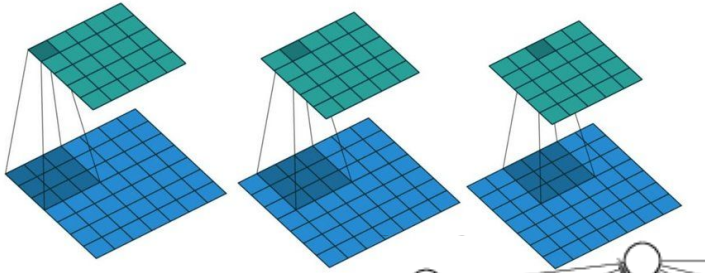
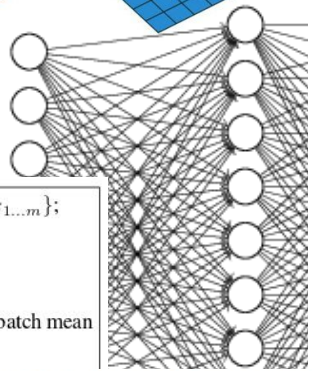
Artificial Neural Networks and Deep Learning

Keras tutorial - 07/10/2020

Francesco Lattari, PhD student (francesco.lattari@polimi.it)

Artificial Intelligence and Robotics Laboratory





Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

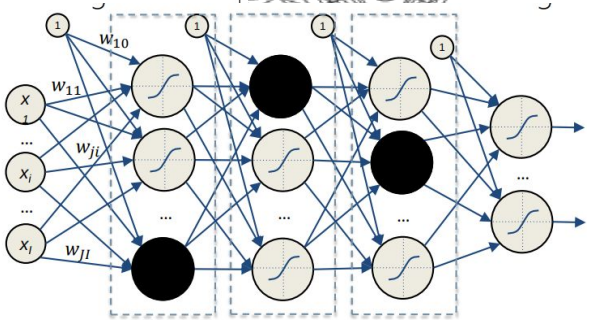
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

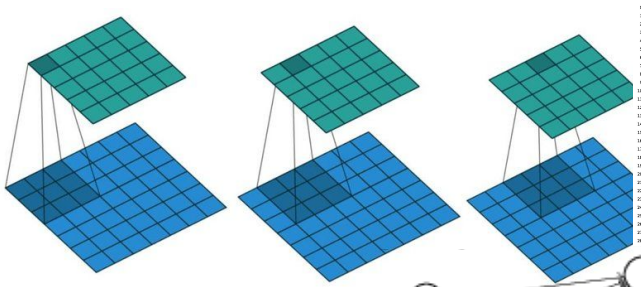
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

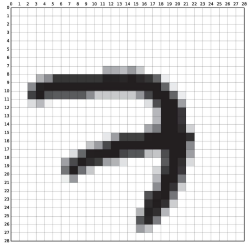
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

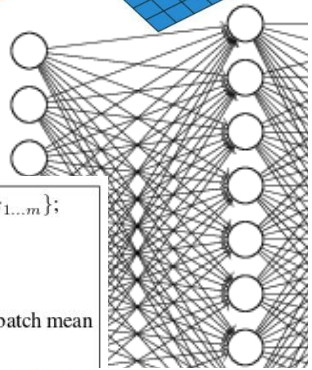
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}($$

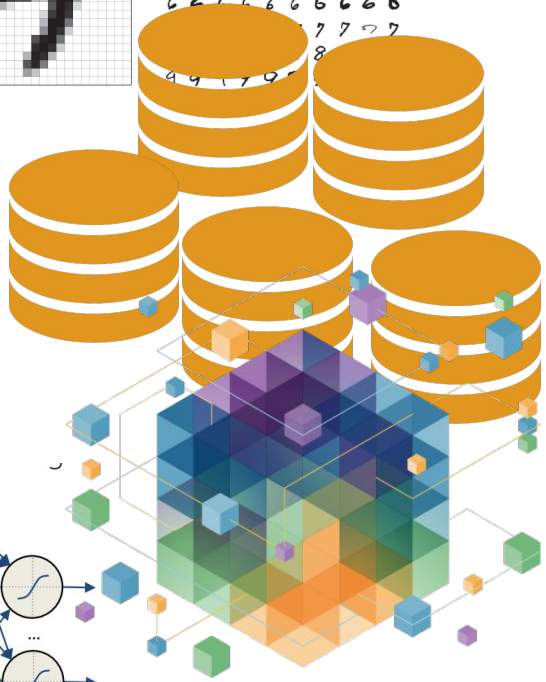
Algorithm 1: Batch Norma
activation x over a mini-bat











Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

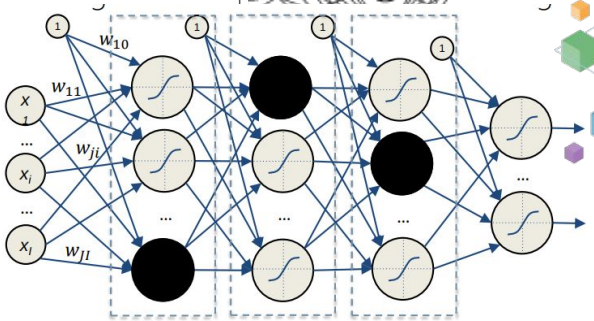
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

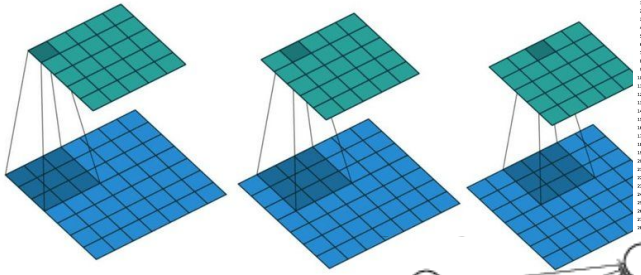
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

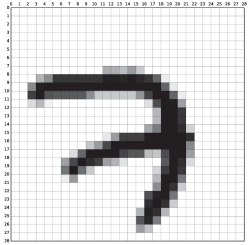
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

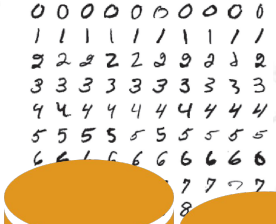
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}($$

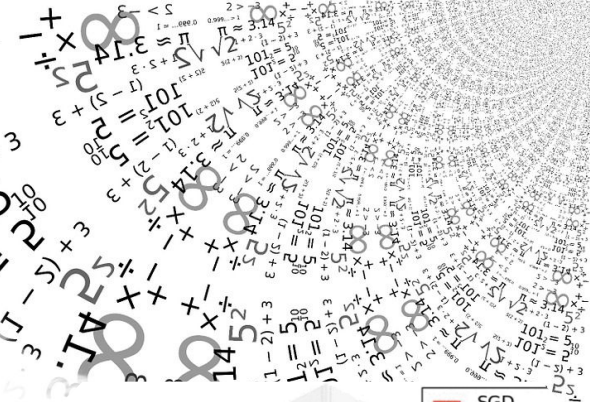
Algorithm 1: Batch Norma
activation x over a mini-bat

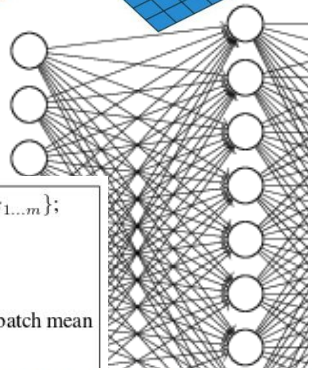












Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

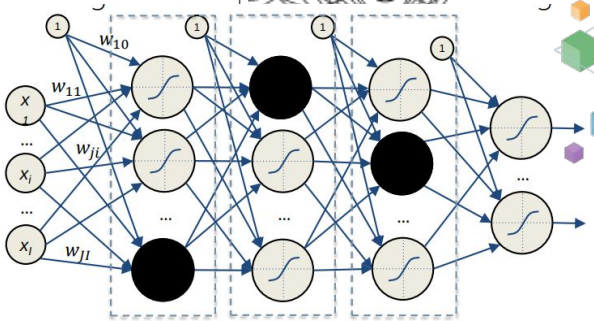
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

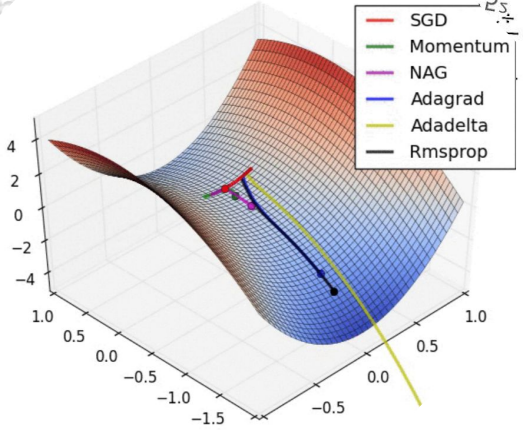
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

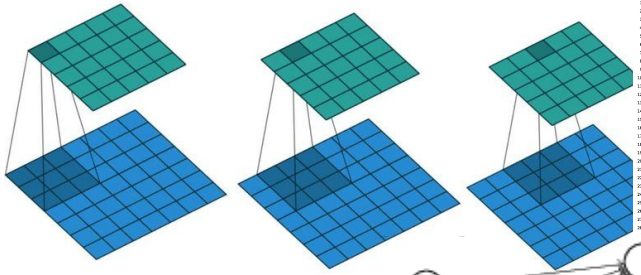
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

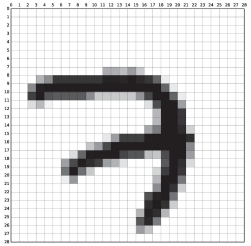
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}$$

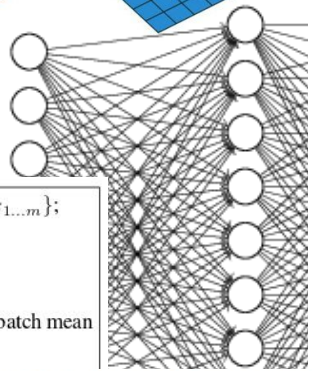
Algorithm 1: Batch Norma
activation x over a mini-bat











Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

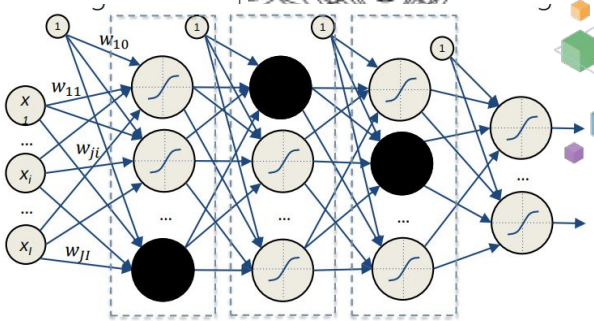
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

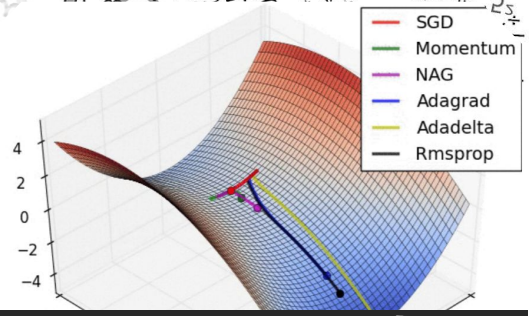
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

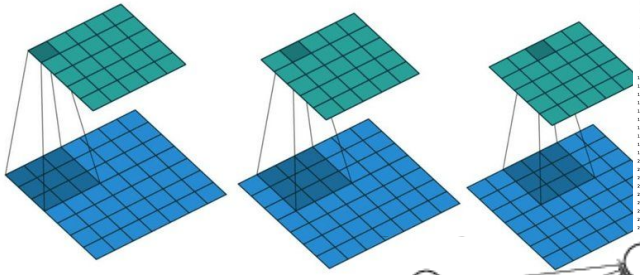
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

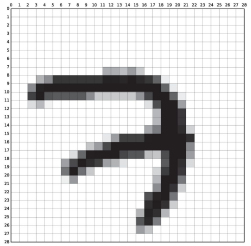
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}$$

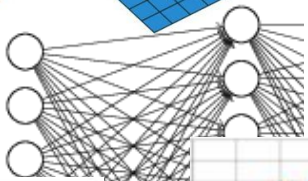
Algorithm 1: Batch Norma
activation x over a mini-bat

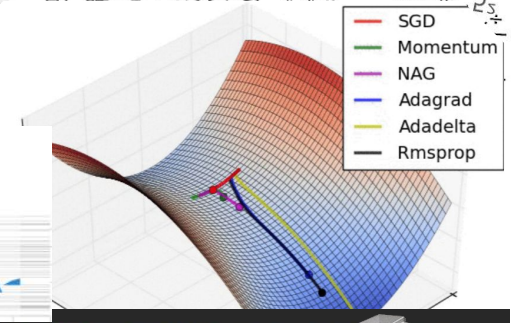












Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

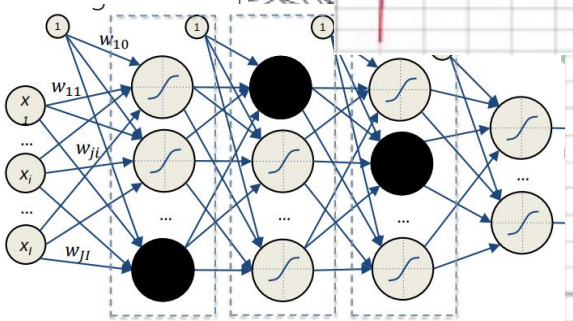
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

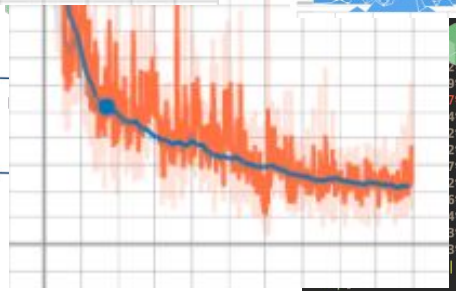
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

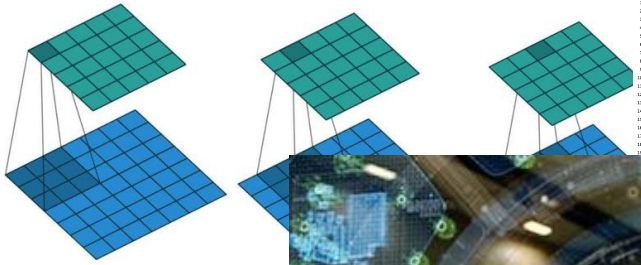
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

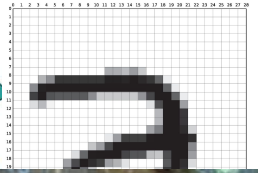
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}$$

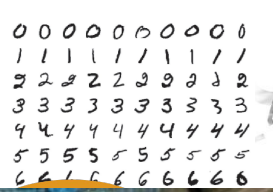
Algorithm 1: Batch Norma
activation x over a mini-bat











Input: Values of x over a mini-ba
Parameters to be learned:
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

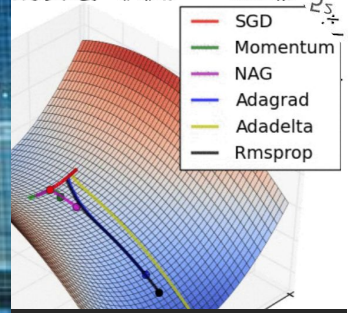
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

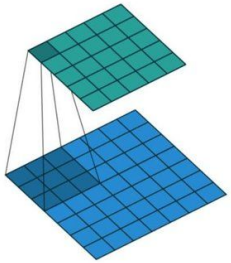
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

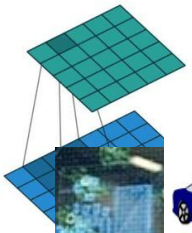
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

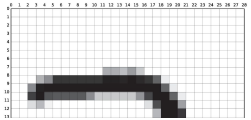
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}$$

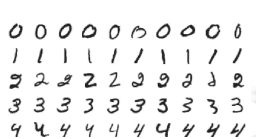
Algorithm 1: Batch Norma
activation x over a mini-bat

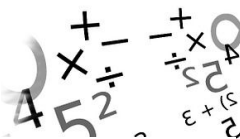


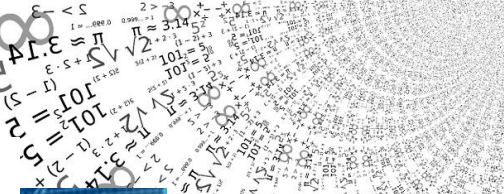


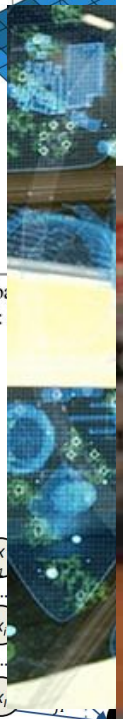


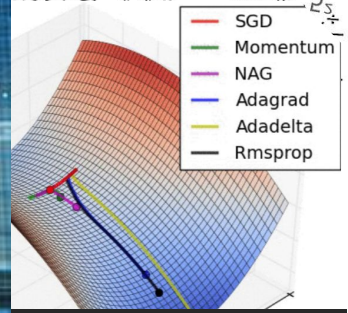












Input: Values of x over a mini-ba
Parameters to be learned:
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}$$

Algorithm 1: Batch Norma
activation x over a mini-bat

```
import keras
```

Being able to go from idea to result with the least possible delay is key to doing good research

https://keras.io/

Keras

```
import tensorflow as tf
```

```
tf.keras
```

Data structure: tf.Tensor


```
import numpy as np

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
array = np.array(matrix, dtype=np.int32)
array
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)
```

Data structure: tf.Tensor


```python
import numpy as np

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
array = np.array(matrix, dtype=np.int32)
array
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)
```


```python
import tensorflow as tf

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tensor = tf.constant(matrix, dtype=np.int32)
tensor
```

```
<tf.Tensor: id=38, shape=(3, 3), dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)>
```

Data structure: tf.Tensor


```
import numpy as np

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
array = np.array(matrix, dtype=np.int32)
array
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)
```


```
import tensorflow as tf

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tensor = tf.constant(matrix, dtype=np.int32)
tensor
```

```
<tf.Tensor: id=38, shape=(3, 3), dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)>
```

```
print("tensor.shape ->", tensor.shape)
print("tensor.ndim ->", tensor.ndim)
print("tensor.dtype ->", tensor.dtype)
print("tensor.device ->\n", tensor.device)
```

```
tensor.shape -> (3, 3)
tensor.ndim -> 2
tensor.dtype -> <dtype: 'int32'>
tensor.device ->
 /job:localhost/replica:0/task:0/device:CPU:0
```

Data structure: tf.Tensor


```
import numpy as np

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
array = np.array(matrix, dtype=np.int32)
array
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)
```


```
import tensorflow as tf

# 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tensor = tf.constant(matrix, dtype=np.int32)
tensor
```

```
<tf.Tensor: id=38, shape=(3, 3), dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]], dtype=int32)>
```

Create a tensor

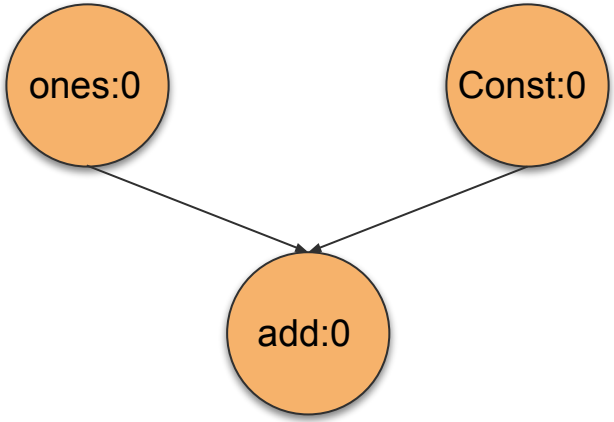
- tf.constant(value, dtype, shape)
- tf.zeros(shape, dtype)
- tf.ones(shape, dtype)
- tf.random
  - normal(
    shape, mean, stddev, dtype, seed)
  - uniform(
    shape, minval, maxval, dtype, seed)

  ….
- tf.range(start, limit, delta, dtype)
- tf.convert_to_tensor(value, dtype)

Operations on tensors - Tensorflow v1

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tensor1 = tf.constant(matrix, dtype=tf.int32)
tensor2 = tf.ones([3, 3], dtype=tf.int32)
out = tensor1 + tensor2

print(tensor1)
print(tensor2)
print(out)
```

```
Tensor("Const:0", shape=(3, 3), dtype=int32)
Tensor("ones:0", shape=(3, 3), dtype=int32)
Tensor("add:0", shape=(3, 3), dtype=int32)
```




```
tensor1.op
tensor2.op
out.op
```

```
<tf.Operation 'Const' type=Const>

<tf.Operation 'ones' type=Const>

<tf.Operation 'add' type=AddV2>
```

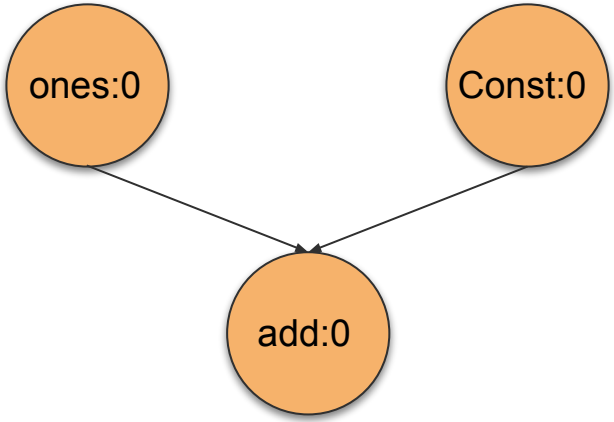
Operations on tensors - Tensorflow v1

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tensor1 = tf.constant(matrix, dtype=tf.int32)
tensor2 = tf.ones([3, 3], dtype=tf.int32)
out = tensor1 + tensor2

print(tensor1)
print(tensor2)
print(out)
```

```
Tensor("Const:0", shape=(3, 3), dtype=int32)
Tensor("ones:0", shape=(3, 3), dtype=int32)
Tensor("add:0", shape=(3, 3), dtype=int32)
```

```
session = tf.compat.v1.Session()
session.run(out)
```

```
array([[ 2,  3,  4],
       [ 5,  6,  7],
       [ 8,  9, 10]], dtype=int32)
```




```
tensor1.op
tensor2.op
out.op
```

```
<tf.Operation 'Const' type=Const>

<tf.Operation 'ones' type=Const>

<tf.Operation 'add' type=AddV2>
```

Operations on tensors - Tensorflow 2.0

Eager execution

```
import tensorflow as tf

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tensor1 = tf.constant(matrix, dtype=tf.int32)
tensor2 = tf.ones([3, 3], dtype=tf.int32)
out = tensor1 + tensor2

print(tensor1)
print(tensor2)
print(out)
```

```
tf.Tensor(
[[1 2 3]
 [4 5 6]
 [7 8 9]], shape=(3, 3), dtype=int32)
tf.Tensor(
[[1 1 1]
 [1 1 1]
 [1 1 1]], shape=(3, 3), dtype=int32)
tf.Tensor(
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]], shape=(3, 3), dtype=int32)
```

- imperative programming environment
  - operations evaluated immediately
- easier debugging
- natural control flow (Python)

Change data type

- tf.cast(x, dtype)

`tf.float16` : 16-bit half-precision floating-point.

`tf.float32` : 32-bit single-precision floating-point.

`tf.float64` : 64-bit double-precision floating-point.

`tf.bfloat16` : 16-bit truncated floating-point.

`tf.complex64` : 64-bit single-precision complex.

`tf.complex128` : 128-bit double-precision complex.

`tf.int8` : 8-bit signed integer.

`tf.uint8` : 8-bit unsigned integer.

`tf.uint16` : 16-bit unsigned integer.

`tf.uint32` : 32-bit unsigned integer.

`tf.uint64` : 64-bit unsigned integer.

`tf.int16` : 16-bit signed integer.

`tf.int32` : 32-bit signed integer.

`tf.int64` : 64-bit signed integer.

`tf.bool` : Boolean.

`tf.string` : String.

`tf.qint8` : Quantized 8-bit signed integer.

`tf.quint8` : Quantized 8-bit unsigned integer.

`tf.qint16` : Quantized 16-bit signed integer.

`tf.quint16` : Quantized 16-bit unsigned integer.

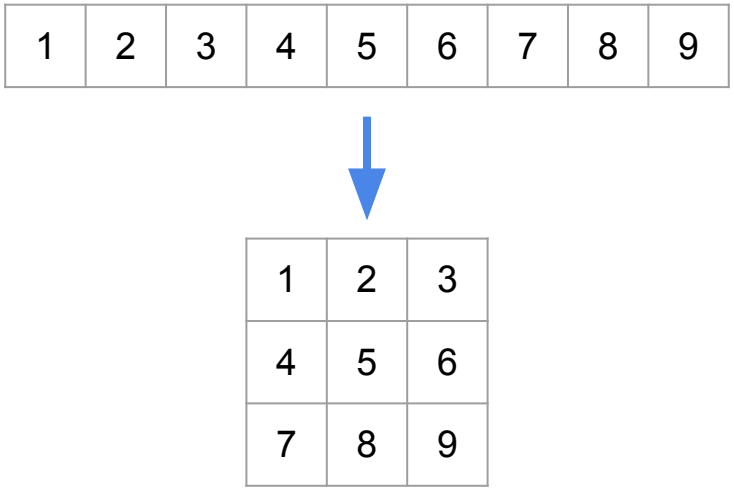
`tf.qint32` : Quantized 32-bit signed integer.

`tf.resource` : Handle to a mutable resource.

`tf.variant` : Values of arbitrary types.

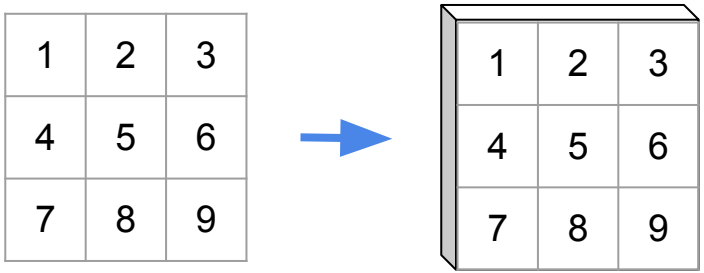
Reshape

- tf.reshape(tensor, shape)



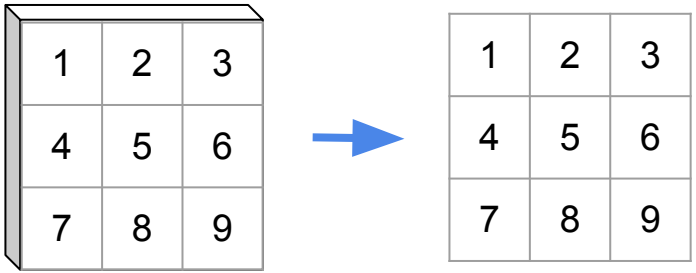
Reshape

- tf.expand_dims(input, axis)



Reshape

- tf.squeeze(input, axis)



Operations on tensors - Tensorflow 2.0

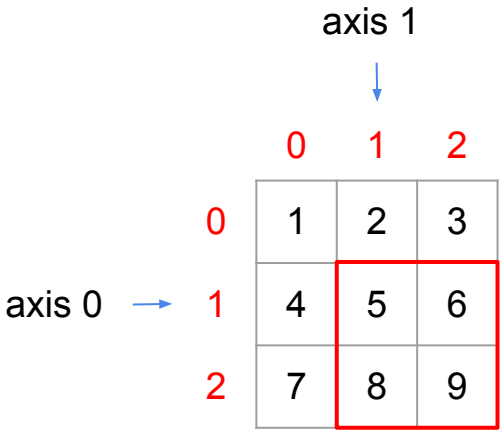
Math operations

- *, +, -, / operators (element-wise)
- tf.add(x, y) (element-wise)
  - tf.add_n(inputs)
- tf.multiply(x, y) (element-wise)
  - tf.tensordot(a, b, axes) (matrix multiplication)
- tf.abs(x)
- tf.pow(x, y)
- tf.transpose(a, perm, conjugate)
- …

Module: tf.math

https://www.tensorflow.org/api_docs/python/tf/math

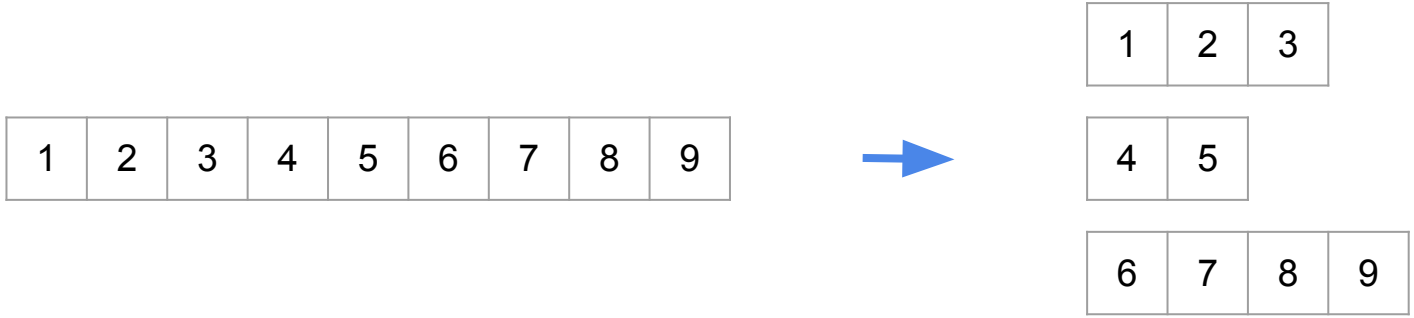
Slicing



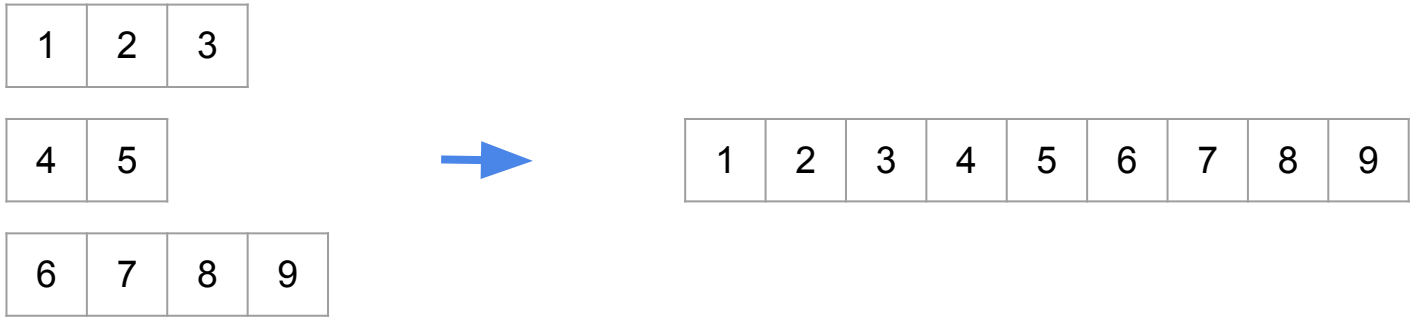
Indexing

- pythonic way: “[ ]”
  - [idx1, idx2, …, idxN]
  - [start:stop:step]
- tf.slice(input_, begin, size)
- using a condition: tf.gather_nd(params, indices)
  - indices = tf.where(condition)

Splitting: tf.split(value, num_or_size_splits, axis)

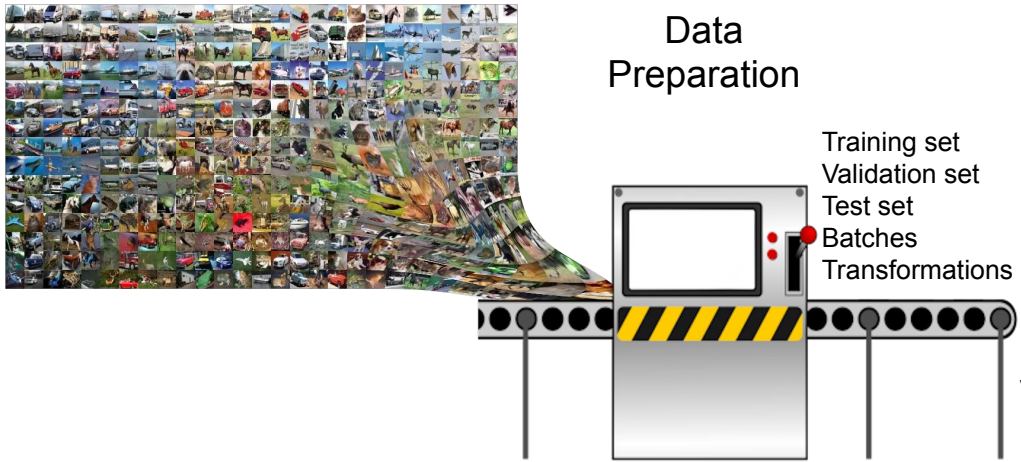


Stacking: tf.concat(values, axis)



Deep Learning Framework

Data



Data Preparation

Training set
Validation set
Test set
Batches
Transformations

Training Loop

Optimization

Model

Visualization

Class tf.data.Dataset

https://www.tensorflow.org/api_docs/python/tf/data/Dataset

- Methods (main)
  - __iter__()
  - batch(batch_size, drop_reminder=False)
  - map(map_fn, num_parallel_calls=None)
  - filter(predicate)
  - shuffle(buffer_size, seed=None, reshuffle_each_iteration=False)
  - …

Class tf.data.Dataset

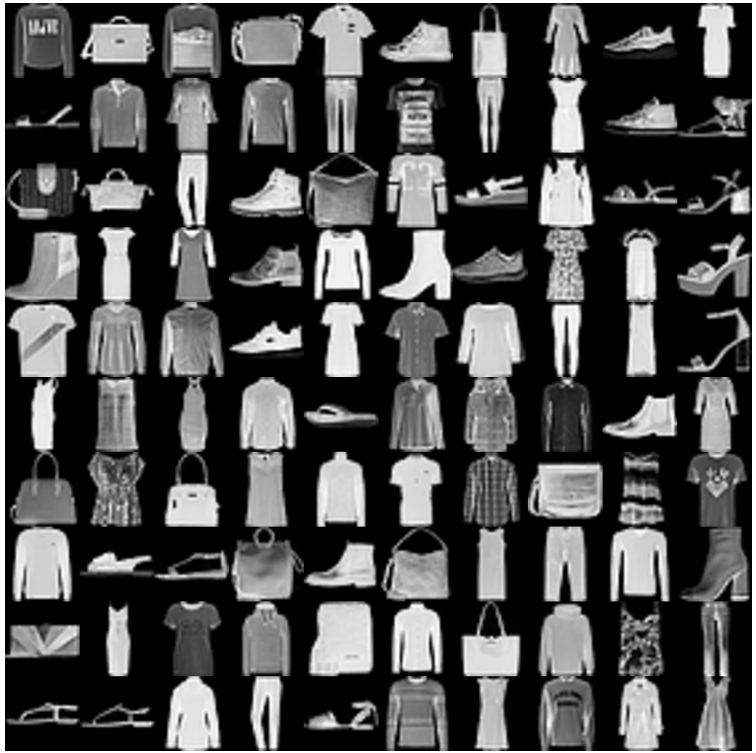
https://www.tensorflow.org/api_docs/python/tf/data/Dataset

- Create a Dataset
  - range(start, stop, step)
  - from_tensors(tensors)
  - from_tensor_slices(tensors)
  - ...

tf.keras.datasets

https://www.tensorflow.org/api_docs/python/tf/keras/datasets

- Fashion MNIST
  - 28x28 grayscale images
  - 10 classes:
    1. T-shirt/top
    2. Trouser/pants
    3. Pullover shirt
    4. Dress
    5. Coat
    6. Sandal
    7. Shirt
    8. Sneaker
    9. Bag
    10. Ankle boot



Class tf.keras.layers

https://www.tensorflow.org/api_docs/python/tf/keras/layers

- Classes of basic building blocks
  - e.g., fully-connected layer

```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs,
)
```

Class tf.keras.layers

https://www.tensorflow.org/api_docs/python/tf/keras/layers

- Classes of basic building blocks
  - e.g., fully-connected layer

```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs,
)
```

tf.keras.activations

https://www.tensorflow.org/api_docs/python/tf/keras/activations

Class tf.keras.layers

https://www.tensorflow.org/api_docs/python/tf/keras/layers

- Classes of basic building blocks
  - e.g., fully-connected layer

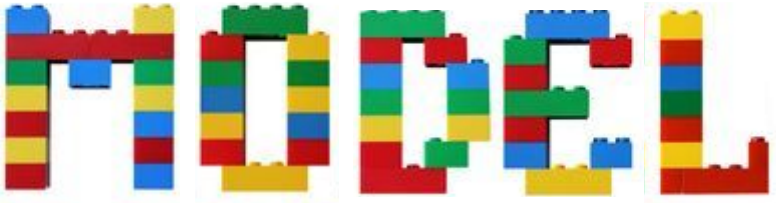
```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs,
)
```

tf.keras.initializers

https://www.tensorflow.org/api_docs/python/tf/initializers

Class tf.keras.Model

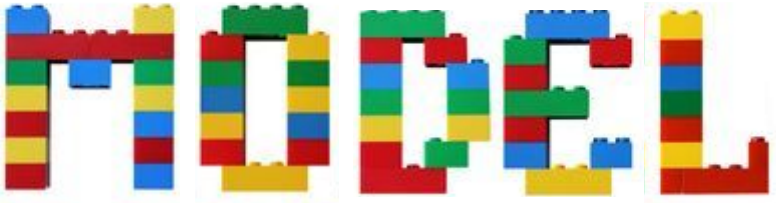
https://www.tensorflow.org/api_docs/python/tf/keras/Model



- Groups multiple layers

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

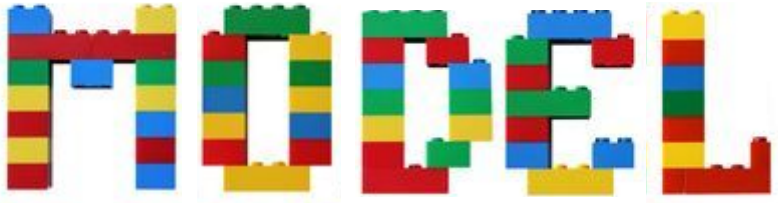


- Groups multiple layers
- Create a Model instance (“functional API”):
    1. Instantiate keras Input tensor
       https://www.tensorflow.org/api_docs/python/tf/keras/Input

       x = tf.keras.Input(shape, dtype, …)

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model



- Groups multiple layers
- Create a Model instance (“functional API”):

1. Instantiate keras Input tensor
   https://www.tensorflow.org/api_docs/python/tf/keras/Input

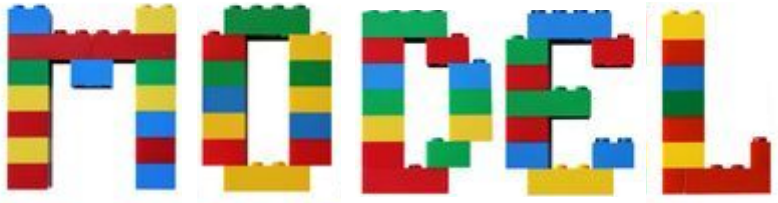
   x = tf.keras.Input(shape, dtype, …)

2. Instantiate and chain keras layers

   layer1 = tf.keras.layers.LAYER_NAME(...)(x)
   layer2 = tf.keras.layers.LAYER_NAME(...)(layer1)
   ….
   out = ...

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model



- Groups multiple layers
- Create a Model instance (“functional API”):

1. Instantiate keras Input tensor
   https://www.tensorflow.org/api_docs/python/tf/keras/Input

   x = tf.keras.Input(shape, dtype, …)

2. Instantiate and chain keras layers

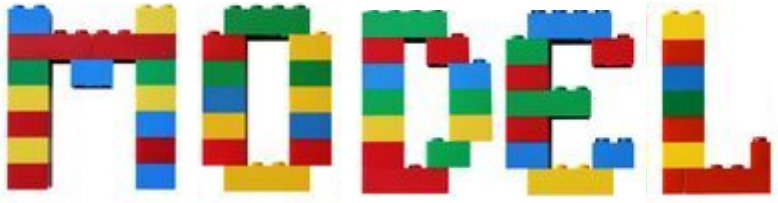
   layer1 = tf.keras.layers.LAYER_NAME(...)(x)
   layer2 = tf.keras.layers.LAYER_NAME(...)(layer1)
   ….
   out = ...

3. Instantiate Model

   model = tf.keras.Model(inputs=x, outputs=out)

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model



- Groups multiple layers
- Create a Model instance (“functional API”):

1. Instantiate keras Input tensor
   https://www.tensorflow.org/api_docs/python/tf/keras/Input

   x = tf.keras.Input(shape, dtype, …)

2. Instantiate and chain keras layers

   layer1 = tf.keras.layers.LAYER_NAME(...)(x)
   layer2 = tf.keras.layers.LAYER_NAME(...)(layer1)
   ….
   out = ...

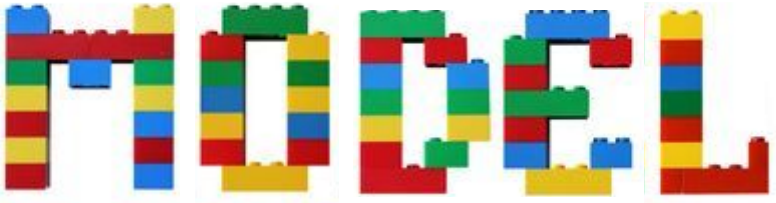
3. Instantiate Model

   model = tf.keras.Model(inputs=x, outputs=out)

OR...

Class tf.keras.Sequential

https://www.tensorflow.org/api_docs/python/tf/keras/Sequential



- Stacks multiple layers
- Create a Sequential instance (“functional API”):
  - 2 modalities:

    1. model = tf.keras.Sequential([layer1, layer2, …, layerN])

    2. model = tf.keras.Sequential()
       model.add(layer1)
       model.add(layer2)
       ....
       model.add(layerN)

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

- 2 main steps through class methods

1. Prepare model for training

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

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

- 2 main steps through class methods
  1. Prepare model for training

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

tf.keras.optimizers

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

- class SGD
- class Adam
- class Adadelta
- ...

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

- 2 main steps through class methods
  1. Prepare model for training

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

tf.keras.losses

https://www.tensorflow.org/api_docs/python/tf/keras/losses

- class BinaryCrossEntropy
- class CategoricalCrossEntropy
- class MeanSquaredError
- ...

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

- 2 main steps through class methods
  1. Prepare model for training

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

tf.keras.metrics

https://www.tensorflow.org/api_docs/python/tf/keras/metrics

- class Accuracy
- class MeanIoU
- class Precision
- class Recall
- ...

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

- 2 main steps through class methods

1. Prepare model for training

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

2. Training model

```
model.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_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    **kwargs)
```

Class tf.keras.Model

https://www.tensorflow.org/api_docs/python/tf/keras/Model

- 2 main steps through class methods

1. Prepare model for training

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

2. Training model

```
model.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_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    **kwargs)
```

Returns a History object containing a record of loss and metric values at each epoch