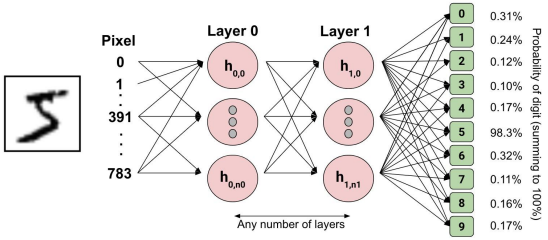
TensorFlow

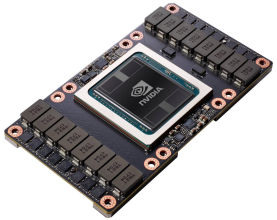
Getting Started with TensorFlow on GPUs

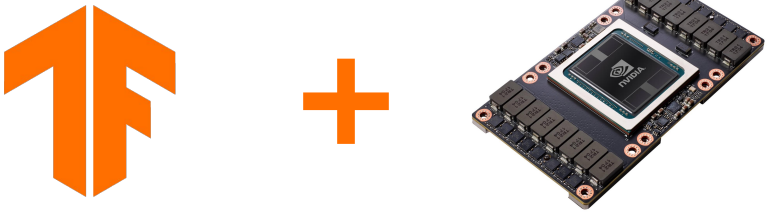
Magnus Hyttsten

@MagnusHyttsten


Pixel
0
1
391
783
Layer 0
Layer 1
$h_{0,0}$
$h_{0,n0}$
$h_{1,0}$
$h_{1,n1}$
Any number of layers
0 0.31%
1 0.24%
2 0.12%
3 0.10%
4 0.17%
5 98.3%
6 0.32%
7 0.11%
8 0.16%
9 0.17%
Probability of digit (summing to 100%)

Agenda





An Awkward Social Experiment

(that I'm afraid you will be part of...)

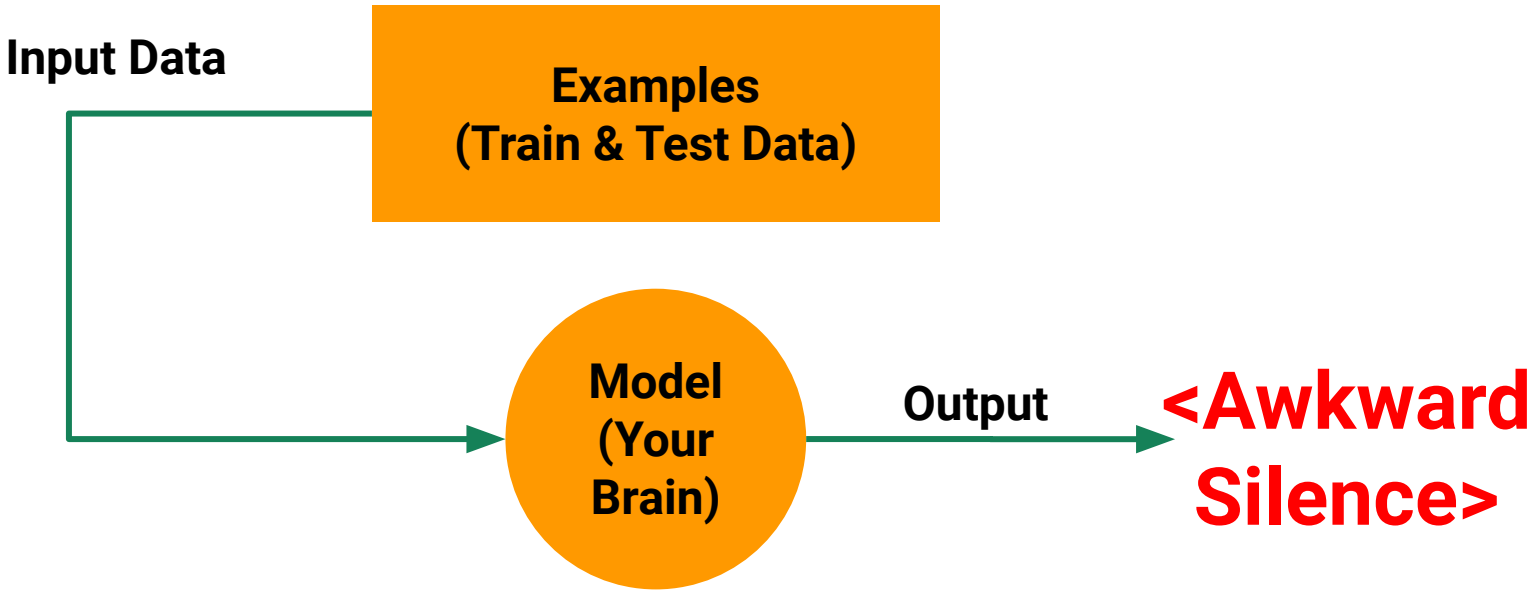
GPU TECHNOLOGY CONFERENCE

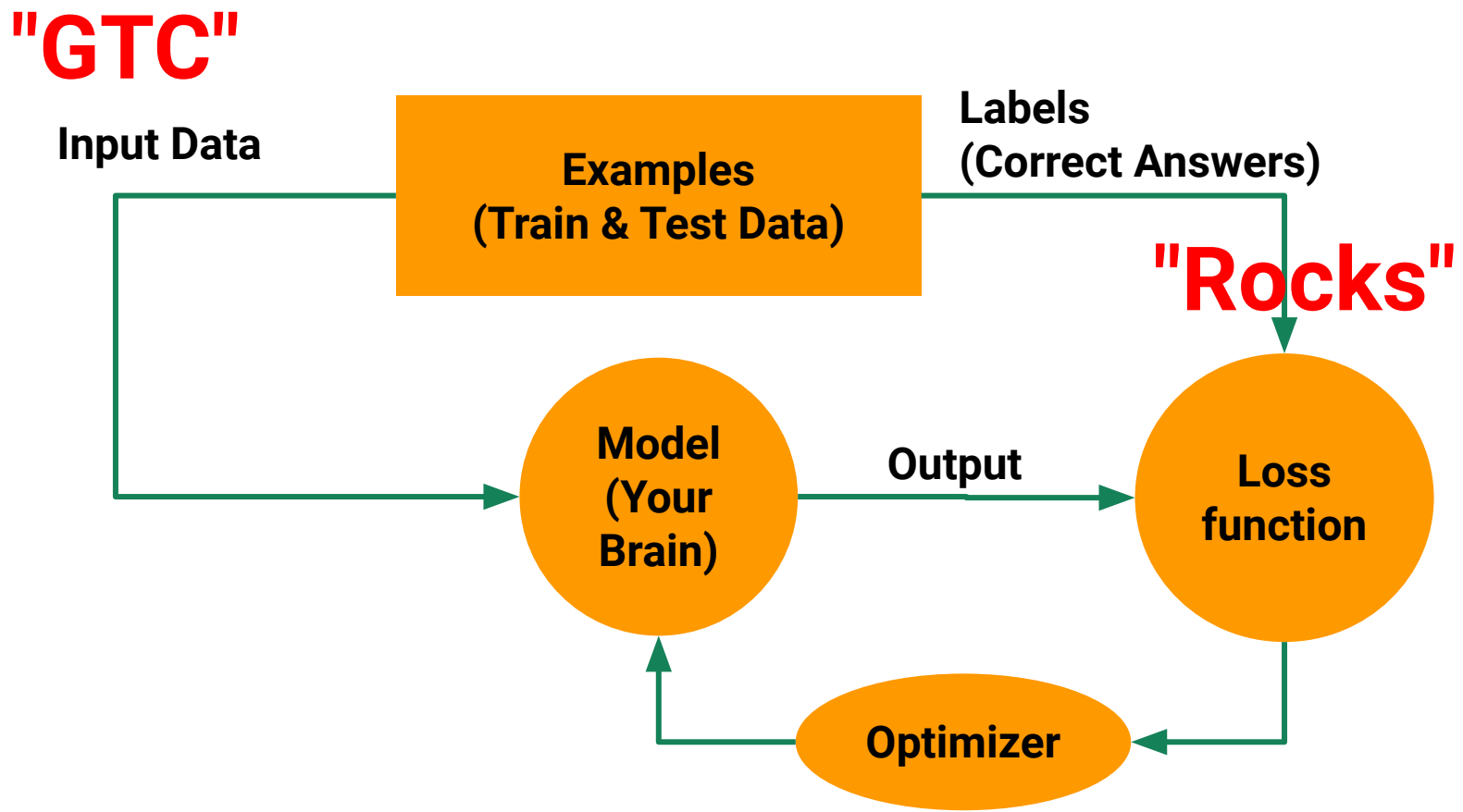
SILICON VALLEY

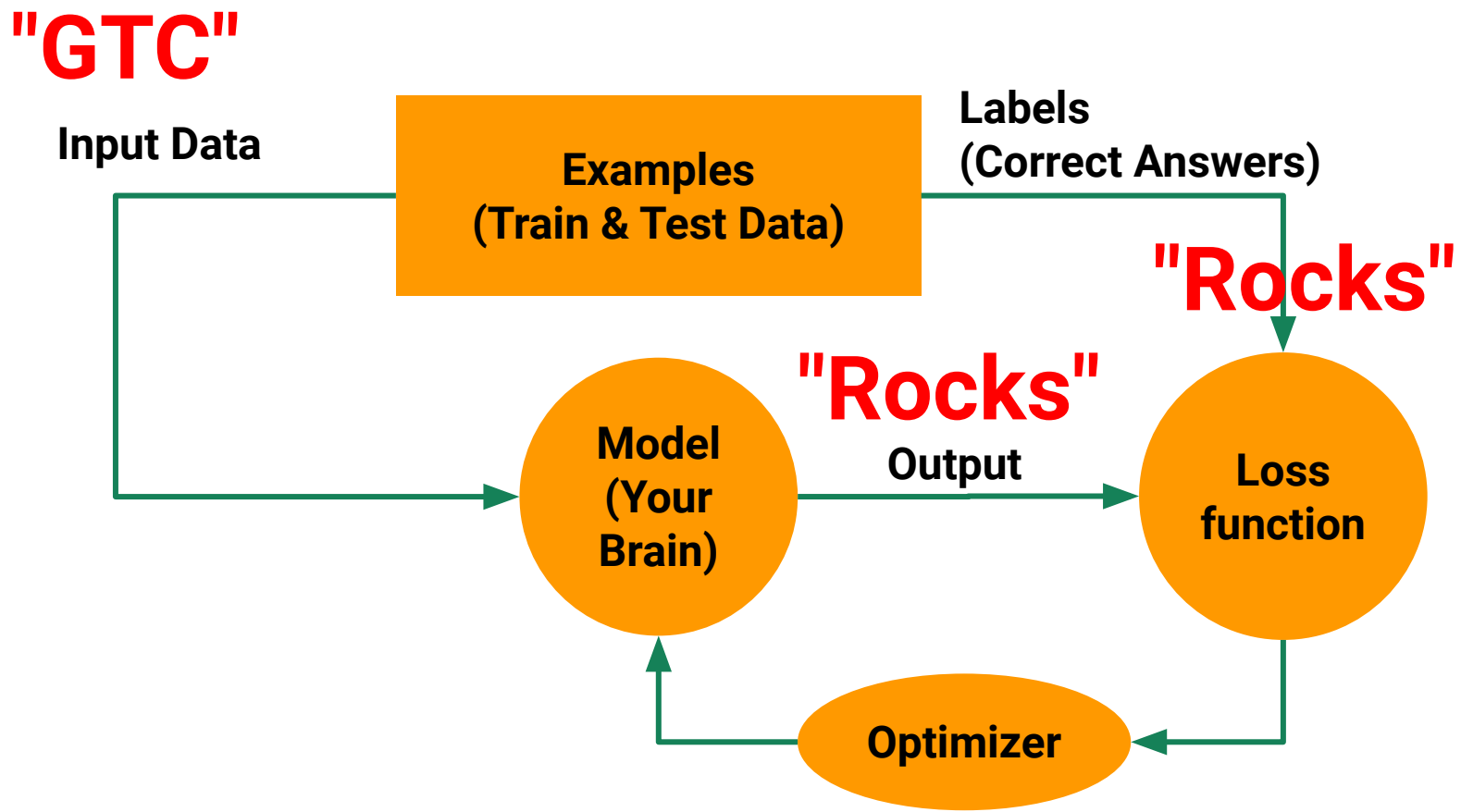
GPU TECHNOLOGY CONFERENCE

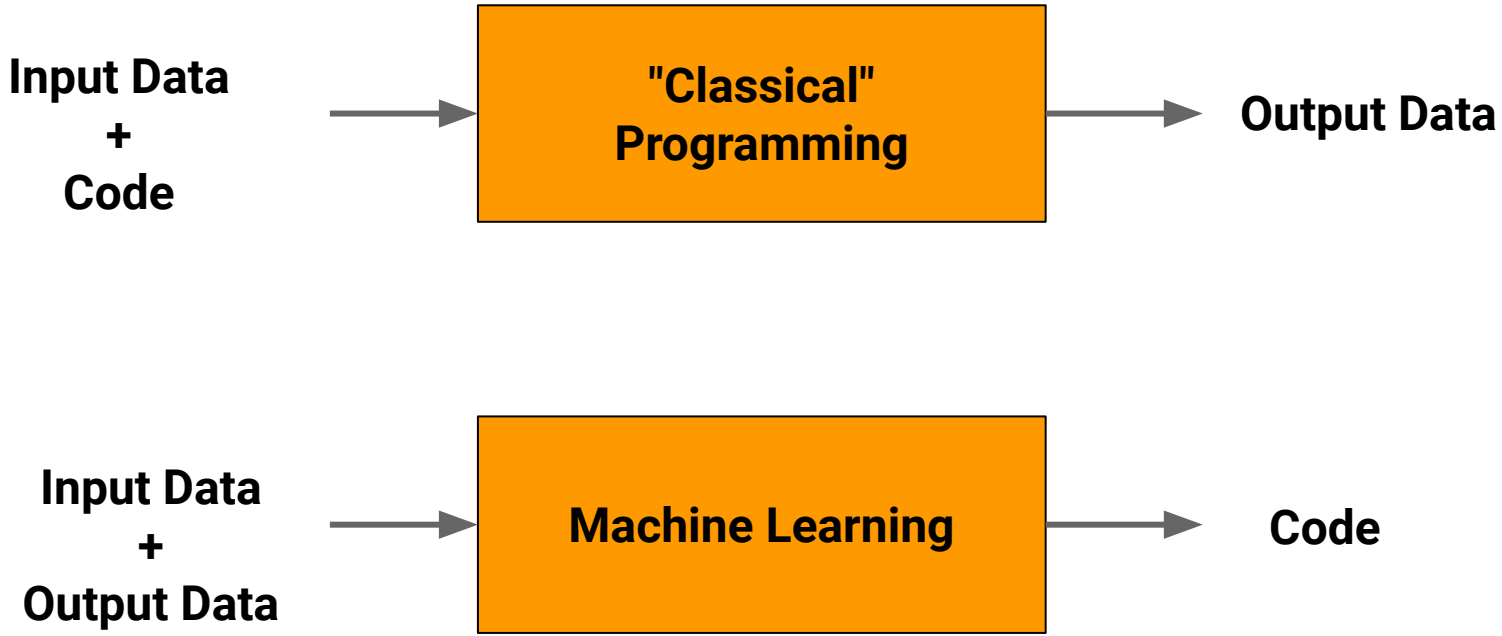
ROCKS!

"GTC"


Input Data
Examples
(Train & Test Data)
Model
(Your
Brain)
Output
<Awkward
Silence>


"GTC"
Input Data
Examples
(Train & Test Data)
Labels
(Correct Answers)
"Rocks"
Model
(Your
Brain)
Output
Loss
function
Optimizer


"GTC"
Input Data
Examples
(Train & Test Data)
Labels
(Correct Answers)
"Rocks"
"Rocks"
Output
Model
(Your
Brain)
Loss
function
Optimizer


Input Data
+
Code
"Classical"
Programming
Output Data
Input Data
+
Output Data
Machine Learning
Code

IS THIS A

CAT or DOG?

CAT DOG

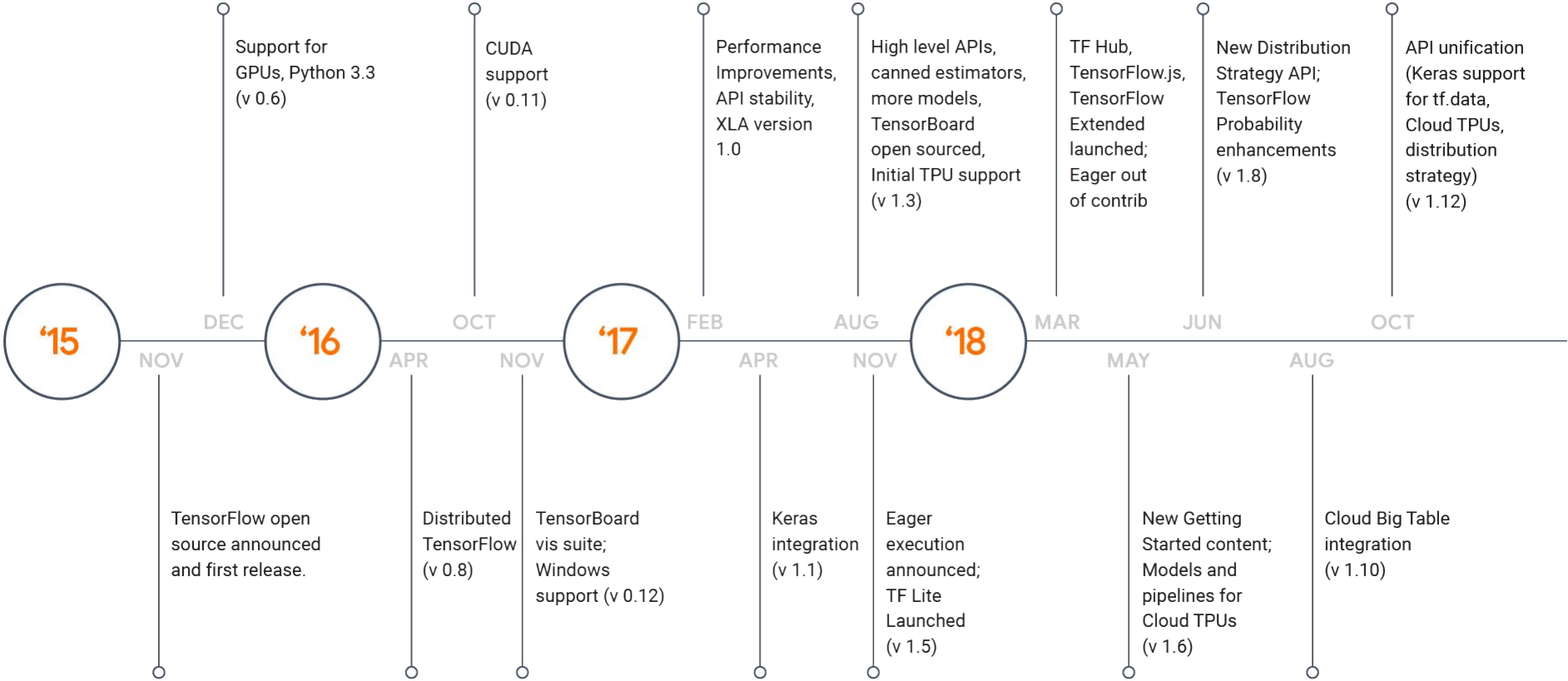
OUTPUT LAYER

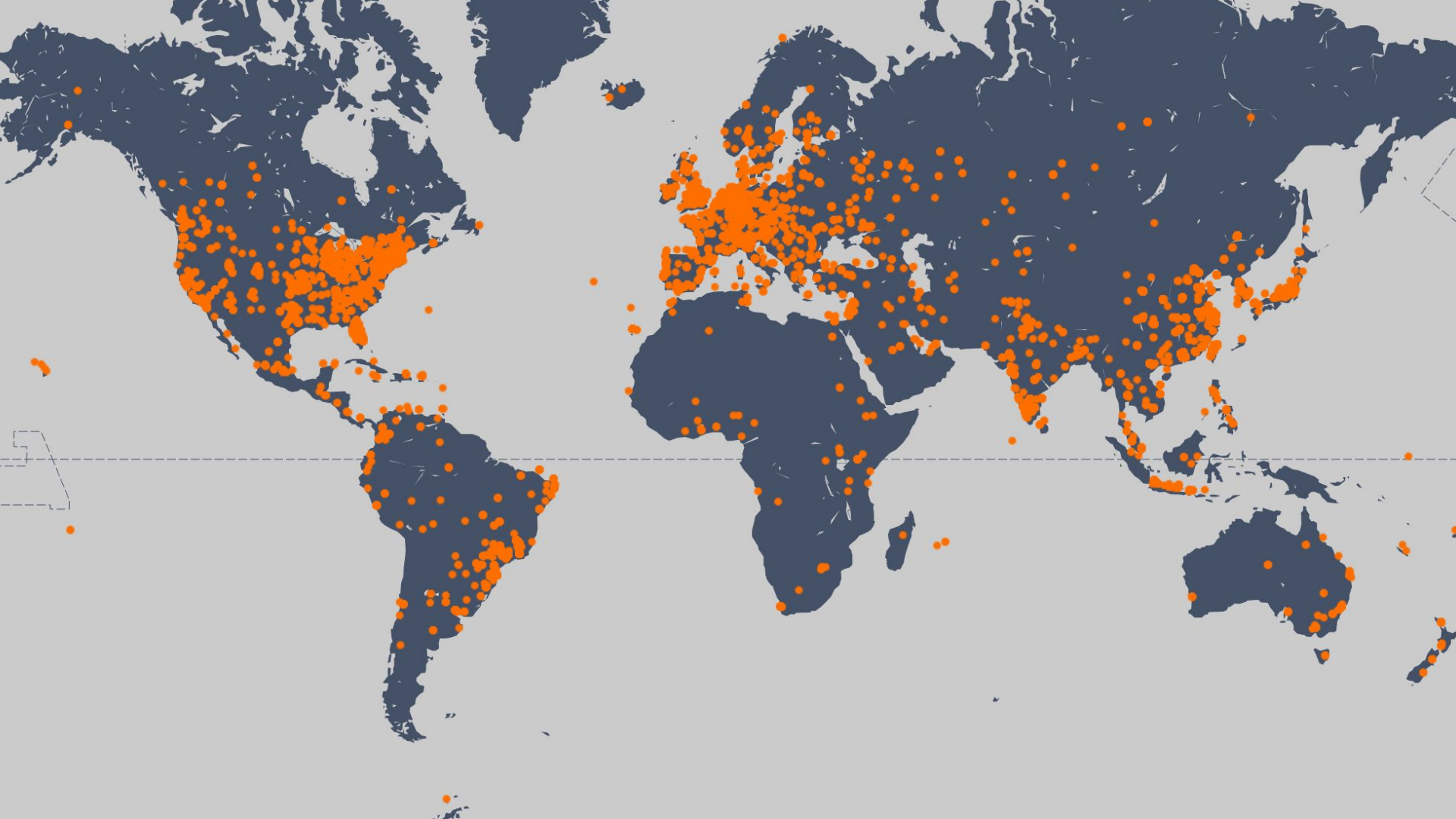
ACTIVATED NEURONS

INPUT LAYER

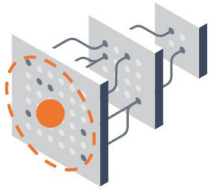
DEEP *neural* NETWORK


7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 1 2 3 4 5 6


'15
NOV
TensorFlow open source announced and first release.
DEC
Support for GPUs, Python 3.3 (v 0.6)
'16
APR
Distributed TensorFlow (v 0.8)
OCT
CUDA support (v 0.11)
NOV
TensorBoard vis suite; Windows support (v 0.12)
'17
FEB
Performance Improvements, API stability, XLA version 1.0
APR
Keras integration (v 1.1)
AUG
High level APIs, canned estimators, more models, TensorBoard open sourced, Initial TPU support (v 1.3)
NOV
Eager execution announced; TF Lite Launched (v 1.5)
'18
MAR
TF Hub, TensorFlow.js, TensorFlow Extended launched; Eager out of contrib
MAY
New Getting Started content; Models and pipelines for Cloud TPUs (v 1.6)
JUN
New Distribution Strategy API; TensorFlow Probability enhancements (v 1.8)
AUG
Cloud Big Table integration (v 1.10)
OCT
API unification (Keras support for tf.data, Cloud TPUs, distribution strategy) (v 1.12)



TensorFlow 2.0 Alpha is out



Easy

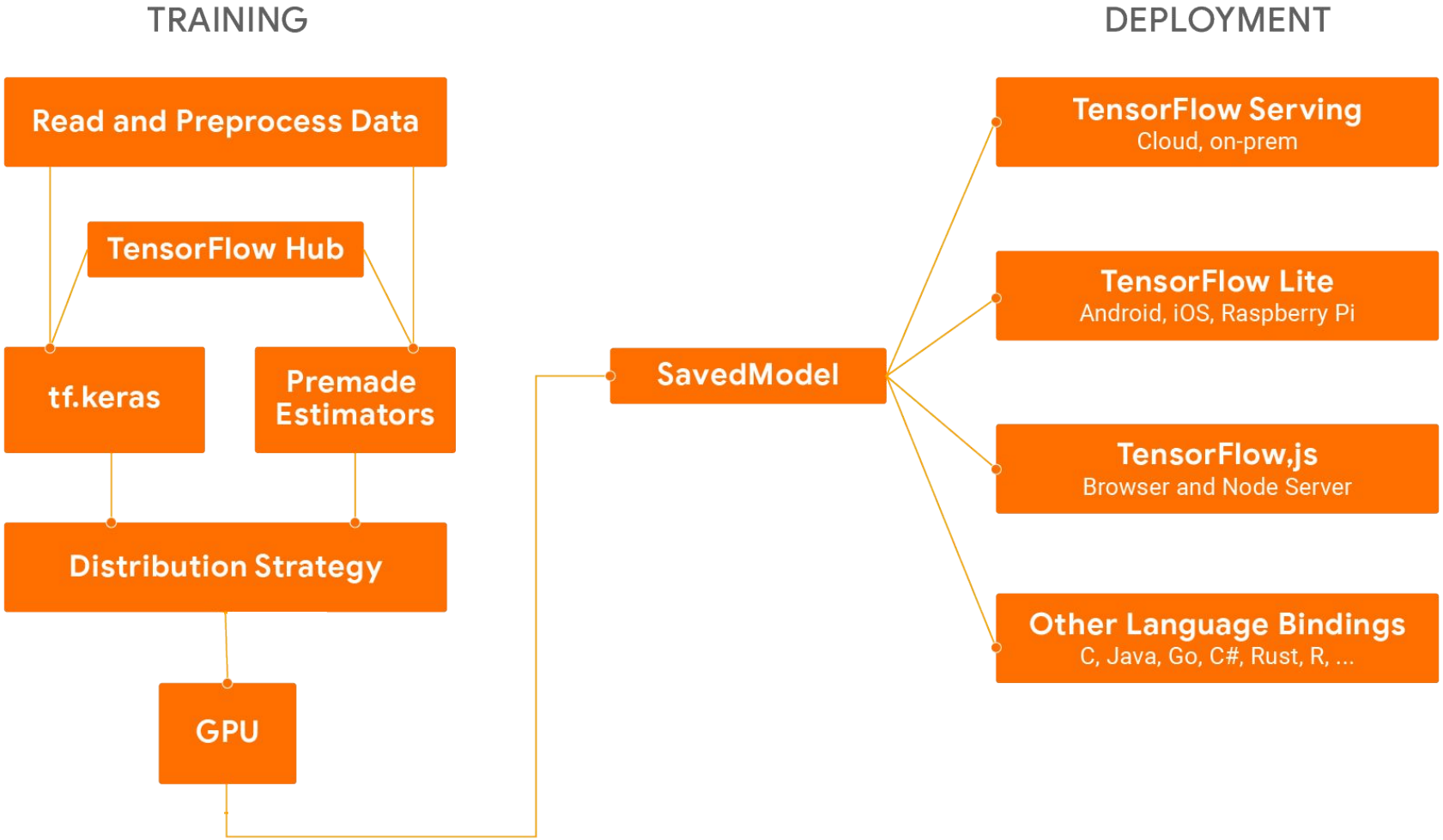
Simplified APIs.
Focused on Keras and
eager execution

Powerful

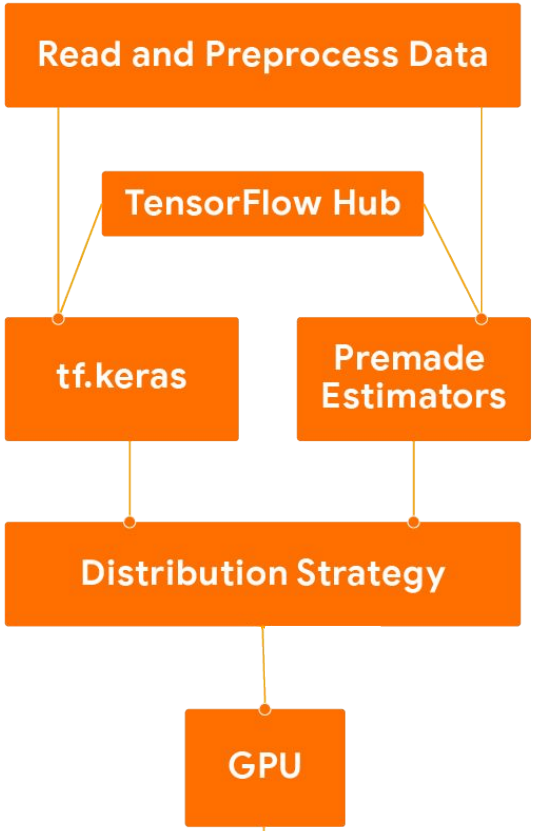
Flexibility and performance.
Power to do cutting edge research
and scale to > 1 exaflops

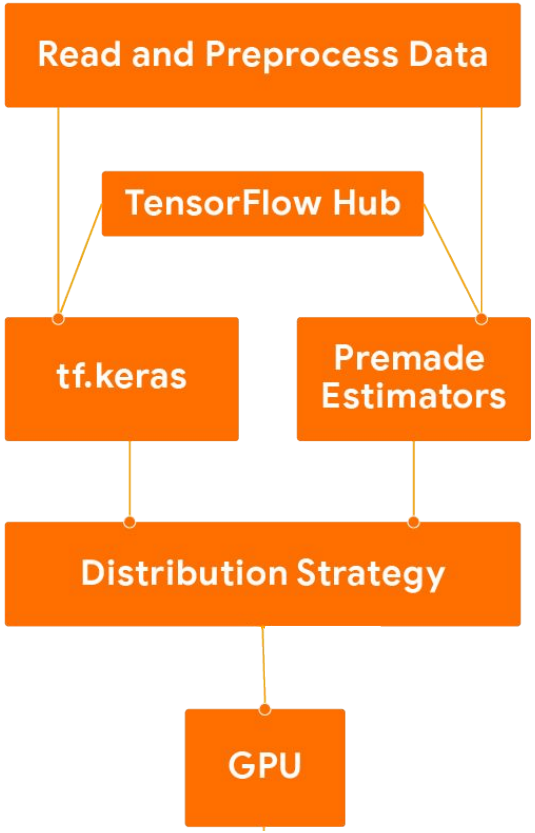
Scalable

Tested at Google-scale.
Deploy everywhere


TRAINING
DEPLOYMENT
Read and Preprocess Data
TensorFlow Hub
tf.keras
Premade Estimators
Distribution Strategy
GPU
SavedModel
TensorFlow Serving
Cloud, on-prem
TensorFlow Lite
Android, iOS, Raspberry Pi
TensorFlow.js
Browser and Node Server
Other Language Bindings
C, Java, Go, C#, Rust, R, ...

TRAINING


Read and Preprocess Data
TensorFlow Hub
tf.keras
Premade Estimators
Distribution Strategy
GPU



tf.data (Dataset)
tf.feature_column

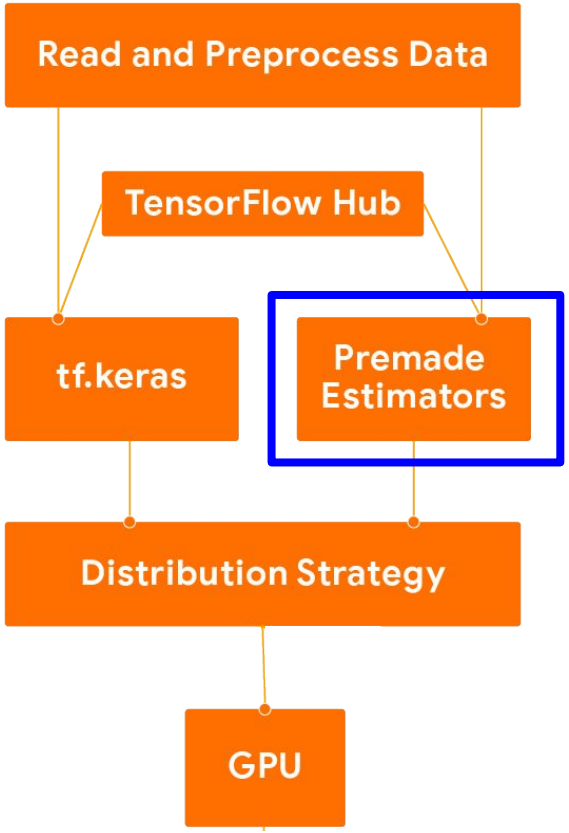
(Transfer Learning)

High-level APIs

Perform Distributed Training (talk @1pm)

E.g. V100

TRAINING



tf.data (Dataset)
tf.feature_column

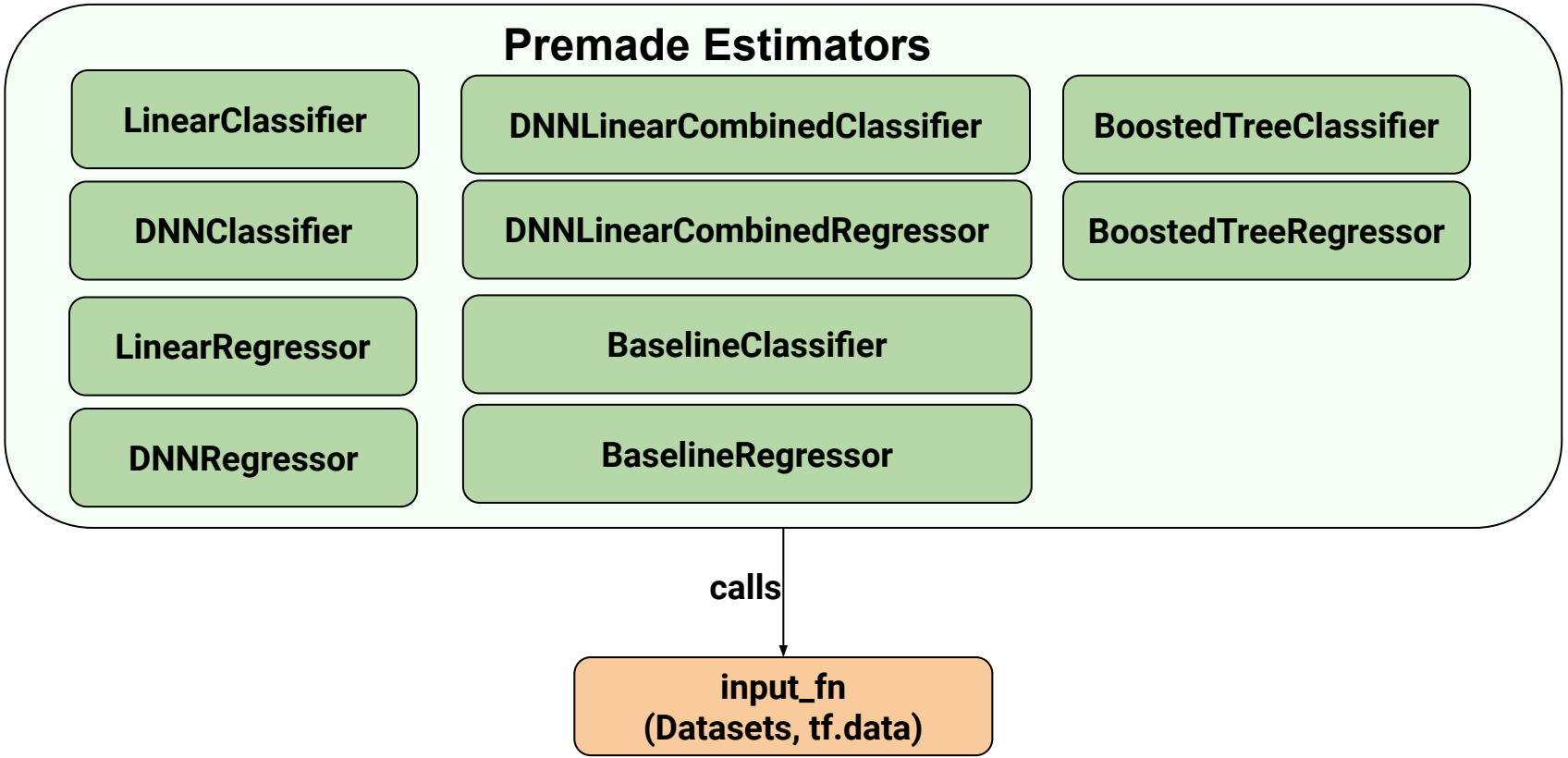
(Transfer Learning)

High-level APIs

Perform Distributed Training (talk @1pm)

E.g. V100

Built to Distribute and Scale



Premade Estimators

Premade Estimators

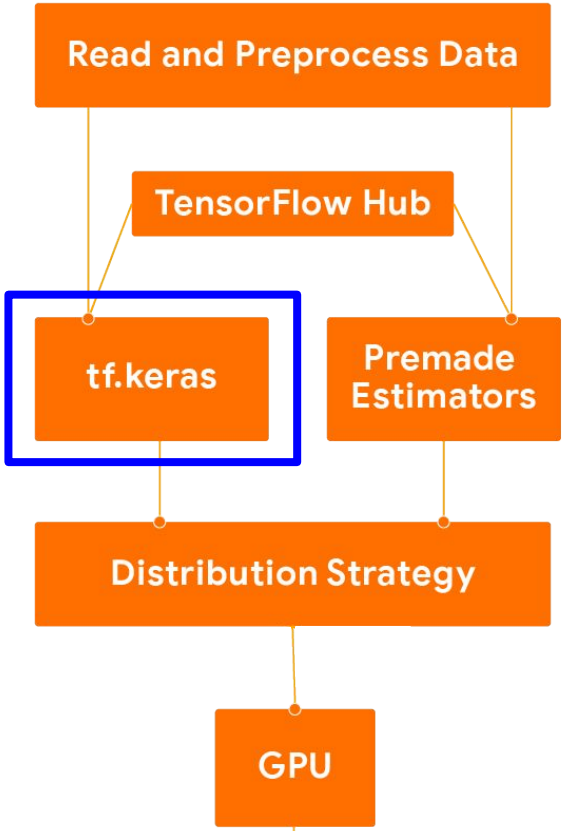
Datasets

```
estimator =  LinearRegressor(...)
             LinearClassifier(...)
             DNNRegressor(...)
             DNNClassifier(...)
             DNNLinearCombinedRegressor(...)
             DNNLinearCombinedClassifier(...)
             BaselineRegressor(...)
             BaselineClassifier(...)
             BoostedTreeRegressor(...)
             BoostedTreeClassifier(...)

# Train locally
estimator.train   (input_fn=...  ...
estimator.evaluate(input_fn=...  ...)
estimator.predict (input_fn=...  ...)
```

Premade Estimator - Wide & Deep

```
wide_columns = [
    tf.feature_column.bucketized_column(
        'age',=[18, 27, 40, 65])]
deep_columns = [
    tf.feature_column.numeric_column('visits'),
    tf.feature_column.numeric_column('clicks')]

tf.estimator.DNNLinearCombinedClassifier(
        linear_feature_columns=wide_columns,
        dnn_feature_columns=deep_columns,
        dnn_hidden_units=[100, 75, 50, 25])
```

TRAINING



tf.data (Dataset)
tf.feature_column

(Transfer Learning)

Perform Distributed Training

E.g. V100

Custom Models

tf.keras

tf.keras.layers

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit     (dataset, epochs=5)
model.evaluate(dataset)
model.predict (dataset)
```

Datasets

TensorFlow Datasets

```
import tensorflow_datasets as tfds
train_ds = tfds.load("imdb_reviews",
                     split="train",
                     as_supervised=True)
```

- 30+ available
- Add your own

- audio
  - "nsynth"
- image
  - "celeb_a"
  - "cifar10"
  - "coco2014"
  - "diabetic_retinopathy_detection"
  - "imagenet2012"
  - "mnist"
  - "open_images_v4"
- structured
  - "titanic"
- text
  - "imdb_reviews"
  - "lm1b"
  - "squad"
- translate
  - "wmt_translate_ende"
  - "wmt_translate_enfr"
- video
  - "bair_robot_pushing_small"
  - "moving_mnist"
  - "starcraft_video"

TensorFlow Summary

- Datasets (tf.data) for the input pipeline
  a. TensorFlow Datasets is great
  b. tf.feature_columns are cool too

- Premade Estimators

- Keras Models (tf.keras)

The V-100

And why is it so good @ Machine Learning???

Disclaimer

- High-Level - We look at only parts of the power of GPUs
- Simple Overview - More optimal designs exist
- Reduced Scope - Only considering fully-connected layers, etc

Strengths of V100

- Built for Massively Parallel Computations
- Specific hardware / software to manage Deep Learning Workloads (Tensor Cores, mixed-precision execution, etc)

Strengths of V100

- Built for Massively Parallel Computations
- Specific hardware / software to manage Deep Learning Workloads (Tensor Cores, mixed-precision execution, etc)

Tesla SXM V100

- 5376 cores (FP32)

My Questions Around the GPU

What are we going to do with 5376 FP32 cores?

The Unsatisfactory Answer

What are we going to do with 5376 FP32 cores?

"Execute things in parallel"!

What are we going to do with 5376 FP32 cores?

"Execute things in parallel"!

Yes, but how can we exactly do that for ML Workloads?

What are we going to do with 5376 FP32 cores?

"Execute things in parallel"!

Yes, but how can we exactly do that for ML Workloads?

"Hey, that's your job - That's why we're here listening"!

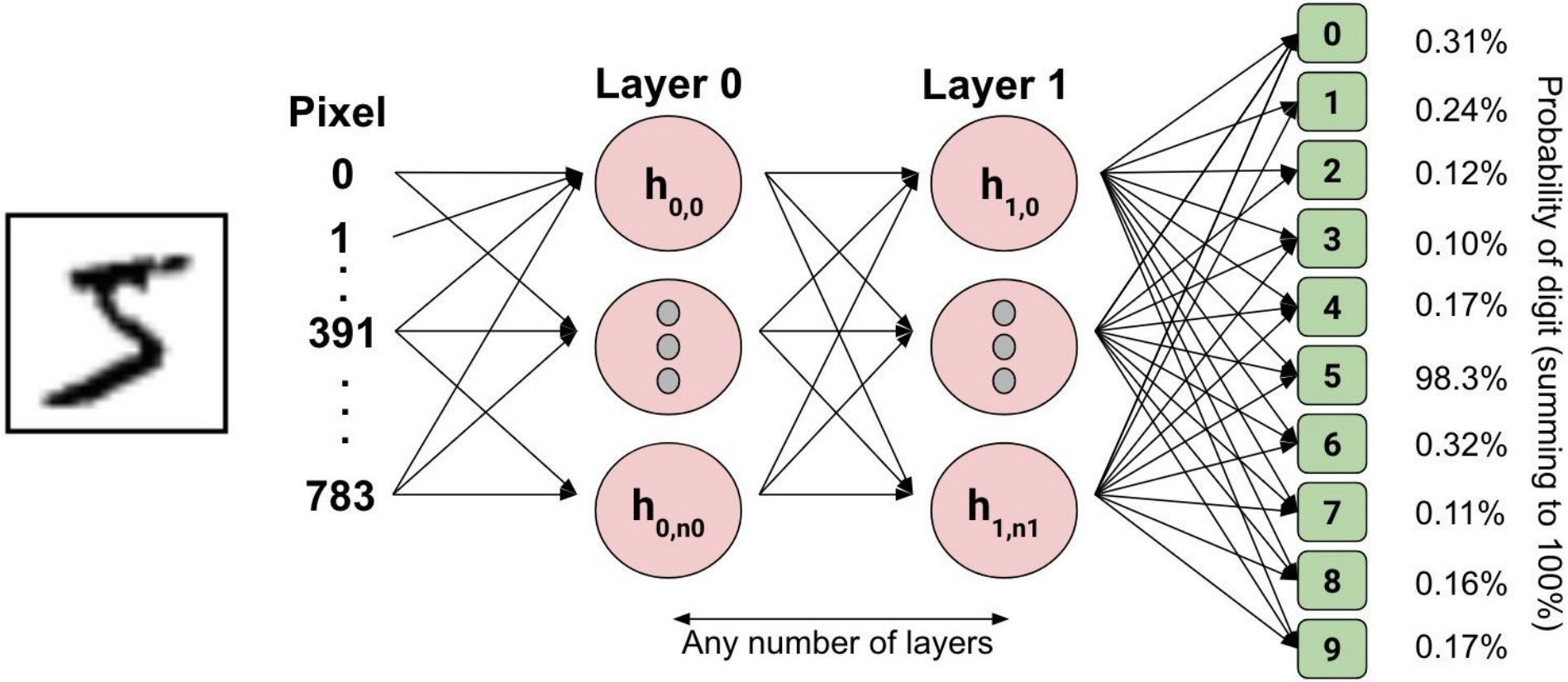
What are we going to do with 5376 FP32 cores?

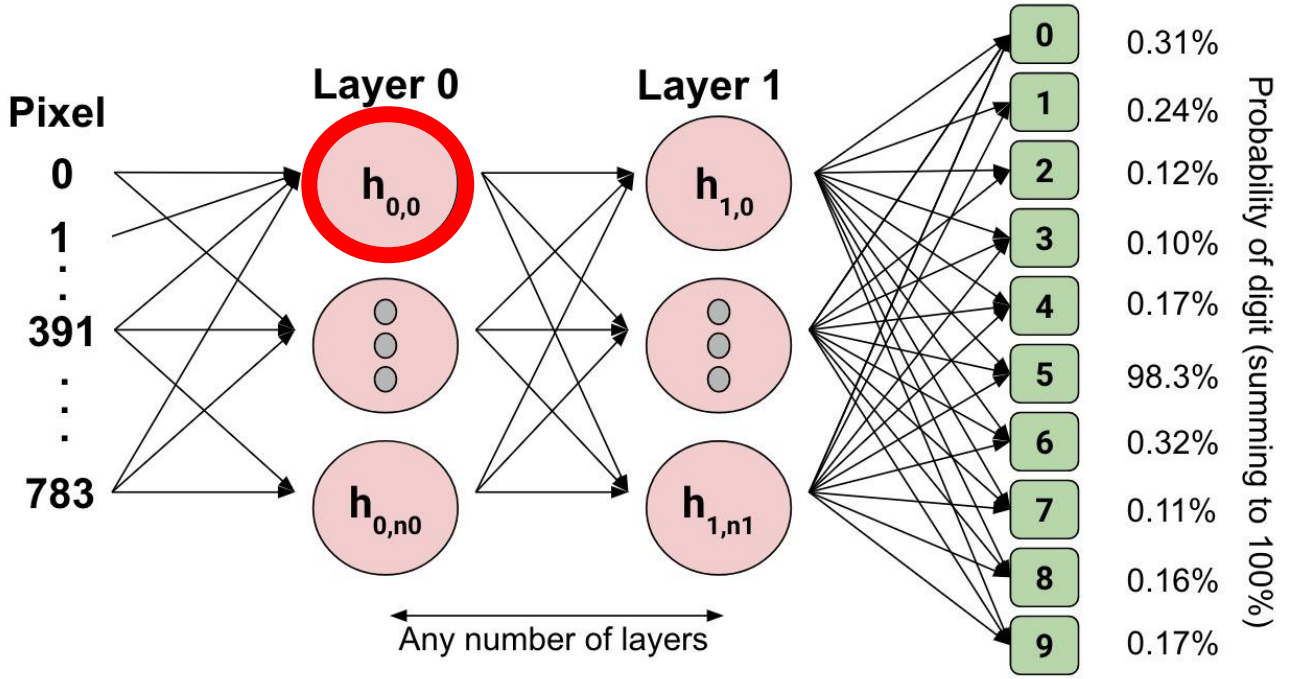
"Execute things in parallel"!

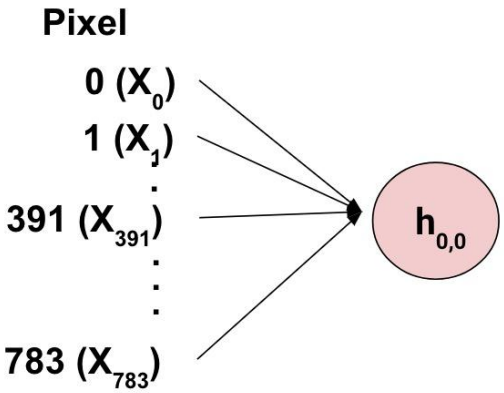
Yes, but how can we exactly do that for ML Workloads?

"Hey, that's your job - That's why we're here listening"!

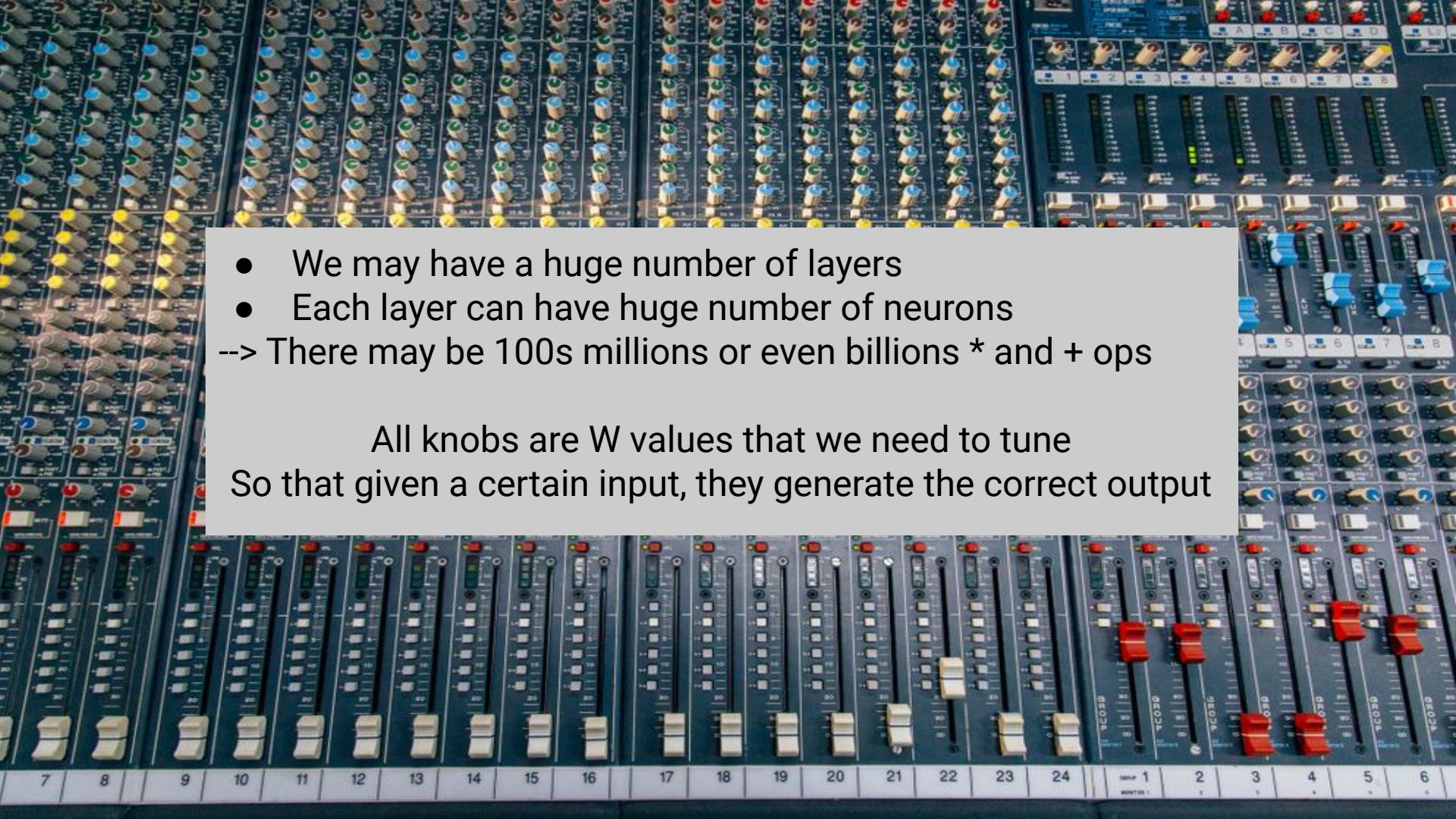
Alright, let me try to talk about that then


Pixel
0
1
391
783
Layer 0
$h_{0,0}$
$h_{0,n0}$
Layer 1
$h_{1,0}$
$h_{1,n1}$
Any number of layers
0
1
2
3
4
5
6
7
8
9
0.31%
0.24%
0.12%
0.10%
0.17%
98.3%
0.32%
0.11%
0.16%
0.17%
Probability of digit (summing to 100%)





$$h_{0,0} = X_0 * W_0 + X_1 * W_1 + \ldots + X_{391} * W_{391} + \ldots + X_{783} * W_{783}$$



- We may have a huge number of layers
- Each layer can have huge number of neurons

--> There may be 100s millions or even billions * and + ops

All knobs are W values that we need to tune
So that given a certain input, they generate the correct output

"Matrix Multiplication is

EATING (the computing resources of) THE WORLD"

$$h_{i\_j} = [X_0, X_1, X_{2, \ldots}] * [W_0, W_1, W_{2, \ldots}]$$

$$h_{i\_j} = X_0*W_0 + X_1*W_1 + X_2*W_2 + \ldots$$

Matmul

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_{0,0} = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

Single-threaded Execution

Single-threaded Execution

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

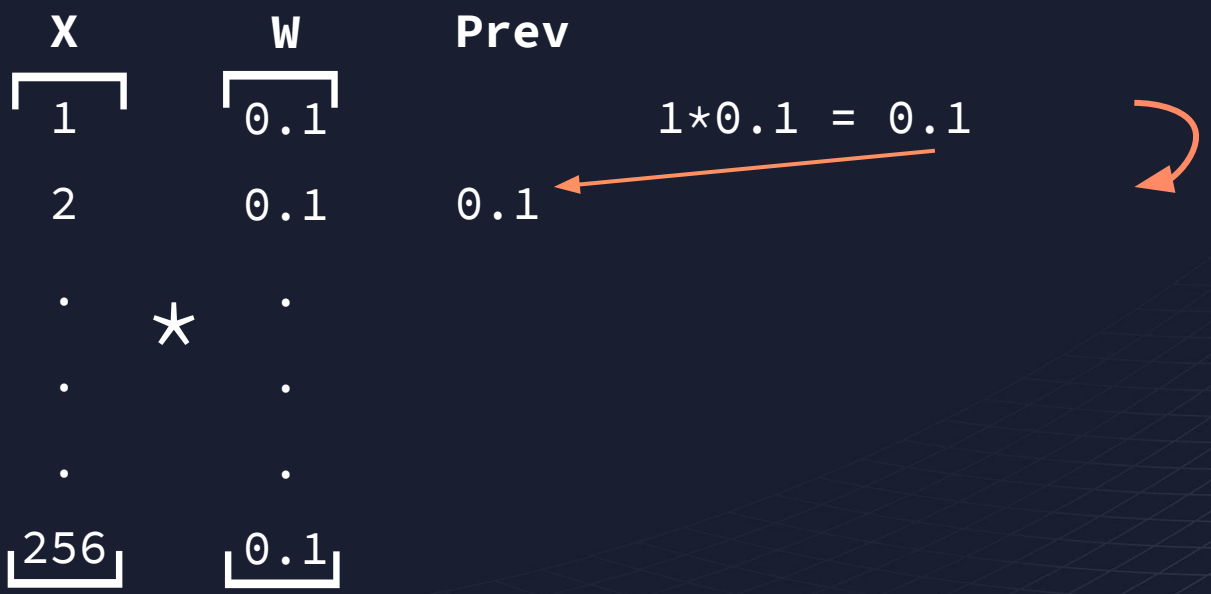
Single-threaded Execution

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

1*0.1 = 0.1

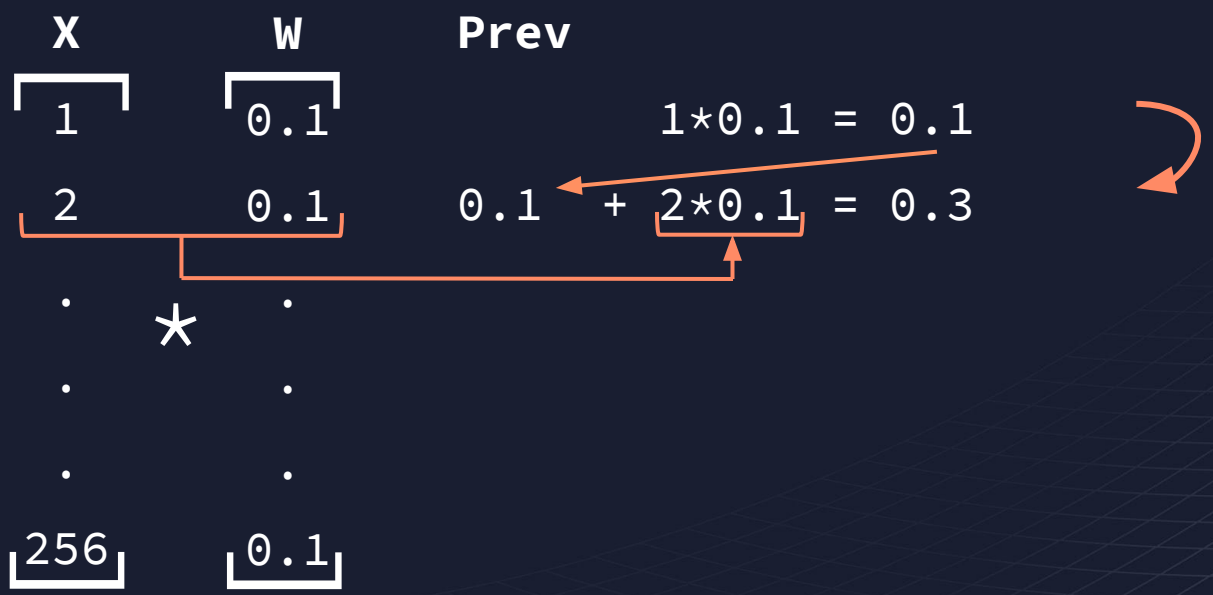
Single-threaded Execution

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```



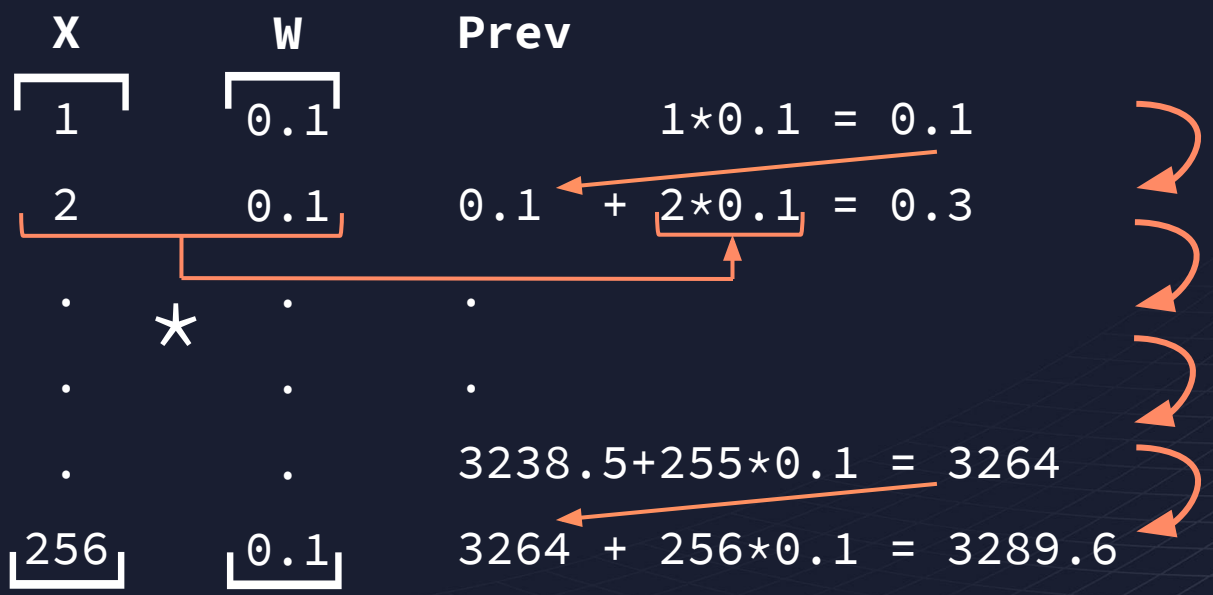
Single-threaded Execution

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```



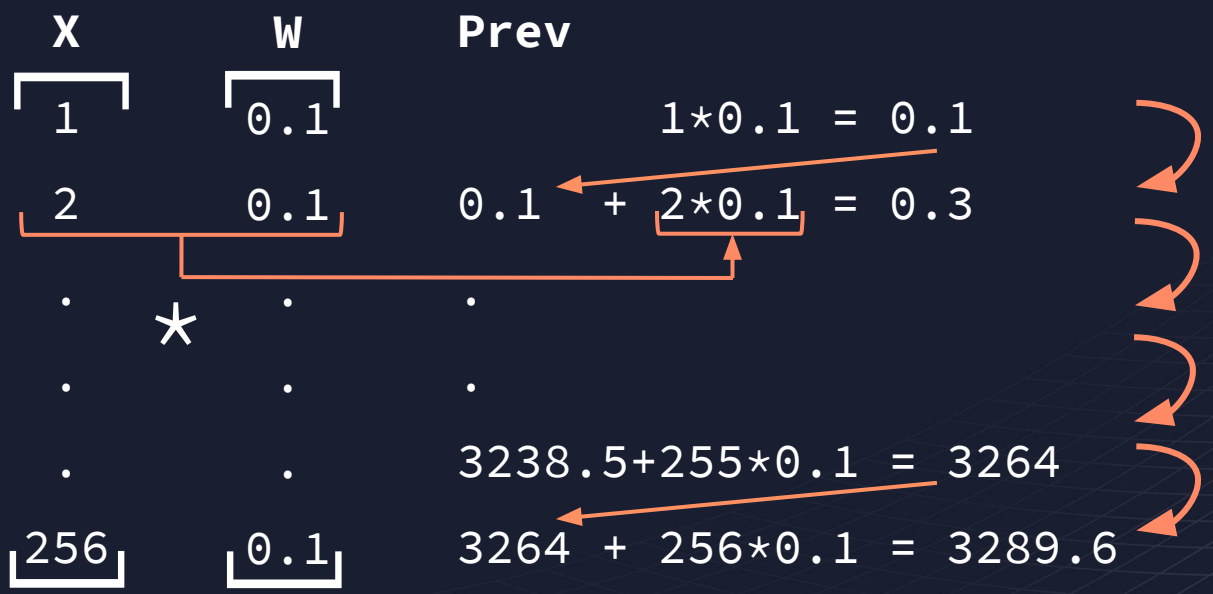
Single-threaded Execution

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```



Single-threaded Execution

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```



Single-threaded Execution

256 * ▲t

GPU Execution

GPU - #1 Multiplication Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
```

$h_{0,0}$ = X * W # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6

GPU - #1 Multiplication Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

Tesla SXM V100

5376 cores (FP32)

GPU - #1 Multiplication Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

GPU - #1 Multiplication Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

X		W	X1_mul_vector
1		0.1	1*0.1 = **0.1**
2		0.1	2*0.1 = **0.2**
.	*	.	.
.		.	.
.		.	.
256		0.1	256*0.1 = **25.6**

GPU - #1 Multiplication Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

X		W	X1_mul_vector
1		0.1	1*0.1 = **0.1**
2		0.1	2*0.1 = **0.2**
.	*	.	.
.		.	.
.		.	.
256		0.1	256*0.1 = **25.6**

Multi-threaded Execution (256 Threads)

GPU - #1 What about Summation?

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

X		W	X1_mul_vector
1		0.1	1*0.1 = **0.1**
2		0.1	2*0.1 = **0.2**
.	*	.	.
.		.	.
.		.	.
256		0.1	256*0.1 = **25.6**

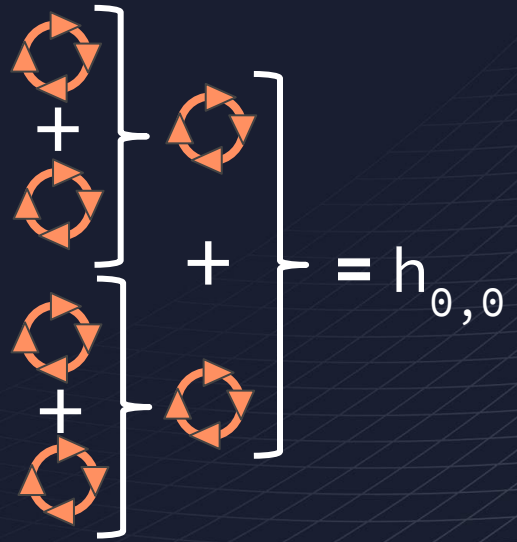
Multi-threaded Execution (256 Threads)

▲t

GPU - #2 Summary Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

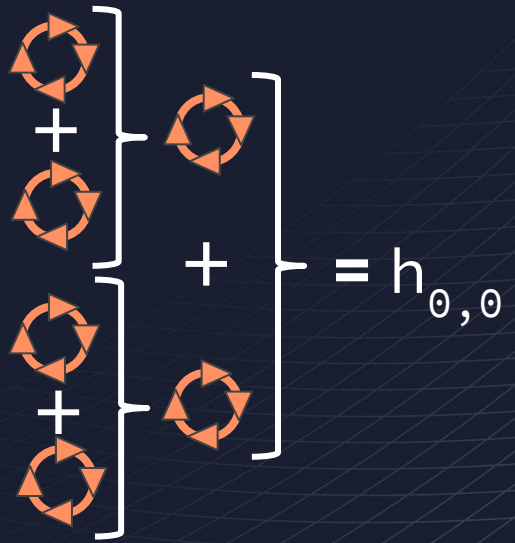
X		W	X1_mul_vector
1		0.1	1*0.1 = **0.1**
2		0.1	2*0.1 = **0.2**
.	*	.	.
.		.	.
.		.	.
256		0.1	256*0.1 = **25.6**



GPU - #2 Summary Step

```
X = [1.0, 2.0, ..., 256.0] # Let's say we have 256 input values
W = [0.1, 0.1, ..., 0.1]   # Then we need to have 256 weight values
h_0,0 = X * W  # [1*0.1 + 2*0.1 + ... + 256*0.1] == 32389.6
```

X		W	X1_mul_vector
1		0.1	1*0.1 = **0.1**
2		0.1	2*0.1 = **0.2**
.	*	.	.
.		.	.
.		.	.
256		0.1	256*0.1 = **25.6**



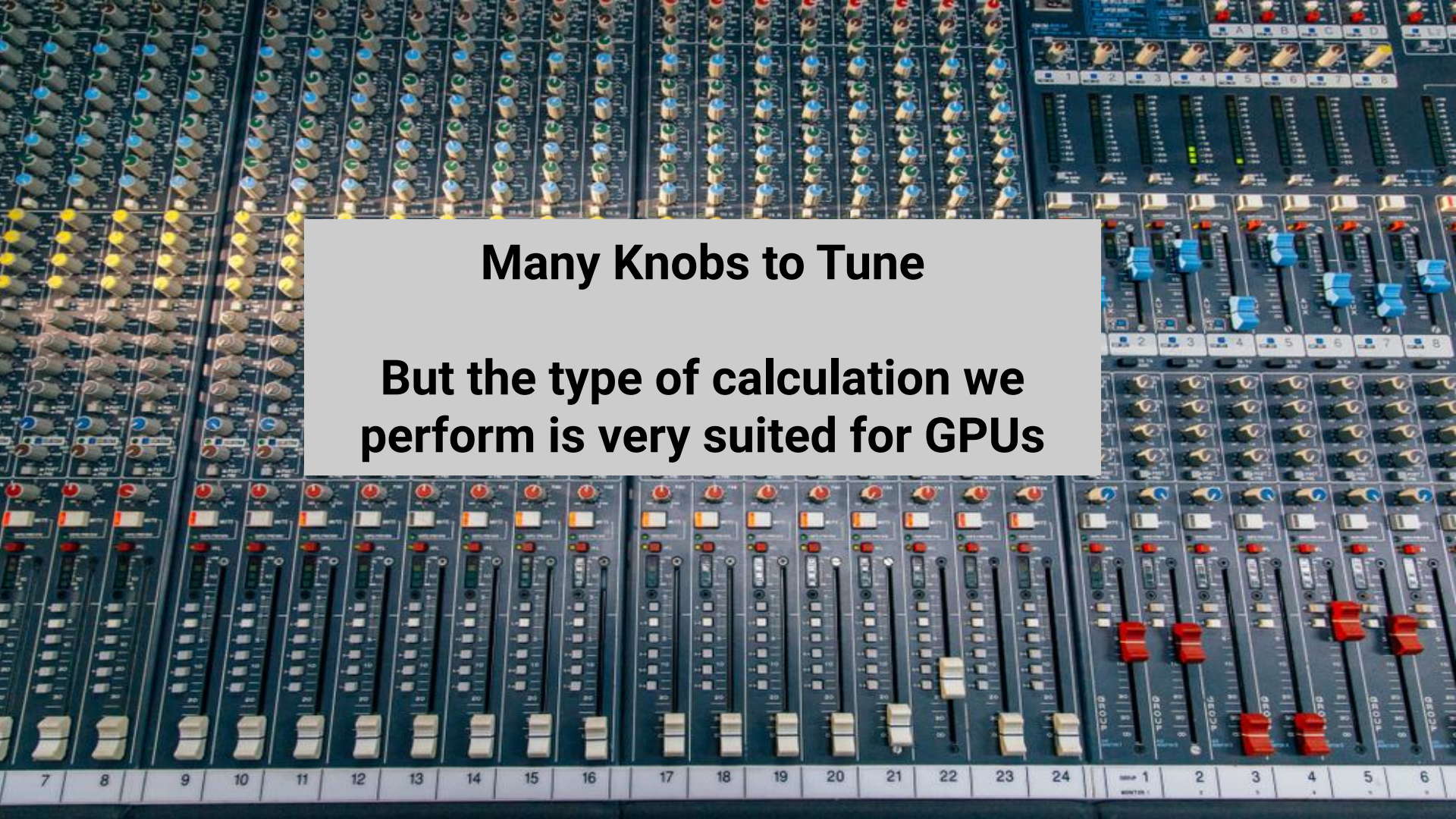
Multi-threaded Execution (256 Threads)

$\log_2 128 =$

$7 * \blacktriangle t$

Comparing - Order of Magnitude (sequences)

Single-Threaded Execution	GPU Multi-Threaded Execution
256 *△t	1 *△t + 7 *△t =



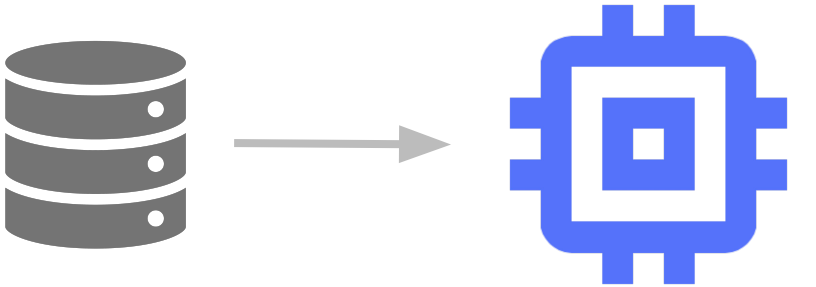
Many Knobs to Tune

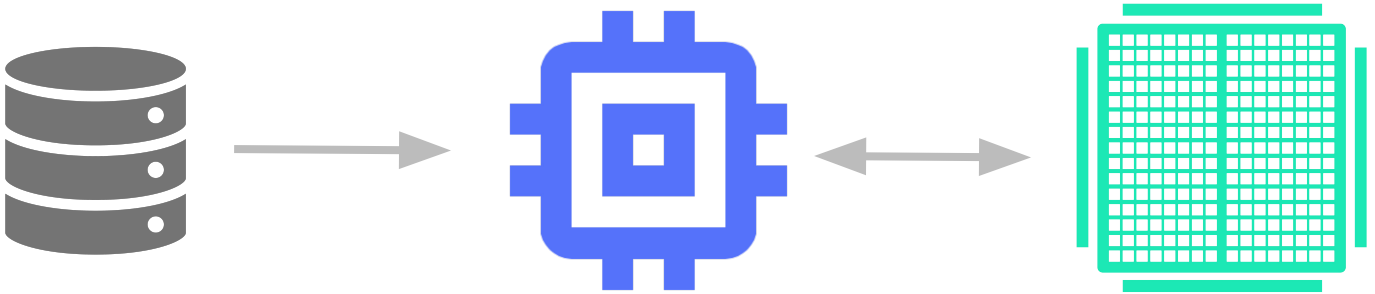
But the type of calculation we perform is very suited for GPUs

Summary

- GPUs == Many Threads == Great for ML Workloads
- And now you know how this works
- Fortunately, you don't need to worry about implementation details

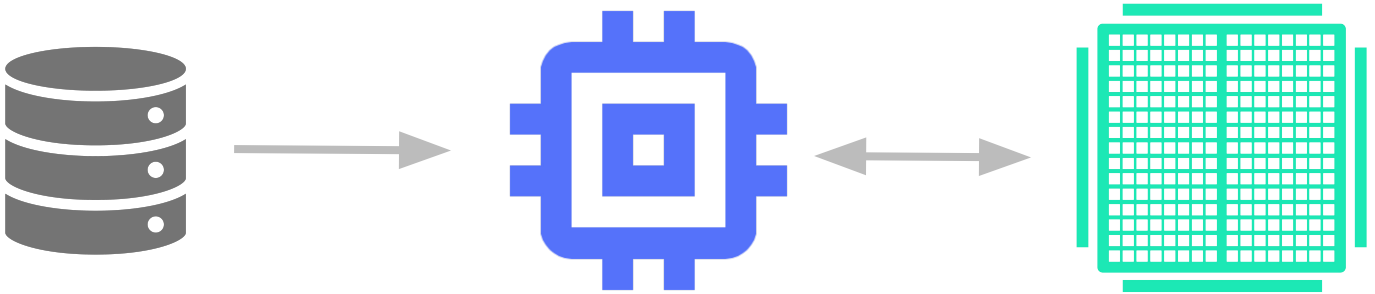

NVIDIA


multi-core
CPU


multi-core
CPU
GPU

Work needed: NONE

(just use a GPU build)



Beyond That

Use Distribution Strategy API

There's a talk for that (@ 1pm)

You Can...

tensorflow.org/learn

TensorFlow Courses

coursera.org/learn/introduction-tensorflow

udacity.com/tensorflow

Distribution Strategies

tensorflow.org/alpha/guide/distribute_strategy

@MagnusHyttsten