Deep Learning with TensorFlow

Deep Learning — Units 7 & 8

Dr. Jon Krohn
jon@untapt.com

Slides available at jonkrohn.com/talks

November 23, 2019

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Outline

Assessing

Your Deep Learning Project III

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Assessing

Your Deep Learning Project III

Where are you at with respect to the following?

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark

Outline

Leading DL Libraries

	Caffe	**Torch**	**MXNet**	**TensorFlow**
Language	Python, Matlab	Lua, C	Python, R, C++ Julia, Matlab JavaScript, Go Scala, Perl	**Python, C**, C++ Java, Go, JS, Swift (*Haskell, Julia, R, Scala, Rust, C#)*
Programming Style	Symbolic	Imperative	Imperative	Imperative *(in 2.0)*
Parallel GPUs: Data	Yes	Yes	Yes	Yes
Parallel GPUs: Model		Yes	Yes	Yes
Pre-Trained Models	Model Zoo	Model Zoo	Model Zoo	github.com/tensorflow/models
High-Level APIs		PyTorch	in-built	Keras
Particular Strength	CNNs	interactivity		production deployment

Leading DL Libraries

PyTorch	TensorFlow
"NumPy", optimized for GPUs	ported to Python from C++
dynamic auto-differentiation (autodiff)	static computational graph
debugging is easier	
fast.ai API	Keras API
	more widely adopted
TorchScript Just-In-Time compilation	TensorFlow Serving, .js, Lite, tf.data, tf.io
better for interactively building models	**better for production deployments**

Outline

Outline

TensorFlow Graphs

1. build graph
2. initialize session
3. fetch and feed data

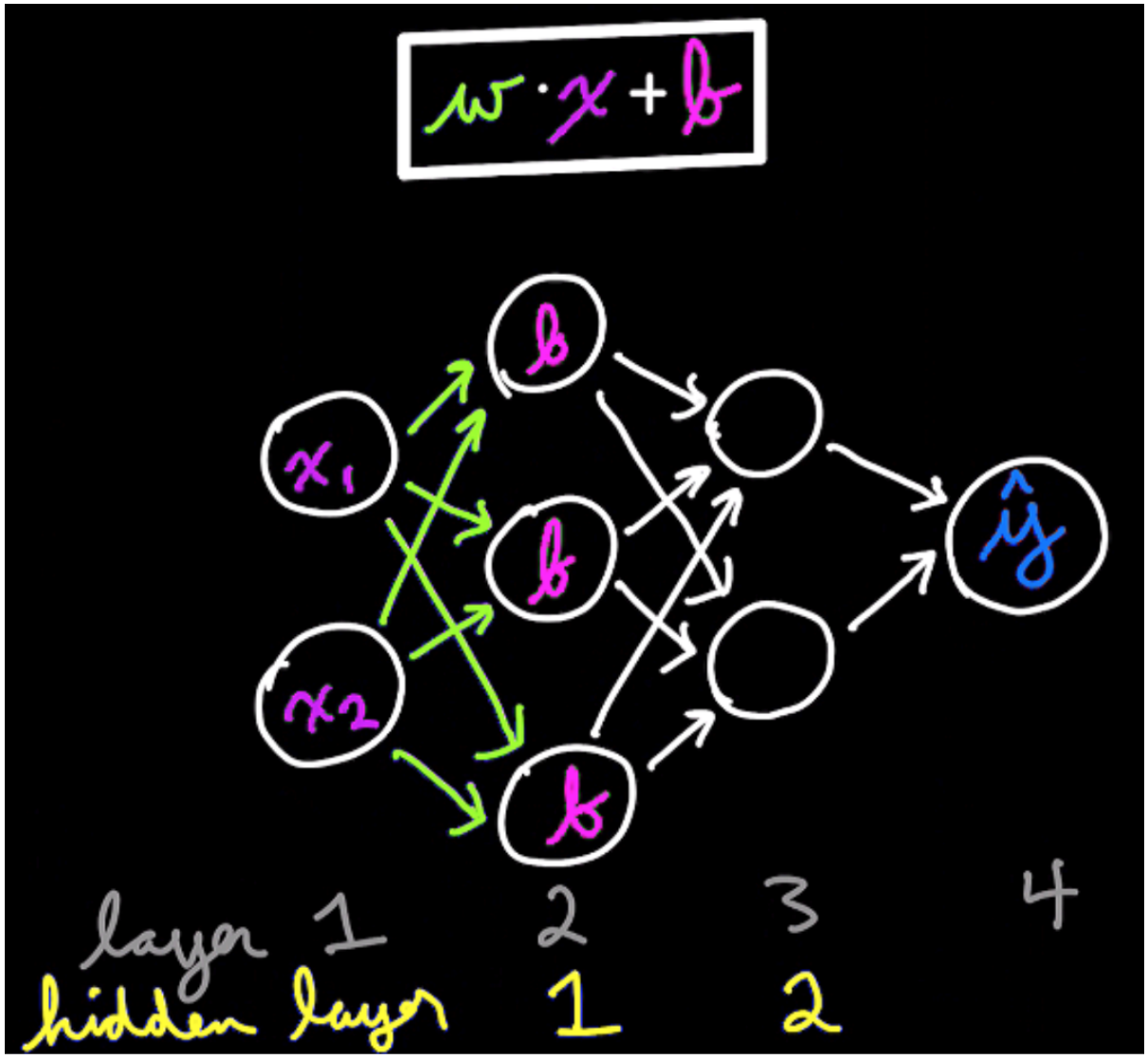
TensorFlow Graphs

1. build graph
2. initialize session
3. fetch and feed data

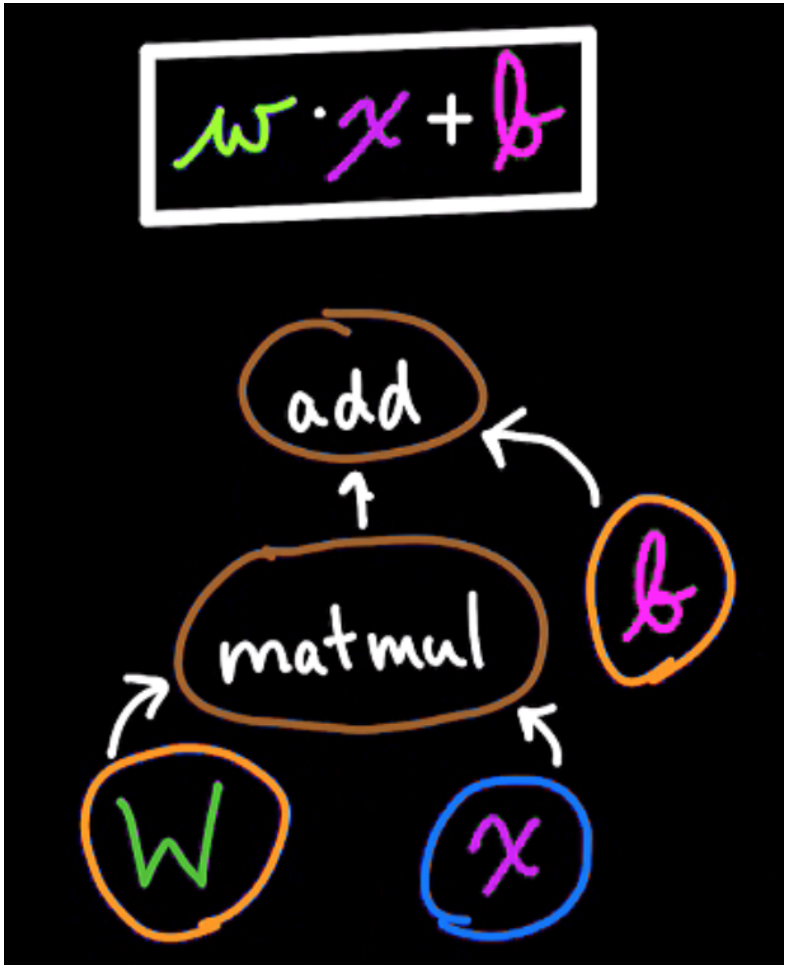
TensorFlow Graphs

1. build graph
2. initialize session
3. fetch and feed data

A Familiar Equation



TensorFlow Graphs



Outline

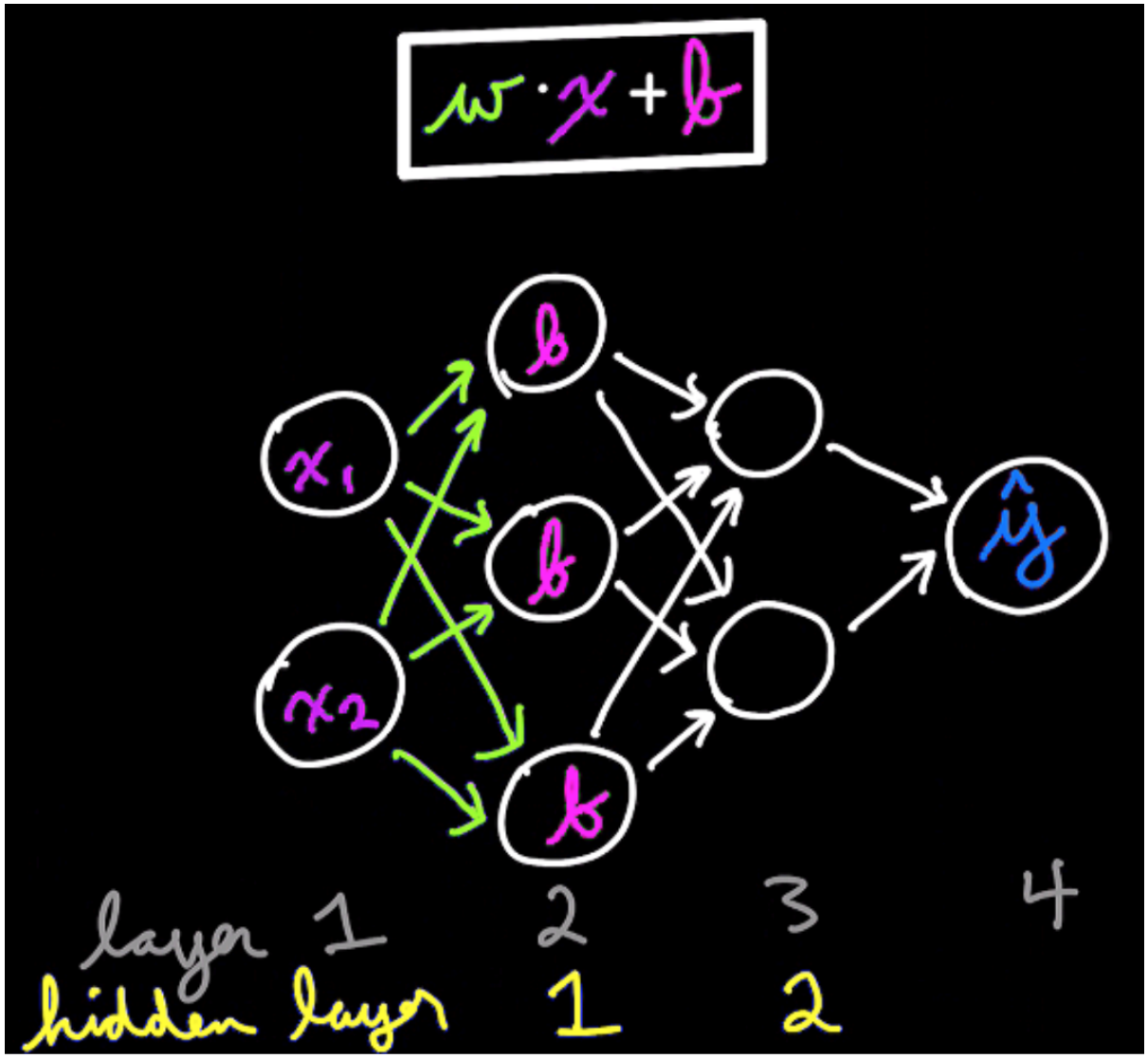
TensorFlow Graph

Programming

[*first TensorFlow graphs* notebook]

Outline

A Familiar Equation



Neurons in TensorFlow

Programming

[*first TensorFlow neurons* notebook]

Outline

Outline

Fitting Eight Points

[*point by point intro to TensorFlow* notebook]

Fitting Eight Points

with Tensors

[*tensor-fied intro to TensorFlow* notebook]

Outline

Fitting Eight Million Points

[*intro to TensorFlow times a million* notebook]

Outline

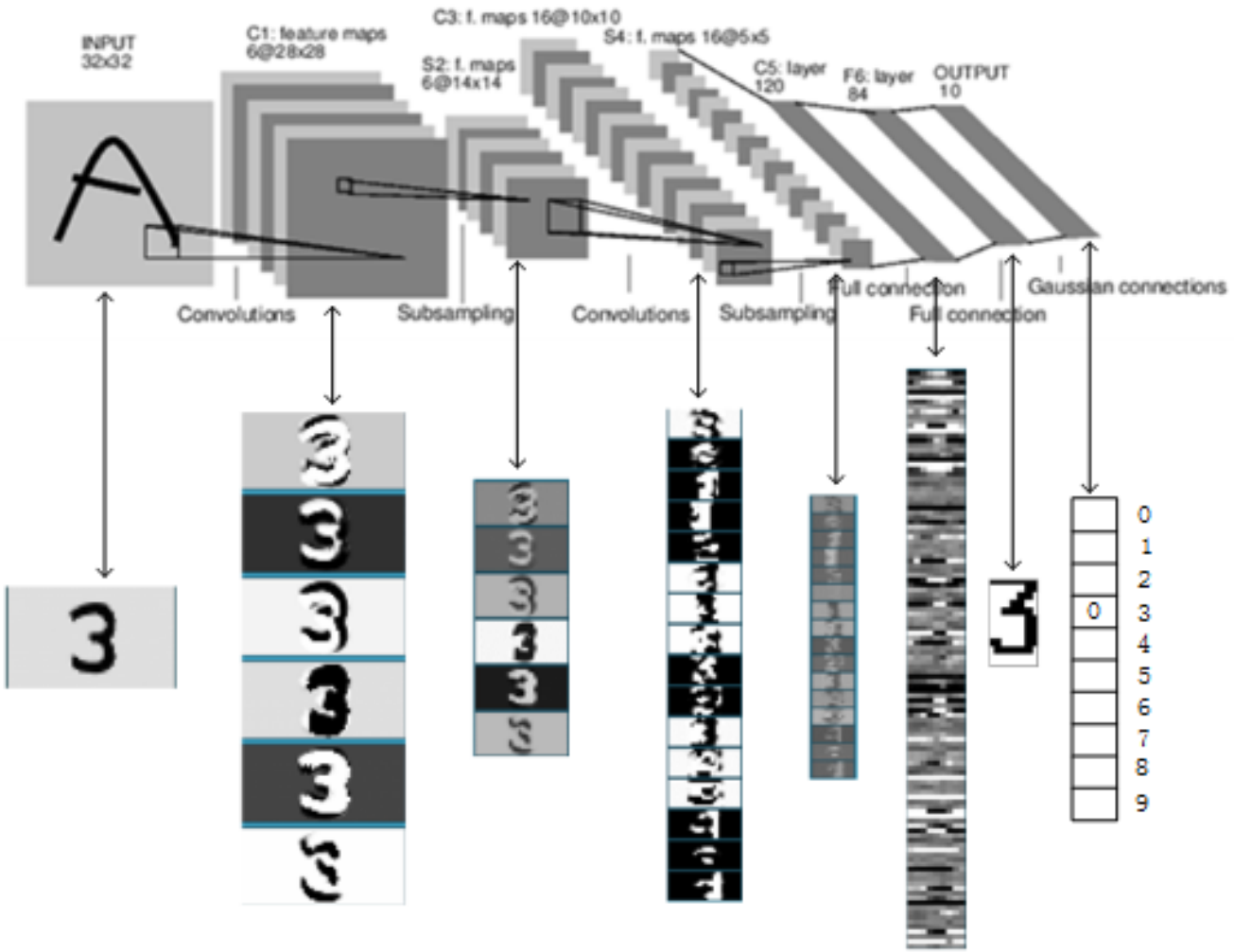
Dense Nets

[*intermediate net in TensorFlow* notebook] [*deep net in TensorFlow* notebook]

Outline

LeNet-5

LeCun et al. (1998)



LeNet-5

LeCun et al. (1998)

[*LeNet in TensorFlow* notebook]

Outline

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

- use layers (Keras) by default
- autodiff Eager execution by default
- JIT compiler for optimization, especially across devices
- subclassing for unlimited flexibility, e.g.:
  - custom loss, optimizers, layers, training loops
  - repeating layers, blocks of layers
- data pipelines & processing with tf.data & tf.io
- portability with:
  - *TensorFlow Serving*
  - *TensorFlow Lite* for mobile/embedded
  - *TensorFlow.js*
- [to update from TF 1.x]

Outline

PyTorch	TensorFlow
"NumPy", optimized for GPUs	ported to Python from C++
dynamic auto-differentiation (autodiff)	static computational graph
debugging is easier	
fast.ai API	Keras API
	more widely adopted
TorchScript Just-In-Time compilation	TensorFlow Serving, .js, Lite, tf.data, tf.io
better for interactively building models	**better for production deployments**

[shallow net in PyTorch]
[deep net in PyTorch]

Outline

Outline

Improving

Your Deep Learning Project IV

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Improving

Your Deep Learning Project IV

1. Splitting your data
   - training set (80% — for optimizing parameters)
   - validation set (10% — for hyperparameters)
   - test set (10% — don't touch yet!)
2. Building and assessing architecture
   - get above chance (simplifying problem, if necessary)
   - do existing performance benchmarks exist?
   - if not, use a simple architecture as benchmark
3. Improving performance & tuning hyperparameters...

Outline

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

1. Parameter initialization
2. Cost function selection
3. Get above chance
4. Layers
   - Type
   - Number
   - Width
5. Avoid overfitting
   - Dropout
   - Data augmentation
   - Batch normalization
   - Early stopping / loading low-loss weights
6. Learning rate
7. Batch size

Outline

GANs

Deep RL