow and TensorLayer and make sure this code is runnable

https://github.com/tensorlayer/tensorlayer/blob/master/examples/basic_tutorials/tutorial_mnist_mlp_static.py

Thanks