Introduction to TensorFlow 2

Oswald Campesato
ocampesato@yahoo.com

Highlights/Overview

- What is TensorFlow 2?
- Major Changes in TF 2
- Working with strings/arrays/tensors
- Working with @tf.function decorator
- Working with generators
- Working with tf.data.Dataset
- Datasets in TF 1.x versus TF 2
- Working with tf.keras
- CNNs, RNNs, LSTMs, Bidirectional LSTMs
- Reinforcement Learning

What is TensorFlow 2?

- An open source framework for ML and DL
- Created by Google (TF 1.x released 11/2015)
- Evolved from Google Brain
- Linux and Mac OS X support (VM for Windows)

- TF home page: https://www.tensorflow.org/

Status of TensorFlow 2

- Alpha release: around March/2019
- Other “Nightly” builds:

https://pypi.org/project/tf-nightly-2.0-preview/2.0.0.dev20190417/#files

- Production release: TBD (later this year)

TF 2 Platform Support

- Tested and supported on 64-bit systems

- Ubuntu 16.04 or later
- Windows 7 or later
- macOS 10.12.6 (Sierra) or later (no GPU support)
- Raspbian 9.0 or later

TF 2 Docker Images

- https://hub.docker.com/r/tensorflow/tensorflow/

$ docker run -it --rm tensorflow/tensorflow bash

- Start a CPU-only container with Python 2:

$ docker run -it --rm --runtime=nvidia tensorflow/tensorflow:latest-gpu-py3 python

- Start a GPU container with Python 3 & Python interpreter:

$ docker run -it --rm -v $(realpath ~/notebooks):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:late

Status of TensorFlow 1.x

- 1.13.1: current stable release

- 1.14: latest release

- After TF 2 is in production:

Only security-related updates to TF 1.x

TF 1.x support for one year after TF 2 production

What is TensorFlow 2?

- Support for Python, Java, C++
- Desktop, Server, Mobile, Web
- CPU/GPU/TPU support

- Visualization via TensorBoard
- Can be embedded in Python scripts

- Installation: pip install tensorflow
- `Ex: pip install tensorflow==2.0.0-alpha0`

TensorFlow Use Cases (Generic)

- Image recognition
- Computer vision
- Voice/sound recognition
- Time series analysis
- Language detection
- Language translation
- Text-based processing
- Handwriting Recognition

Major Changes in TF 2

- TF 2 Eager Execution: default mode
- @tf.function decorator: instead of tf.Session()
- AutoGraph: graph generated in @tf.function()
- Generators: used with tf.data.Dataset (TF 2)
- Differentiation: calculates gradients
- tf.GradientTape: automatic differentiation

Removed from TensorFlow 2

- tf.Session()
- tf.placeholder()
- tf.global_initializer()
- feed_dict
- Variable scopes
- tf.contrib code
- tf.flags and tf.contrib
- Global variables

- => TF 1.x functions moved to compat.v1

Upgrading Files/Directories to TF 2

- The TF 2 upgrade script:

https://www.tensorflow.org/alpha/guide/upgrade

- Script name: tf_upgrade_v2

- install TF to get the upgrade script:

pip install tensorflow (NOT pip3)

Upgrading Files/Directories to TF 2

- 1) upgrade one file:

tf_upgrade_v2 --infile oldtf.py --outfile newtf.py

- The preceding command creates report.txt

- Upgrade an entire directory:

tf_upgrade_v2 –intree mydir --outtree newtf.py –copyotherfiles False

- NB: do NOT make manual upgrade changes

Migrating to TensorFlow 2

- 1) do not manually update parts of your code
- 2) script won't work with functions with reordered arguments
- 3) script assumes this TF statement: "import tensorflow as tf"
- 4) the script does not reorder arguments
- 5) the script adds keyword arguments to functions that have their arguments reordered.

- Upgrade Jupyter notebooks and Python files in a GitHub repo:

http://tf2up.ml/

=> Replace prefix https://github.com/ with http://tf2up.m

Check your TF Version

- `import tensorflow as tf`
- `import numpy`

- `print("TF version:    ",tf.__version__)`
- `print("Keras version:",tf.keras.__version__)`
- `print("Numpy version:",numpy.__version__)`

- `# my system:`
- `#TF version:    2.0.0-alpha0`
- `#Keras version: 2.2.4-tf`
- `#Numpy version: 1.15.4`

What is a Tensor?

- TF tensors are n-dimensional arrays
- TF tensors are very similar to numpy ndarrays

- scalar number: a zeroth-order tensor
- vector: a first-order tensor
- matrix: a second-order tensor
- 3-dimensional array: a 3rd order tensor

- https://dzone.com/articles/tensorflow-simplified-examples

TensorFlow Eager Execution

- An imperative interface to TF
- Fast debugging & immediate run-time errors
- => TF 2 requires Python 3.x (not Python 2.x)
- No static graphs or sessions

TensorFlow Eager Execution

- integration with Python tools
- Supports dynamic models + Python control flow
- support for custom and higher-order gradients
- => Default mode in TensorFlow 2.0

TensorFlow Eager Execution

```
import tensorflow as tf

x = [[2.]]
# m = tf.matmul(x, x)  <= deprecated
m = tf.linalg.matmul(x, x)

print("m:",m)
print("m:",m.numpy())
#Output:
#m: tf.Tensor([[4.]], shape=(1,1),dtype=float32)
#m: [[4.]]
```

Basic TF 2 Operations

- Working with Constants
- Working with Strings
- Working with Tensors
- Working Operators
- Arrays in TF 2
- Multiplying Two Arrays
- Convert Python Arrays to TF Arrays
- Conflicting Types

TensorFlow “primitive types”

- tf.constant:

> initialized immediately and immutable

```
import tensorflow as tf
aconst = tf.constant(3.0)
print(aconst)
```

output for TF 2:

```
#tf.Tensor(3.0, shape=(), dtype=float32)
```

output for TF 1.x:

```
#Tensor("Const:0", shape=(), dtype=float32)
```

TensorFlow “primitive types”

- tf.Variable (a class):

+ initial value is required

+ updated during training

+ in-memory buffer (saved/restored from disk)

+ can be shared in a distributed environment

+ they hold learned parameters of a model

TensorFlow Arithmetic

```
import tensorflow as tf # arith1.py
a = tf.add(4, 2)
b = tf.subtract(8, 6)
c = tf.multiply(a, 3)
d = tf.div(a, 6)
```

- `print(a) # 6`
- `print(b) # 2`
- `print(c) # 18`
- `print(d) # 1`
- `print("a:",a.numpy())`
- `print("b:",b.numpy())`
- `print("c:",c.numpy())`
- `print("d:",d.numpy())`

TensorFlow Arithmetic

- tf.Tensor(6, shape=(), dtype=int32)
- tf.Tensor(2, shape=(), dtype=int32)
- tf.Tensor(18, shape=(), dtype=int32)
- tf.Tensor(1.0, shape=(), dtype=float64)

- 6
- 2
- 18
- 1.0

TF 2 Loops and Arrays

- Using “for” loops
- Using “while” loops
- tf.reduce_prod()
- tf.reduce_sum()

A “for” Loop Example

- `import tensorflow as tf`
- `x = tf.Variable(0, name='x')`
- `for i in range(5):`
- `    x = x + 1`
- `    print("x:",x)`

A “for” Loop Example

- x: tf.Tensor(1, shape=(), dtype=int32)
- x: tf.Tensor(2, shape=(), dtype=int32)
- x: tf.Tensor(3, shape=(), dtype=int32)
- x: tf.Tensor(4, shape=(), dtype=int32)
- x: tf.Tensor(5, shape=(), dtype=int32)

A “while” Loop Example

```
import tensorflow as tf

a = tf.constant(12)

while not tf.equal(a, 1):
  if tf.equal(a % 2, 0):
    a = a / 2
  else:
    a = 3 * a + 1
  print(a)
```

A “while” Loop Example

- tf.Tensor(6.0, shape=(), dtype=float64)
- tf.Tensor(3.0, shape=(), dtype=float64)
- tf.Tensor(10.0, shape=(), dtype=float64)
- tf.Tensor(5.0, shape=(), dtype=float64)
- tf.Tensor(16.0, shape=(), dtype=float64)
- tf.Tensor(8.0, shape=(), dtype=float64)
- tf.Tensor(4.0, shape=(), dtype=float64)
- tf.Tensor(2.0, shape=(), dtype=float64)
- tf.Tensor(1.0, shape=(), dtype=float64)

Working with Arrays

- `import tensorflow as tf`
- `import numpy as np`

- `# create a Python array:`
- `array_1d = np.array([1.3, 1, 4.0, 23.5])`
- `tf_tensor = tf.convert_to_tensor(value=array_1d, dtype=tf.float64)`

- `print(tf_tensor)`
- `print("0:",tf_tensor[0])`
- `print("2:",tf_tensor[2])`

Working with Arrays

- `tf.Tensor([ 1.3  1.   4.  23.5], shape=(4,), dtype=float64)`

- `0: tf.Tensor(1.3, shape=(), dtype=float64)`

- `2: tf.Tensor(4.0, shape=(), dtype=float64)`

Random Numbers

- The tf.random.normal()
- The tf.truncated_normal()
- TF Arrays with Random Values

Random Numbers: examples

```
import tensorflow as tf


# normal distribution:
w=tf.Variable(tf.random.normal([784,10],stddev=0.01))


# mean of an array:
b = tf.Variable([10,20,30,40,50,60],name='t')


print("w: ",w)
print("b: ",tf.reduce_mean(input_tensor=b))
```

Random Numbers: examples

- w: <tf.Variable 'Variable:0' shape=(784, 10) dtype=float32, numpy=
- array([[0.00199239, 0.00285635, 0.00804297, ..., -0.00323935,
-          0.00138759, 0.00941323],
-        [-0.00017284, -0.00708508, -0.00670188, ..., 0.00085814,
-         -0.01298123, 0.01133613],
-        [-0.00506489, 0.01542902, -0.00710952, ..., -0.00294803,
-         -0.00767813, 0.00815126],
-        ...,

Random Numbers: examples

- [0.01556268, 0.01296226, 0.01230366, ..., 0.01154588,
- -0.01639041, 0.00107052],
- [0.0109421 , 0.00486962, 0.02887715, ..., -0.00792963,
- -0.00337163, 0.0041892],
- [-0.0199652 , -0.00471972, 0.00246108, ..., -0.00995417,
- -0.00854415, -0.01487656]], dtype=float32)>

Useful TF 2 APIs

- tf.shape()
- tf.rank()
- tf.range()
- tf.reshape()
- tf.ones()
- tf.zeros()
- tf.fill()

The tf.range() API

```
import tensorflow as tf

a1 = tf.range(3, 18, 3)
a2 = tf.range(0, 8, 2)
a3 = tf.range(-6, 6, 3)
a4 = tf.range(-10, 10, 4)

print('a1:',a1)
print('a2:',a2)
print('a3:',a3)
print('a4:',a4)
```

The tf.range() API

```
a1: tf.Tensor([3 6 9 12 15],  shape=(5,),dtype=int32)

a2: tf.Tensor([0 2 4 6],      shape=(4,), dtype=int32)

a3: tf.Tensor([-6 -3 0 3],    shape=(4,), dtype=int32)

a4: tf.Tensor([-10 -6 -2 2 6], shape=(5,),dtype=int32)
```

Some TF 2 “lazy operators”

- `map()`
- `filter()`
- `flatmap()`
- `batch()`
- `take()`
- `zip()`
- `flatten()`

- Combined via “method chaining”

TF 2 “lazy operators”

- filter():
- uses Boolean logic to "filter" the elements in an array to determine which elements satisfy the Boolean condition

- map(): a projection
- this operator "applies" a lambda expression to each input element

- flat_map():
- maps a single element of the input dataset to a Dataset of elements

TF 2 “lazy operators”

- batch(n):
- processes a "batch" of n elements during each iteration

- repeat(n):
- repeats its input values n times

- take(n):
- operator "takes" n input values

TF 2 tf.data.Dataset

- TF “wrapper” class around data sources
- Located in the tf.data.Dataset namespace
- Supports lambda expressions
- Supports lazy operators
- They use generators instead of iterators (TF 1.x)

tf.data.Dataset.from_tensors()

- Import tensorflow as tf

- #combine the input into one element
- t1 = tf.constant([[1, 2], [3, 4]])
- ds1 = tf.data.Dataset.from_tensors(t1)

- # output: [[1, 2], [3, 4]]

tf.data.Dataset.from_tensor_slices()

- Import tensorflow as tf

- #separate element for each item
- t2 = tf.constant([[1, 2], [3, 4]])
- ds1 = tf.data.Dataset.from_tensor_slices(t2)

- # output: [1, 2], [3, 4]

TF 2 Datasets: code sample

- import tensorflow as tf # tf2-dataset.py
- import numpy as np

```
x = np.arange(0, 10)
```

- # create a dataset from a Numpy array

```
ds = tf.data.Dataset.from_tensor_slices(x)
```

TF 1.x Datasets: iterator

```
import tensorflow as tf # tf1x-plusone.py
import numpy as np

x = np.arange(0, 10)
# create dataset object from Numpy array
ds = tf.data.Dataset.from_tensor_slices(x)
ds.map(lambda x: x + 1)

# create a one-shot iterator <= TF 1.x
iterator = ds.make_one_shot_iterator()

for i in range(10):
  val = iterator.get_next()
  print("val:",val)
```

TF 2 Datasets: generator

```
import tensorflow as tf # tf2-plusone.py
import numpy as np
x = np.arange(0, 5) # 0, 1, 2, 3, 4

def gener():
  for i in x:
    yield (3*i)

ds = tf.data.Dataset.from_generator(gener, (tf.int64))

for value in ds.take(len(x)):
  print("1value:",value)

for value in ds.take(2*len(x)):
  print("2value:",value)
```

TF 2 Datasets: generator

- 1value: tf.Tensor(0, shape=(), dtype=int64)
- 1value: tf.Tensor(3, shape=(), dtype=int64)
- 1value: tf.Tensor(6, shape=(), dtype=int64)
- 1value: tf.Tensor(9, shape=(), dtype=int64)
- 1value: tf.Tensor(12, shape=(), dtype=int64)
- 2value: tf.Tensor(0, shape=(), dtype=int64)
- 2value: tf.Tensor(3, shape=(), dtype=int64)
- 2value: tf.Tensor(6, shape=(), dtype=int64)
- 2value: tf.Tensor(9, shape=(), dtype=int64)
- 2value: tf.Tensor(12, shape=(), dtype=int64)

TF 1.x map() and take()

- `import tensorflow as tf # tf 1.12.0`
- `tf.enable_eager_execution()`

- `ds = tf.data.TextLineDataset("file.txt")`
- `ds = ds.map(lambda line: tf.string_split([line]).values)`

```
try:
  for value in ds.take(2):
    print("value:",value)
except tf.errors.OutOfRangeError:
  pass
```

TF 2 map() and take(): file.txt

- this is file line #1
- this is file line #2
- this is file line #3
- this is file line #4
- this is file line #5

TF 2 map() and take(): output

- ('value:', <tf.Tensor: id=16, shape=(5,), dtype=string, numpy=array(['this', 'is', 'file', 'line', '#1'], dtype=object)>)

- ('value:', <tf.Tensor: id=18, shape=(5,), dtype=string, numpy=array(['this', 'is', 'file', 'line', '#2'], dtype=object)>)

TF 2 map() and take()

```
import tensorflow as tf
import numpy as np

x = np.array([[1],[2],[3],[4]])

# make a ds from a numpy array
ds = tf.data.Dataset.from_tensor_slices(x)
ds = ds.map(lambda x: x*2).map(lambda x: x+1).map(lambda x: x**3)

for value in ds.take(4):
  print("value:",value)
```

TF 2 map() and take(): output

- value: tf.Tensor([27], shape=(1,), dtype=int64)
- value: tf.Tensor([125], shape=(1,), dtype=int64)
- value: tf.Tensor([343], shape=(1,), dtype=int64)
- value: tf.Tensor([729], shape=(1,), dtype=int64)

TF 2 batch(): EXTRA CREDIT

```
import tensorflow as tf
import numpy as np

x = np.arange(0, 12)

def gener():
  i = 0
  while(i < len(x/3)):
    yield (i, i+1, i+2) # three integers at a time
    i += 3

ds = tf.data.Dataset.from_generator(gener, (tf.int64,tf.int64,tf.int64))
third = int(len(x)/3)
for value in ds.take(third):
  print("value:",value)
```

TF 2 batch() & zip(): EXTRA CREDIT

```
import tensorflow as tf

ds1 = tf.data.Dataset.range(100)
ds2 = tf.data.Dataset.range(0, -100, -1)
ds3 = tf.data.Dataset.zip((ds1, ds2))
ds4 = ds3.batch(4)

for value in ds.take(10):
  print("value:",value)
```

Working with tf.keras

- tf.keras.layers
- tf.keras.models
- tf.keras.optimizers
- tf.keras.utils
- tf.keras.regularizers

Working with tf.keras.models

- tf.keras.models.Sequential(): most common

- clone_model(...): Clone any Model instance
- load_model(...): Loads a model saved via save_model
- model_from_config(...): Instantiates a Keras model from its config
- model_from_json(...):
- Parses a JSON model config file and returns a model instance
- model_from_yaml(...):
- Parses a yaml model config file and returns a model instance
- save_model(...): Saves a model to a HDF5 file

Working with tf.keras.layers

- tf.keras.models.Sequential()

- tf.keras.layers.Conv2D()
- tf.keras.layers.MaxPooling2D()
- tf.keras.layers.Flatten()
- tf.keras.layers.Dense()
- tf.keras.layers.Dropout(0.1)
- tf.keras.layers.BatchNormalization()
- tf.keras.layers.embedding()
- tf.keras.layers.RNN()
- tf.keras.layers.LSTM()
- tf.keras.layers.Bidirectional (ex: BERT)

TF 2 Activation Functions

- tf.keras.activations.relu
- tf.keras.activations.selu
- tf.keras.activations.elu
- tf.keras.activations.linear
- tf.keras.activations.sigmoid
- tf.keras.activations.softmax
- tf.keras.activations.softplus
- tf.keras.activations.tanh
- Others …

TF 2 Datasets

- tf.keras.datasets.boston_housing
- tf.keras.datasets.cifar10
- tf.keras.datasets.cifar100
- tf.keras.datasets.fashion_mnist
- tf.keras.datasets.imdb
- tf.keras.datasets.mnist
- tf.keras.datasets.reuters

TF 2 “Experimental”

- tf.keras.experimental.CosineDecay
- tf.keras.experimental.CosineDecayRestarts
- tf.keras.experimental.LinearCosineDecay
- tf.keras.experimental.NoisyLinearCosineDecay
- tf.keras.experimental.PeepholeLSTMCell

TF 2 Optimizers

- tf.optimizers.Adadelta
- tf.optimizers.Adagrad
- tf.optimizers.Adam
- tf.optimizers.Adamax
- tf.optimizers.SGD

TF 2 Callbacks

- tf.keras.callbacks.Callback
- tf.keras.callbacks.EarlyStopping
- tf.keras.callbacks.LambaCallback
- tf.keras.callbacks.ModelCheckpoint
- tf.keras.callbacks.TensorBoard
- tf.keras.callbacks.TerminateOnNan

TF 2 Losses

- tf.losses.BinaryCrossEntropy
- tf.losses.CategoricalCrossEntropy
- tf.losses.CategoricalHinge
- tf.losses.CosineSimilarity
- tf.losses.Hinge
- tf.losses.Huber
- tf.losses.KLD
- tf.losses.KLDivergence
- tf.losses.MAE
- tf.losses.MSE
- tf.losses.SparseCategoricalCrossentropy

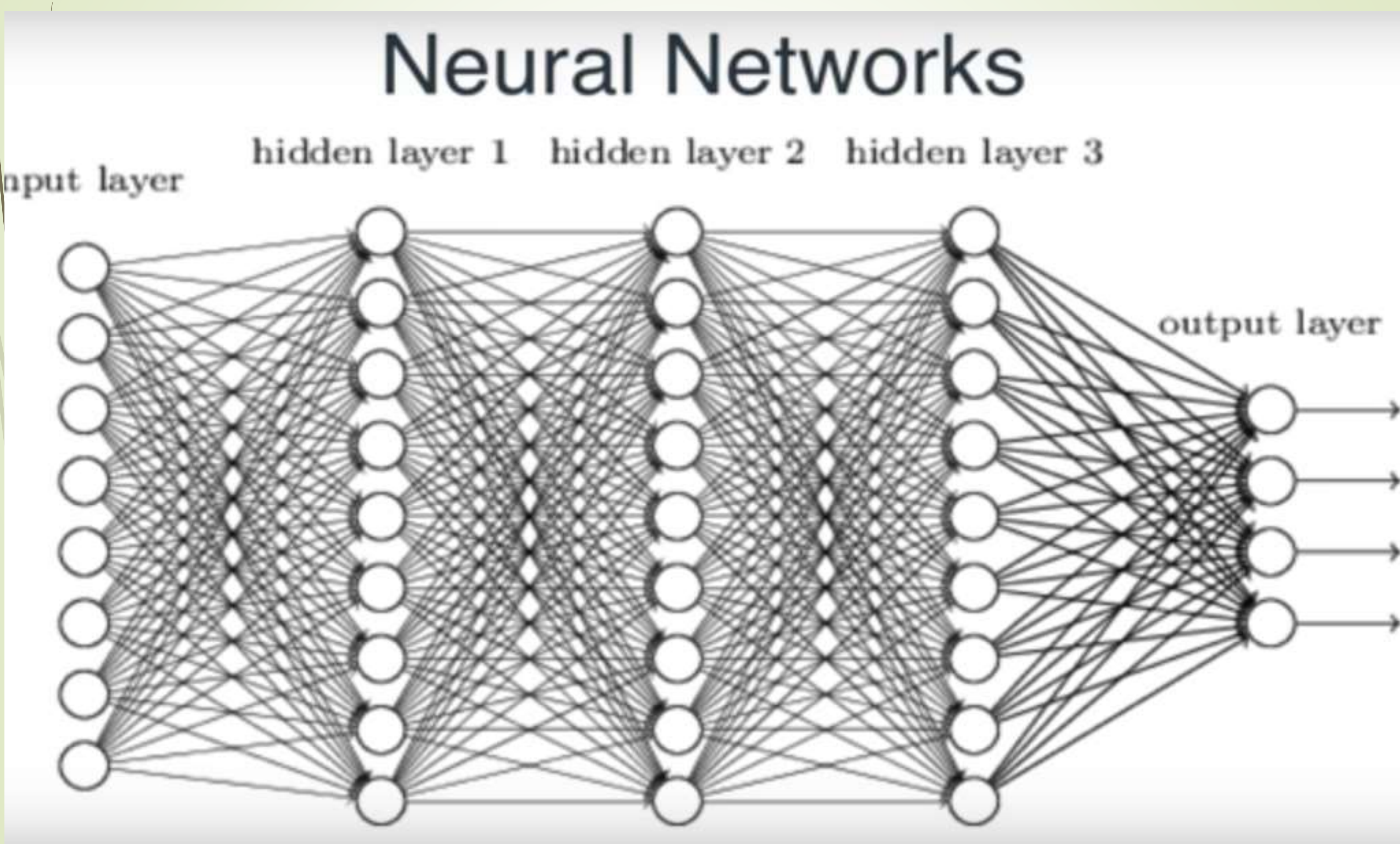
Other TF 2 Namespaces

- tf.keras.regularizers (L1 and L2)
- tf.keras.utils (to_categorical)
- tf.keras.preprocessing.text.Tokenizer

TF 1.x Model APIs

- tf.core (“reduced” in TF 2)
- tf.layers (deprecated in TF 2)
- tf.keras (the primary API in TF 2)
- tf.estimator.Estimator (implementation in tf.keras)
- tf.contrib.learn.Estimator (Deprecated in TF 2)

Neural Network: 3 Hidden Layers



TF 2/Keras and MLPs

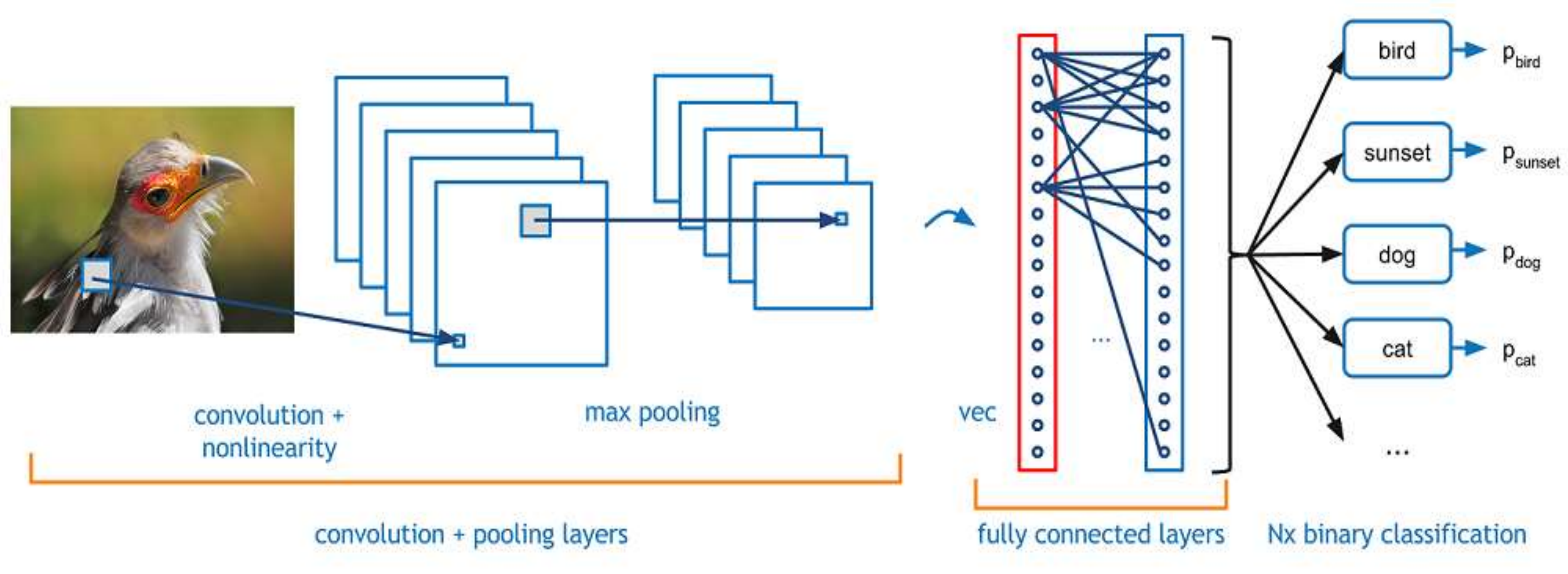
- import tensorflow as tf
- . . .
- model = tf.keras.models.Sequential()
- model.add(tf.keras.layers.Dense(10, input_dim=num_pixels, activation='relu'))
- model.add(tf.keras.layers.Dense(30, activation='relu'))
- model.add(tf.keras.layers.Dense(10, activation='relu'))
- model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))
- model.compile(tf.keras.optimizers.Adam(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])

TF 2/Keras and MLPs

- import tensorflow as tf
- . . .
- model = tf.keras.models.Sequential([

```
tf.keras.layers.Dense(10, input_dim=num_pixels, activation='relu'),
tf.keras.layers.Dense(30, activation='relu'),
tf.keras.layers.Dense(10, activation='relu'),
tf.keras.layers.Dense(num_classes, activation='softmax'),
tf.keras.optimizers.Adam(lr=0.01), loss='categorical_crossentropy',
metrics=['accuracy'])
])
```

CNNs: Convolution and Pooling



TF 2/Keras and CNNs

- import tensorflow as tf
- . . .
- `model = tf.keras.models.Sequential([`
- tf.keras. layers.Conv2D(30,(5,5), input_shape=(28,28,1),activation='relu')),
- tf.keras.layers.MaxPooling2D(pool_size=(2, 2))),
- tf.keras.layers.Flatten(),
- tf.keras.layers.Dense(500, activation='relu’),
- tf.keras.layers.Dropout(0.5),
- tf.keras.layers.Dense(num_classes, activation='softmax’)
- `])`

What are RNNs?

- RNNs = Recurrent Neural Networks
- RNNs capture/retain events in time
- FF networks do not learn from the past
- => RNNs DO learn from the past
- RNNs contain layers of recurrent neurons

Common Types of RNNs

- 1) Simple RNNs (minimal structure)
- 2) LSTMs (input, forget, output gates)
- 3) Gated Recurrent Unit (GRU)
- NB: RNNs can be used with MNIST data

Some Use Cases for RNNs

- 1) Time Series Data

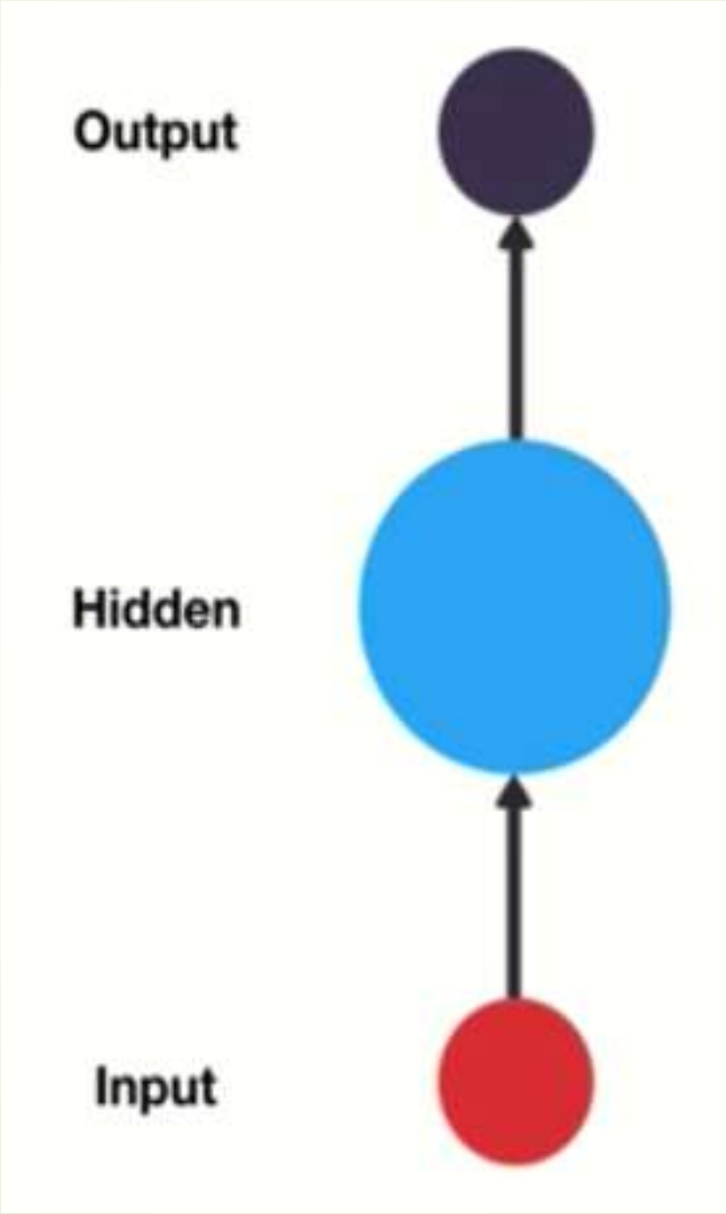
- 2) NLP (Natural Language Processing):

Processing documents

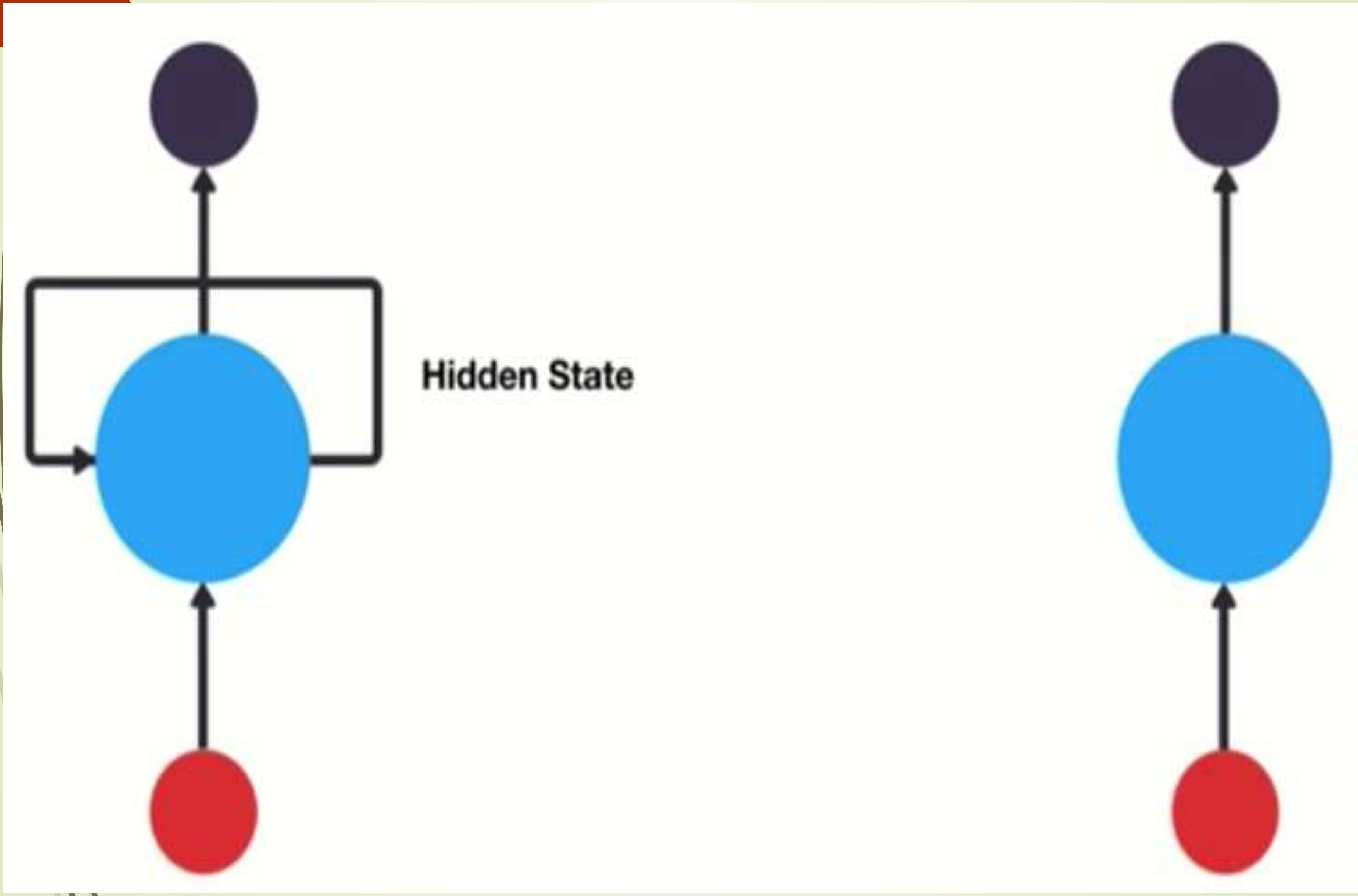
“analyzing” Trump’s tweets

- 3) RNNs and MNIST dataset

Recall: Feed Forward Structure



RNN Cell (left) and FF (right)

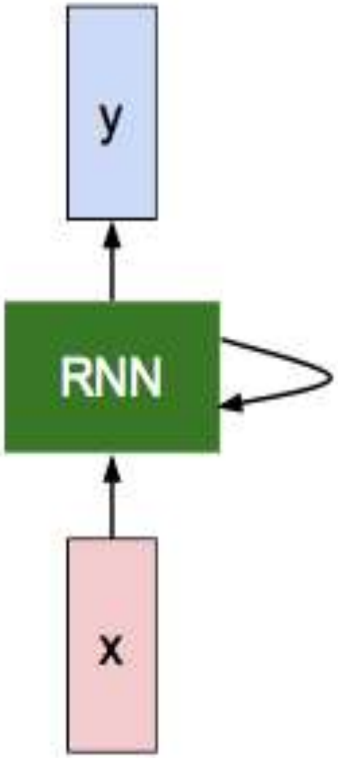


RNN “Big Picture” (cs231n)

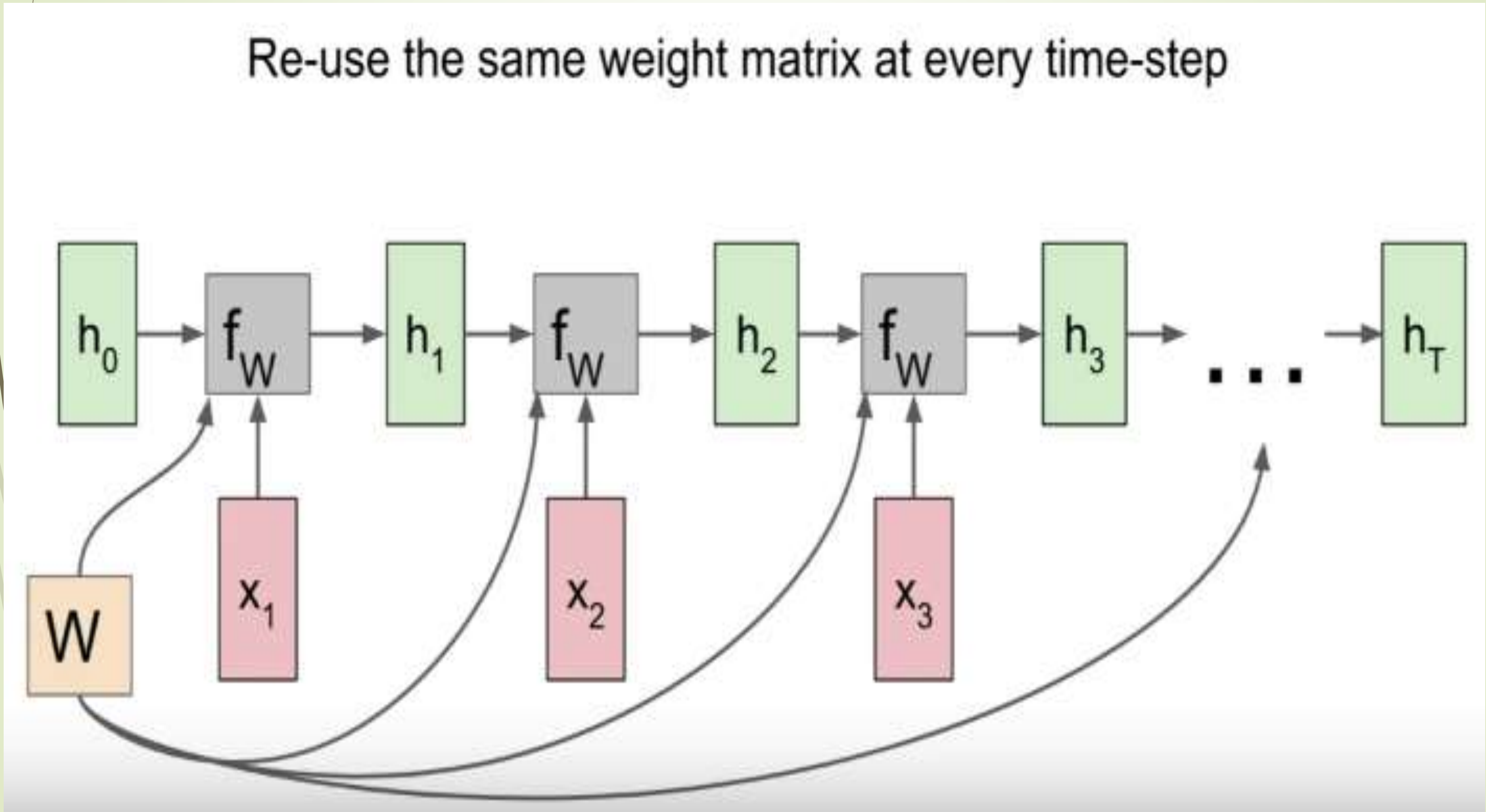
Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

$$\underbrace{\boxed{h_t}}_{\text{new state}} = \underbrace{\boxed{f_W}}_{\substack{\text{some function}\\\text{with parameters W}}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\substack{\text{input vector at}\\\text{some time step}}})$$



RNN Neuron: same W for all steps



RNN and MNIST Dataset

- Sample initialization values for RNN:

```
n_steps   = 28  # # of time steps
n_inputs  = 28  # number of rows
n_neurons = 150 # 150 neurons in one layer
n_outputs = 10  # 10 digit classes (0->9)
```

- Remember that:

one layer = one memory cell

MNIST class count = size of output layer

Problems with RNNs

- lengthy training time
- Unrolled RNN will be very deep
- Propagating gradients through many layers
- Prone to vanishing gradient
- Prone to exploding gradient

What are LSTMs?

- Long short-term memory (LSTM):

a model for the short-term memory

Contains more state than a “regular” RNN

which can last for a long period of time

can forget and remember things selectively

- Hochreiter/Schmidhuber proposed in 1997

- improved in 2000 by Felix Gers' team

- set accuracy records in multiple apps

Features of LSTMs

- Used in Google speech recognition + Alpha Go
- they avoid the vanishing gradient problem
- Can track 1000s of discrete time steps
- Used by international competition winners

Use Cases for LSTMs

- Connected handwriting recognition
- Speech recognition
- Forecasting
- Anomaly detection
- Pattern recognition

Advantages of LSTMs

- well-suited to classify/process/predict time series
- More powerful: increased memory (=state)
- Can explicitly add long-term or short-term state
- even with time lags of unknown size/duration between events

The Primary LSTM Gates

- Input, Forget, and Output gates (FIO)
- The main operational block of LSTMs
- an "information filter"
- a multivariate input gate
- some inputs are blocked
- some inputs "go through"
- "remember" necessary information

- FIO gates use the sigmoid activation function
- The cell state uses the tanh activation function

LSTM Gates and Cell State

- 1) the input gate controls:

the extent to which a new value flows into the cell state

- 2) the forget gate controls:

the extent to which a value remains in the cell state

- 3) the output gate controls:

The portion of the cell state that’s part of the output

- 4) the cell state maintains long-term memory

- => LSTMs store more memory than basic RNN cells

Common LSTM Variables

- # LSTM unrolled through 28 time steps (with MNIST):
- time_steps = 28

- # number of hidden LSTM units:
- num_units = 128

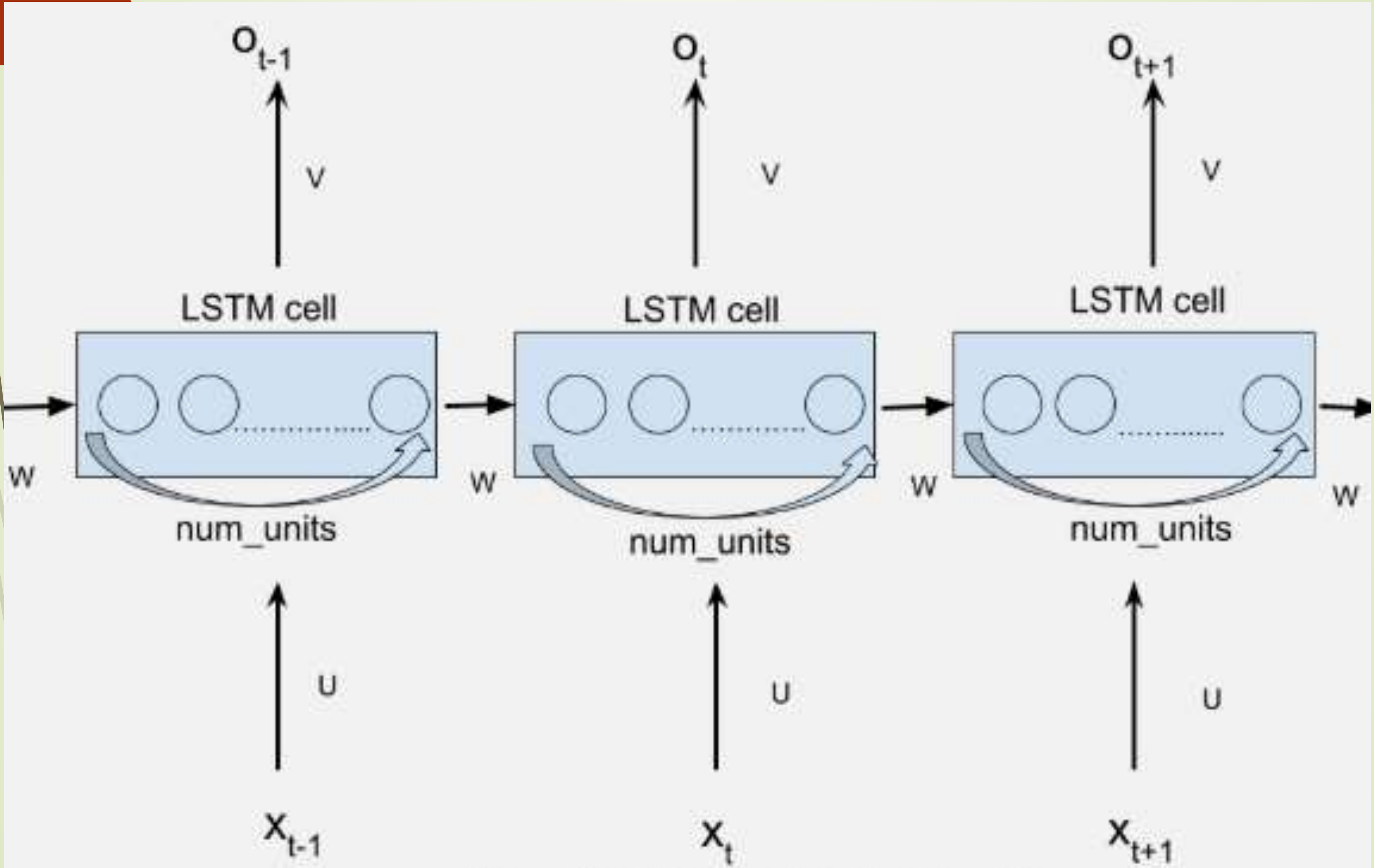
- # number of rows of 28 pixels:
- n_inputs = 28

- # number of pixels per row:
- input_size = 28

- # mnist has 10 classes (0-9):
- n_classes = 10

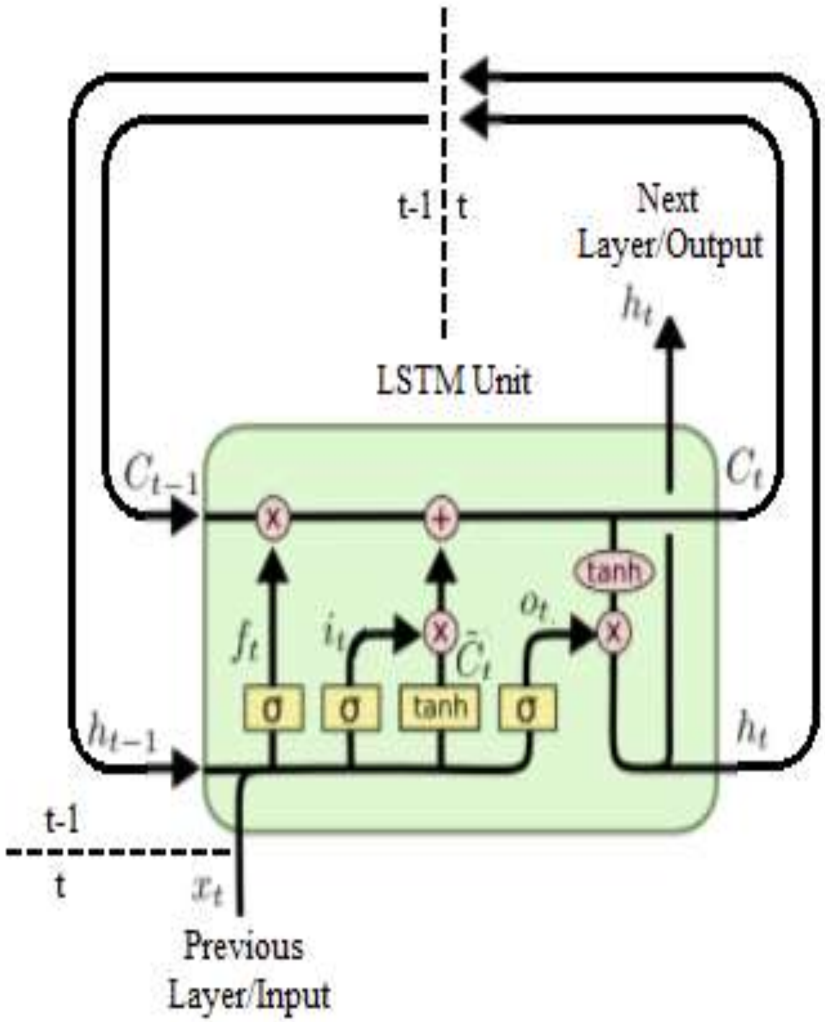
- # size of input batch:
- batch_size = 128

Basic LSTM Cells



Interpreting LSTM cell and num_units

LSTM Formulas



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

TF 2/Keras and LSTMs

import tensorflow as tf

- . . .

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTMCell(6,batch_input_shape=(1,1,1),kernel_initializer='ones',stateful=True))
model.add(tf.keras.layers.Dense(1))
. . .
```

=> LSTM versus LSTMCell:

https://stackoverflow.com/questions/48187283/whats-the-difference-between-lstm-and-lstmcell

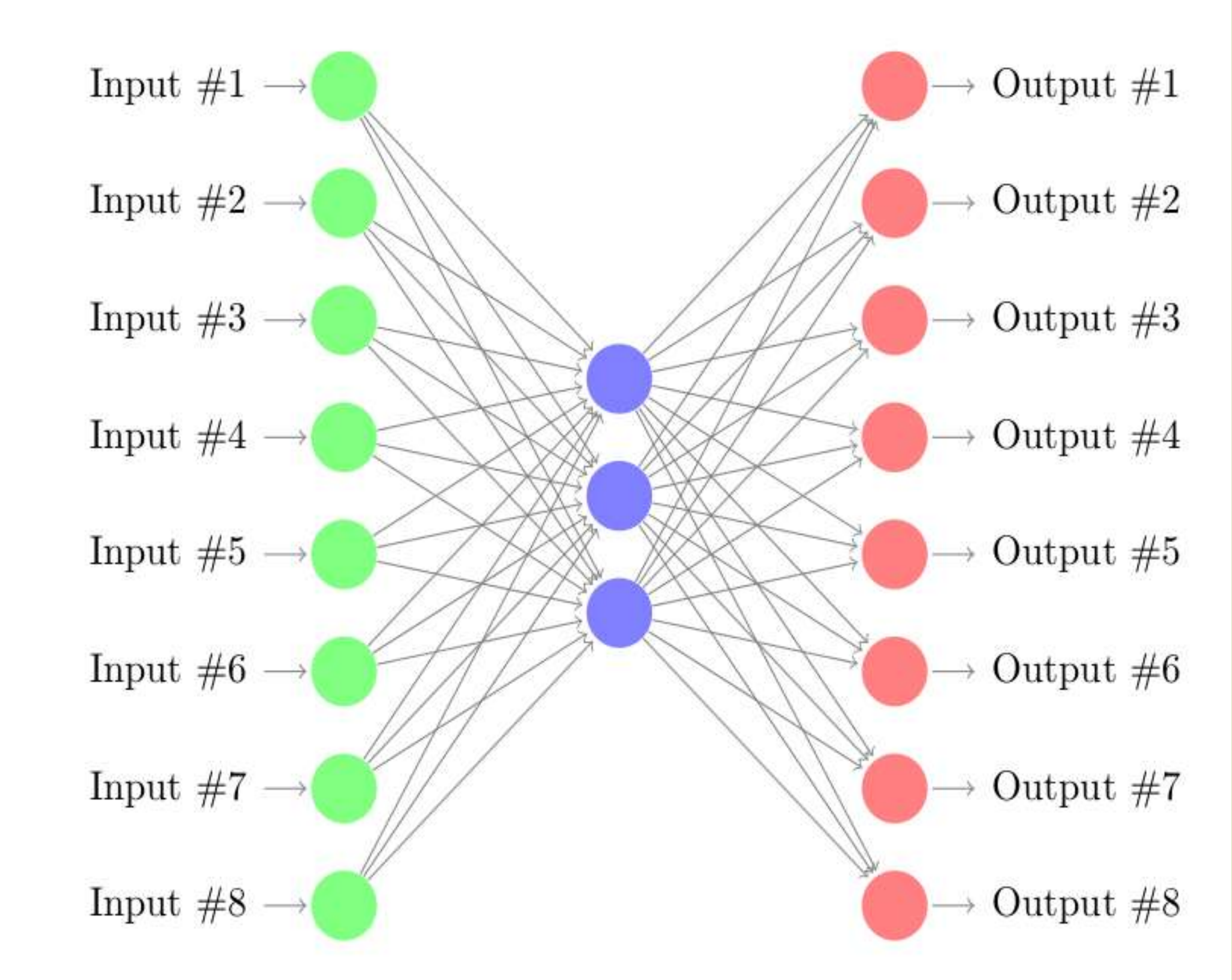
=> How to define a custom LSTM cell:

https://stackoverflow.com/questions/54231440/define-custom-lstm-cell-in-keras

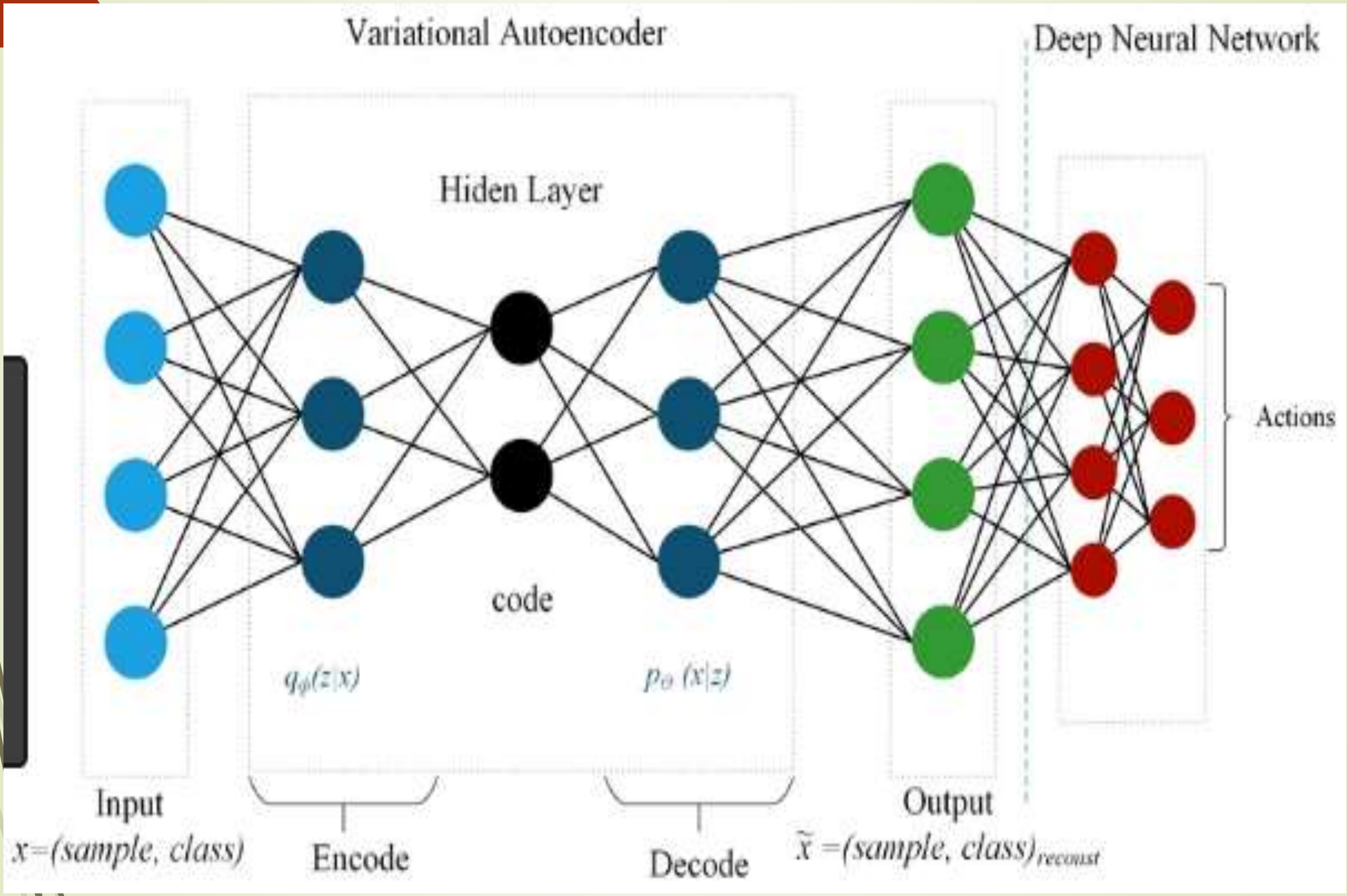
TF 2/Keras & BiDirectional LSTMs

- import tensorflow as tf
- . . .
- model = Sequential()
- `model.add(Bidirectional(LSTM(10, return_sequences=True), input_shape=(5,10)))`
- `model.add(Bidirectional(LSTM(10)))`
- model.add(Dense(5))
- model.add(Activation('softmax'))
- model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

Autoencoders



Variational Autoencoders (2013)



AEs, VAEs, and GANs

- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

- TensorFlow 2 and Autoencoders:

https://gist.github.com/AFAgarap/326af55e36be0529c507f1599f88c06e

- TensorFlow 2 and Variational Autoencoders:

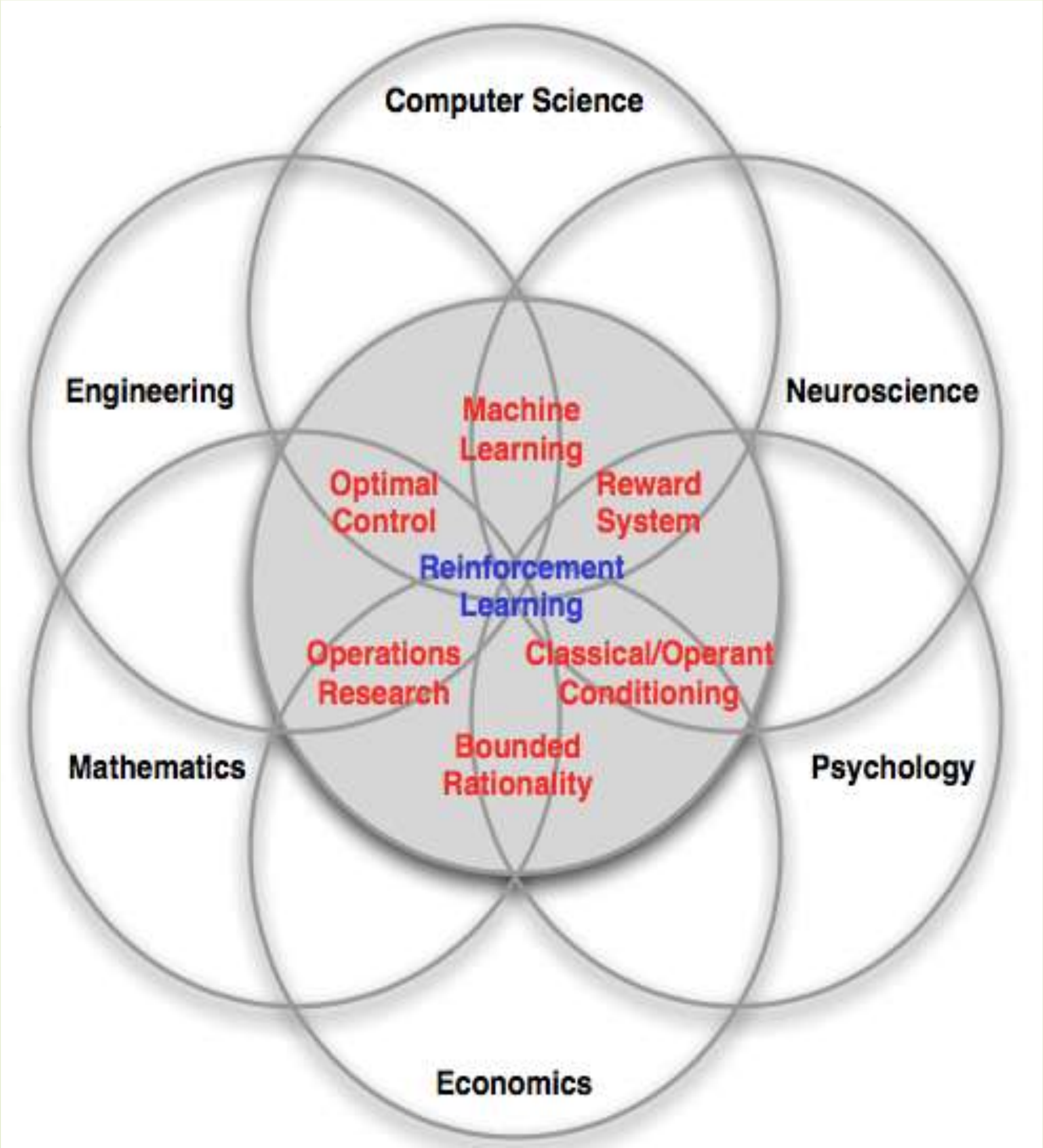
- Convolutional Variational Autoencoder:

https://www.tensorflow.org/alpha/tutorials/generative/cvae

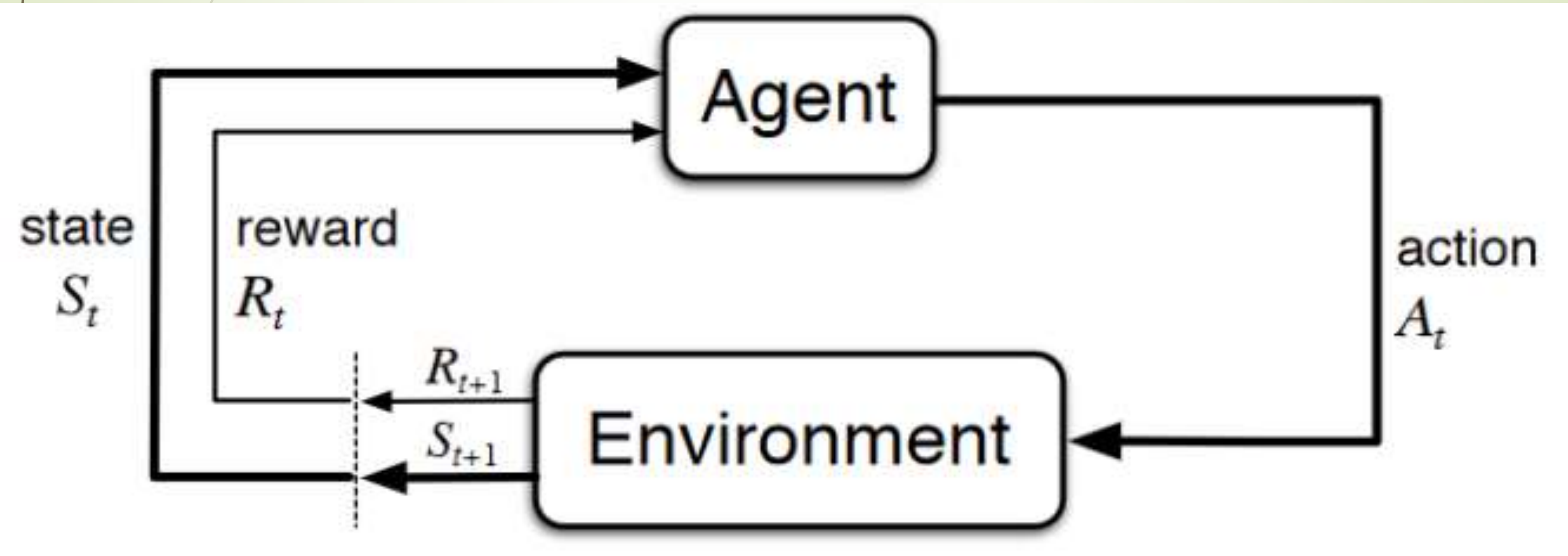
- Variational Autoencoders and GANS:

https://www.youtube.com/watch?v=KFX4apL7a4s

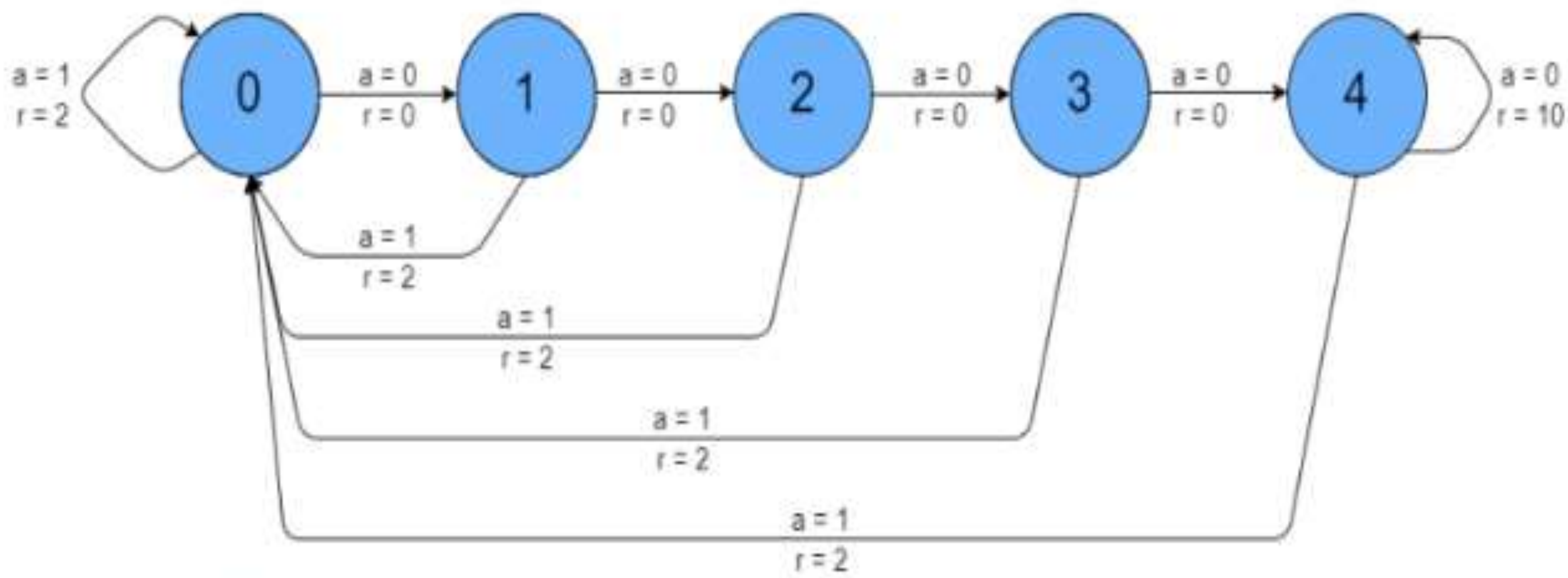
Reinforcement Learning



Reinforcement Learning

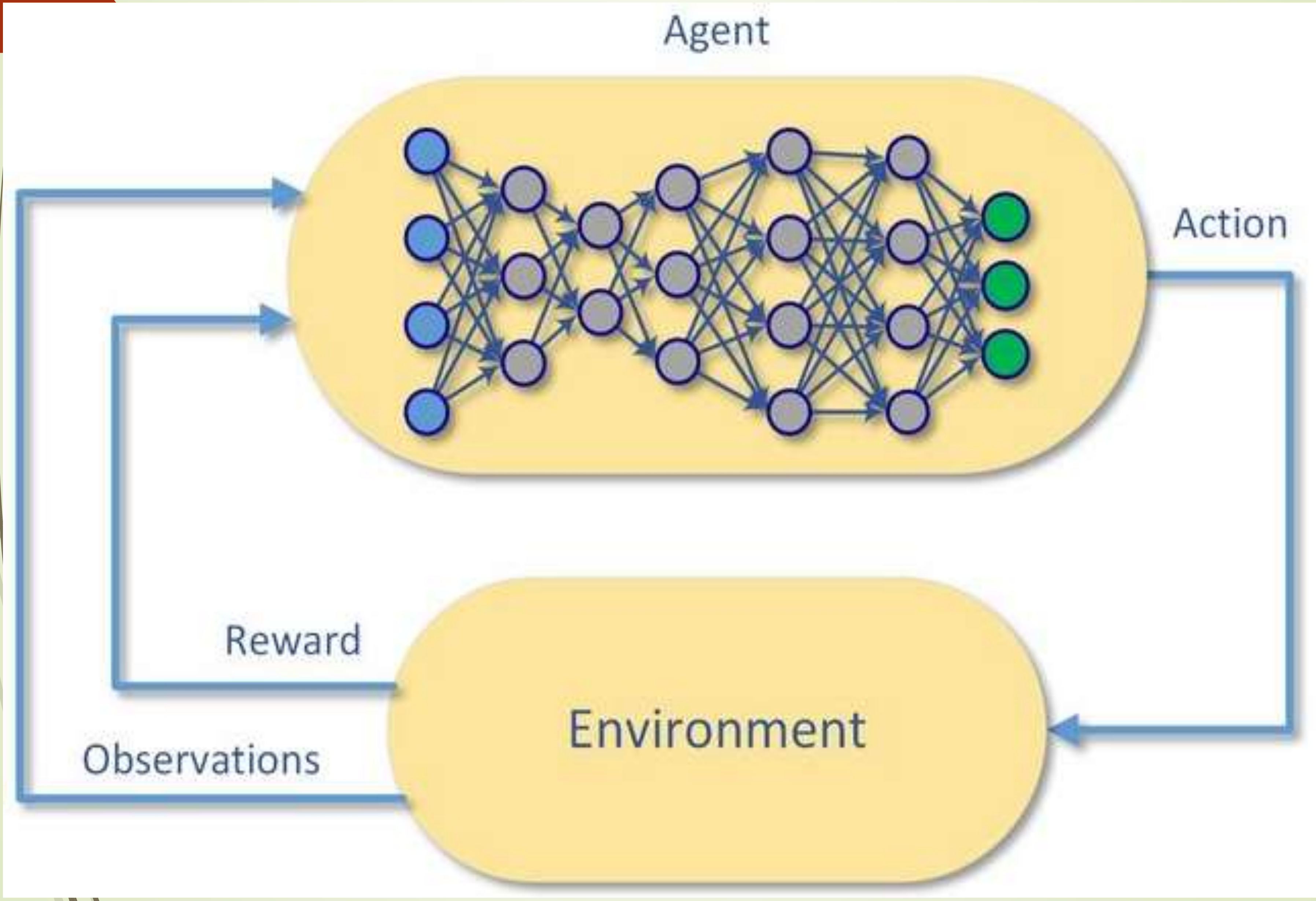


Reinforcement Learning



Open AI Gym's NChain environment

Deep Reinforcement Learning



Reinforcement Learning

- Dopamine on TensorFlow (Research Framework):

https://github.com/google/dopamine

- TF-Agents library for RL in TensorFlow:

https://github.com/tensorflow/agents

- Keras and Reinforcement Learning:

https://github.com/keras-rl/keras-rl

- OpenAI universe, DeepMind Lab, TensorForce