Machine Learning and Deep Learning
Lecture-10

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Topics

- Regularization: BatchNorm
- Recurrent Neural Networks (RNNs)
- Distributed Deep learning
- TensorFlow

Regularization: BatchNorm

BatchNorm

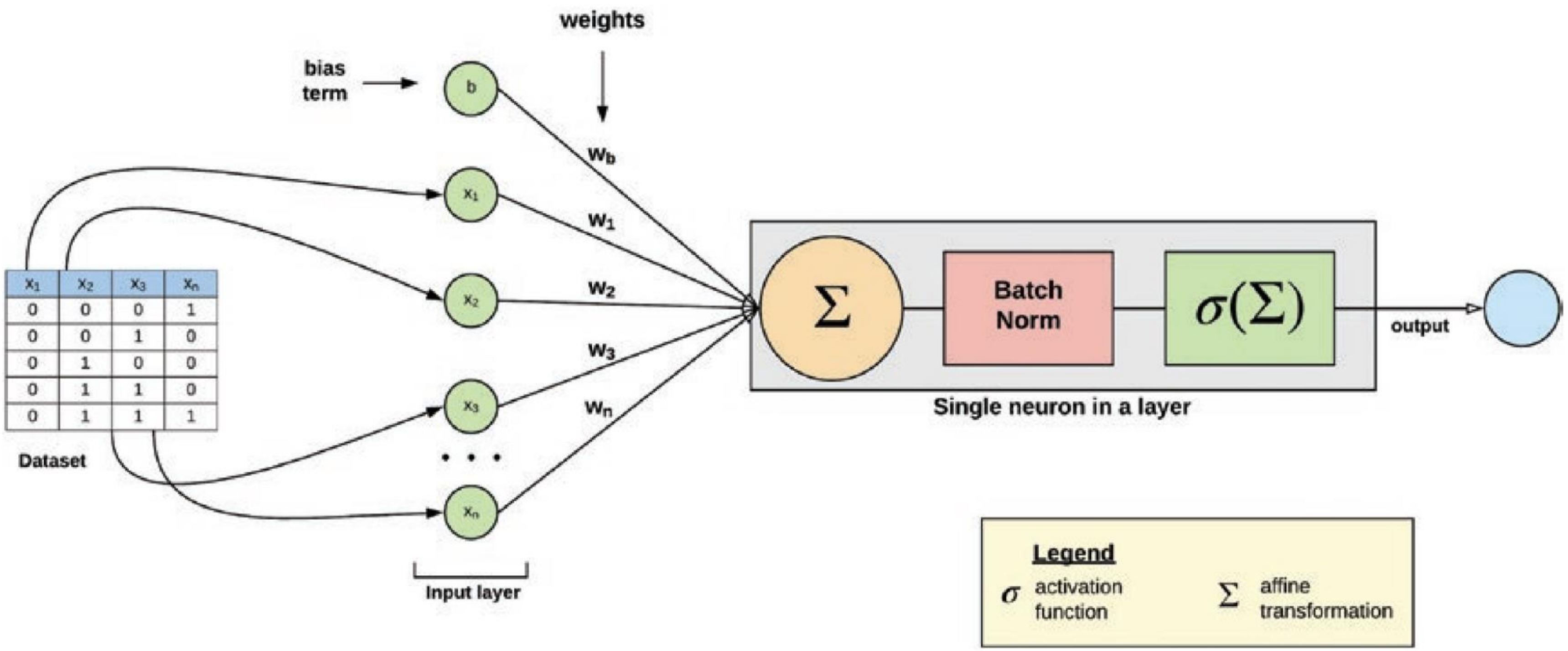
- Vanishing gradients problem can be alleviated with better weight initialization, better optimizers or Batch Normalization[1].
- BatchNorm most successful architectural innovations in deep learning[2].
- BatchNorm aims to stabilize distribution (over a minibatch) of inputs to a given network layer during training.

- BatchNorm Working: Operation lets model learn optimal scale and mean of each of layer’s inputs.

1. First add an operation in model just before or after activation function of each hidden layer. This operation simply zero-center and normalizes each input.
2. Next, scales and shifts result using scaling and shifting vectors [per layer].

1. Sergey Ioffe and Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Proceedings of the 32nd International Conference on Machine Learning (2015): 448–456.
2. Santurkar, Shibani, et al. "How does batch normalization help optimization?." *Advances in Neural Information Processing Systems*. 2018.

BatchNorm



x_1	x_2	x_3	x_n
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
0	1	1	1

- Note: If you add a BN layer as the very first layer of your NN, you do not need to standardize your training set; the BN layer will do it for you.
- Vanishing gradients problem can be reduced to a point that activation functions can be used for further solution.
- BN can improve many deep neural networks.

Bisong E. (2019) More on Optimization Techniques. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA

BatchNorm

- Batch Normalization is tricky to use in RNNs but sufficient for other nets.
  - Gradient Clipping* is often used RNNs to mitigate exploding gradients problem.
    - Gradient Clipping clips the gradients during backpropagation so that they never exceed some threshold.
  - Gradient clipping does not help with vanishing gradients.
- BN acts like a regularizer reducing need for other regularization techniques (such as dropout).
- BN adds runtime penalty to neural network.
  - Training is rather slow because each epoch takes much more time when BN in use.

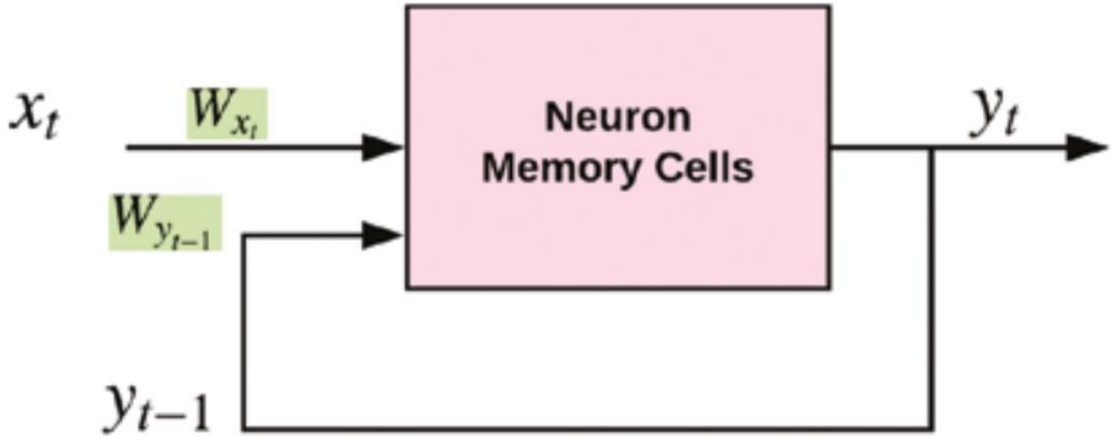
*Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International conference on machine learning*. 2013.

RNN

Recurrent Neural Networks (RNNs)

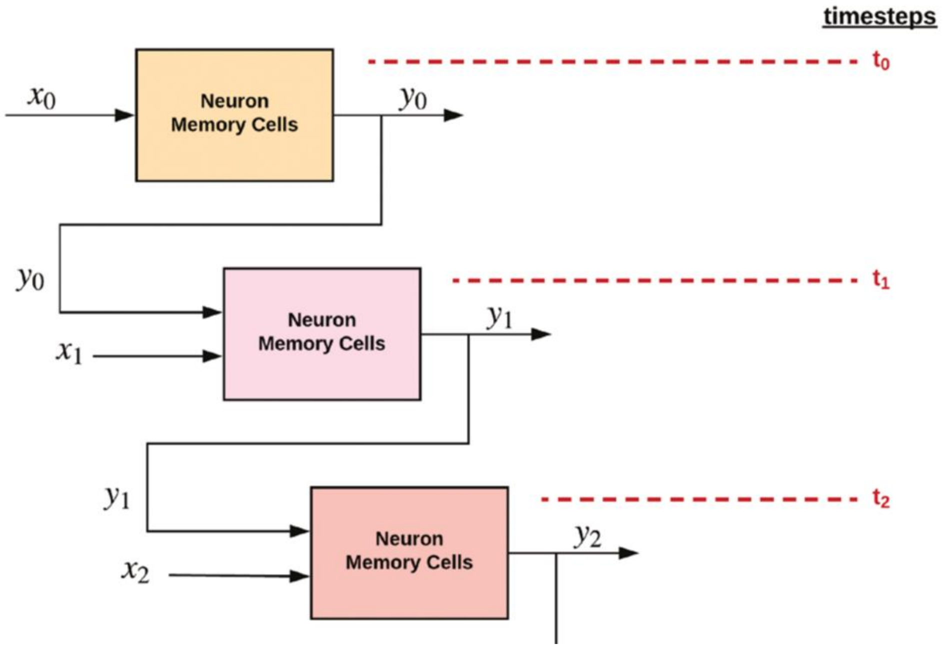
- RNNs developed to solve learning problems (time series or sequential tasks) where information about past (i.e., past instants/events) is directly linked to making future predictions.

- Recurrent Neuron: maintains a memory or a state from past computations.
  - Data is looped back into same neuron at every new time instant.
  - It takes input as output of previous instant y_{t-1} in addition to its current input at instant x_t.
  - The recurrent neuron has two input weights, W_{xt} and W_{yt-1}



Recurrent Neural Networks (RNNs)

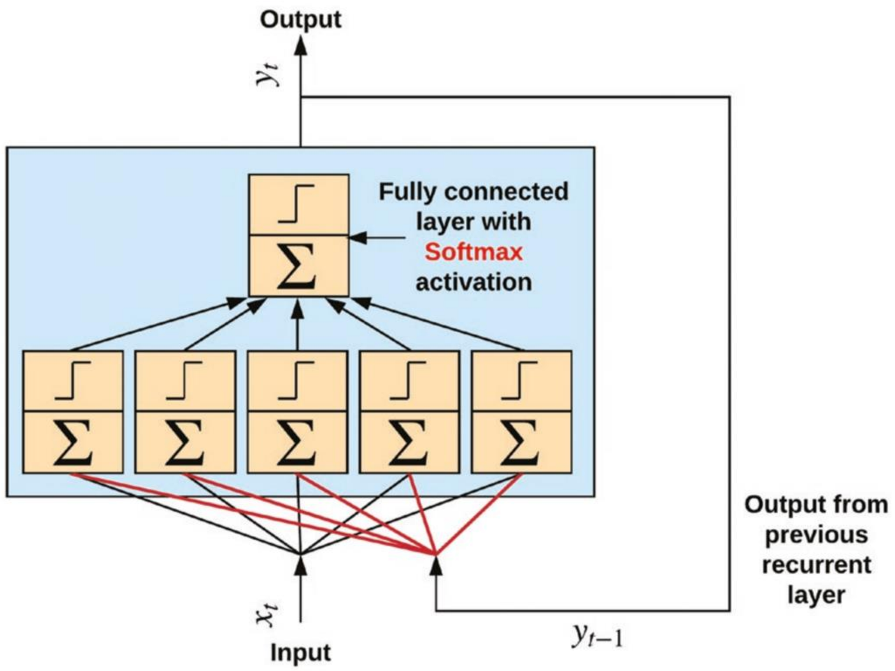
- Recurrent Computational Graph: RNN formalized as an unfolded computational graph.
- An unfolded computational graph shows information flow through recurrent layer at every time instant in the sequence.



A sequence of five-time steps: We will unfold recurrent neuron five-times across number of instants.

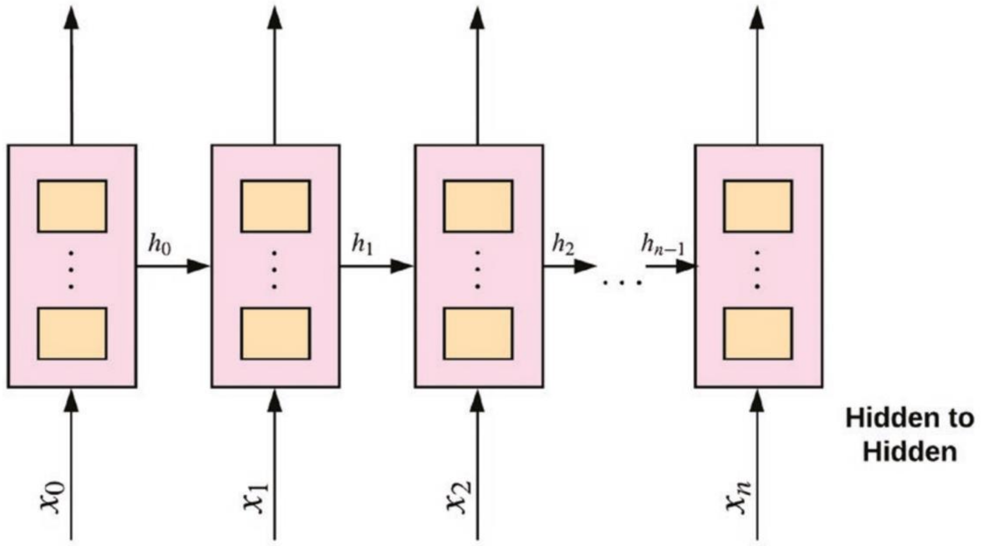
Recurrent Neural Networks (RNNs)

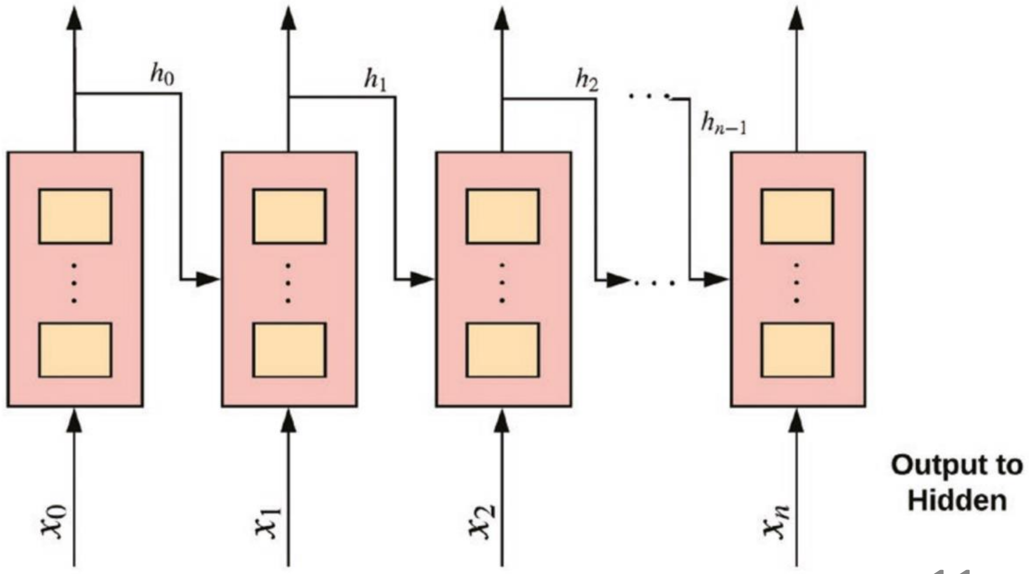
- Computations within a recurrent layer:
  - Each neuron in a recurrent layer (RL) receives as input, output of previous layer and its current input.
  - Neurons each have two weight vectors.
  - Neurons perform an affine transformation of inputs and pass it through a non-linear activation function (tanh).
  - Within RL, output of neurons is moved to a dense or fully connected layer with a softmax activation function as output of class probabilities.



Recurrent Connection Schemes

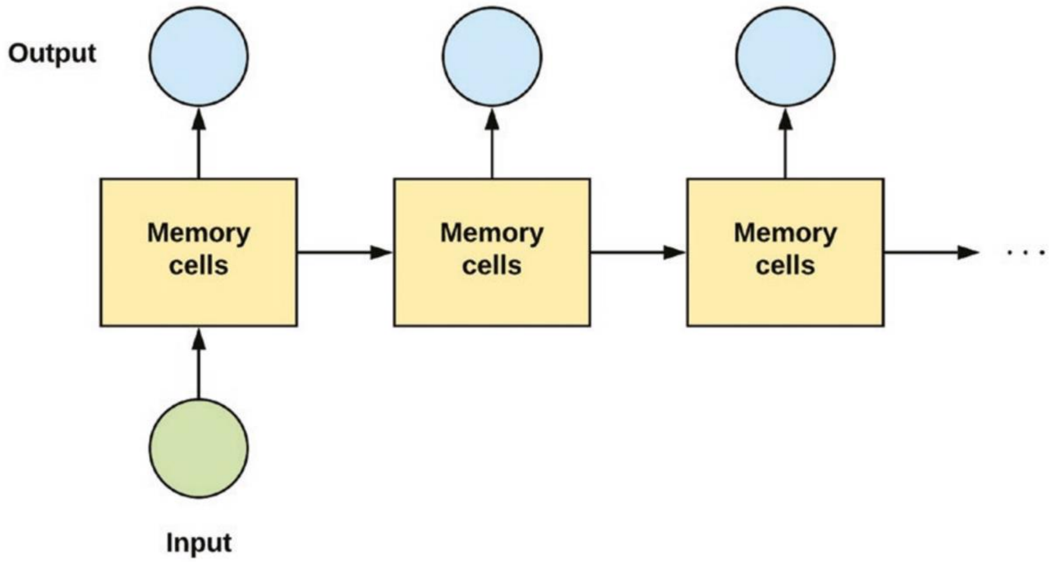
- Two main schemes for forming recurrent connections from one recurrent layer to another:
  - Recurrent connections between hidden units.
    - Better captures high-dimensional features about past.
  - Recurrent connections between output of previous layer and hidden unit.
    - Easy to compute and parallelizable.



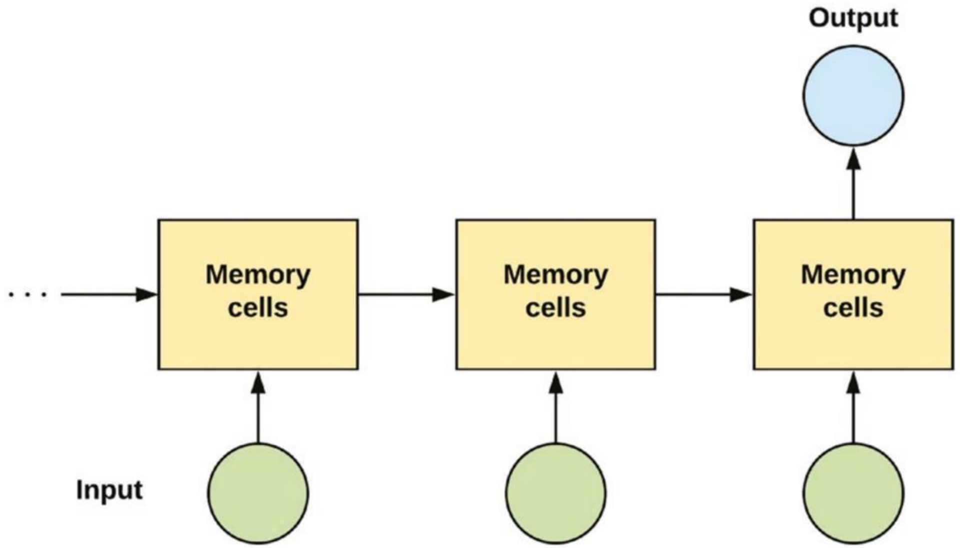


RNN for Sequence Problems

- An input to a sequence of output: when an image is passed as an input to network, and output is a sequence of words (Image captioning problem)



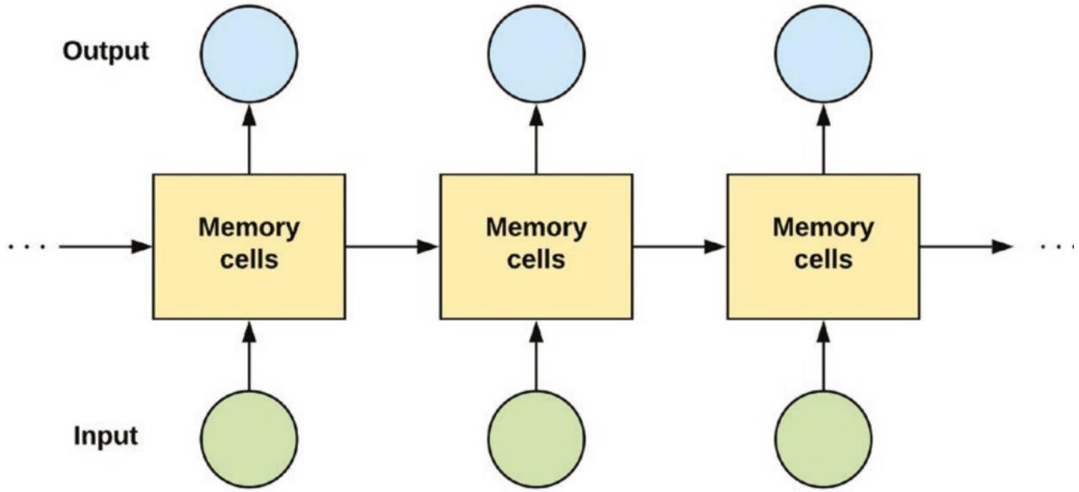
- A sequence of inputs to an output: Pass a sequence of words as input to network, and output is a class indicating either a positive or negative review or sentiment (Sentiment Analysis)

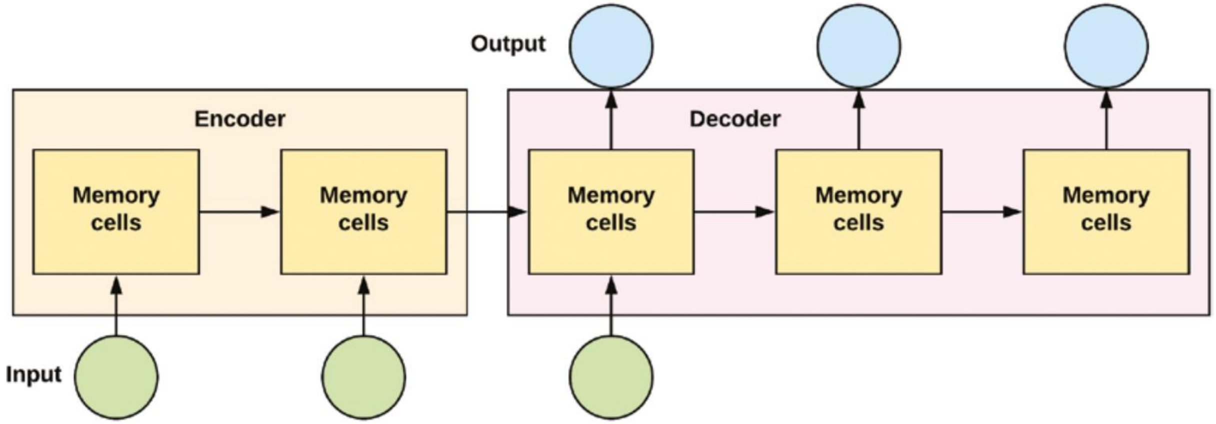


RNN for Sequence Problems

- Synced sequence input to output: need to label each video frame (Video Classification).

- Encoder-decoder/sequence-to-sequence architecture: A sequence of words in a language as input, and we want a sequence of words as output in another language (Machine translation and Speech recognition)



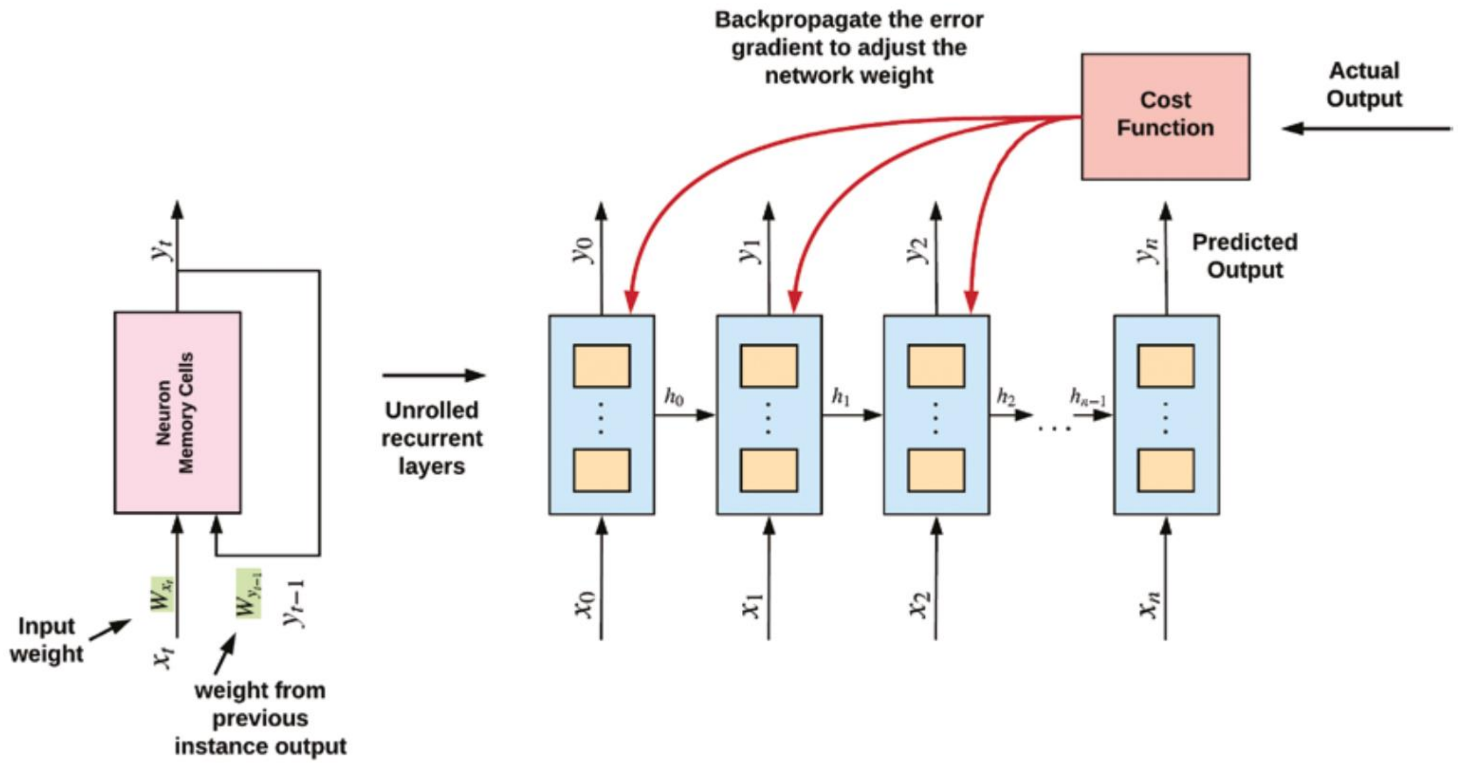


Recurrent Network Training

- RNN is trained by backpropagation through time (BPTT).
  - Because standard backpropagation cannot work in loop or recurrent structure.
    - Training a network using backpropagation involves calculating error gradient, moving backward from output layer through hidden layers of network and adjusting network weights.
  - However, this operation cannot work in recurrent neuron because we have just one neural cell with recurrent connections to itself.
    - Note: Deep RNN is the way to stack multiple layers of cells.
      - Each time step t (called a frame).

Recurrent Network Training

BPTT:
1. Unroll recurrent neuron across time instants
2. Apply backpropagation to unrolled neurons at each time layer same way it is done for a traditional feedforward NN.



Recurrent Network Training

- Challenge of training RNN is vanishing and exploding gradient problem.
  - Due to long-term dependencies or time instant of unrolled recurrent neuron RNN suffers.
  - Gradient clipping, BatchNorm and ReLU can be used for vanishing and exploding gradient problem.

- (Exploding and vanishing gradients + discards early time instances) leads to development of a memory cell called the Long Short-Term Memory/LSTM.

Recurrent Network Example

- Built-in RNN layers: keras.layers.SimpleRNN
- fully-connected RNN where the output from previous timestep is to be fed to next timestep.

```
tf.keras.layers.SimpleRNN(
    units, activation='tanh', use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros', kernel_regularizer=None,
    recurrent_regularizer=None, bias_regularizer=None,
activity_regularizer=None,
    kernel_constraint=None, recurrent_constraint=None,
bias_constraint=None,
    dropout=0.0, recurrent_dropout=0.0,
return_sequences=False, return_state=False,
    go_backwards=False, stateful=False, unroll=False, **kwargs
)
```

```
inputs = np.random.random([32, 10, 8]).astype(np.float32)
simple_rnn = tf.keras.layers.SimpleRNN(4)

output = simple_rnn(inputs)  # The output has shape `[32, 4]`.

simple_rnn = tf.keras.layers.SimpleRNN(
    4, return_sequences=True, return_state=True)

# whole_sequence_output has shape `[32, 10, 4]`.
# final_state has shape `[32, 4]`.
whole_sequence_output, final_state = simple_rnn(inputs)
```

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

Distributed DL

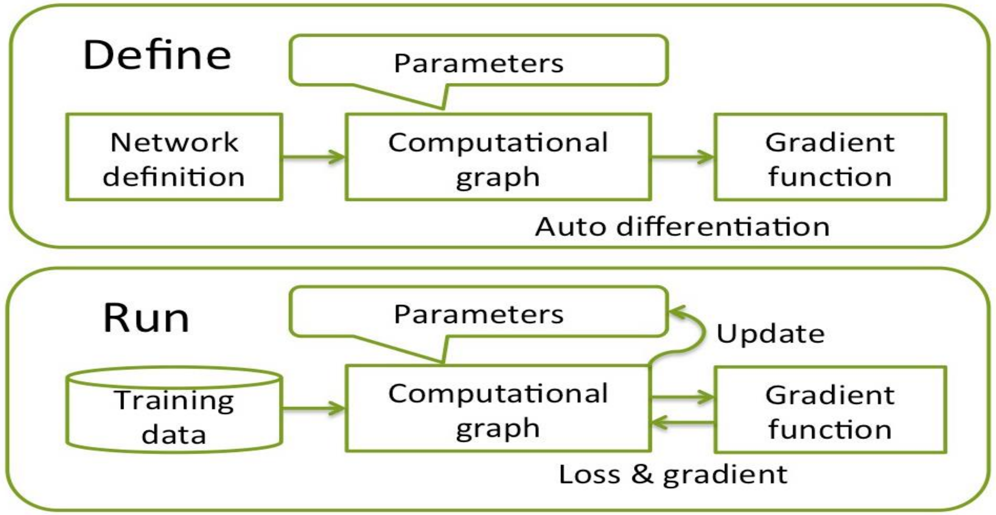
DL Framework

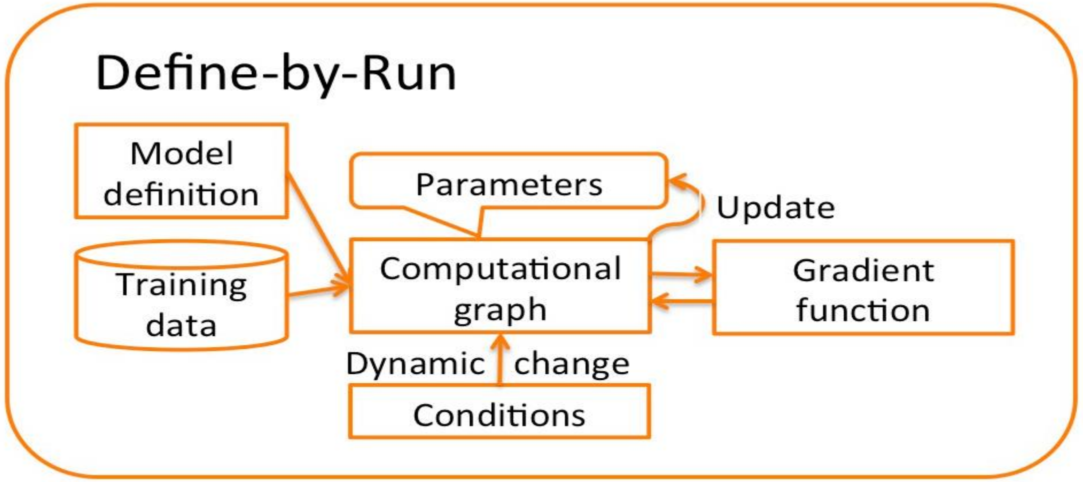
- DL frameworks hide mathematics and focus on design of neural nets.
- Deep Learning frameworks
  - Google TensorFlow/Keras
  - PyTorch(https://pytorch.org/)
  - Caffe (https://caffe.berkeleyvision.org/)
  - Microsoft Cognitive Toolkit (https://cntk.ai)

- Larger and Deeper models examples:
  - LeNet (1998)
  - AlexNet (2012, groundwork for VGG and ResNet.)
  - Residual neural network (ResNet-50, 2015)
  - Transformer (2017)

Model Training

- Steps: 1) build a computational graph from network definition, 2) input training data and compute loss function, 3) update parameters.





- Define-and-run: DL frameworks complete step one in advance of step two (TensorFlow, Caffe).

- Define-by-run: Combines steps one and two into a single step (PyTorch).
  - Computational graph is not given before training but obtained while training.

https://www.oreilly.com/content/complex-neural-networks-made-easy-by-chainer/

Good to know....ONNX

- ONNX: Open Neural Network eXchange (https://onnx.ai/)
- Open-source shared model representation for framework interoperability and shared optimization.
- ONNX defines a common set of operators, data types and a common file format to enable developers to use models with a variety of frameworks, tools, runtimes, and compilers.
- ONNX provides a definition of an extensible computation graph model (Tensor Flow supports it!!)

Non-distributed ML Way

- Standard way to execute ML model

$$\mathrm{argmin}_{\boldsymbol{x}}\, f(\boldsymbol{x})$$

Solved via **optimization procedure**, e.g. **stochastic gradient descent** (SGD).

$$f(x) = \sum_{i=1}^{M} loss(x, exi)$$

Notion of "quality," e.g. **squared distance**

model $\boldsymbol{x}$

E.g., **neural network** or **linear model**

Task

E.g., **classification**

Data

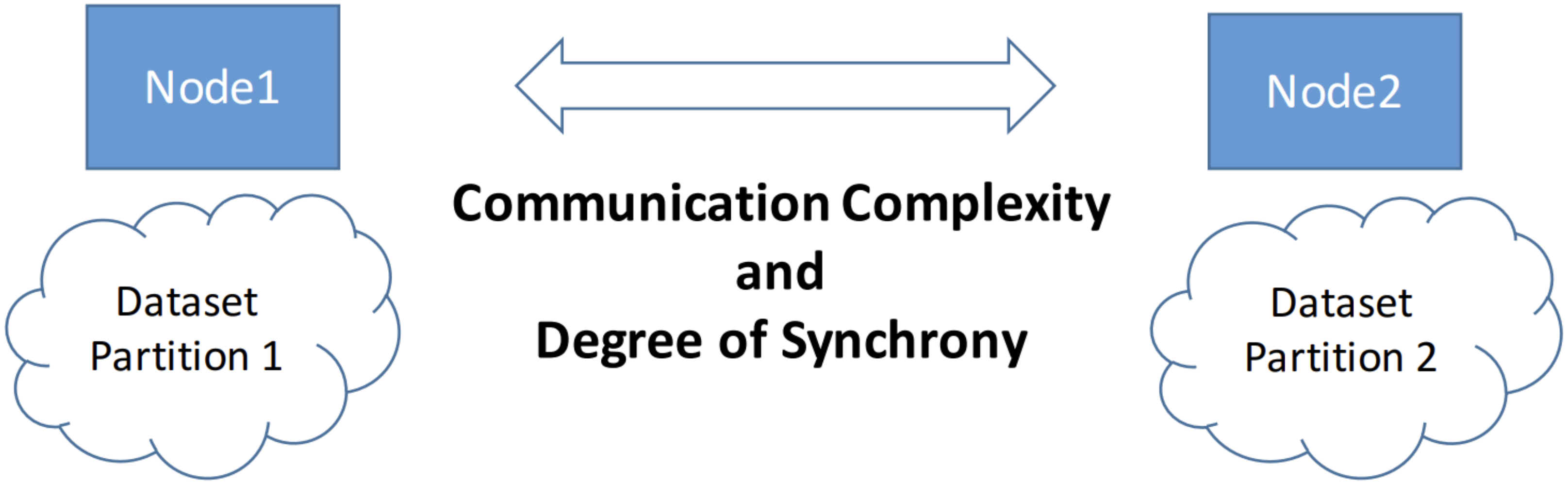
Distributed ML Way

- (Standard) data parallel paradigm, but there are also model parallel or hybrid approaches.

$$\operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$$

$$f_1(x) = \sum_{i=1}^{M/2} l(x, ei) \qquad f_2(x) = \sum_{i=\frac{M}{2}+1}^{M} l(x, ei)$$

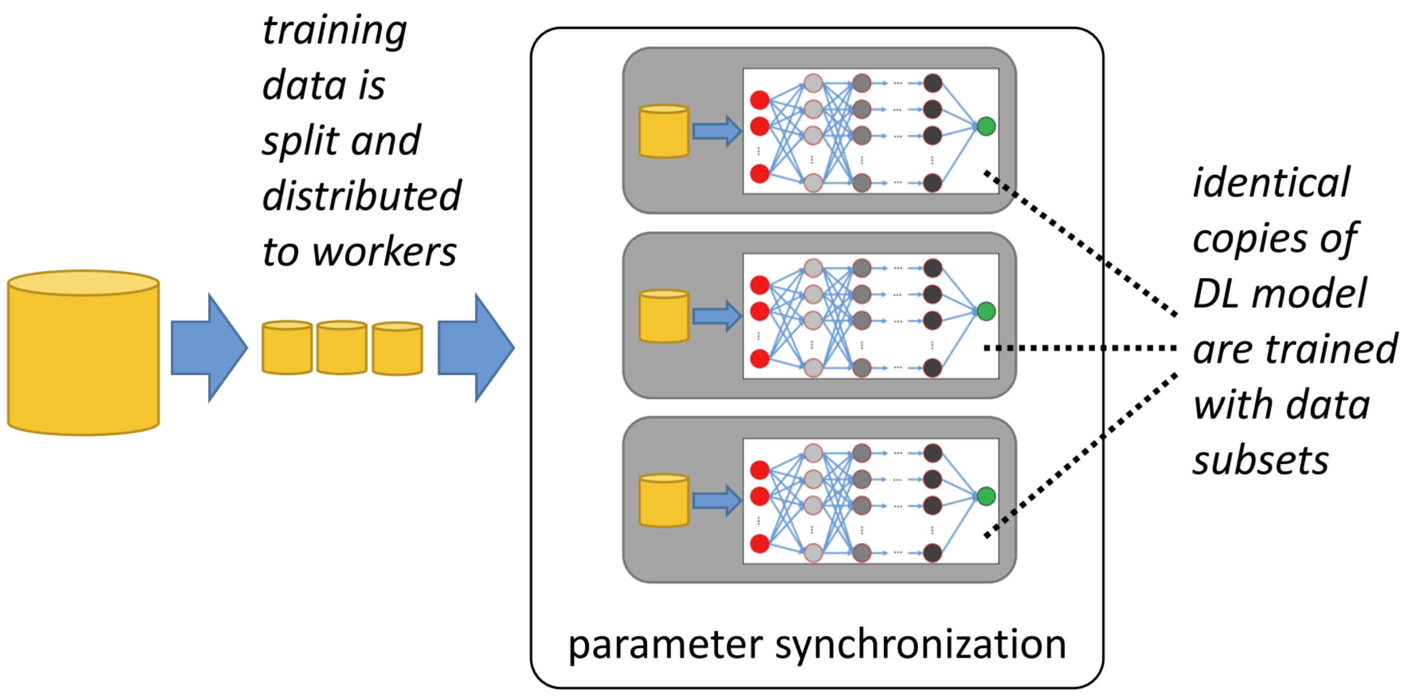
model $\boldsymbol{x}$ model $\boldsymbol{x}$



Distributed ML: Parallelization

- Parallelization methods in DL: data, model and *pipeline*. Hybrid as well.

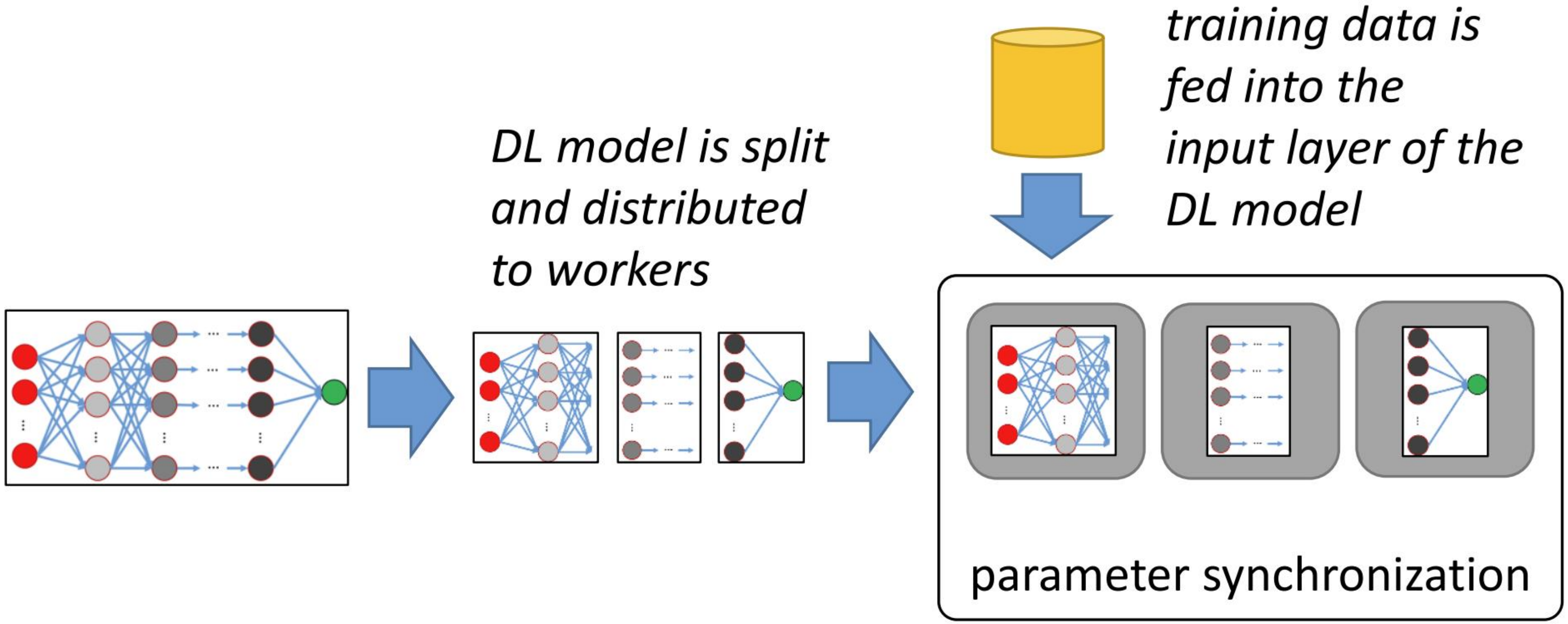
- In data parallelism, a number of machines loads an identical copy of a DL model.
- Training data split into non-overlapping chunks and fed into model replicas of workers for training.
- Each worker performs training on its training data, which leads to updates of model parameters.
- Hence, model parameters between workers need to be synchronized.



Mayer, Ruben, and Hans-Arno Jacobsen. "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools." *ACM Computing Surveys (CSUR)* 53.1 (2020): 1-37.

Distributed ML: Parallelization

- Model parallelism: DL model is split, and each worker loads a different part of DL model for training.
- Worker(s) hold input layer of DL model are fed with training data.
- In forward pass, they compute their output signal which is propagated to workers that hold next layer of DL model.
- In backpropagation pass, gradients are computed starting at workers that hold output layer of DL model, propagating to workers that hold input layers of DL model.



Mayer, Ruben, and Hans-Arno Jacobsen. "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools." *ACM Computing Surveys (CSUR)* 53.1 (2020): 1-37.

TensorFlow

TensorFlow and Others

	TensorFlow	PyTorch	Keras
API Level	Both (High and Low)	Low	High
Architecture	Not easy to use	Complex, less readable	Simple, concise, readable
Datasets	Large datasets, high-performance	Large datasets, high-performance	Smaller datasets
Debugging	Difficult to conduct debugging	Good debugging capabilities	Simple network, so debugging is not often needed
Pretrained models?	Yes	Yes	Yes
Popularity	Second most popular of the three	Third most popular of the three	Most popular of the three
Speed	Fast, high-performance	Fast, high-performance	Slow, low performance
Written In	C++, CUDA, Python	Lua	Python

https://www.datacamp.com/tutorial/pytorch-vs-tensorflow-vs-keras

What is TensorFlow?

- TensorFlow (tf) offers tools, libraries and an open-source platform for ML.
  - Keras python based deep learning framework (high-level API of tf).
- Perform numerical computation using data flow graphs.

- Developed by Google Brain Team to conduct ML research
- TensorFlow provides Python and C++ APIs.

- To install:
  - $ pip install tensorflow
  - $ pip install tensorflow-cpu (CPU-only package)
- Sample Code:

```
import tensorflow as tf
tf.add(1, 2).numpy()
```

What is TensorFlow?

- Its core is very similar to NumPy, but with GPU support.
- Key idea: express a numeric computation as a graph.
    - Graph nodes are operations with any number of inputs and outputs.
    - Graph edges are tensors which flow between nodes.
    - Computation graphs can be exported to a portable format, train here- run there.

- Core features: tf.keras, data loading and preprocessing (tf.data, tf.io), image processing (tf.image), signal processing (tf.signal)

What is TensorFlow?

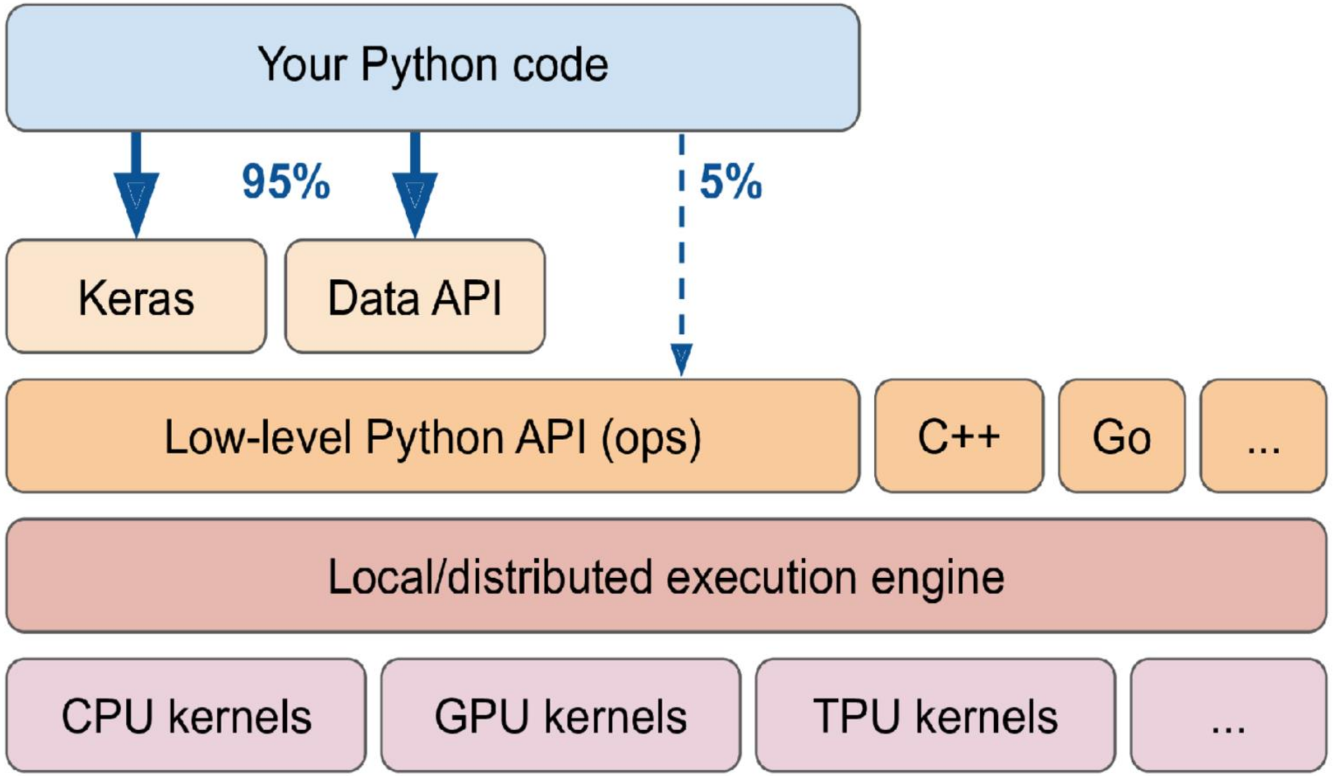
- TensorFlow's API revolves around tensors which flow from operation to operation hence name TensorFlow.

- Tensor is like a NumPy ndarray
  - Can create a tensor from a NumPy array and vice versa.
    - Can apply TensorFlow operations to NumPy arrays and NumPy operations to tensors.
  - Ideally a multi-dimensional array but can hold a scalar.
  - Helps during custom cost functions, custom metrics, custom layers.

TensorFlow’s Architecture

- TensorFlow’s execution engine take care of running operations efficiently, even across multiple devices and machines.

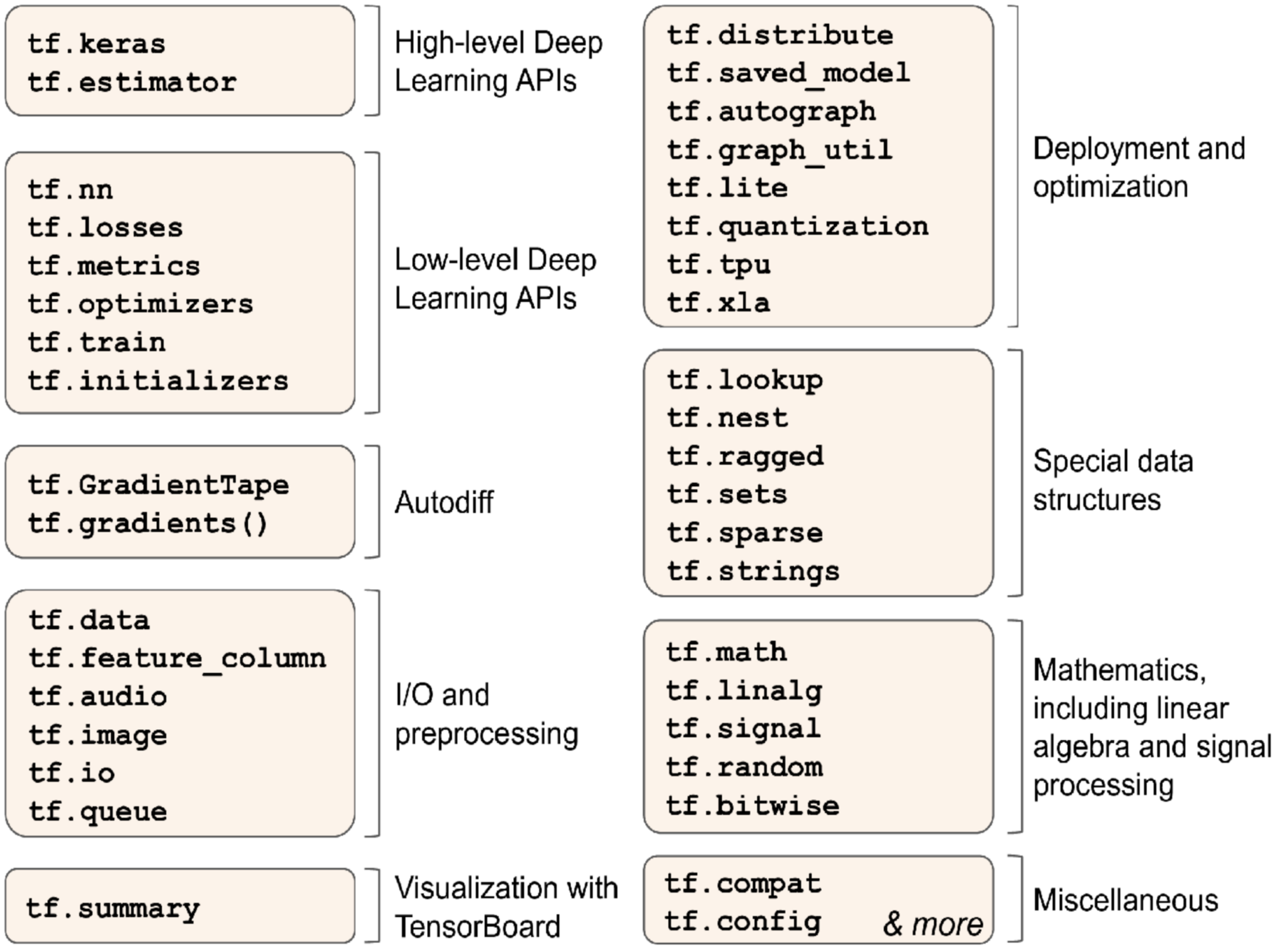
- JavaScript implementation called TensorFlow.js run your models directly in your browser.



- Execution Steps:
- Built a graph using variables and placeholders.
- Deploy the graph for execution.
- Train model by defining loss function and gradients computations.

TensorFlow's Python API

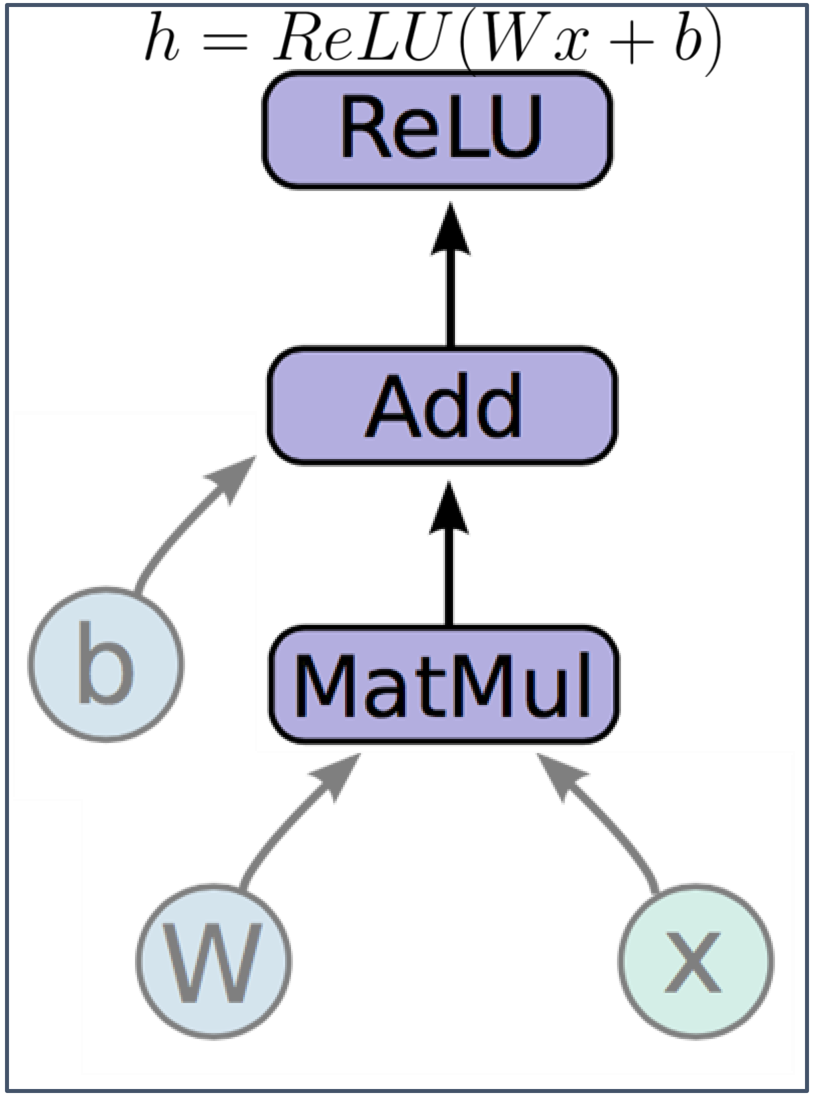
- At lowest level, each TensorFlow operation is implemented using C++ code.
- Many operations have multiple implementations called kernels.
- Each dedicated to a specific device (CPUs/GPUs/TPUs).
- GPUs can speed up computations by splitting them into many smaller chunks and running them in parallel.
- TPUs are custom ASIC chips built specifically for DL operations.



How it works?

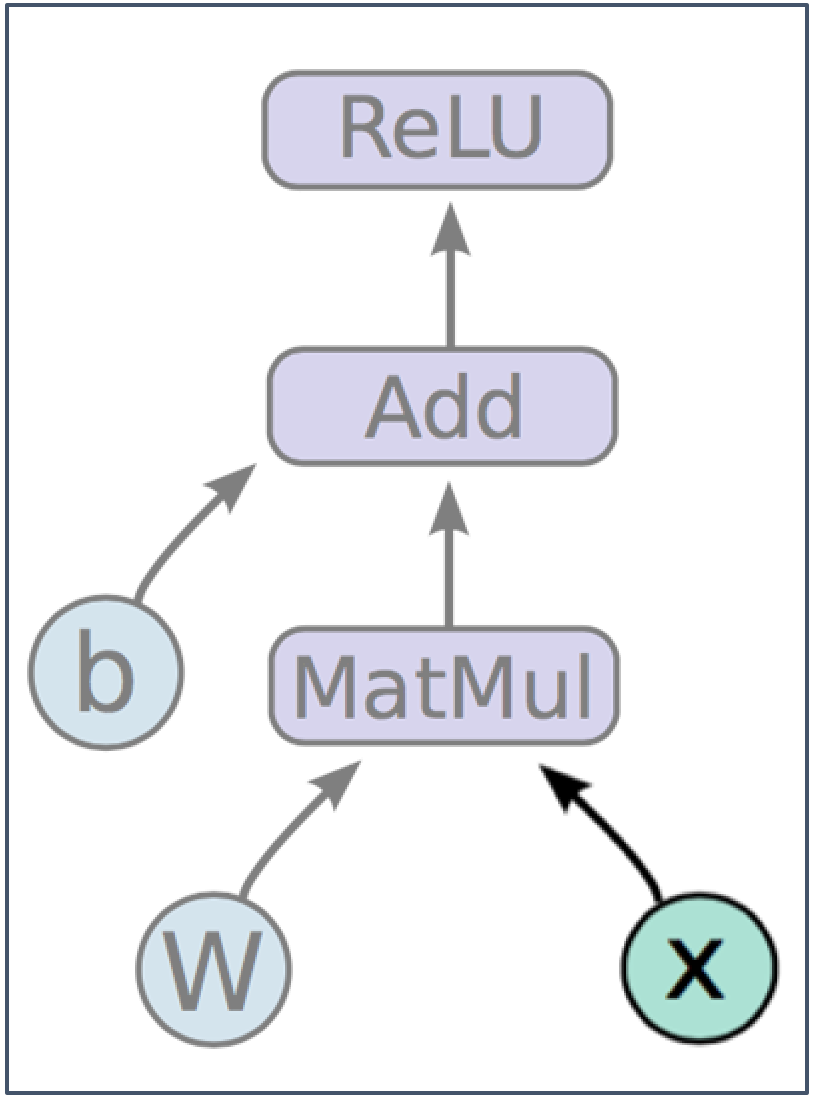
- MatMul: Multiply two matrices
- Add: Add element wise
- ReLU: Activate with elementwise rectified linear function

$$ReLu(x) = \begin{cases} 0, & x <= 0 \\ x, & x > 0 \end{cases}$$



- Variables are stateful nodes which output their current value.
- State is retained across multiple executions of a graph (mostly parameters).

How it works?

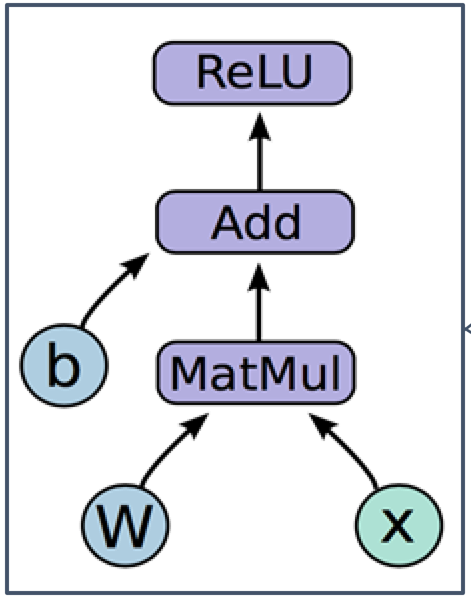


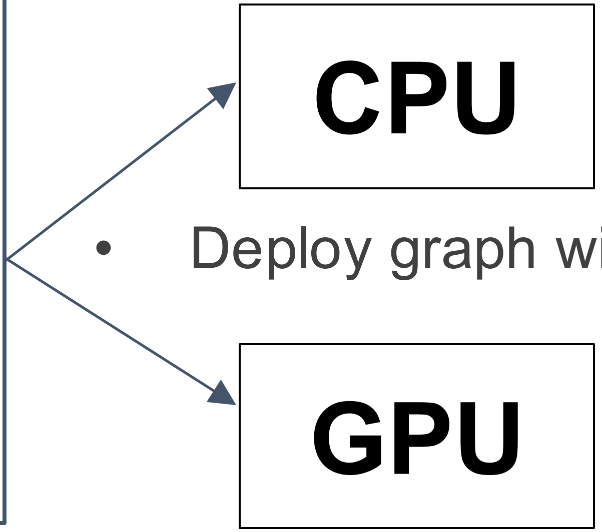
$$h = ReLU(Wx + b)$$

- Placeholders are nodes whose value is fed in at execution time (inputs, labels, …).
- Code for:

$$h = ReLU(Wx + b)$$

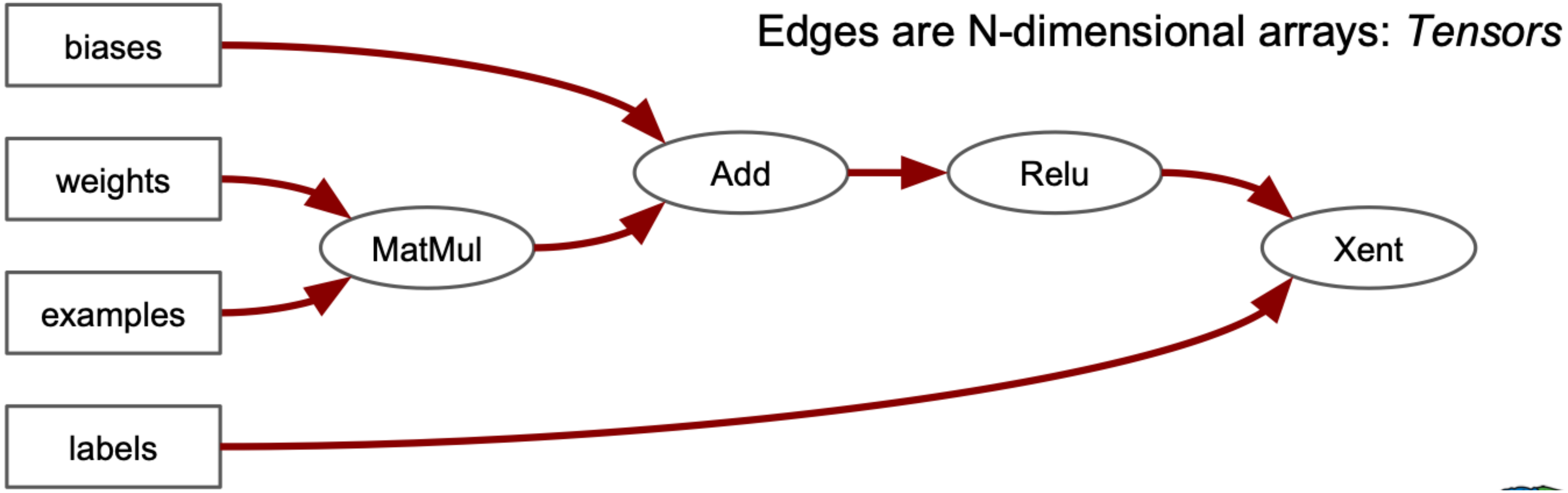
```
import tensorflow as tf
b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))
x = tf.placeholder(tf.float32, (1, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)
```



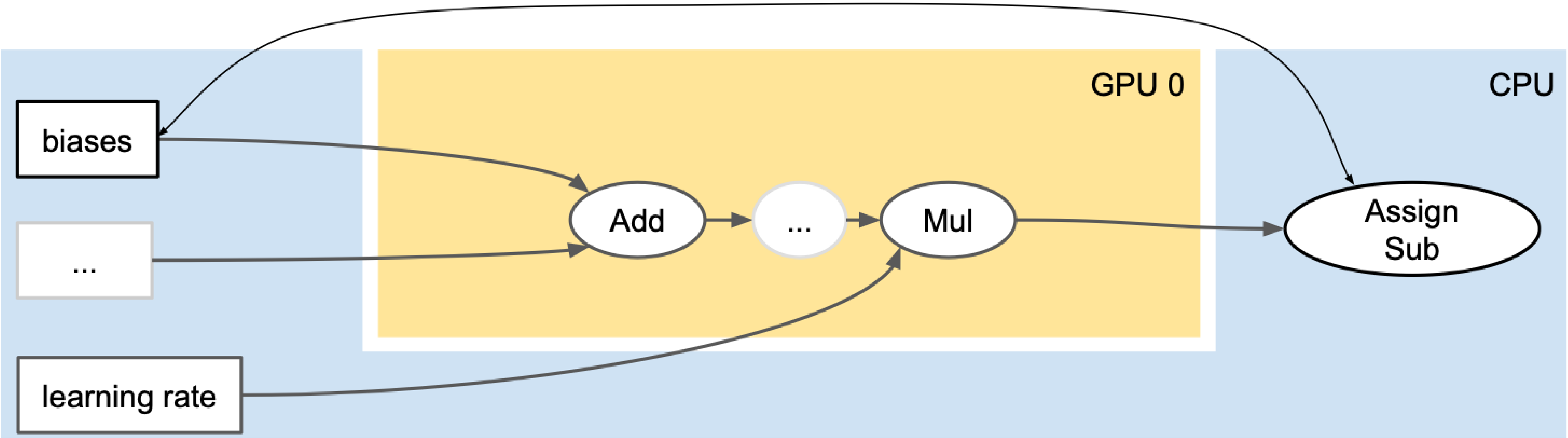


- Deploy graph with a session

TensorFlow’s Computation: Dataflow Graph

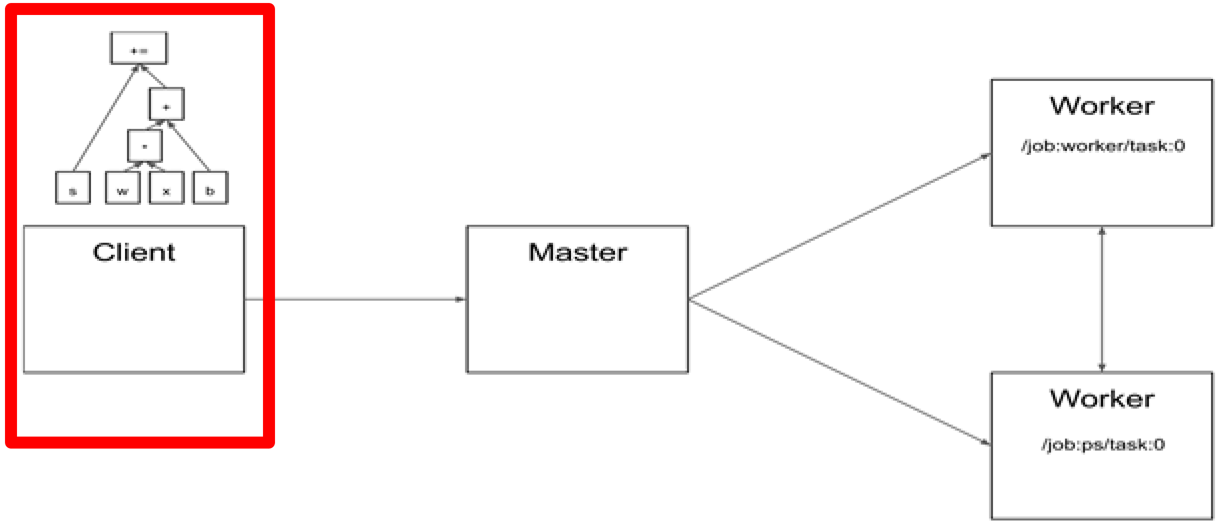


TensorFlow's Computation: Dataflow Graph Distribution

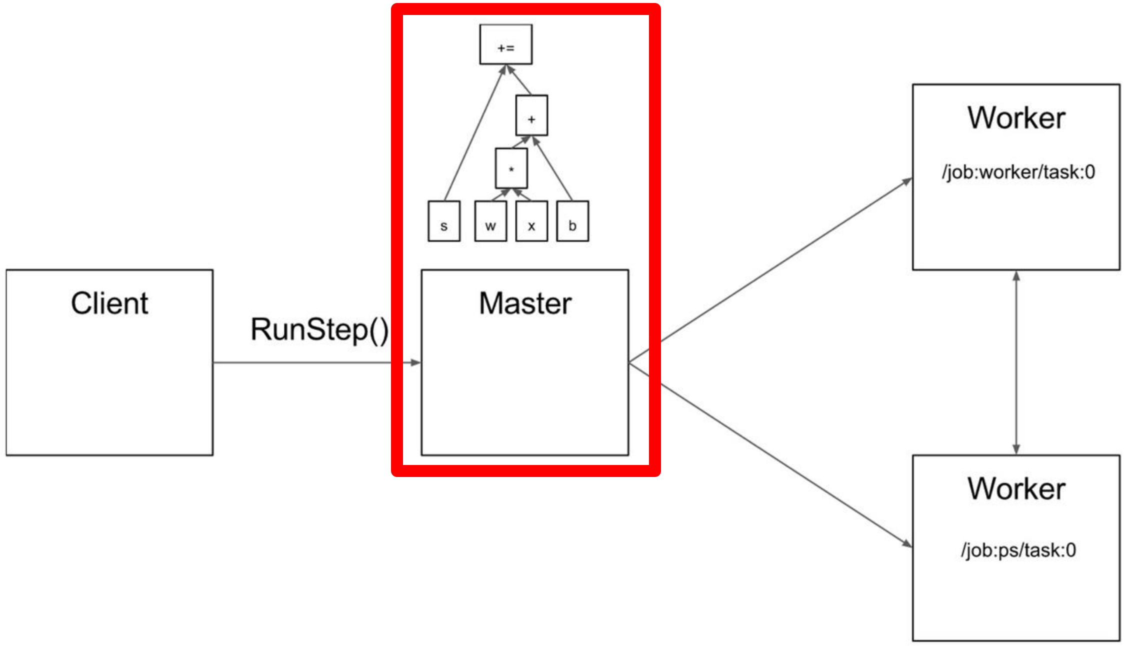


CPUS & GPUS

TensorFlow’s Execution Framework



- Client



- Master

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

References

- Building Machine Learning and Deep Learning Models on Google Cloud Platform
  By E. Bisong.
- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron.
- Deep Learning By Ian Goodfellow, Yoshua Bengio and Aaron Courville, The MIT Press.
- TensorFlow and Clipper (Lecture 24, cs262a) By Ali Ghodsi and Ion Stoica, UC Berkeley, 2018.
- Large-Scale Deep Learning With TensorFlow by Jeff Dean & Google Brain team.
- https://projects.apache.org/projects.html