DERIVING KNOWLEDGE FROM DATA AT SCALE

Lecture 08

Drew Bryant, Ph.D

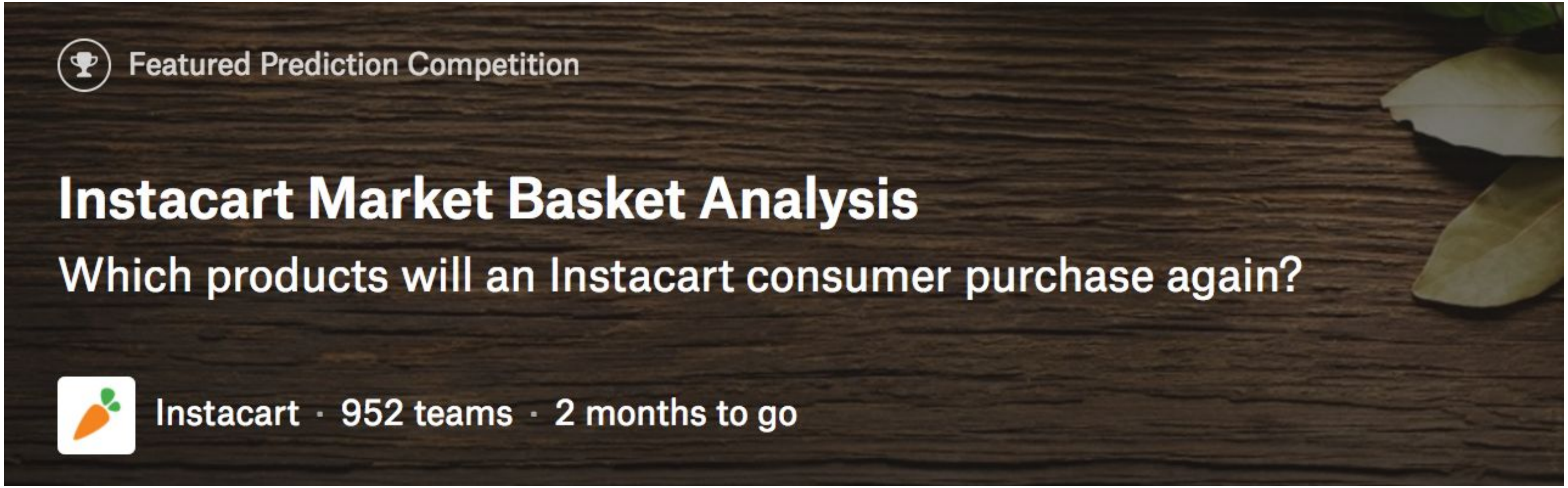
Lecture overview

- Capstone project walkthrough
- Scalable ML training techniques
- Intro to Neural Nets
- Intro to TensorFlow

Capstone project

Quick walkthrough

Capstone project: Instacart

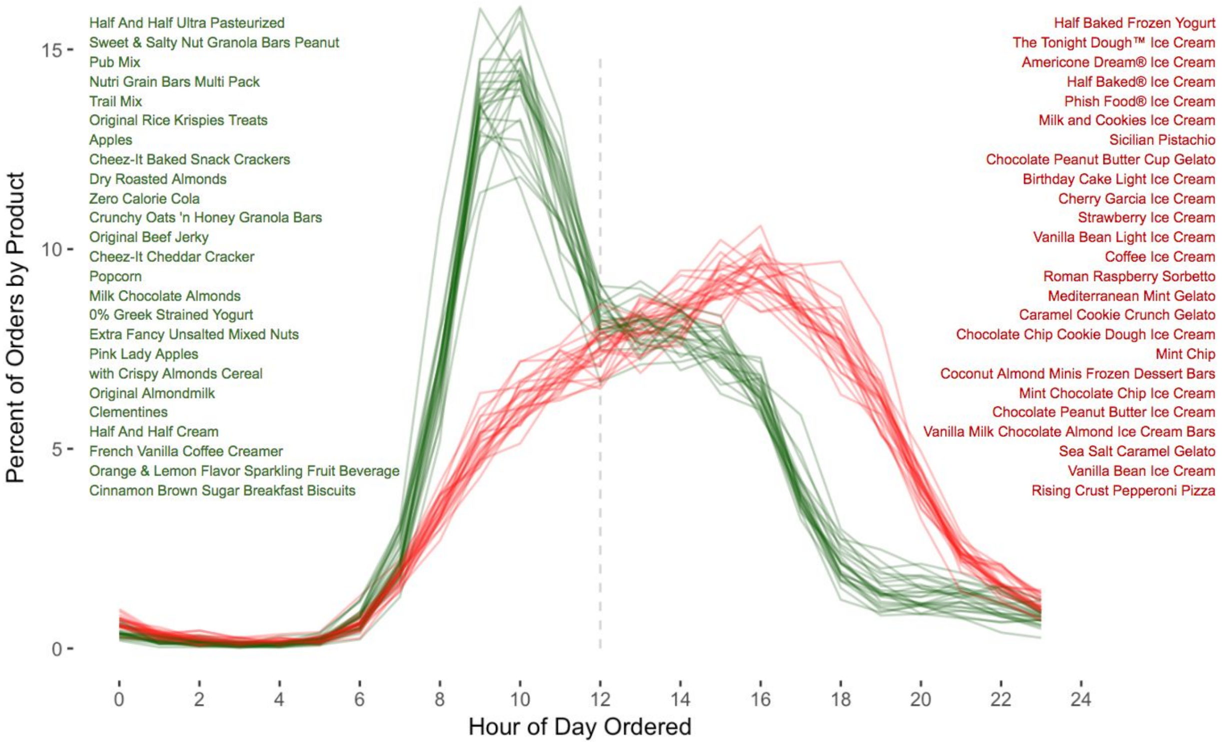


https://www.kaggle.com/c/instacart-market-basket-analysis

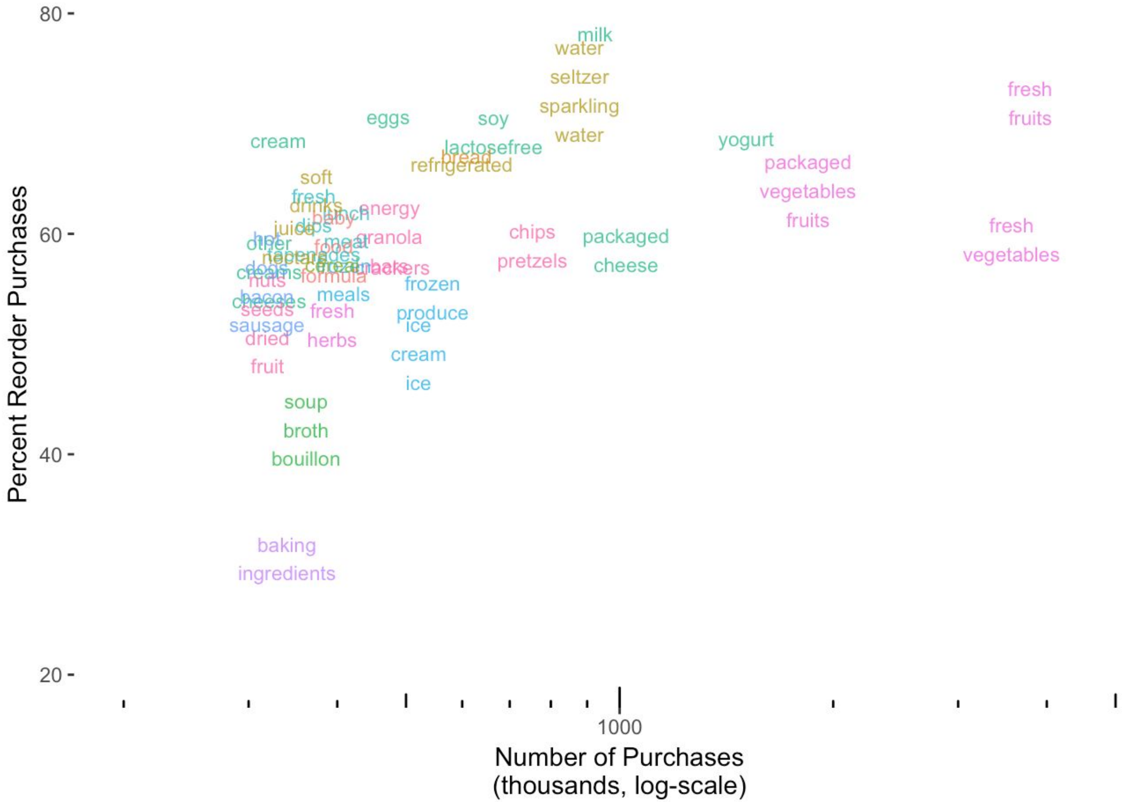
First 2 orders for user_id=1

user id	order id	order number	order dow	order hour of day	add to cart order	product id	product name
1	2539329	1	2	8	1	196	Soda
1	2539329	1	2	8	2	14084	Organic Unsweetened Vanilla Almond Milk
1	2539329	1	2	8	3	12427	Original Beef Jerky
1	2539329	1	2	8	4	26088	Aged White Cheddar Popcorn
1	2539329	1	2	8	5	26405	XL Pick-A-Size Paper Towel Rolls
1	2398795	2	3	7	1	196	Soda
1	2398795	2	3	7	2	10258	Pistachios
1	2398795	2	3	7	3	12427	Original Beef Jerky
1	2398795	2	3	7	4	13176	Bag of Organic Bananas
1	2398795	2	3	7	5	26088	Aged White Cheddar Popcorn
1	2398795	2	3	7	6	13032	Cinnamon Toast Crunch

Ordered products by time of day



Reorders from common aisles



Demo: Instacart dataset

[demo]

kaggle-instacart-overview.ipynb

Project submission details

- 1-2 page write up data exploration and feature engineering strategy
- Code to train and evaluate your model
- Predicted orders for users in the capstone-test set: "predictions.csv"
  - CSV file with two columns: "order_id","products"
  - "products" is a space-delimited list of (integer) product_ids
  - Use "None" if you predict that the order_id will be empty in lieu of product_ids list

Project submission details

```
"order_id","products"

1234,13 204 11

4567,None

8901,11
```

Each line corresponds to a single order_id in the capstone-test list.

To the left, we have predictions for 3 orders

Project evaluation

- Kaggle evaluation score: F1 (harmonic mean of precision and recall)
  - Discourage guessing "all products" (low precision)
  - Discourage guessing only high-confidence products (low recall)
- DATASCI 450 evaluation score: "was-bought" accuracy
  - Predict any one of the products that was purchased in the given order
  - Classification problem with "k-hot" target vector
    - All products in order have 1, products not in order have 0
    - [0, 0, 1, 1, 0, 1, 0, .., 1]
    - Your prediction is "correct" if you predict *any* of the 1s
    - Your prediction is "wrong" if you predict a 0
- **Choose an eval, but specify in your writeup**

Project submission details [alt]


```
"order_id","was_bought"

1234,13

4567,None

8901,11
```

Each line corresponds to a single order_id in the capstone-test list.

To the left, we have predictions for 3 orders

Tips if you're lost

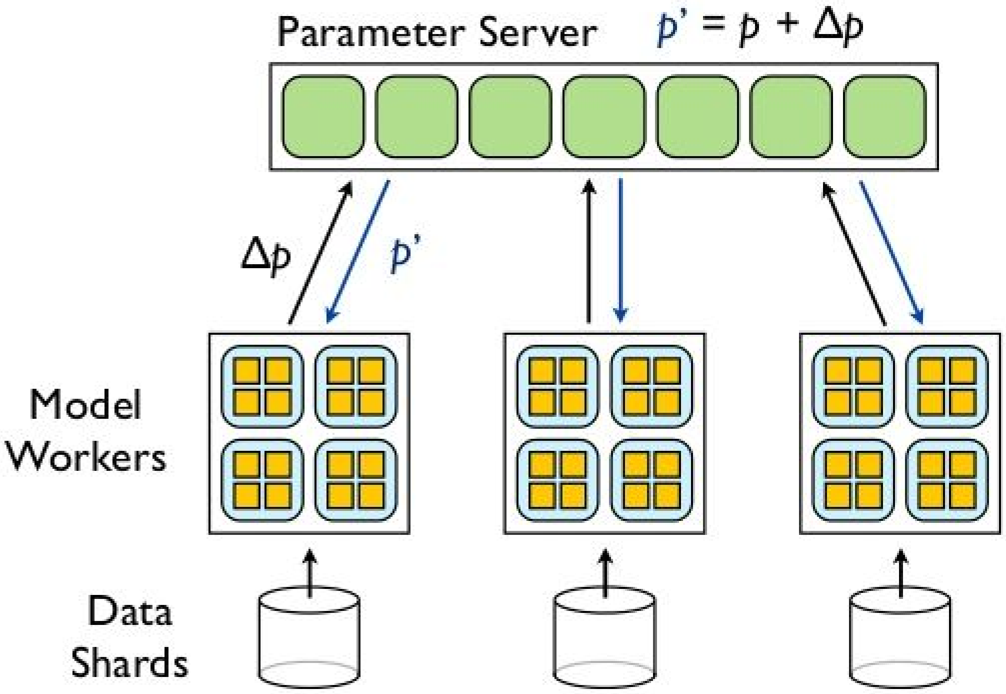
- Try some of the time-based feature transformations we discussed in class w.r.t. taxi dataset
- Try and bucket products and customers to reduce the complexity of the learning problem
- Related to your own purchasing behavior for ideas about what might drive user decision making
- Take a look at the exploratory data analysis of others on Kaggle

More details about the dataset here

Scalable ML training

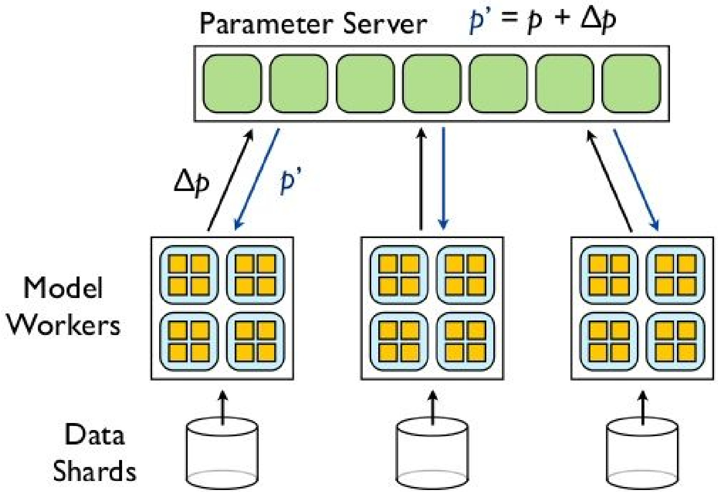
Let the tensors flow!

Async distributed SGD

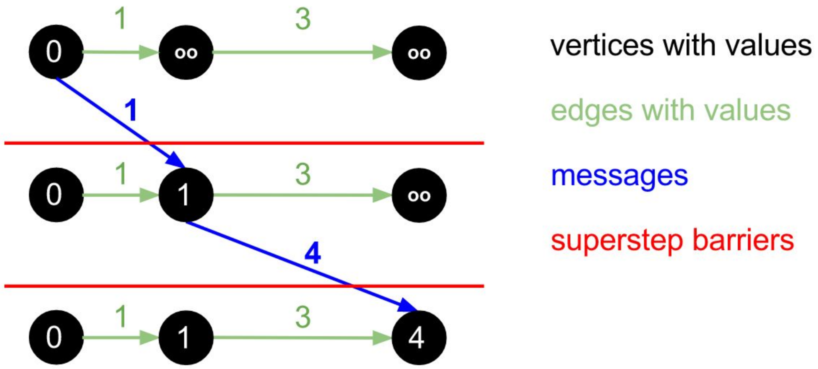


Unblocking training at scale

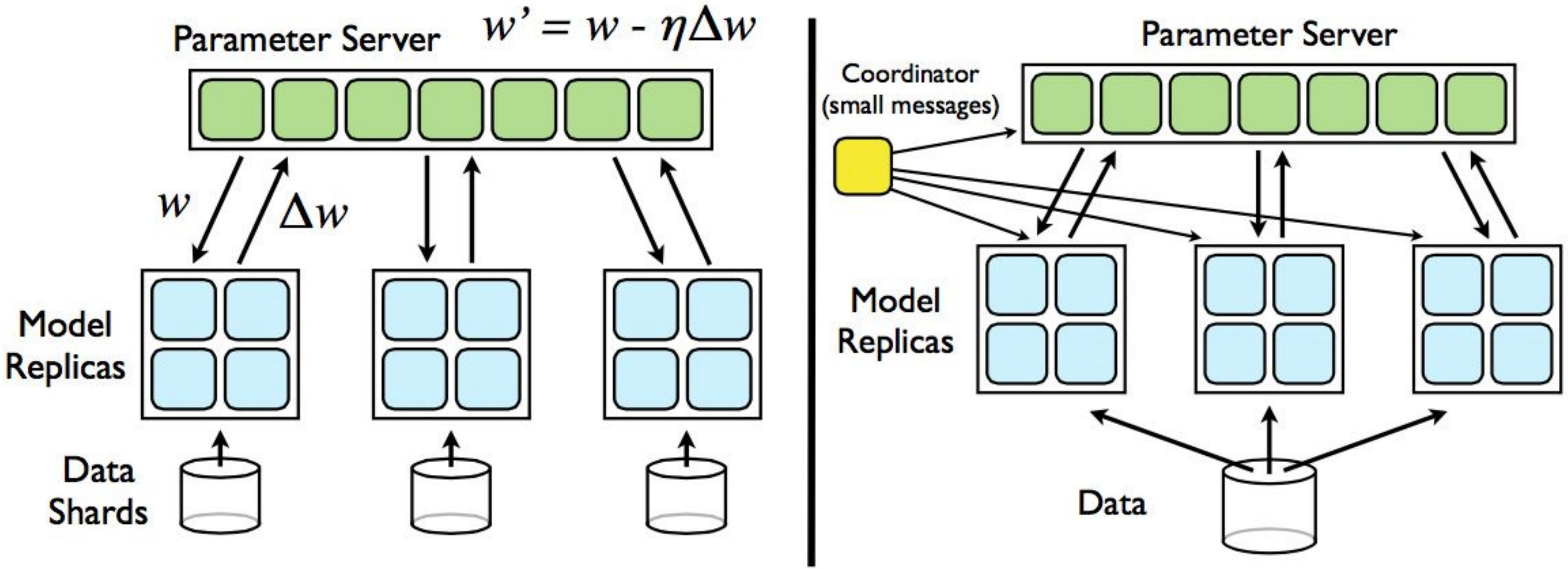
Async distributed training



Synchronous distributed training



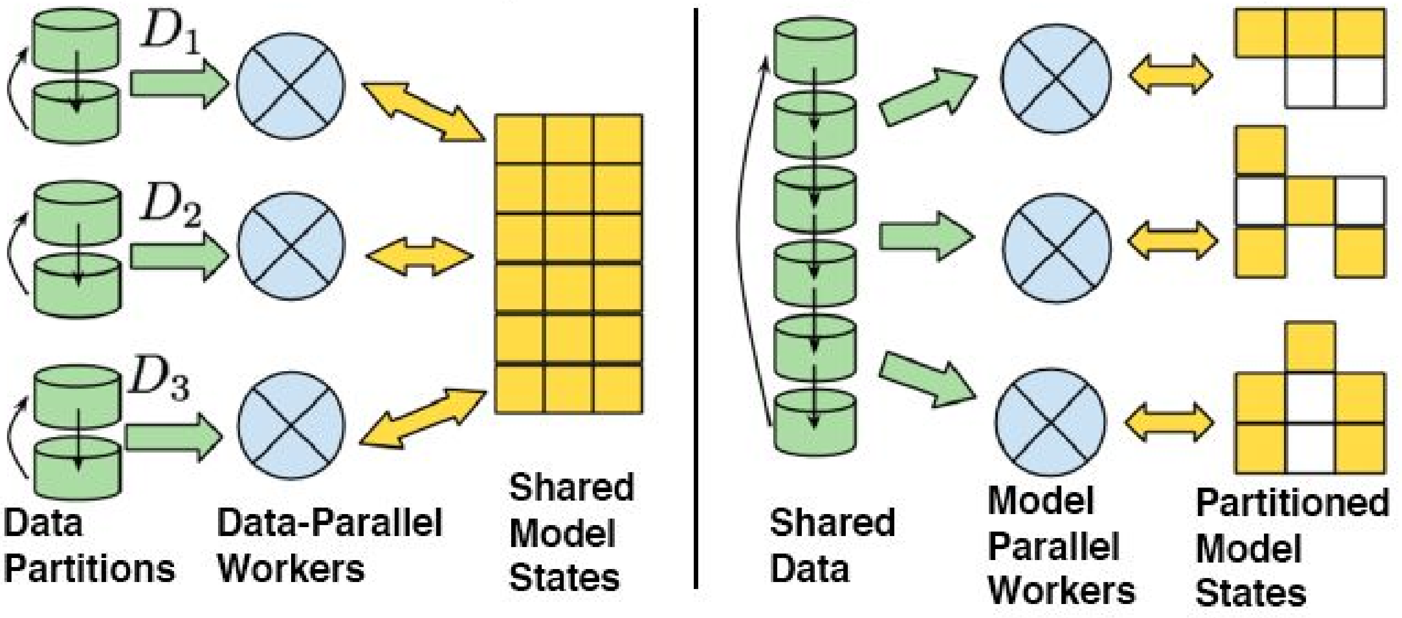
Distributed training setups



Data vs model parallelism

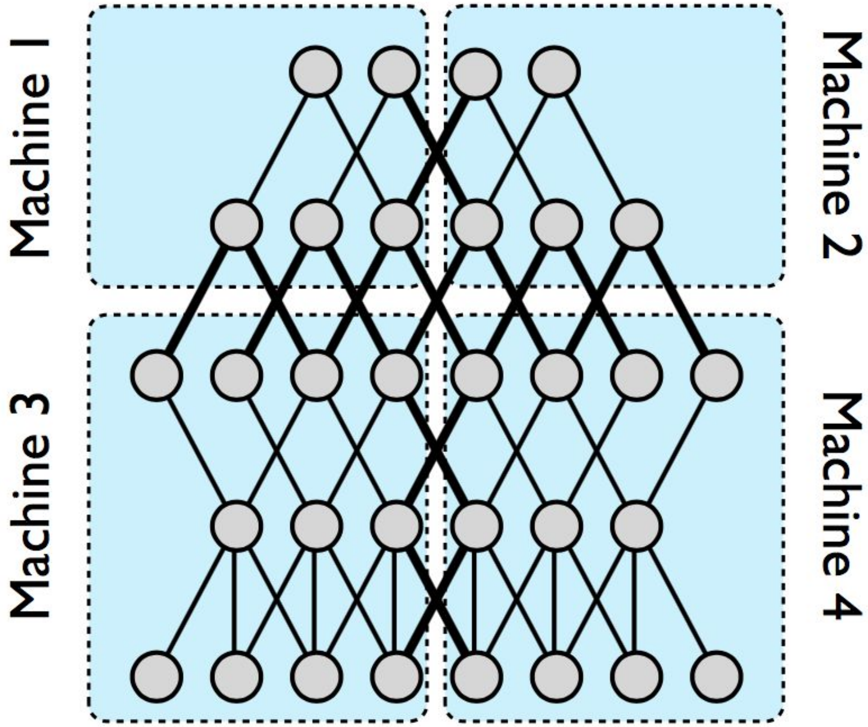
Data parallelism

Model parallelism



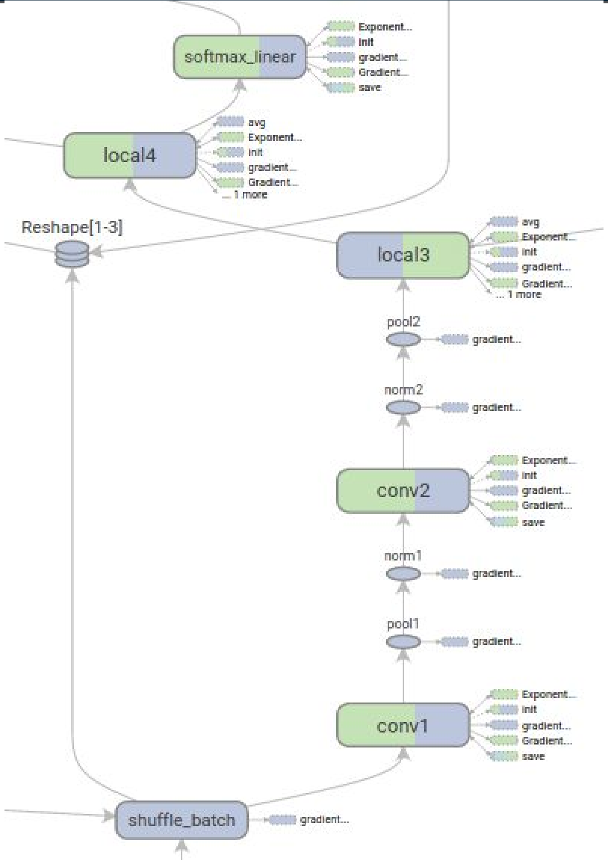
Model parallelism

Model parallelism allows large networks to be split across nodes



Visualizing and monitoring

- **TensorBoard**
- Examine how your model is distributed across nodes
- Identify performance bottlenecks

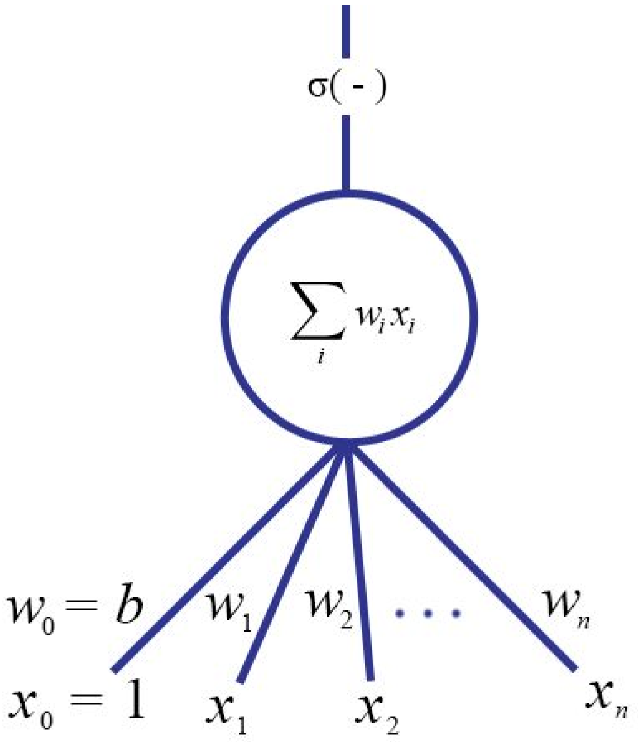


Intro to Neural Nets

Universal function approximators

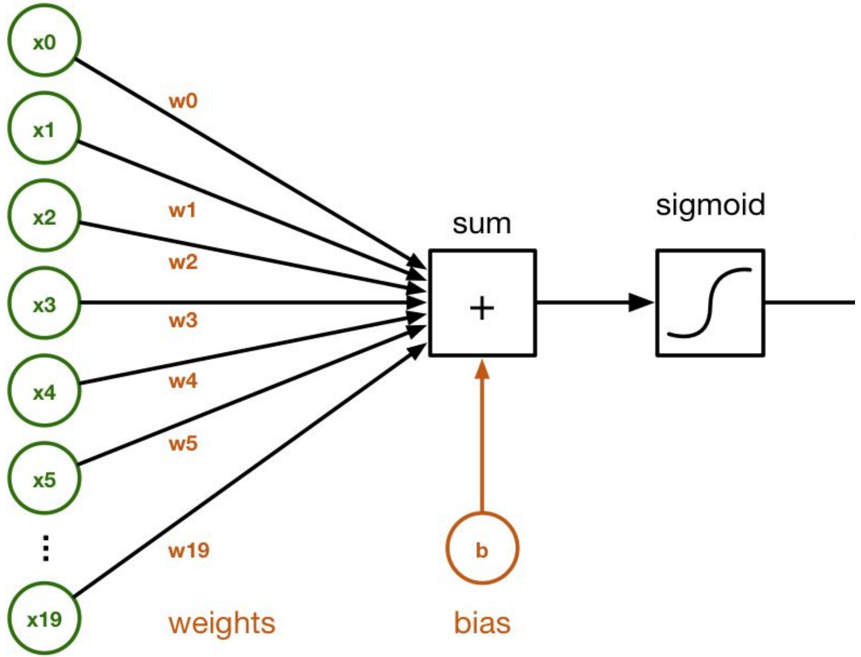
Single node: logistic regression

- Single linear node
- Activation function applied to output (sigma)
- Sigmoid for logistic regression

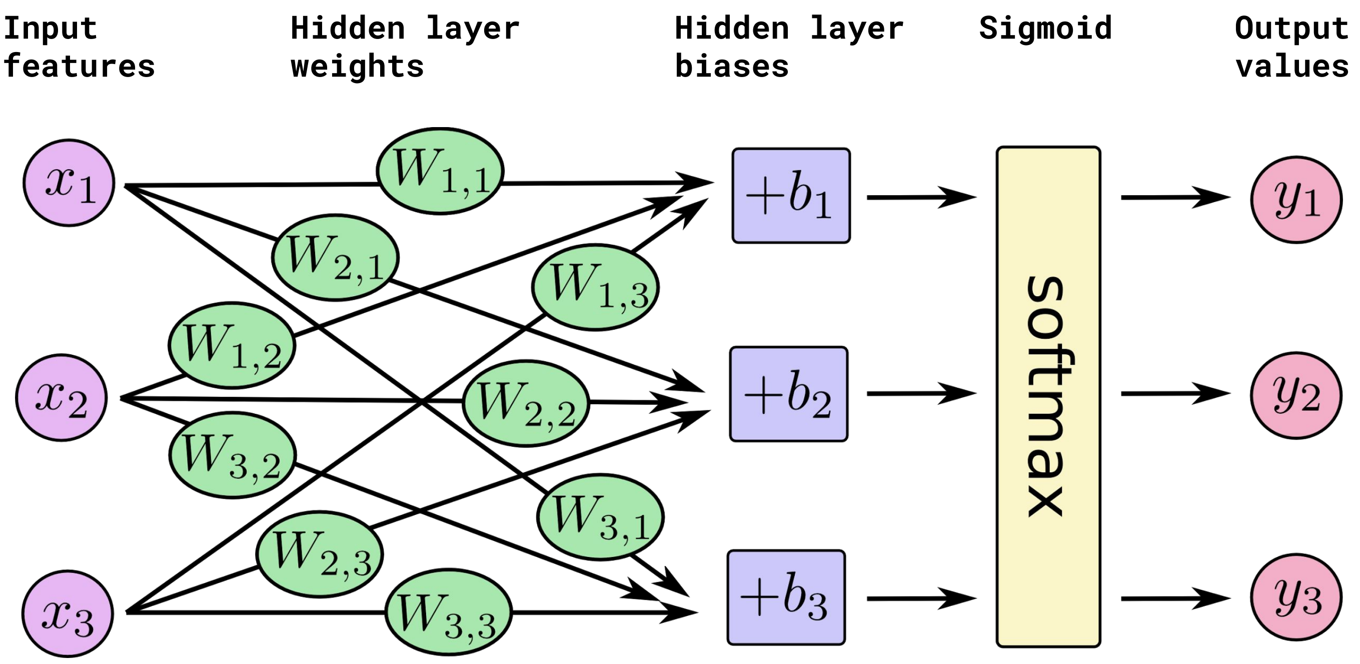


Single node: logistic regression

Another perspective



Anatomy of a neural network



Just a bunch of math

- Matrix multiply-add operations comprise most of the network evaluation (GPUs!)

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

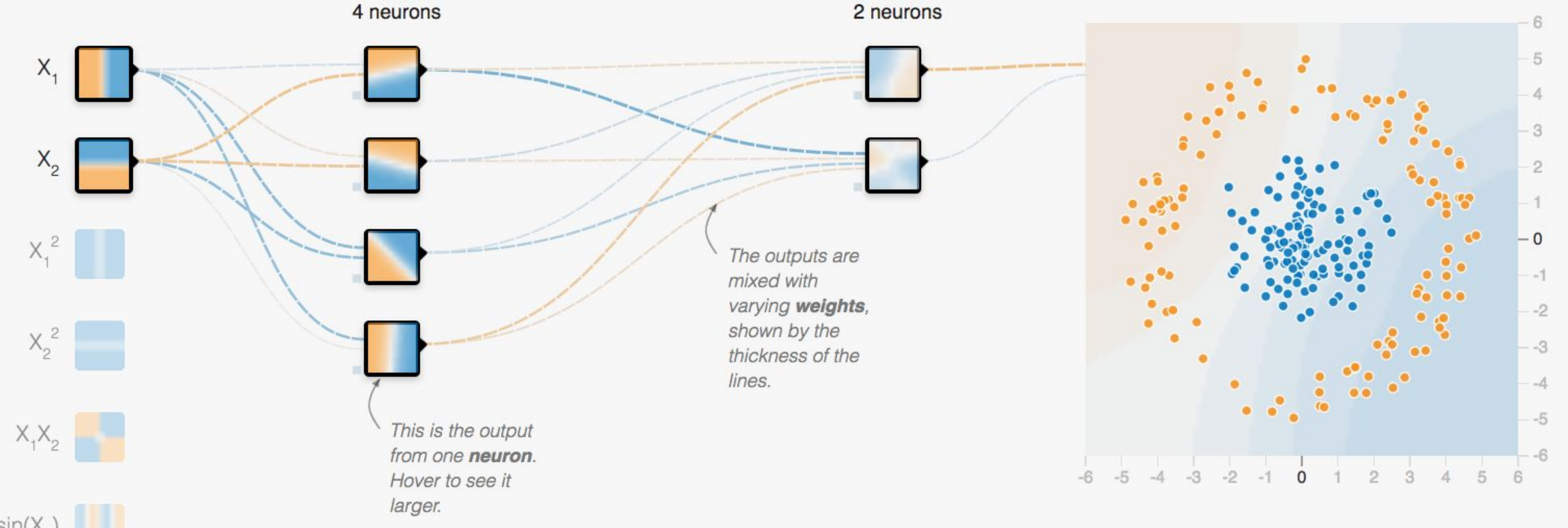
i.e., y = softmax(Wx + b)

- Multiplying out the terms gets us...

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

Intro to neural nets

- A 2-layer neural network with input features <x1, x2>

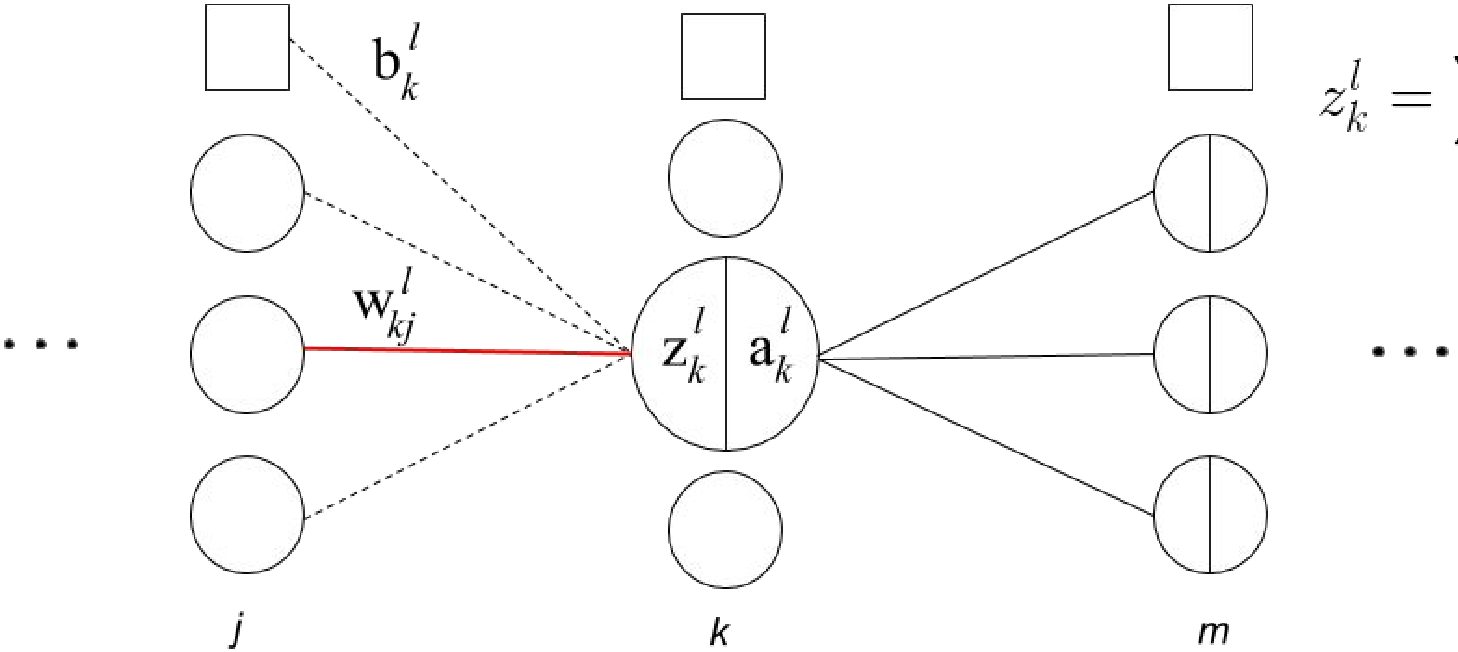


Demo: TensorFlow Playground

http://playground.tensorflow.org

TensorFlow Playground

Single neuron activation



$$z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l$$

Softmax multi-classification

- Softmax is differentiable
- Output is a multinomial distribution
- "squashes" log-odds for each class

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_K^{k=1} e^{z_k}} \text{for } j = 1, \ldots, k$$

Cross entropy loss

For a 3-class problem

predicted distribution

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

true distribution

Cross entropy loss

For a 3-class problem

predicted distribution [0.1, 0.6, 0.3]

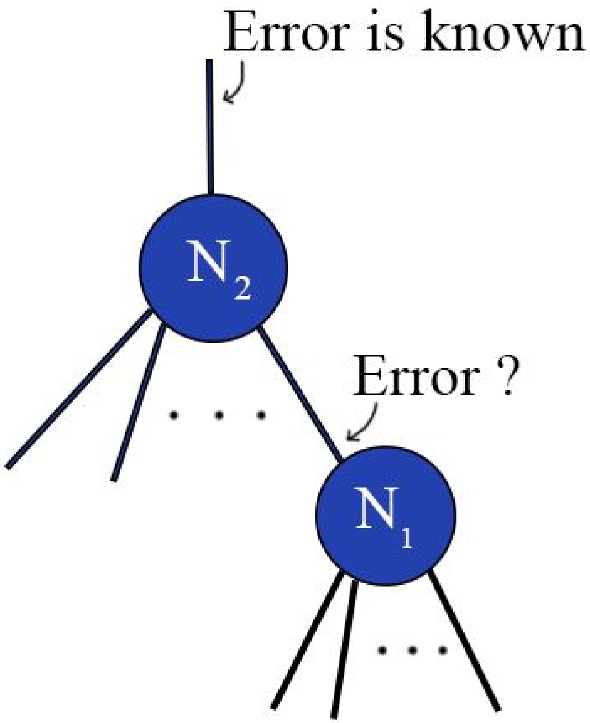
$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

true distribution [0.0, 1.0, 0.0]

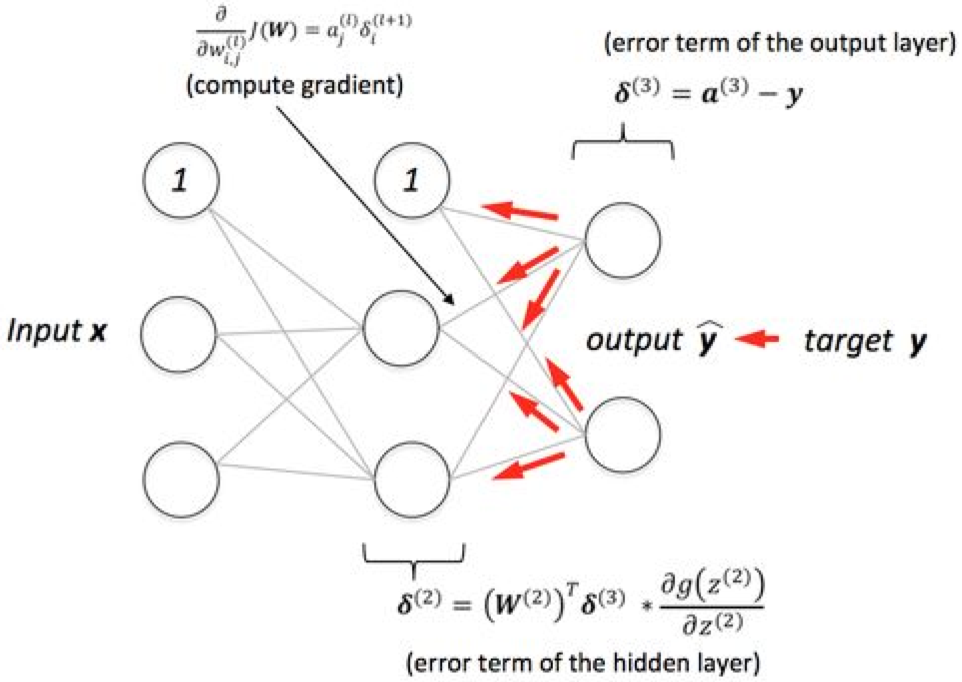
H(y) = -[0*log(.1) + 1*log(.6) + 0*log(.3)]

Backpropagation: loss gradient

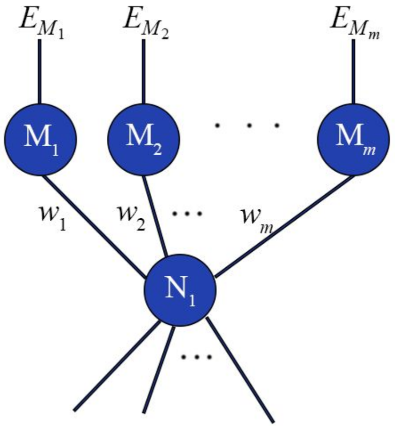
Main idea: use the error at the output layer to compute the error at previous layers via the chain rule



Backpropagation: loss gradient

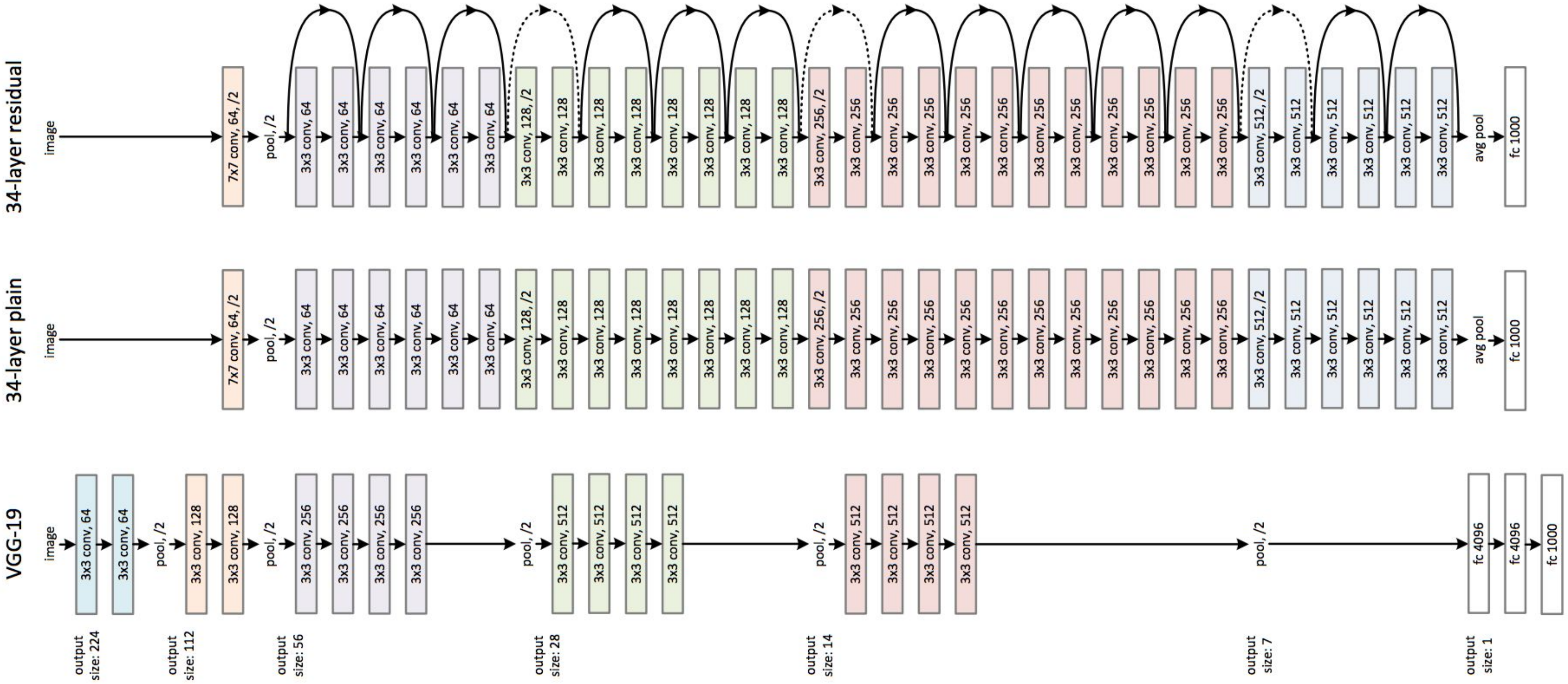


The gradient for each node depends only on its direct descendents!



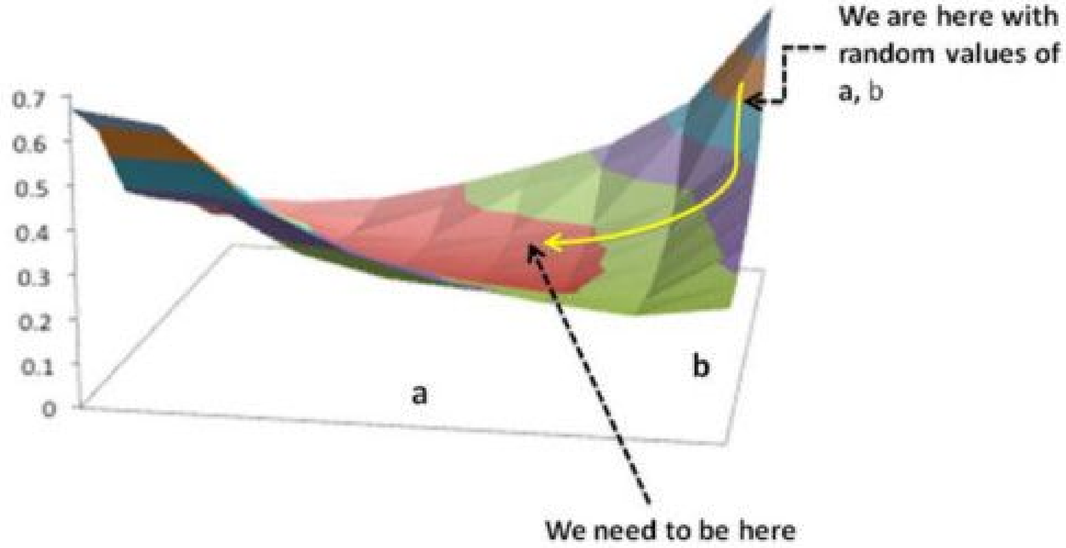
Letting the gradients flow

DATASCI 450 SU 17
LECTURE 0008
TENSORFLOW



Gradient descent

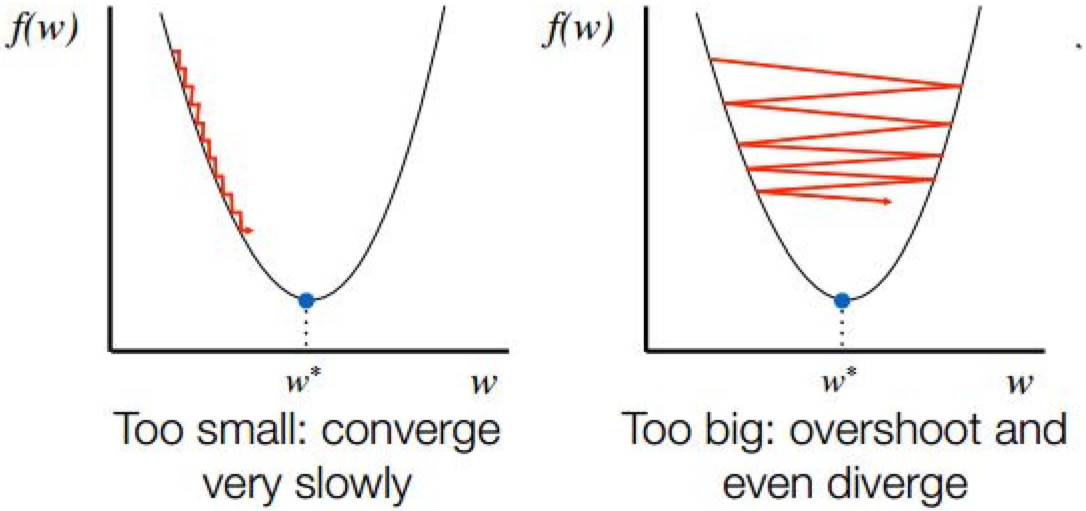
- Follow the breadcrumbs from our loss gradient
- Hope to avoid local minima along the way



$$w_{t+1} \leftarrow w_t - \eta \nabla(w_t, x_t, y_t)$$

Tuning the learning rate

The learning rate we select (hyperparameter) determines the size of each step we'll take



Adaptive learning rates

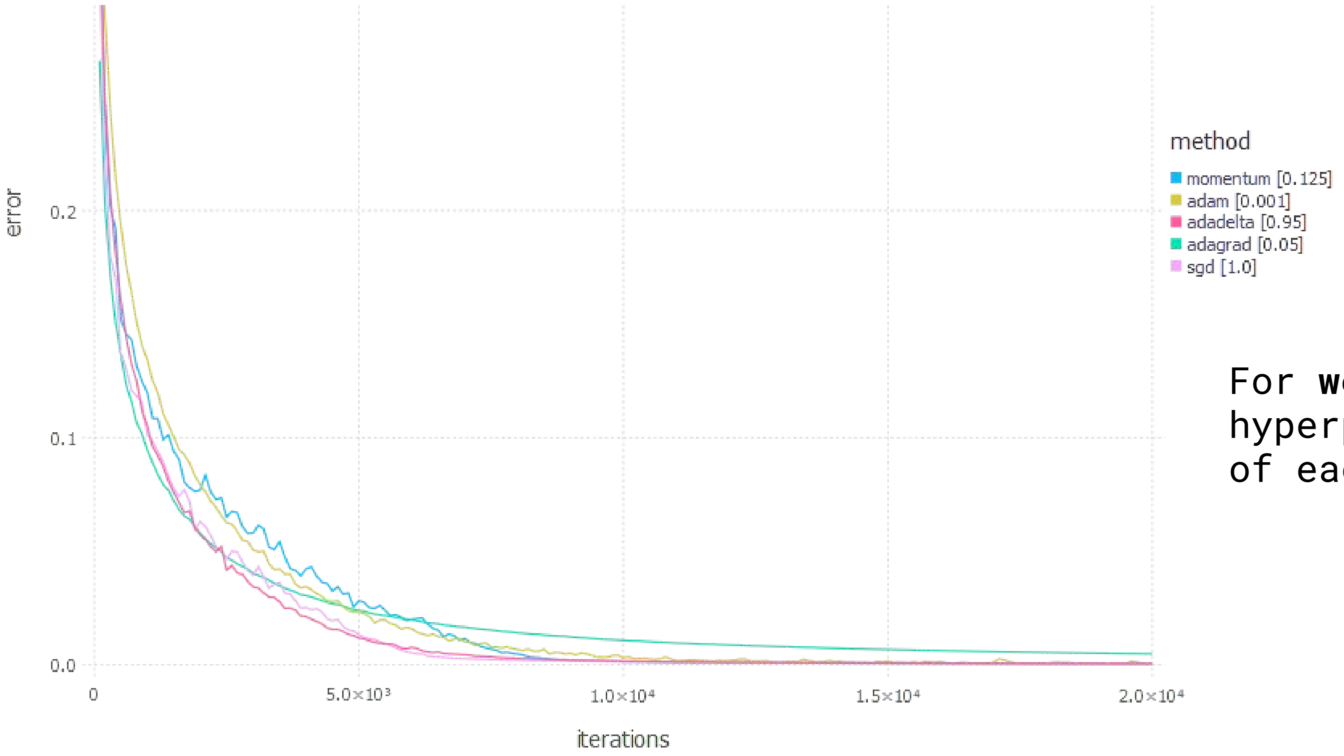
$$w_{t+1} \leftarrow w_t - \frac{\eta}{t + t_0} \nabla(w_t, x_t, y_t)$$

Popular choices

- Momentum
- Adagrad

[Good overview of how the various adaptive learning rates differ]()

Optimization methods compared

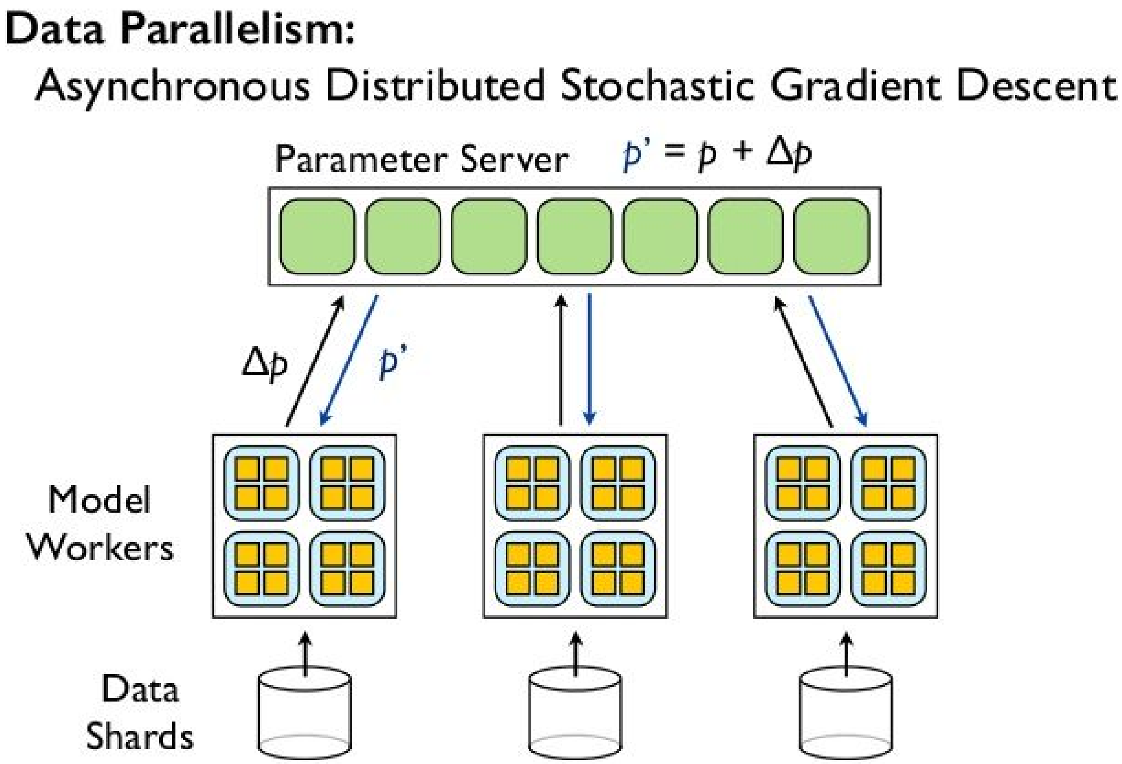


For **well-chosen** hyperparameters of each!

Stochastic gradient descent

- Standard gradient descent
  - Compute gradient from ALL examples for each update of the parameters
- **Stochastic** gradient descent (SGD)
  - Use a single (random) example for each update
- **Mini-batch** stochastic gradient descent
  - Use a random sample of examples for each update

Stochastic gradient descent



Intro to TensorFlow

Computation graphs and symbolic diffs

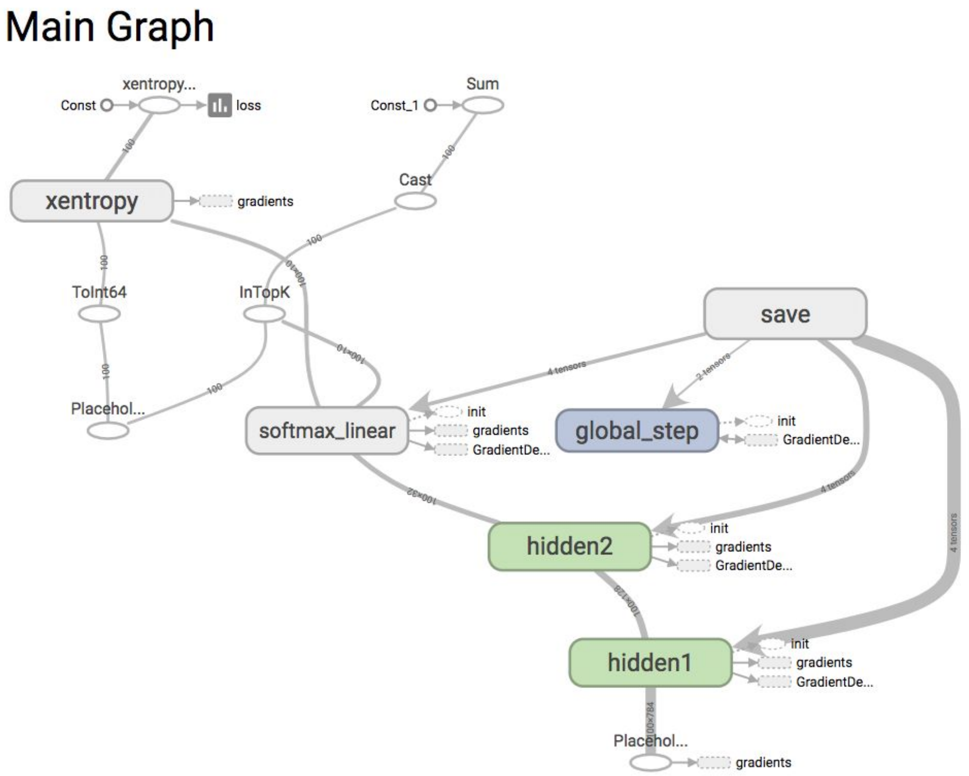
What is TensorFlow?

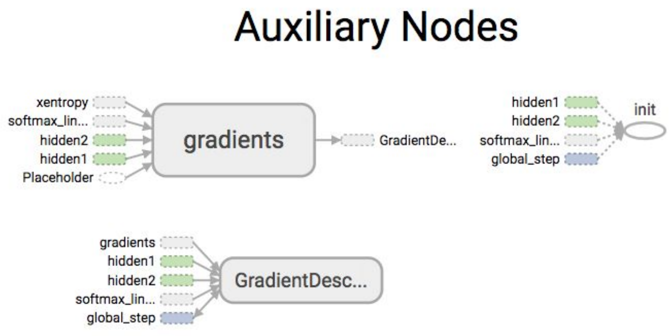
- Symbolic differentiation engine
- Language-agnostic computation graph
- Distributed programming abstraction
- Library of ML algorithms (defined via symbolic graphs)
- Infrastructure coordinating distributed optimization
- Inference serving, deployment, and monitoring tools
- Backend compute for various high-level APIs: e.g., keras, theano

Symbolic computation graph

- A computational graph is a series of TensorFlow operations arranged into a graph of nodes.

Symbolic computation graph





"Tensors"

A rank 0 tensor; this is a scalar with shape []

```
3
```

"Tensors"

A rank 1 tensor; this is a vector with shape [3]

```
[1. ,2., 3.]
```

"Tensors"

A rank 2 tensor; a matrix with shape [2, 3]

```
[[1., 2., 3.],
 [4., 5., 6.]])
```

"Tensors"

A rank 3 tensor with shape [2, 1, 3]

```
[[[1., 2., 3.]],
 [[7., 8., 9.]]]
```

Nodes and operations

Your first operation: add two constants

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
node3 = tf.add(node1, node2)
```

TensorFlow sessions

Creating a session and running an operation

```
.... define your graph of operations here ....

# Evaluate a node within the graph
sess = tf.Session()
sess.run([...any nodes go here...])
```

Structure of a TensorFlow program

TensorFlow Core programs as consisting of two discrete sections:

1. Building the computational graph.
2. Running the computational graph.

TensorFlow sessions

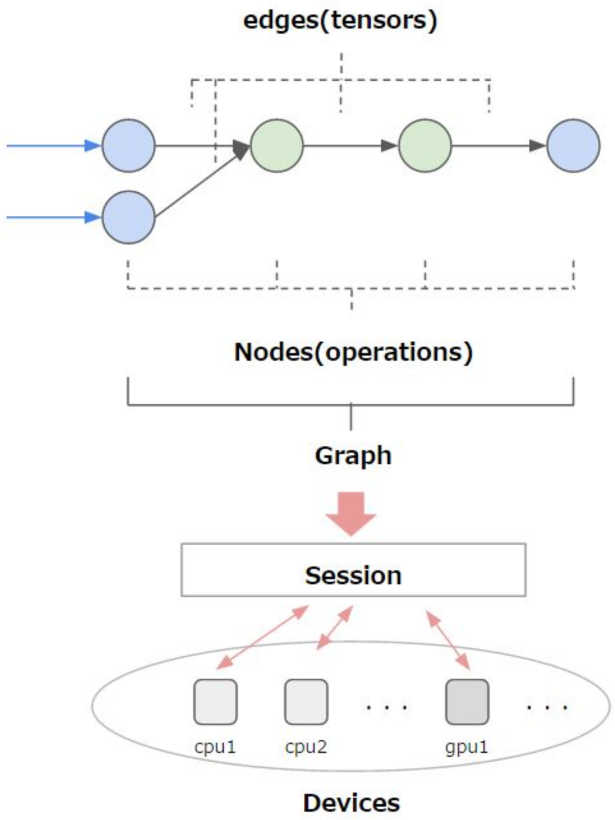
Creating a session and running an operation

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
sum_node = tf.add(node1, node2)

sess = tf.Session()
sess.run([sum_node])
```

Recap: elements of TensorFlow

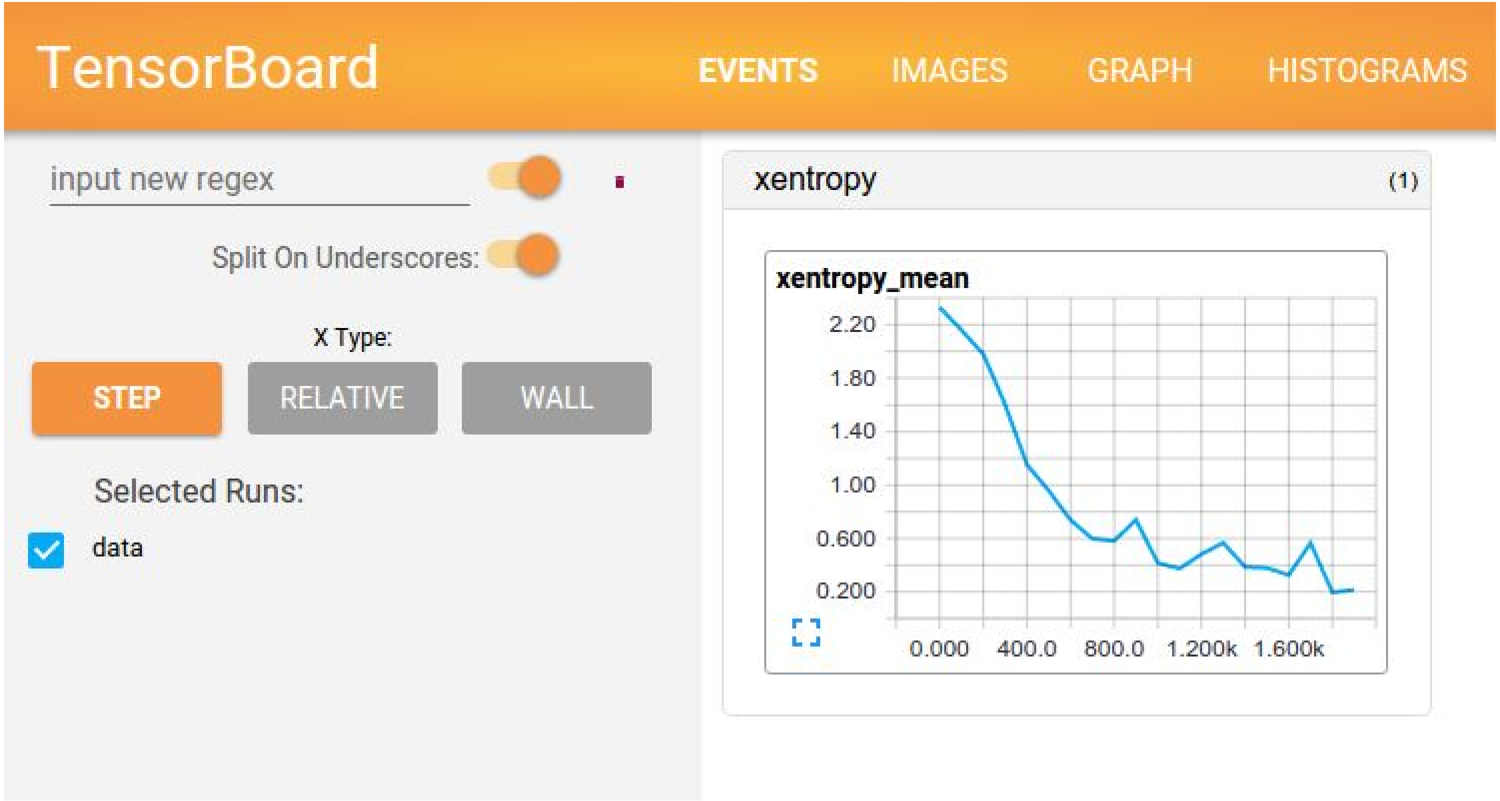
- Tensors
- Nodes
- Operations
- Graph
- Session
- Devices



Demo: Hello, TensorFlow

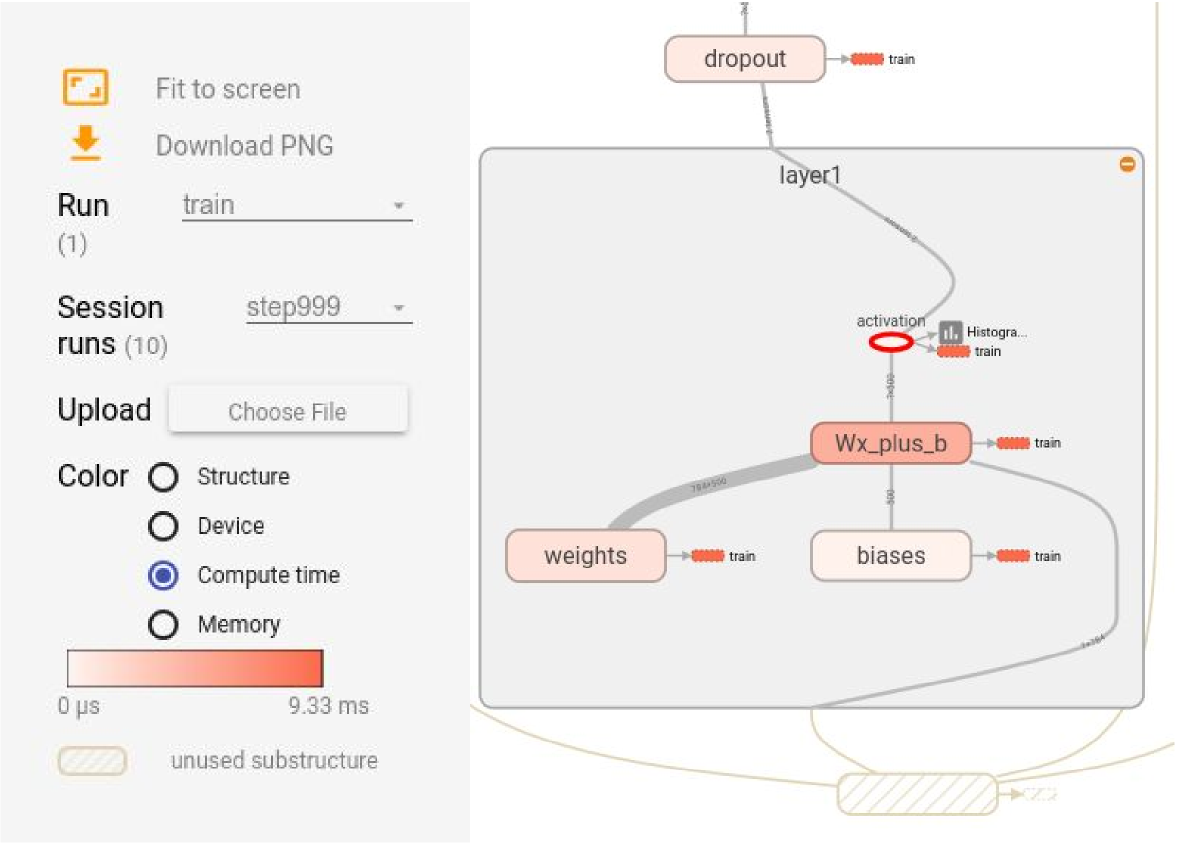
[demo] **tf-hello.ipynb**

TensorBoard



TensorBoard: advanced use cases

- Analyze time spent in parts of graph
- Identify bottlenecks



Demo: TensorFlow graph viz

[demo]

tf-graph-viz.ipynb

Demo: linear regression in TF

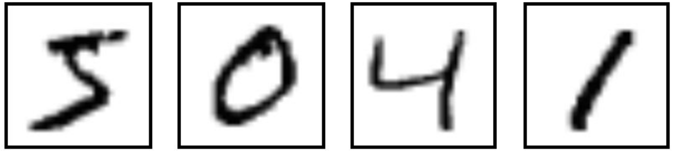
[demo]

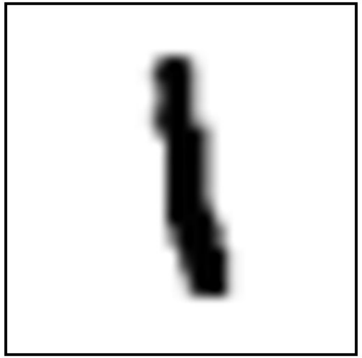
tf-linear-regression.ipynb

Demo: basic neural network

[demo]

tf-mnist.ipynb





$$\approx \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.6&.8&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.7&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.7&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&.5&1&.4&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&.4&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&.4&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&.7&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&.9&1&.1&0&0&0&0 \\ 0&0&0&0&0&0&0&.3&1&.1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

Demo: end-to-end modelling

[demo]

tf-e2e.ipynb

Questions?