

INTRODUCTION TO DEEP LEARNING

Yogesh Kulkarni

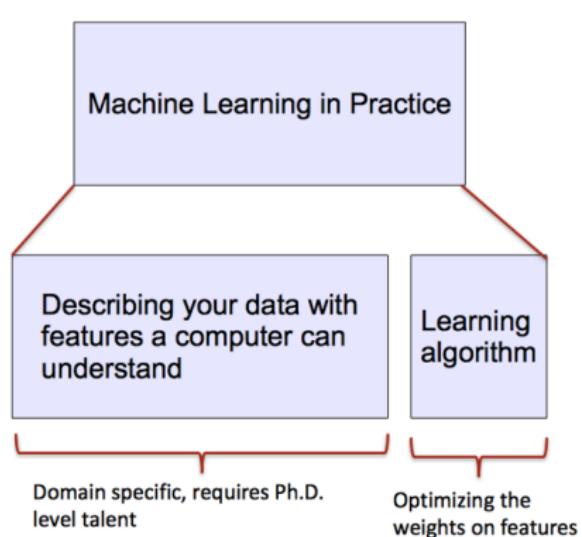
February 18, 2020

Introduction to Deep Learning

What is Deep Learning?

- ▶ Artificial Intelligence: mimicking human intelligence
- ▶ Machine Learning: Automating Learning with features.
- ▶ ML: human-designed representations and input features. So, its just optimizing weights to best make a final prediction
- ▶ There could be programmed (hand coded) AI, that's not Machine Learning
- ▶ Machine Learning could be for non AI activities, like automation
- ▶ Deep Learning: Neural network with no input features

ML vs DL: What's the difference?



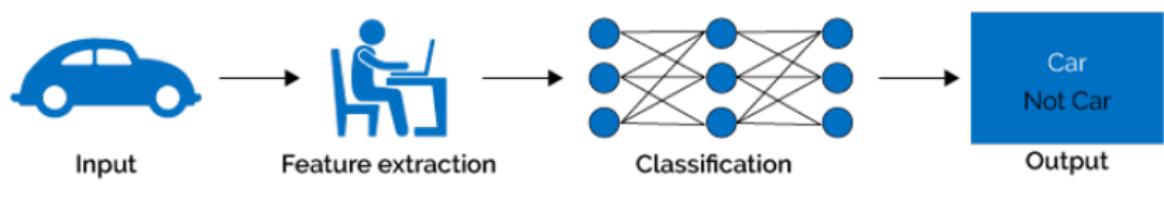
Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

(Reference: Introduction to Deep Learning - Ismini Lourentzou)

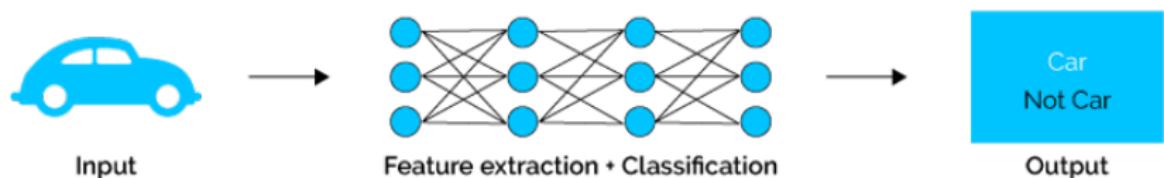
ML vs DL: What's the difference?

Deep learning algorithms attempt to learn (multiple levels of) representation by using a hierarchy of multiple layers

Machine Learning

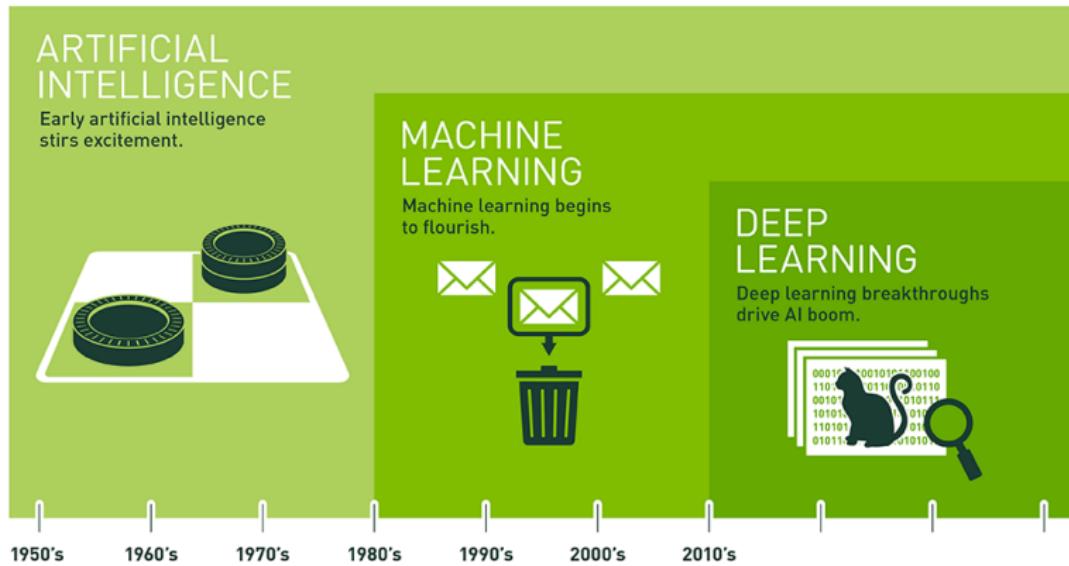


Deep Learning



(Reference: <https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png>)

AI ML DL: What's the difference?



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

(Reference: The Difference Between AI, Machine Learning, and Deep Learning - NVIDIA Blog)

Use Deep Learning When ...

- ▶ You have lots of data (about 10k+ examples)
- ▶ The problem is “complex” - speech, vision, natural language
- ▶ The data is unstructured
- ▶ You need the absolute “best” model

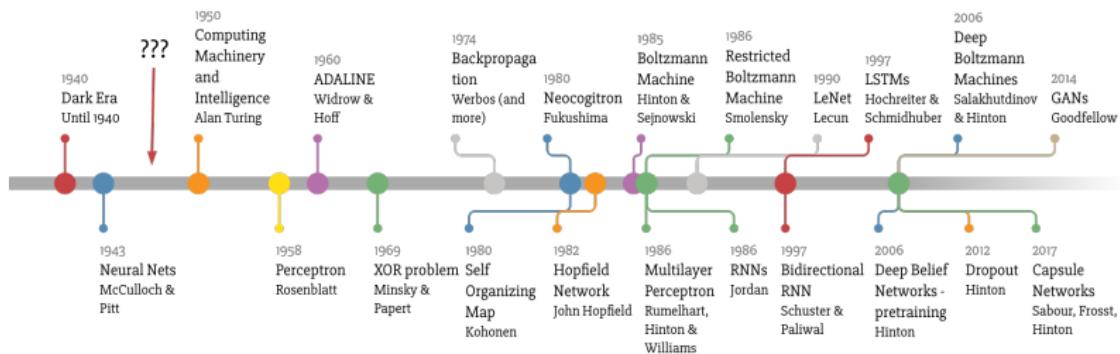
(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Don't use Deep Learning When . . .

- ▶ You don't have a large dataset
- ▶ You are performing sufficiently well with traditional ML methods
- ▶ Your data is structured and you possess the proper domain knowledge
- ▶ Your model should be explainable

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

History



Made by Favio Vázquez

(Reference: Deep Learning basics - Rodrigo Agundez)

History

- ▶ 1958 - Perceptron unit - Frank Rosenblatt
- ▶ 1986 - Backpropagation - Geoffrey Hinton
- ▶ 1986 - RNN - Schuster & Paliwal
- ▶ 1989 - LeNet Backpropagation to multi-layer perceptron - Yan LeCun
- ▶ 1997 - LSTM - Sepp Hochreiter and Jürgen Schmidhuber
- ▶ 1998 - LeNet-5 Convolutional neural networks - Yan LeCun
- ▶ 2007 - Fei Fei Li Princeton ImageNet competition
- ▶ 2009 - GPU for deep learning - Andrew Ng
- ▶ 2011 - Demonstration of ReLU for deep neural networks - Yoshua Bengio
- ▶ 2012 - AlexNet wins ImageNet 25% to 16% error
- ▶ 2012 - Dropout technique - Geoffrey Hinton
- ▶ 2014 - Generative adversarial networks - Ian Goodfellow & Yoshua Bengio
- ▶ 2015 - CNN beats human error in ImageNet 5% to 3%
- ▶ 2016 - AlphaGo - Google DeepMind
- ▶ 2016 - Detectic cancer beats human pathologist .96 vs 0.99 AUC
- ▶ 2017 - Capsule networks - Geoffrey Hinton

History

“



{

There are many interesting recent development in deep learning...The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This, and the variations that are now being proposed, is the most interesting idea in the last 10 years in ML.

Yann LeCun

}

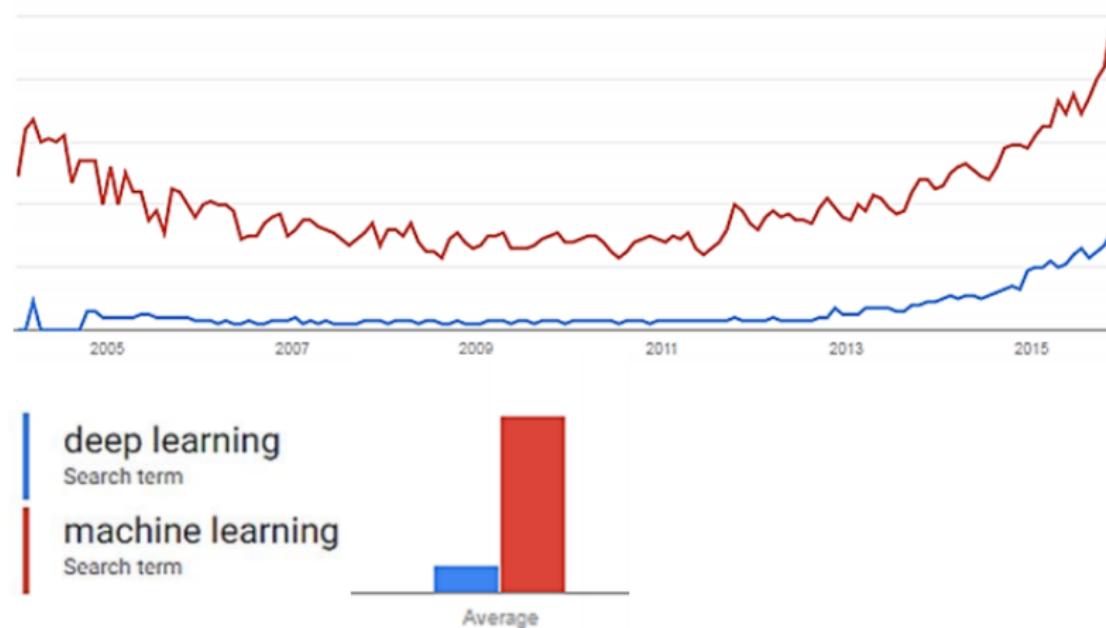
(Reference: Deep Learning basics - Rodrigo Agundez)

Why is DL useful?

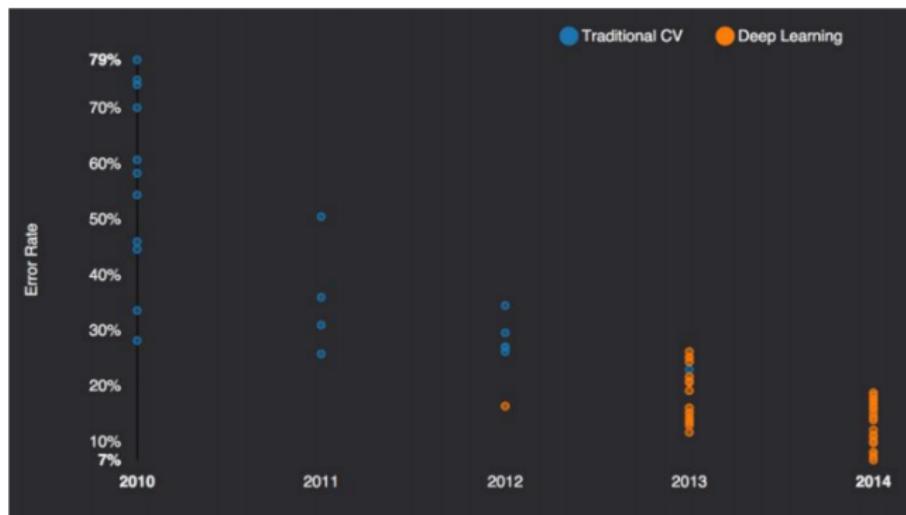
- ▶ ML features could be over-specified, incomplete and take longer to design
- ▶ DL “invents” features.
- ▶ Learned Features are easy to adapt, fast to learn
- ▶ Deep learning provides a very flexible, (almost?) universal, learnable framework for representing world, visual and linguistic information

Why is DL useful?

- ▶ In 2010 DL started outperforming other ML techniques
- ▶ First in speech and vision, then NLP

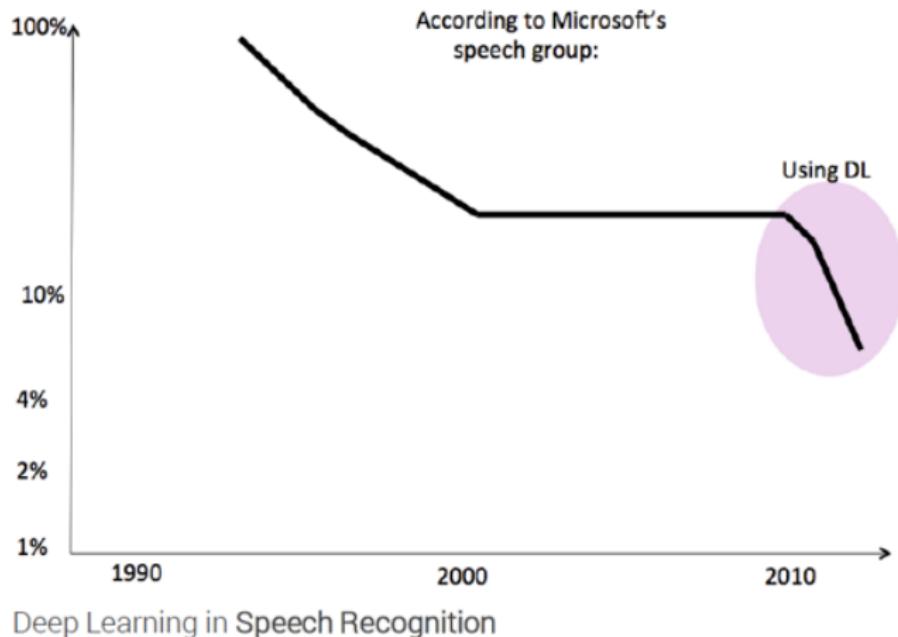


Big Break-through in Vision



(Reference: Introduction to Deep Learning - Ismini Lourentzou)

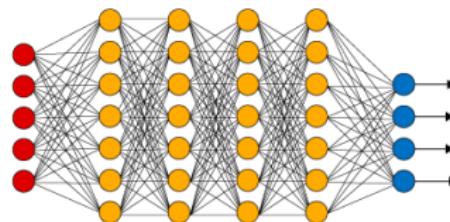
Big Break-through in Speech



(Reference: Introduction to Deep Learning - Ismini Lourentzou)

Deep Learning == Neural Nets

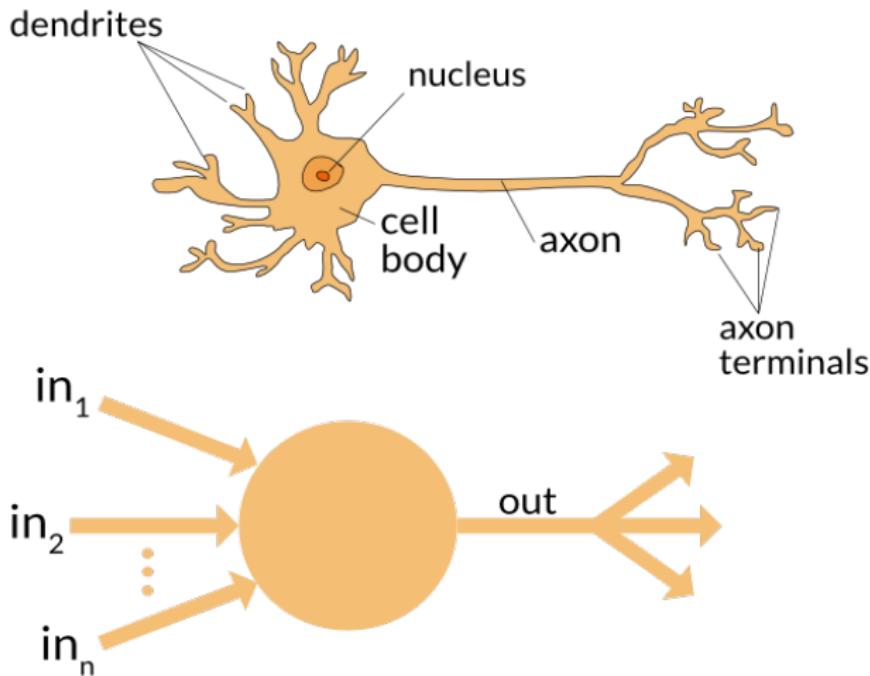
- ▶ Main idea of deep learning: transform the input space into outputs via higher level abstractions.
- ▶ Neural Net architectures are made up of perceptrons (similar to neurons)
- ▶ Each neuron carries certain transformations on inputs coming to it.
- ▶ Collection of such neurons with various types of transformations, can create desired overall transformation.



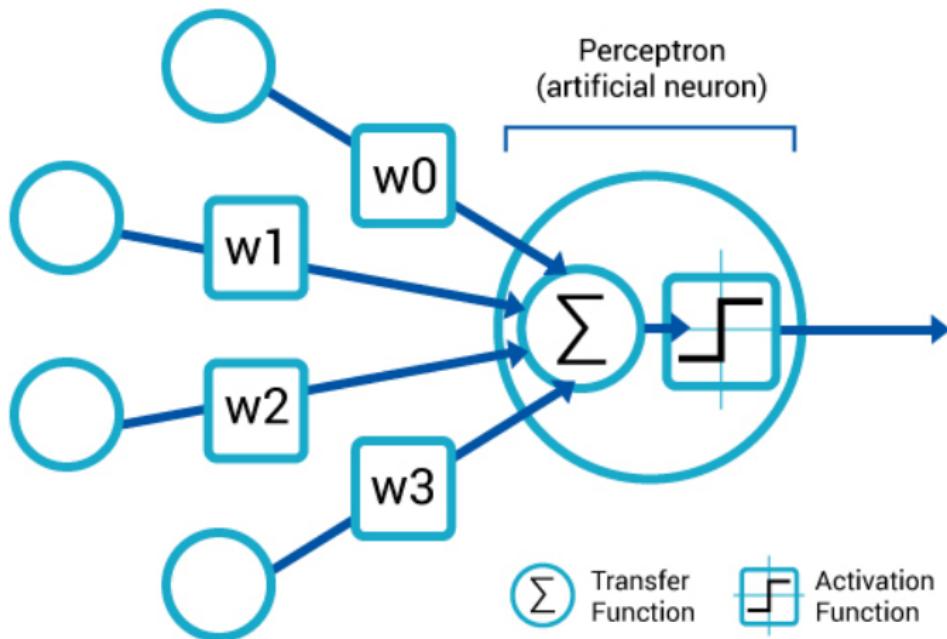
(Reference: Deep Learning basics - Rodrigo Agundez)

Deep Learning == Neural Nets

First artificial neuron proposed in 1943!



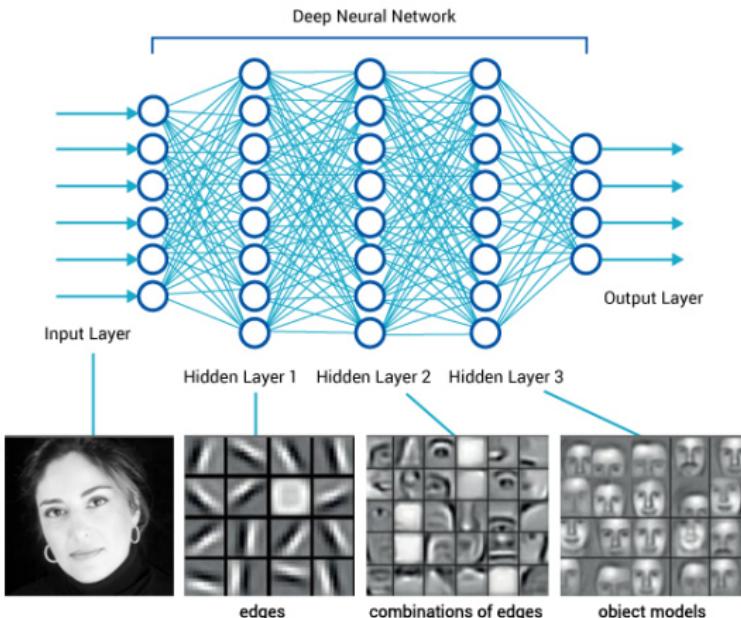
Artificial neuron



(Reference: Deep Learning basics - Rodrigo Agundez)

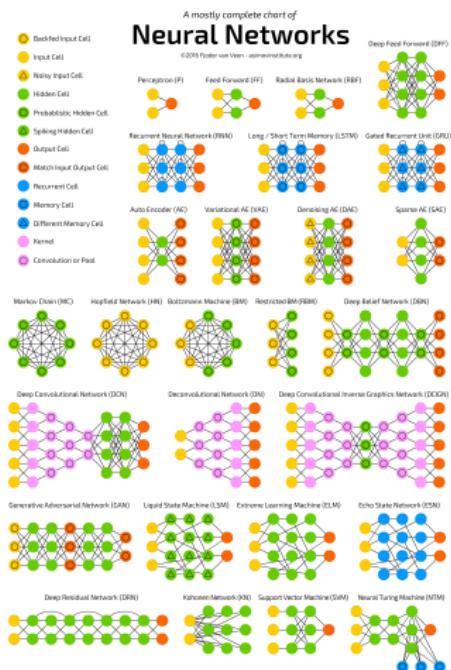
Layers

Hierarchical feature representations



(Reference: Deep Learning basics - Rodrigo Agundez)

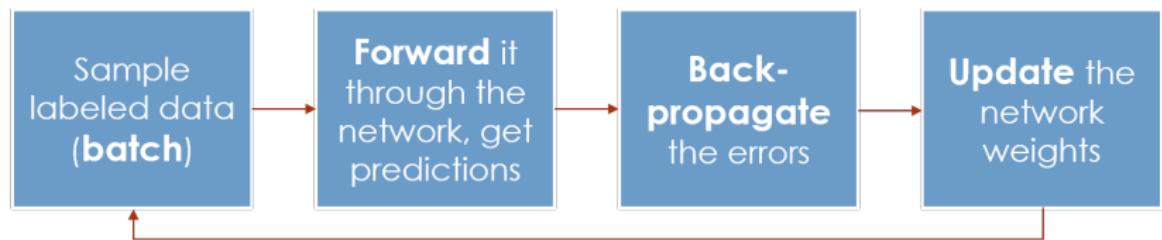
Neural Networks



(Reference: Deep Learning basics - Rodrigo Agundez)

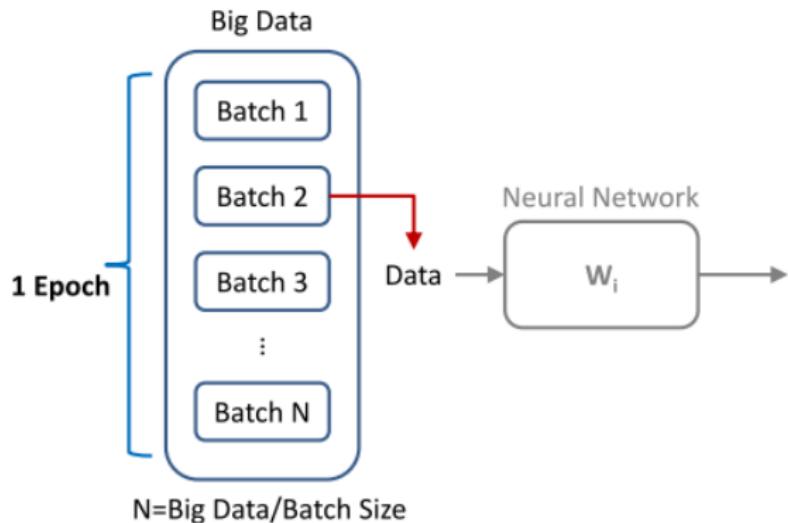
Neural Networks Training Process: Non Mathematical

Training Process: Non Mathematical



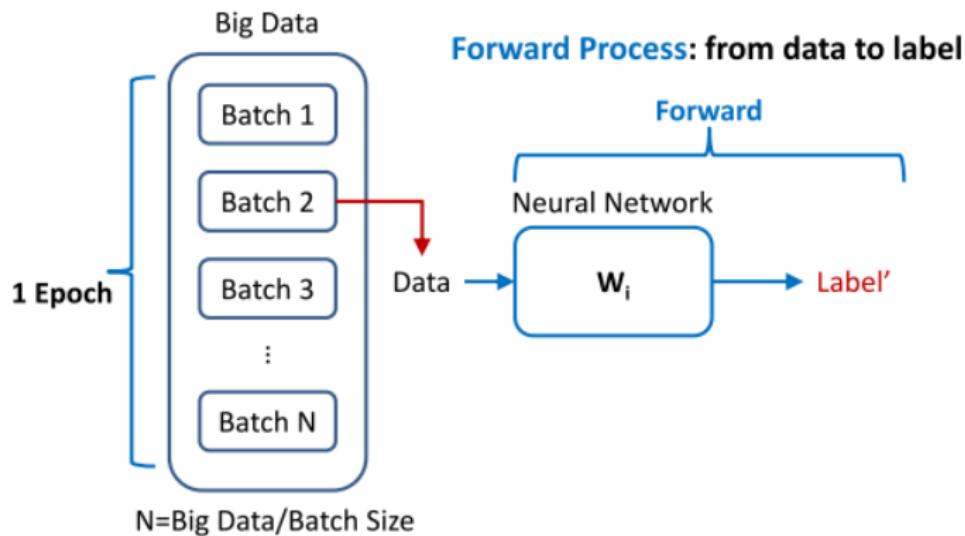
(Reference: Introduction to Deep Learning - Ismini Lourentzou)

Data Enters



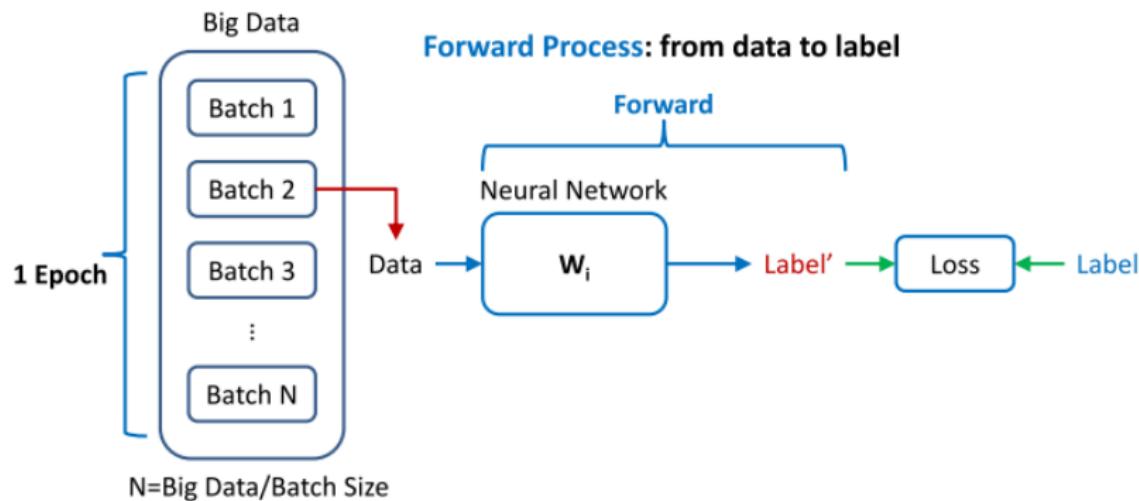
(Reference:PyTorch Tutorial-NTU Machine Learning Course-Lyman Lin)

Forward Pass



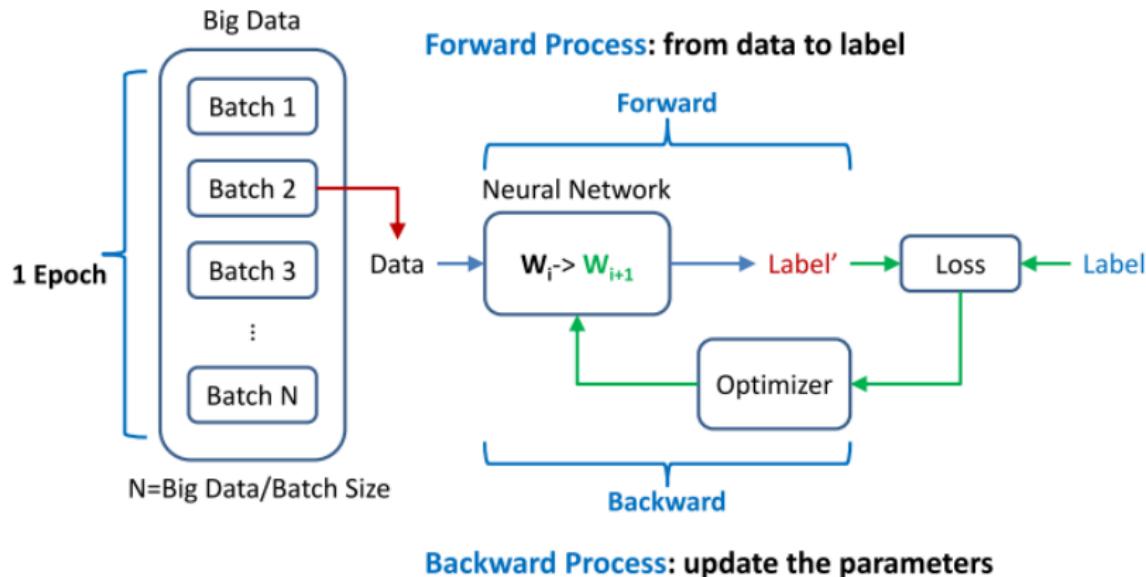
(Reference:PyTorch Tutorial-NTU Machine Learning Course-Lyman Lin)

Loss Calculations



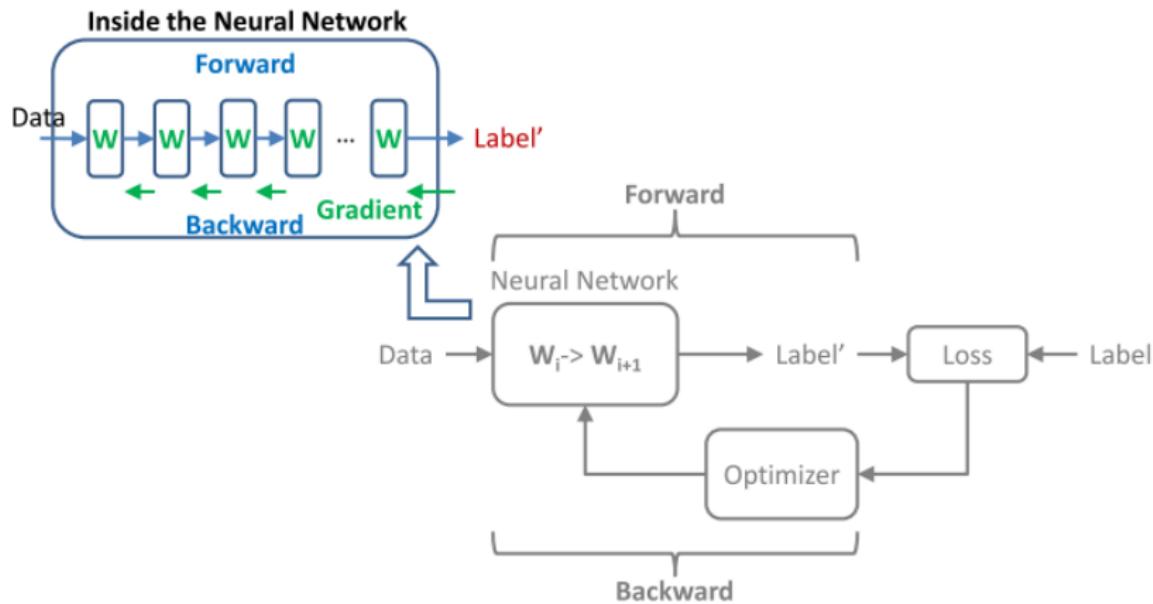
(Reference:PyTorch Tutorial-NTU Machine Learning Course-Lyman Lin)

Back Propagation



(Reference:PyTorch Tutorial-NTU Machine Learning Course-Lyman Lin)

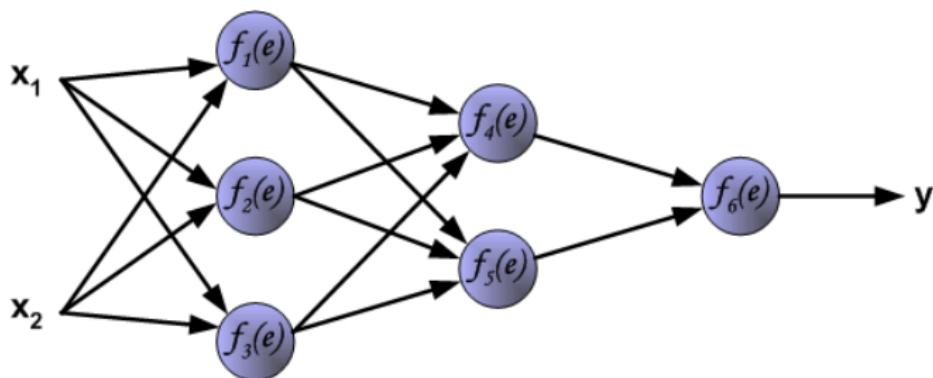
Overall Process



(Reference:PyTorch Tutorial-NTU Machine Learning Course-Lyman Lin)

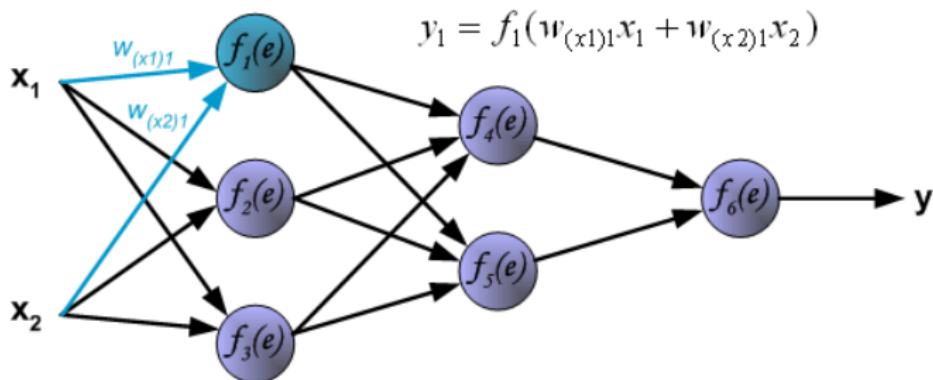
Neural Networks Training Process: Mathematical

Start



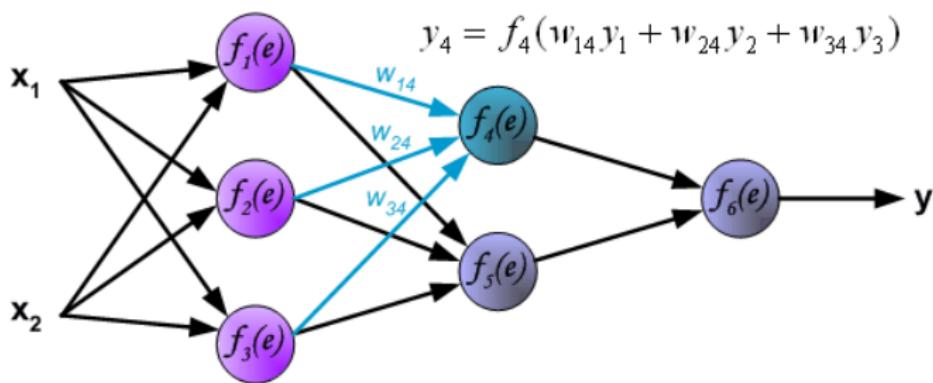
(Reference: Deep Learning basics - Rodrigo Agundez)

Forward pass



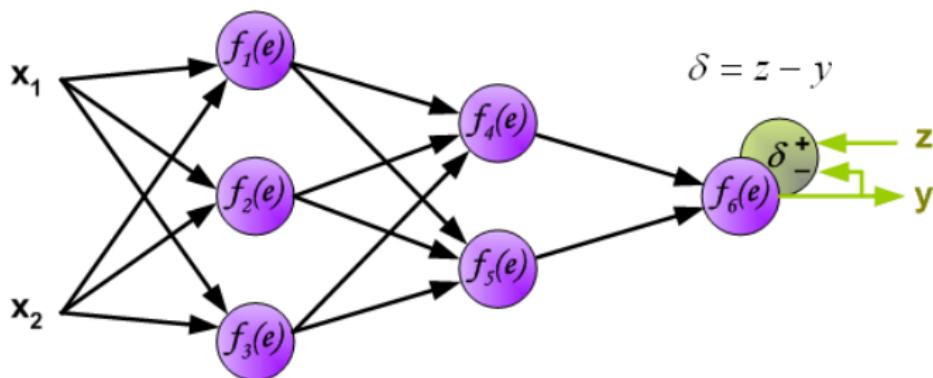
(Reference: Deep Learning basics - Rodrigo Agundez)

Forward pass



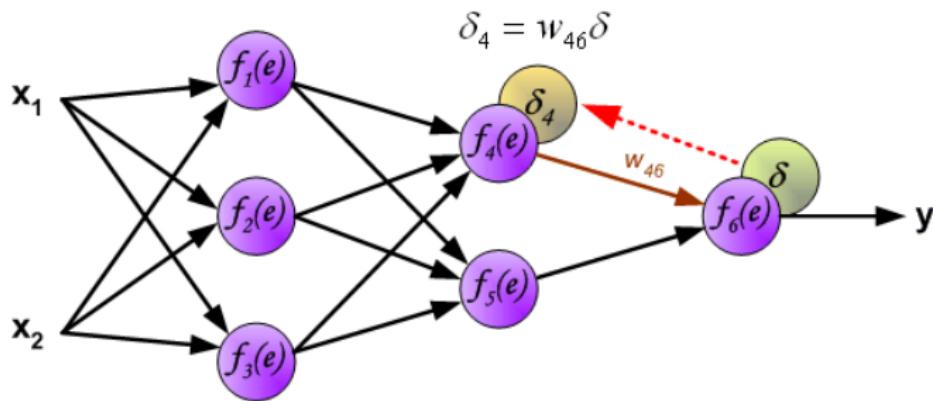
(Reference: Deep Learning basics - Rodrigo Agundez)

Forward pass



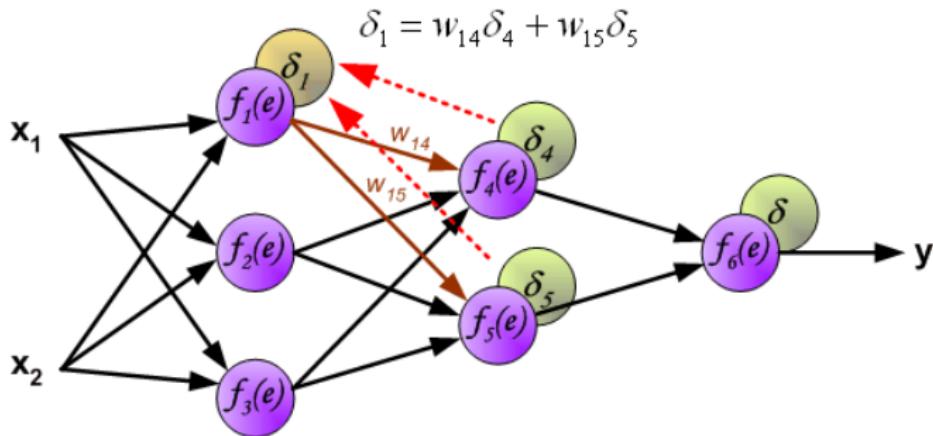
(Reference: Deep Learning basics - Rodrigo Agundez)

Backpropagation



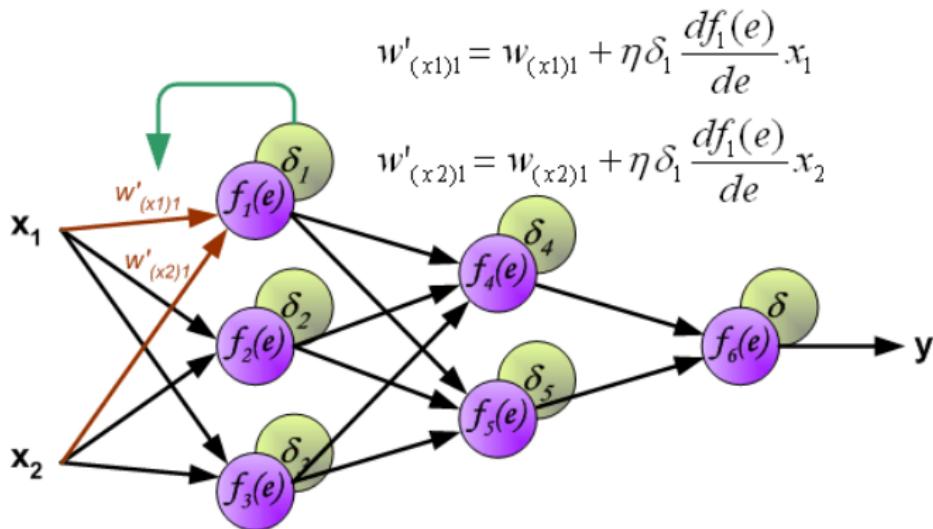
(Reference: Deep Learning basics - Rodrigo Agundez)

Backpropagation



(Reference: Deep Learning basics - Rodrigo Agundez)

Backpropagation

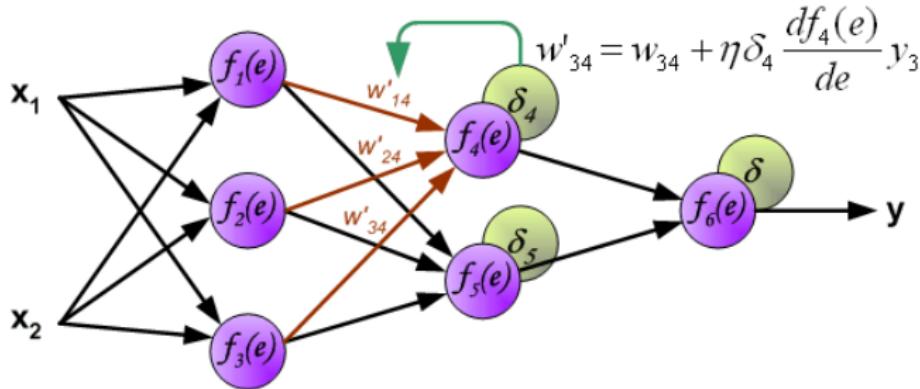


(Reference: Deep Learning basics - Rodrigo Agundez)

Backpropagation

$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

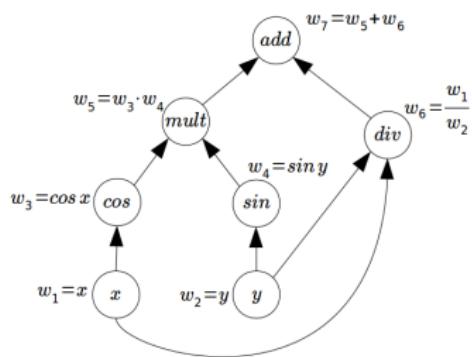
$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$



(Reference: Deep Learning basics - Rodrigo Agundez)

Backpropagation

- ▶ Starts at the end of the net and tunes each layer using the gradient of the loss function. Repeatedly applies the chain rule.
- ▶ Numeric approximation: $f'(x) \approx \frac{f(x+h) - f(x)}{h}$
- ▶ Symbolic differentiation: Symbolic, exact representation of the derivative.
- ▶ Reverse automatic differentiation

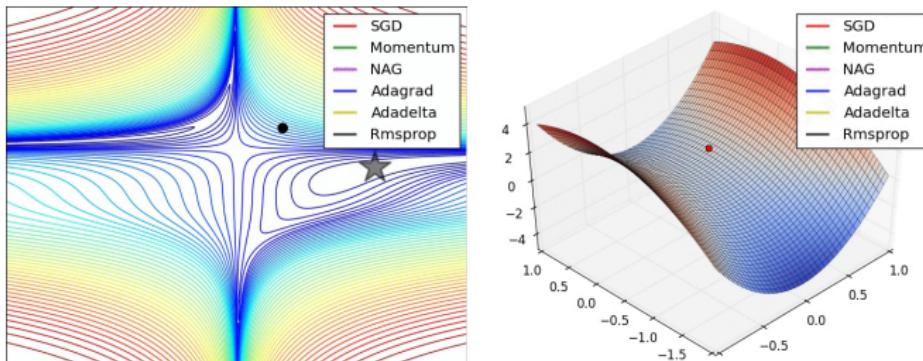


$$\begin{aligned}
 \frac{\partial w_7}{\partial w_1} &= \frac{\partial w_3}{\partial w_1} \frac{\partial w_7}{\partial w_3} + \frac{\partial w_6}{\partial w_1} \frac{\partial w_7}{\partial w_6} = -\sin w_1 \frac{\partial w_7}{\partial w_3} + \frac{1}{w_2} \frac{\partial w_7}{\partial w_6} \\
 \frac{\partial w_7}{\partial w_2} &= \frac{\partial w_4}{\partial w_2} \frac{\partial w_7}{\partial w_4} + \frac{\partial w_6}{\partial w_2} \frac{\partial w_7}{\partial w_6} = \cos w_2 \frac{\partial w_7}{\partial w_4} - \frac{w_1}{w_2^2} \frac{\partial w_7}{\partial w_6} \\
 \frac{\partial w_7}{\partial w_3} &= \frac{\partial w_5}{\partial w_3} \frac{\partial w_7}{\partial w_5} = w_4 \frac{\partial w_7}{\partial w_5} \\
 \frac{\partial w_7}{\partial w_4} &= \frac{\partial w_5}{\partial w_4} \frac{\partial w_7}{\partial w_5} = w_3 \frac{\partial w_7}{\partial w_4} \\
 \frac{\partial w_7}{\partial w_5} &= 1 \\
 \frac{\partial w_7}{\partial w_6} &= 1
 \end{aligned}$$

(Reference: Deep Learning basics - Rodrigo Agundez)

Optimization by backpropagation

- ▶ Loss/cost function
- ▶ Gradient descent



(Reference: Deep Learning basics - Rodrigo Agundez)

Other Aspects

Challenges of Deep Learning

- ▶ Explain-ability - How do you learn what you learn?
- ▶ Debugging - What has gone wrong?
- ▶ Why is this not converging?

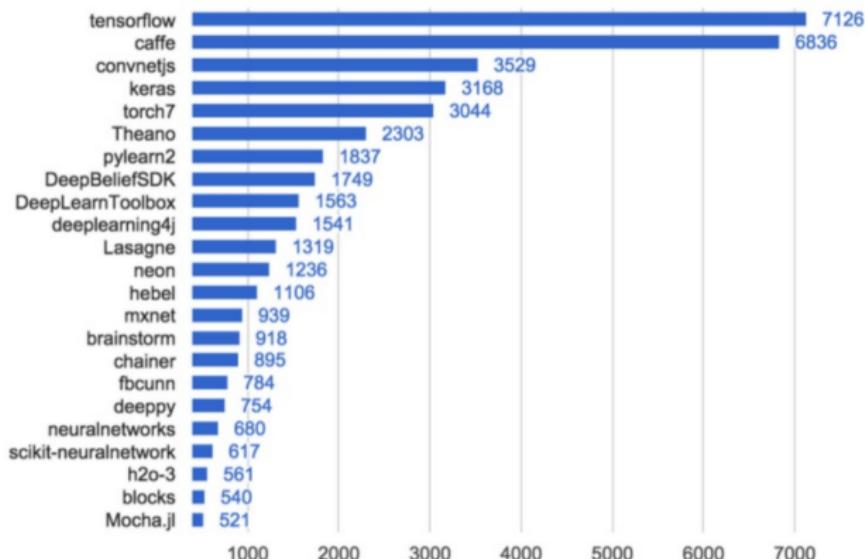
(Reference: AI and Deep Learning - Subrat Panda)

Usage Requirements

- ▶ Large data set with good quality (input-output mappings)
- ▶ Measurable and describable goals (define the cost)
- ▶ Enough computing power (AWS GPU Instance)
- ▶ Excels in tasks where the basic unit (pixel, word) has very little meaning in itself, but the combination of such units has a useful meaning.

(Deep Learning - The Past, Present and Future of Artificial Intelligence - Lukas Masuch)

Deep Learning Tools



(Reference: Introduction to Deep Learning - Ismini Lourentzou)

Deep Learning Outlook

- ▶ Significant advances in deep reinforcement and unsupervised learning
- ▶ Bigger and more complex architectures
- ▶ Harder problems being attempted

Introduction to TensorFlow 2.0

TensorFlow is

- ▶ Open source, Free library, with Python bindings, by Google Brain team
- ▶ Other libraries are: Caffe (Berkeley), Torch (Facebook), Cntk (Microsoft),
- ▶ Can deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- ▶ Flexibility: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- ▶ Till 2019 Keras was popular as a separate library (with back-end as Tensorflow) but with Tensorflow 2.0, Keras has become its default front end API.
- ▶ TensorFlow 2.0 merges keras as "tf.keras". It allows you to design, fit, evaluate deep learning models.

Open Source Community

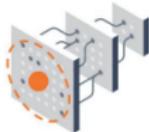
41,000,000+ 69,000+ 12,000+ 2,200+

downloads commits pull requests contributors

As of Oct 2019 ...

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Tensorflow 2.0



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

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Deploy Anywhere

Servers



Edge devices



JavaScript



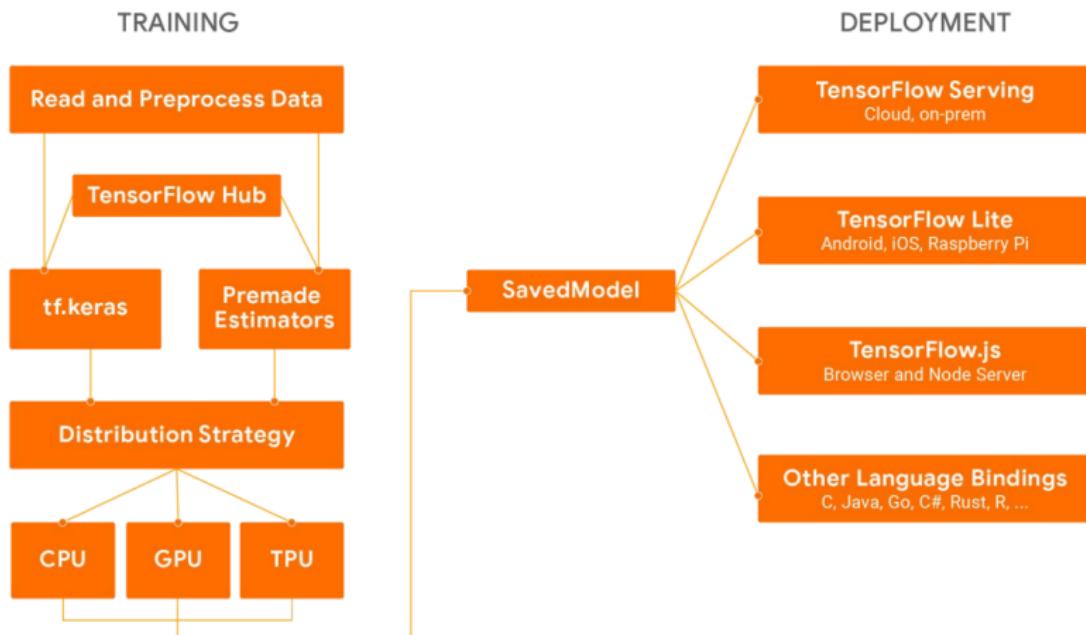
TensorFlow
Extended

TensorFlow
Lite

TensorFlow
.JS

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Training and Deployment



(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Ecosystem/Verticals

TF Probability

TF Agents

Tensor2Tensor

TF Ranking

TF Text

TF Federated

TF Privacy

...

Ecosystem/Verticals

```
1 import tensorflow as tf # Assuming TF 2.0 is installed
2 a = tf.constant([[1, 2],[3, 4]])
3 b = tf.matmul(a, a)
4 print(b)
5 # tf.Tensor( [[ 7 10] [15 22]], shape=(2, 2), dtype=int32)
6 print(type(b.numpy()))
7 # <class 'numpy.ndarray'>
```

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Compared to TF 1.0

What's Gone

- ▶ `Session.run`
- ▶ `tf.control_dependencies`
- ▶ `tf.global_variables_initializer`
- ▶ `tf.cond`, `tf.while_loop`
- ▶ `tf.contrib`
(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

What's New

- ▶ Eager execution by default
- ▶ `tf.function`
- ▶ Keras as main high-level api

tf.keras



(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Installation

- ▶ Have Python installed, such as Python 3.6 or higher.
- ▶ Easy way to install TensorFlow
- ▶ Linux:

```
sudo pip install tensorflow
```

- ▶ Windows:

```
1 pip install tensorflow
```

Installation

The screenshot shows the TensorFlow website's main page. At the top, there is a navigation bar with links for "Install", "Learn", "API", "Resources", "Community", and "Why TensorFlow". A search bar and a "GitHub" link are also present. The main content area features a large orange banner with the text "An end-to-end open source machine learning platform". Below the banner, there are four tabs: "TensorFlow" (which is selected), "For JavaScript", "For Mobile & IoT", and "For Production". A text box below the tabs states: "The core open-source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser." A prominent "Get started with TensorFlow" button is located in this box. To the right of the text box is a complex 3D-style illustration depicting various machine learning applications. It shows a laptop displaying a neural network diagram, a smartphone with a camera icon, a server tower, a car, a speech bubble, a leaf, and an airplane, all interconnected by a network of lines and nodes.

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Installation Check

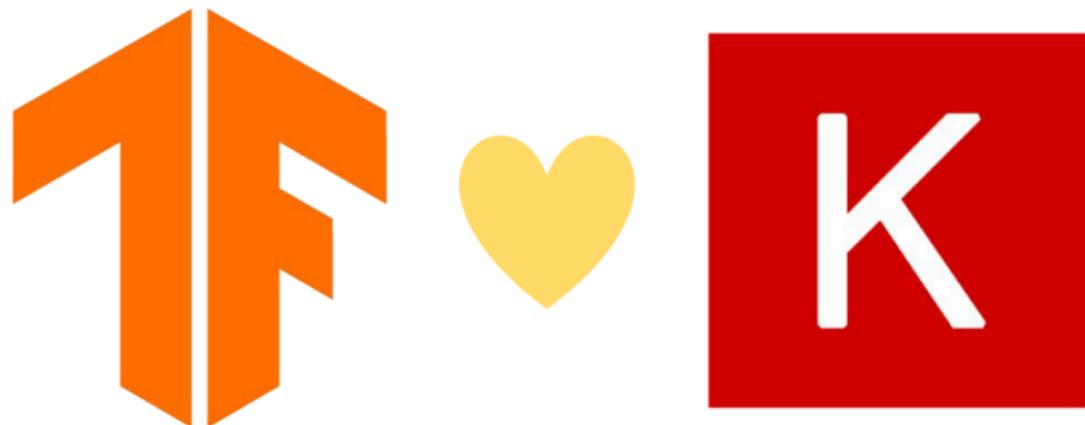
- ▶ Confirm the installation by:

```
1 # check version
  import tensorflow
3 print(tensorflow.__version__)
```

- ▶ It must be 2.0 onwards
- ▶ If you get warning like below, Don't worry, just IGNORE.

```
1 Your CPU supports instructions that this TensorFlow binary was not
  compiled to use: AVX2 FMA
  XLA service 0x7fde3f2e6180 executing computations on platform Host.
  Devices:
3 StreamExecutor device (0): Host, Default Version
```

tf.keras



(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Keras and tf .keras

- ▶ Fast prototyping, advanced research, and production
- ▶ keras.io = reference implementation `import keras`
- ▶ tf .keras = TensorFlow's implementation (a superset, built-in to TF, no need to install Keras separately) `from tensorflow import keras`

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Steps to use tf.Keras

- ▶ Define the model.
- ▶ Compile the model.
- ▶ Fit the model.
- ▶ Evaluate the model.
- ▶ Make predictions.

Define the Model

- ▶ First, select the type of the model.
- ▶ Choose architecture or network topology.
- ▶ Meaning, define layers, its parameters.
- ▶ There are multiple API ways to define the model (will look at later)

```
...
2 # define the model
model = ...
```

Compile the Model

- ▶ Select loss function that you want to optimize, eg Cross Entropy or Mean Squared Error
- ▶ Select Optimization method, eg Adam, Stochastic Gradient Descent
- ▶ Select performance metrics to be used during Training

```
1 ...
2 # compile the model
3 opt = SGD(learning_rate=0.01, momentum=0.9)
4 model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

Fit the Model

- ▶ Select Training configuration (epochs, batch size, etc)
- ▶ *Epochs: number of full cycles (forward + backward) during training
- ▶ *Batch Size: Number of samples used to estimate model error, or update the weights
- ▶ Can take minutes to hours to days depending on complexity, hardware, training samples size.
- ▶ Progress bar shows status of each epoch, performance, etc.

```
2 ...
# fit the model
model.fit(X, y, epochs=100, batch_size=32)
```

Evaluate the Model

- ▶ Select a holdout dataset (cross validation)
- ▶ This is not used for training but just for evaluation as it has correct answers as well.

```
1 ...
# evaluate the model
3 loss = model.evaluate(X, y, verbose=0)
```

Making Predictions

- ▶ Get Test set for which answers have to be found out.
- ▶ Better to save the model and later load it to make predictions.
- ▶ May choose to fit a model on all of the available data before you start using it.

```
1 ...  
# make a prediction  
3 yhat = model.predict(X)
```

Model Definition

API styles

- ▶ The Sequential Model
 - ▶ Dead simple
 - ▶ Only for single-input, single-output, sequential layer stacks
 - ▶ Good for 70+% of use cases
- ▶ The functional API
 - ▶ Like playing with Lego bricks
 - ▶ Multi-input, multi-output, arbitrary static graph topologies
 - ▶ Good for 95% of use cases
- ▶ Model subclassing
 - ▶ Maximum flexibility
 - ▶ Larger potential error surface

Sequential Model API (Simple)

- ▶ Called “Sequential” because it involves using Sequential class and adding layers to it one-by-one, in a sequence.
- ▶ E.g. 8 inputs, one hidden layer with 10 nodes, and one output layer with one node to predict numerical value would look:

```
1 # example of a model defined with the sequential api
2 from tensorflow.keras import Sequential
3 from tensorflow.keras.layers import Dense
4 # define the model
5 model = Sequential()
6 model.add(Dense(10, input_shape=(8,)))
7 model.add(Dense(1))
```

Note:

- ▶ Input layer, per say, is NOT added. Its an argument for the first HIDDEN layer.
- ▶ Here ‘input_shape’ of (8,) means one sample/row is of 8 values. And such, many samples/rows can come, so left blank.

Functional Model API (Advanced)

- ▶ Need to explicitly connections between layers.
- ▶ Models may have multiple input/output paths (a word and a number)
- ▶ Input layer needs to be defined explicitly, like:

```
1 x_in = Input(shape=(8,))
```

- ▶ Next, a fully connected layer can be connected to the input by calling the layer and passing the input layer. This will return a reference to the output connection in this new layer.

```
1 x = Dense(10)(x_in)
```

- ▶ Once connected, we define a Model object and specify the input and output layers.

```
1 x_in = Input(shape=(8,))
2 x = Dense(10)(x_in)
3 x_out = Dense(1)(x)
4 # define the model
5 model = Model(inputs=x_in, outputs=x_out)
```

Sub-classing Model API (Very Advanced)

```
1 class MyModel(tf.keras.Model):
2     def __init__(self, num_classes=10):
3         super(MyModel, self).__init__(name='my_model')
4         self.dense_1 = layers.Dense(32, activation='relu')
5         self.dense_2 = layers.Dense(num_classes, activation='sigmoid')
6
7     def call(self, inputs):
8         # Define your forward pass here,
9         x = self.dense_1(inputs)
10        return self.dense_2(x)
```

Understanding deferred (symbolic) vs. eager (imperative)

- ▶ Deferred: Build a computation graph that gets compiled first and then once values are filled, executed later
- ▶ Eager: Model is a python exe, Execution is runtime (like Numpy)
- ▶ Deferred: Symbolic tensors don't have a value in your Python code (yet)
- ▶ Eager: tensors have a value in your Python code
- ▶ Eager: can use value-dependent dynamic topologies (tree-RNNs)

Sample eager execution code

```
lstm_cell = tf.keras.layers.LSTMCell(10)
2
def fn(input, state):
4    return lstm_cell(input, state)

6 input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2
lstm_cell(input, state); fn(input, state) # warm up
8 # benchmark
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
```

Let's make this faster

```
1 lstm_cell = tf.keras.layers.LSTMCell(10)
2
3 @tf.function
4 def fn(input, state):
5     return lstm_cell(input, state)
6
7 input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2
8 lstm_cell(input, state); fn(input, state) # warm up
9 # benchmark
10 timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
```

AutoGraph makes this possible

Say, for a sample function

```
1 @tf.function
2 def f(x):
3     while tf.reduce_sum(x) > 1:
4         x = tf.tanh(x)
5     return x
6 # you never need to run this (unless curious)
7 print(tf.autograph.to_code(f))
```

Generated code

We need not understand this, but still ...

```
1 def tf__f(x):
2     def loop_test(x_1):
3         with ag__.function_scope('loop_test'):
4             return ag__.gt(tf.reduce_sum(x_1), 1)
5     def loop_body(x_1):
6         with ag__.function_scope('loop_body'):
7             with ag__.utils.control_dependency_on_returns(tf.print(x_1)):
8                 tf_1, x = ag__.utils.alias_tensors(tf, x_1)
9                 x = tf_1.tanh(x)
10                return x,
11 x = ag__.while_stmt(loop_test, loop_body, (x,), (tf,))
12 return x
```

tf.distribute.Strategy

For the sample code below ...

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, input_shape=[10]),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Multi-GPU

One of the computations distribution strategy could be ...

```
1 strategy = tf.distribute.MirroredStrategy()
2 with strategy.scope():
3     model = tf.keras.models.Sequential([
4         tf.keras.layers.Dense(64, input_shape=[10]),
5         tf.keras.layers.Dense(64, activation='relu'),
6         tf.keras.layers.Dense(10, activation='softmax')])
7     model.compile(optimizer='adam',
8                     loss='categorical_crossentropy',
9                     metrics=['accuracy'])
```

TensorFlow Datasets

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

More at tensorflow.org/datasets

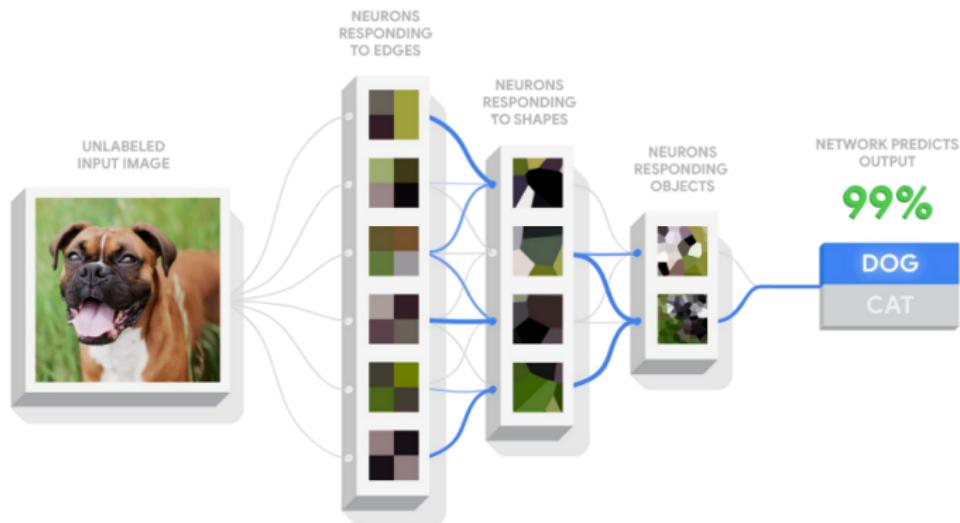
(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Terminologies

In the neural network terminology:

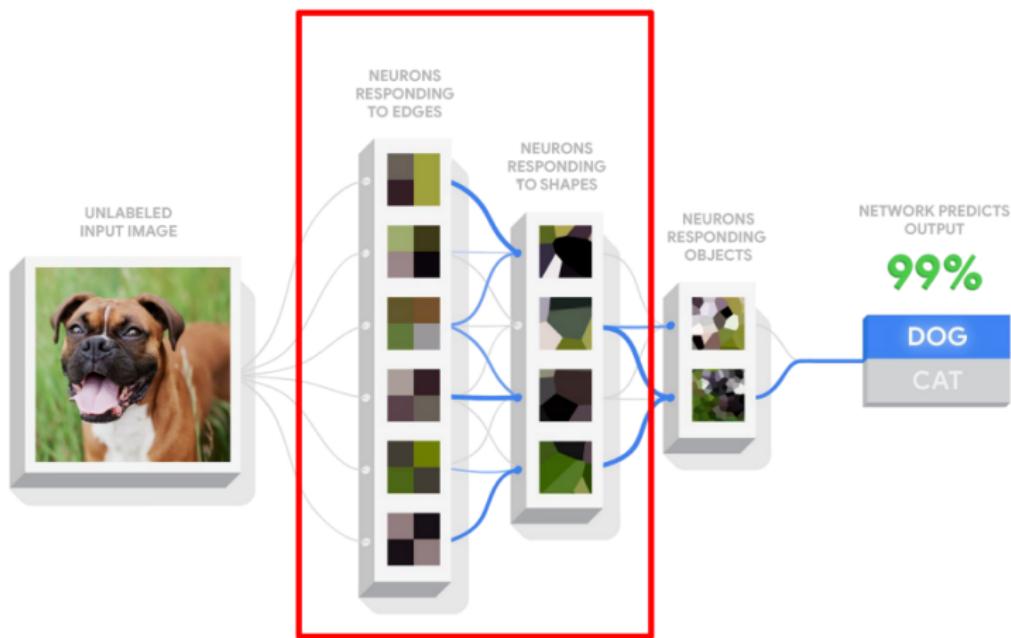
- ▶ one epoch = one forward pass and one backward pass of all the training examples
- ▶ batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- ▶ number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).
- ▶ Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

Transfer Learning



(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Transfer Learning



(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Transfer Learning

```
1 import tensorflow as tf
2 base_model = tf.keras.applications.SequentialMobileNetV2(
3     input_shape=(160, 160, 3),
4     include_top=False,
5     weights='imagenet')
6 base_model.trainable = False
7 model = tf.keras.models.Sequential([
8     base_model,
9     tf.keras.layers.GlobalAveragePooling2D(),
10    tf.keras.layers.Dense(1)
11])
12 # Compile and fit
```

(Ref: Introduction to TensorFlow 2.0 - Brad Miro)

Transfer Learning

≡ TensorFlow Hub 🔍 USER GUIDE

Text Embedding	Text embedding  universal-sentence-encoder By Google text-embedding DAN en Encoder of greater-than-word length text trained on a variety of data.
Image Classification Feature Vector Generator	 nnlm-en-dim128 By Google text-embedding Google News NNLm en Token based text embedding trained on English Google News 200B corpus.
Video Classification	 elmo By Google text-embedding 1 Billion Word Benchmark ELMo en Embeddings from a language model trained on the 1 Billion Word Benchmark.
Publishers Google DeepMind	View more text embeddings Image feature vectors  imagenet/inception_v3/feature_vector By Google image-feature-vector ImageNet (ILSVRC-2012-CLS) Inception V3 Feature vectors of images with Inception V3 trained on ImageNet (ILSVRC-2012-CLS).

References

Many publicly available resources have been refereed for making this presentation. Some of the notable ones are:

- ▶ TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras - Jason Brownlee
- ▶ Deep Learning using Keras- Alyosamah
- ▶ Introduction to Keras - Francois Chollet
- ▶ Michael Nielsen's Neural Networks and Deep Learning: <http://neuralnetworksanddeeplearning.com/>

Thanks ... yogeshkulkarni@yahoo.com