greatlearning
Learning for Life

Week 1

Introduction to neural networks and deep learning

Topics you have covered in week 1 video

- Introduction to neural networks
- Activation functions
  - Types of activation functions
- Feed forward neural networks
  - Basic structure of neural network
  - Function of a neural network
  - Training a neural network
  - Loss function
- Backpropagation and gradient descent
  - Gradient ascent
  - Gradient descent
  - Backpropagation
- Learning rate setting and tuning
  - Decay in learning rate
  - Momentum
- Introduction to TensorFlow
- Introduction to Keras

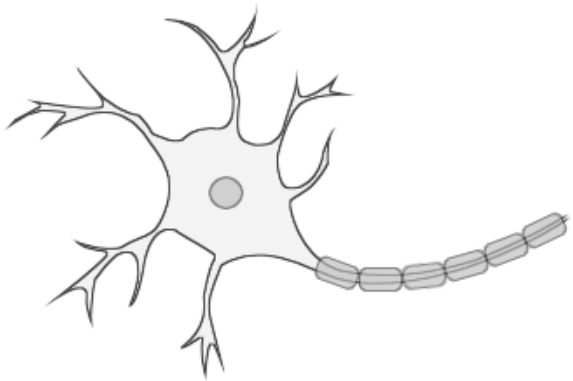
Session agenda

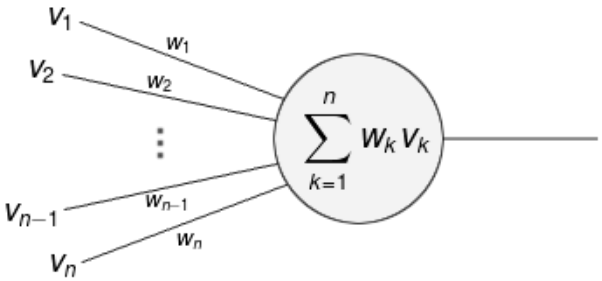
- Introduction to neural networks
- Building blocks of a neural network
- Introduction to TensorFlow
- Introduction to Keras
- Case study
- Questions

Neural networks

Image source: codeproject

Neuron





Artificial neuron is inspired by biological neuron

Image source: hackernoon

Activation function

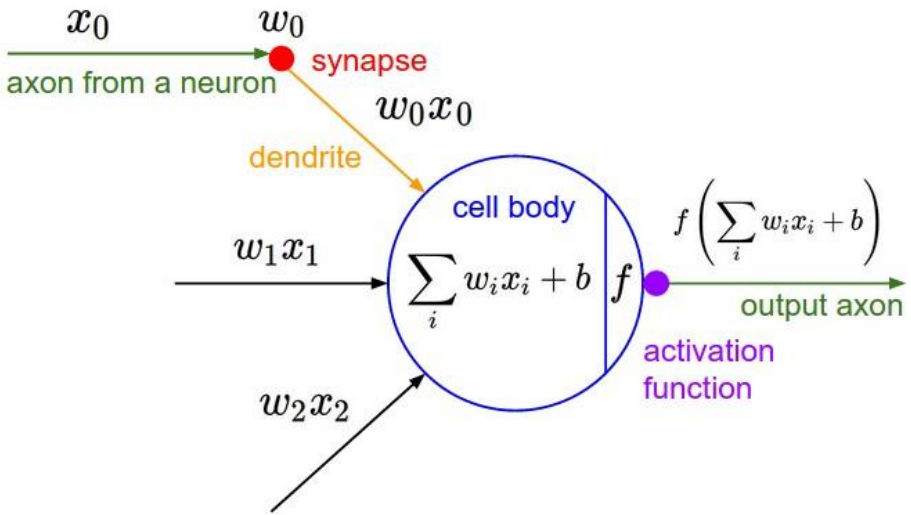
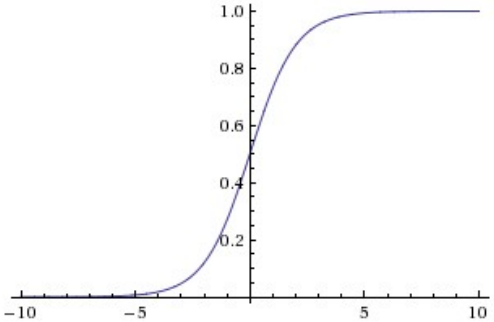
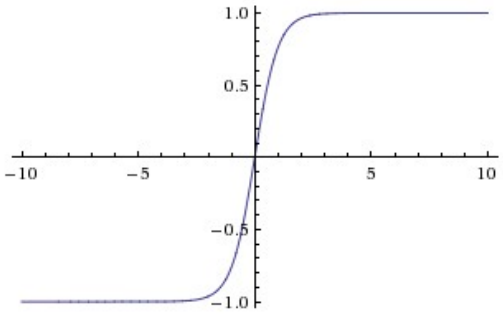


Image source: cs231n stanford

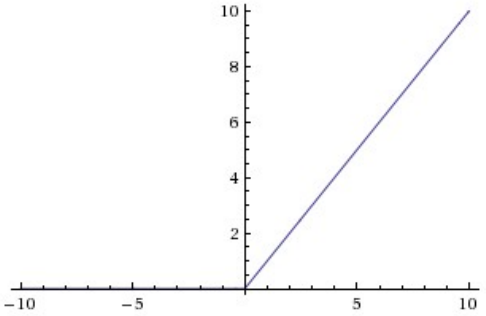
Types of activation function



Sigmoid



Tanh

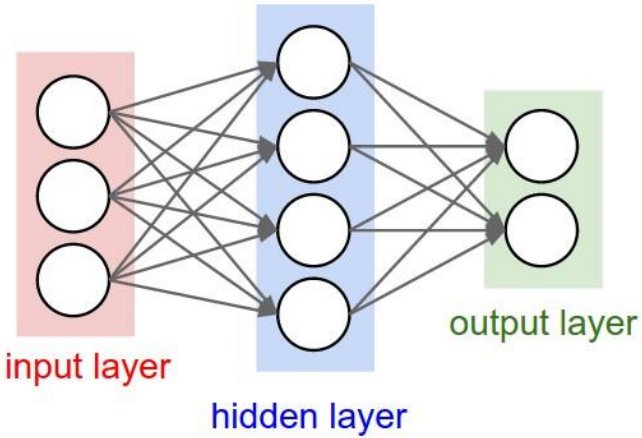


ReLU

Every activation function (or non-linearity) takes a single number and performs certain fixed mathematical operation on it.

Images and text source: cs231n stanford

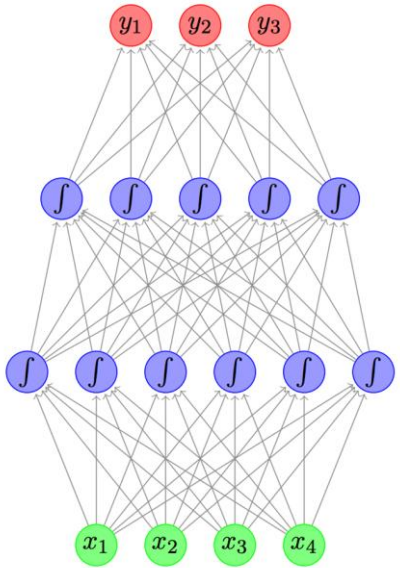
Feed forward neural network



A 2-layer neural network (one hidden layer of 4 neurons (or units) and one output layer with two neurons)

Image and text source: cs231n stanford

Layer details



Output layer

- Represents the output of the neural network
- Most commonly it doesn't have any activation function

Hidden layer(s)

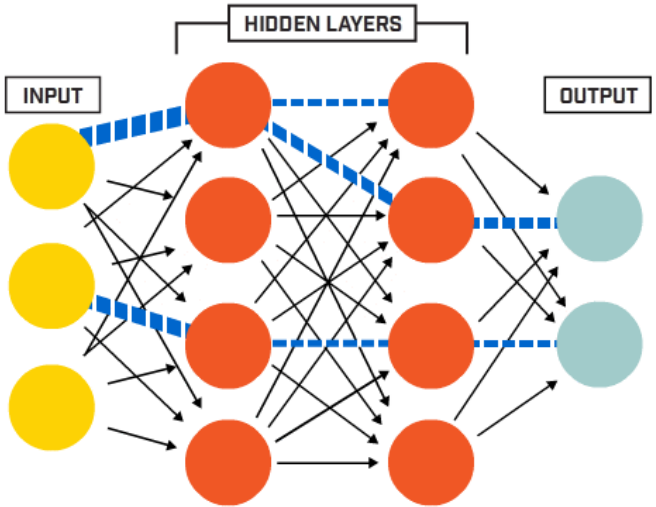
- Represents the intermediary nodes that divide the input space into regions with (soft) boundaries
- Given enough hidden nodes, we can model an arbitrary input-output relation
- It takes in a set of weighted input and produces output through an activation function

Input layer

- Represents dimensions of the input vector (one node for each dimension)

Image source: A Primer on Neural Network Models for Natural Language Processing

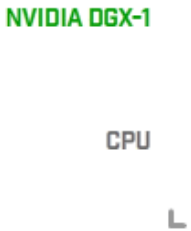
Train a neural network



- Choose hyper parameters
- Choose network design
- Form a neural network
- Compute an estimate value for all samples
- Compute loss
- Reduce loss
- Repeat last three steps

Image source: ptgrey.com

Training on specialized hardware



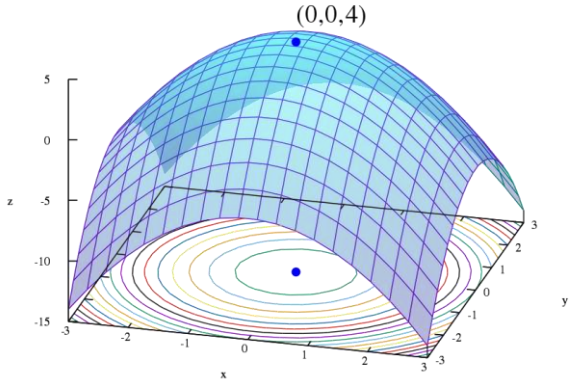
GPU-accelerated training of deep learning systems is much faster than CPU

Image and text source: ptgrey.com

Error and Loss function

- In most learning networks, error is calculated as the difference between the actual output and the predicted output.
- The function that is used to compute this error is known as Loss Function.
- Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model.
- One of the most widely used loss function is mean square error, which calculates the square of difference between actual value and predicted value.
- Different loss functions are used to deal with different type of tasks, i.e. regression and classification.

Text source: xpertup

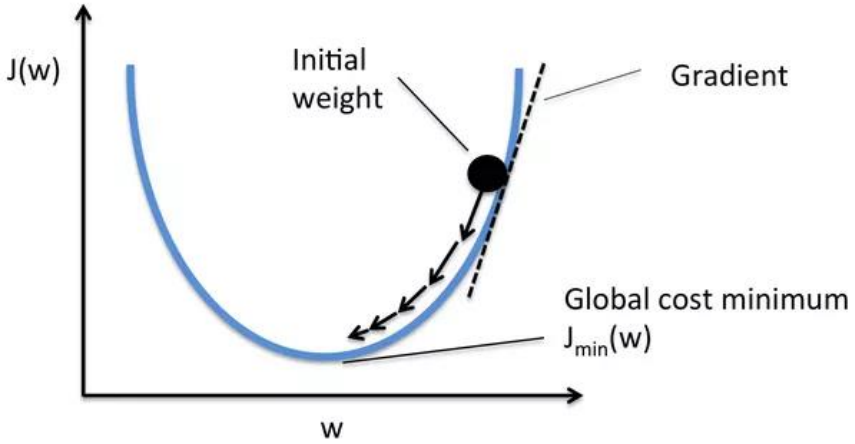


Optimization

The goal of optimization is to find a set of weights that minimizes the loss function

Image source: wikipedia

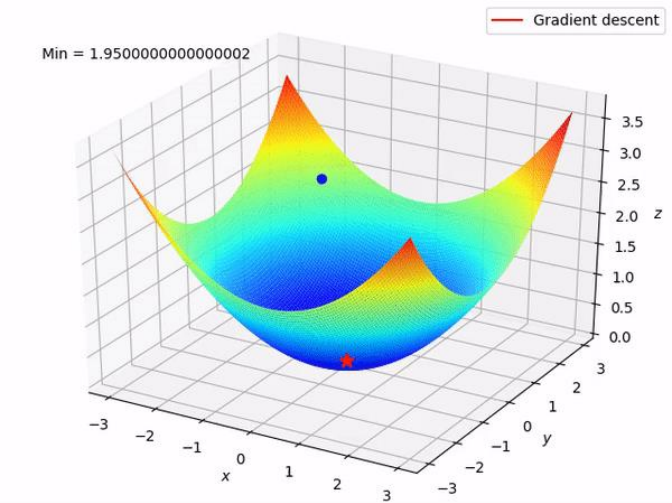
Gradient



Optimisation functions usually calculate the **gradient** i.e. the partial derivative of loss function with respect to weights, and the weights are modified in the opposite direction of the calculated gradient. This cycle is repeated until we reach the minima of loss function.

Image and text source: xpertup

Gradient Descent

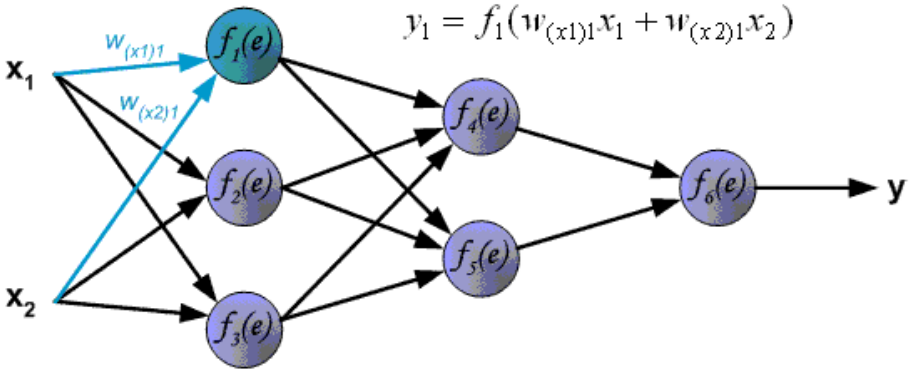


The procedure of repeatedly evaluating the gradient and then performing a parameter update is called Gradient Descent.

Image source: xpertup

Backpropagation

FP



Backpropagation is an efficient method to do gradient descent. It saves the gradient w.r.t the upper layer output to complete the gradient w.r.t the weights immediately below

Image source: webenergeek

Learning rate

- With low learning rate the improvements will be linear
- Higher learning rate can decay the loss faster, but it can get stuck
- Initially keep the learning rate higher
- Later decrease the learning rate

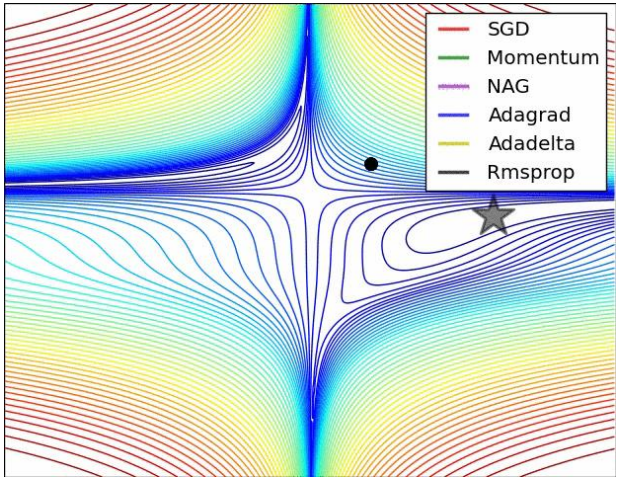
Parameter updation approaches

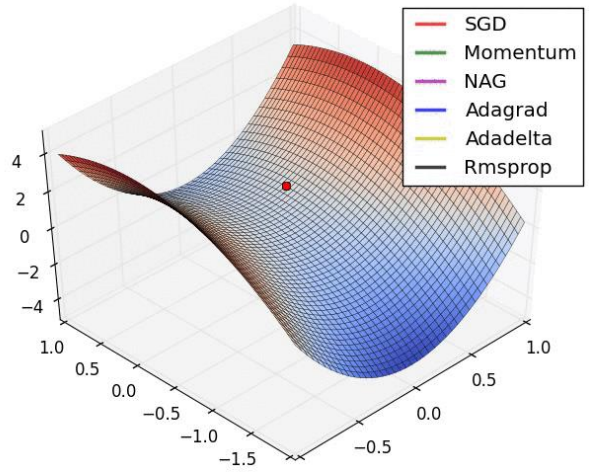
- Vanilla update
- Momentum update
- Nesterov momentum update

Per-parameter adaptive learning rate methods

- Adagrad
- RMSprop
- Adadelta
- Adam

Learning rate visualization





Animations that may help your intuitions about the learning process dynamics. **Left:** Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill. **Right:** A visualization of a saddle point in the optimization landscape, where the curvature along different dimension has different signs (one dimension curves up and another down). Notice that SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSProp proceed.

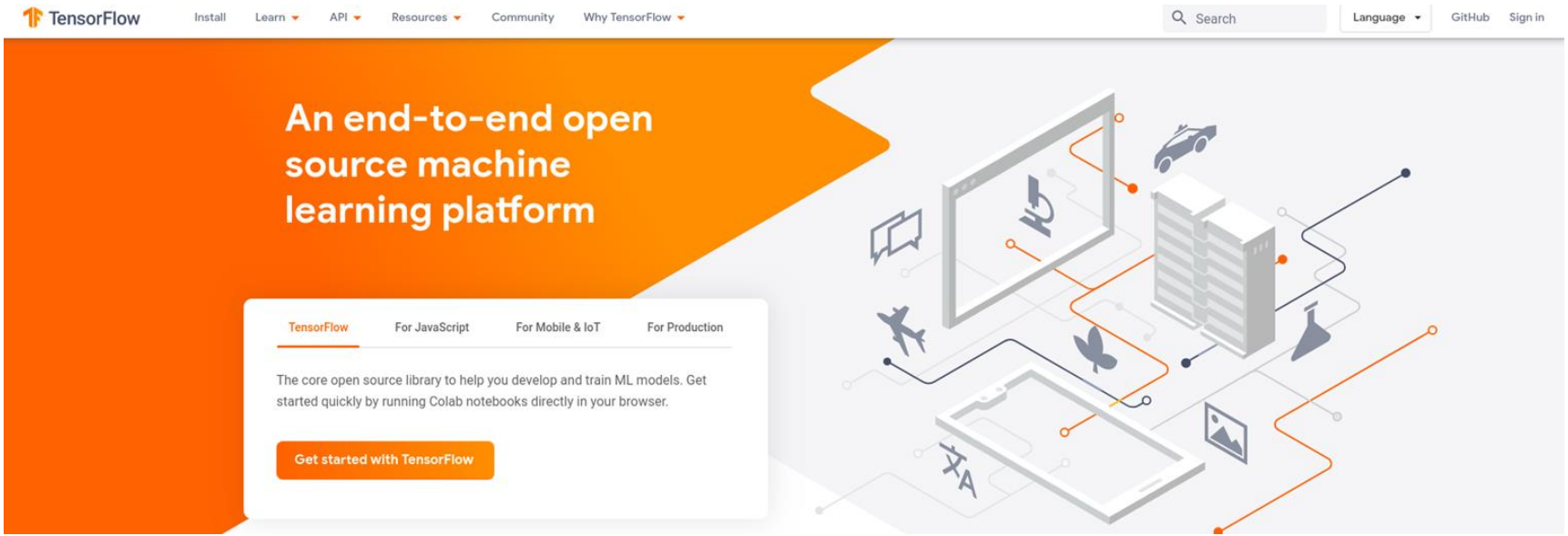
Images and text source: cs231n stanford

TensorFlow

An open source machine learning library for research and production

Image source: tensorflow

What is TensorFlow?



TensorFlow is an open-source machine learning library for research and production. TensorFlow offers APIs for beginners and experts to develop for desktop, mobile, web, and cloud.

Image source: tensorflow

Tensor values

The central unit of data in TensorFlow is the **tensor**. A tensor consists of a set of primitive values shaped into an array of any number of dimensions. A tensor's **rank** is its number of dimensions, while its **shape** is a tuple of integers specifying the array's length along each dimension. Here are some examples of tensor values:

```
3. # a rank 0 tensor; a scalar with shape [],
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

TensorFlow uses numpy arrays to represent tensor **values**.

Text source: tensorflow

Graph

A **computational graph** is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects.

- tf.Operation (or "ops"): The nodes of the graph. Operations describe calculations that consume and produce tensors.
- tf.Tensor: The edges in the graph. These represent the values that will flow through the graph. Most TensorFlow functions return tf.Tensors.

Let's build a simple computational graph. The most basic operation is a constant. The Python function that builds the operation takes a tensor value as input. The resulting operation takes no inputs. When run, it outputs the value that was passed to the constructor. We can create two floating point constants a and b as follows:

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

The print statements produce:

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

Text source: tensorflow

TensorBoard

TensorFlow provides a utility called TensorBoard. One of TensorBoard's many capabilities is visualizing a computation graph. You can easily do this with a few simple commands.

First you save the computation graph to a TensorBoard summary file as follows:

```
writer = tf.summary.FileWriter('.')
writer.add_graph(tf.get_default_graph())
writer.flush()
```

This will produce an `event` file in the current directory with a name in the following format:

```
events.out.tfevents.{timestamp}.{hostname}
```

Now, in a new terminal, launch TensorBoard with the following shell command:

```
tensorboard --logdir .
```

Then open TensorBoard's graphs page in your browser, and you should see a graph similar to the following:

Text and image source: tensorflow

Session

To evaluate tensors, instantiate a tf.Session object, informally known as a **session**. A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations. If a tf.Graph is like a .py file, a tf.Session is like the python executable. The following code creates a tf.Session object and then invokes its run method to evaluate the total tensor we created above:

```
sess = tf.Session()
print(sess.run(total))
```

When you request the output of a node with Session.run TensorFlow backtracks through the graph and runs all the nodes that provide input to the requested output node. So this prints the expected value of 7.0:

```
7.0
```

You can pass multiple tensors to tf.Session.run. The run method transparently handles any combination of tuples or dictionaries, as in the following example:

`print(sess.run({'ab':(a, b), 'total':total}))` which returns the results in a structure of the same layout:

```
{'total': 7.0, 'ab': (3.0, 4.0)}
```

During a call to tf.Session.run any tf.Tensor only has a single value.

Text source: tensorflow

Eager execution

TensorFlow's eager execution is an imperative programming environment that evaluates operations immediately, without building graphs: operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow and debug models, and it reduces boilerplate as well. To follow along with this guide, run the code samples below in an interactive python interpreter.

Eager execution is a flexible machine learning platform for research and experimentation, providing:

- *An intuitive interface*—Structure your code naturally and use Python data structures. Quickly iterate on small models and small data.
- *Easier debugging*—Call ops directly to inspect running models and test changes. Use standard Python debugging tools for immediate error reporting.
- *Natural control flow*—Use Python control flow instead of graph control flow, simplifying the specification of dynamic models.

Eager execution supports most TensorFlow operations and GPU acceleration.

Text source: tensorflow

Keras

Keras is a high-level API to build and train deep learning models. It's used for fast prototyping, advanced research, and production, with three key advantages:

- *User friendly*
  Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.
- *Modular and composable*
  Keras models are made by connecting configurable building blocks together, with few restrictions.
- *Easy to extend*
  Write custom building blocks to express new ideas for research. Create new layers, loss functions, and develop state-of-the-art models.

Import tf.keras

tf.keras is TensorFlow's implementation of the Keras API specification. This is a high-level API to build and train models that includes first-class support for TensorFlow-specific functionality, such as eager execution, tf.data pipelines, and Estimators. tf.keras makes TensorFlow easier to use without sacrificing flexibility and performance.

Text source: tensorflow

Thank you! :)

Questions are always welcome