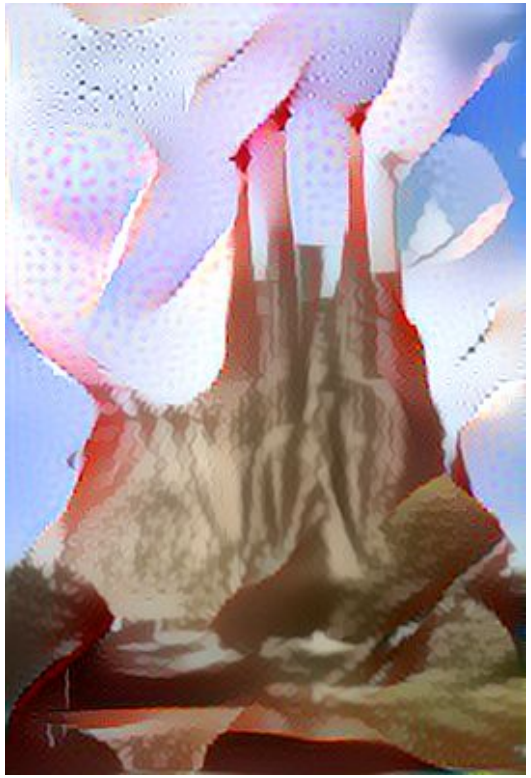
DEEP LEARNING FRAMEWORKS



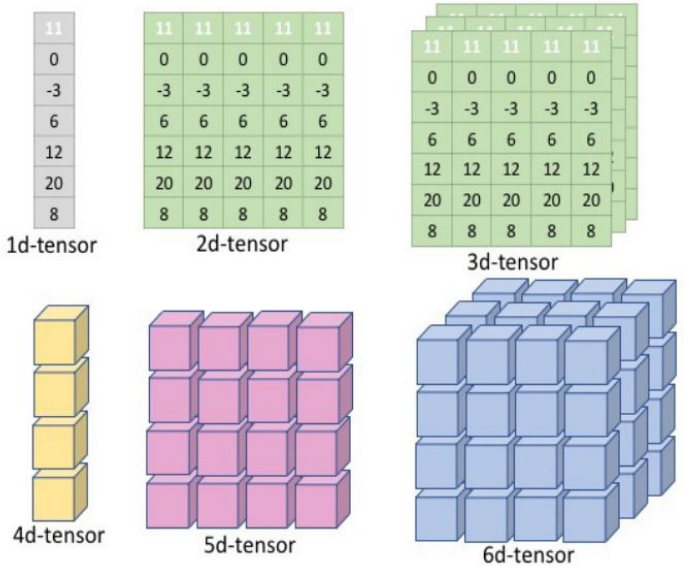
Afi Escuela de Finanzas

Analytical Index

1. **Intro: Into the Tensorverse**
2. **Tensorflow. Keras**
3. **Pytorch**
4. **So… Then what?**

Into the Tensorverse

Into the Tensorverse



- Scalar is the minimum possible unit we are working with, like an integer.
- Vector is a bunch of Scalars in a particular axis, can be drawn as a line of boxes or items.
- Matrix is a 2 dimensional array, usually represented as a table.
- Tensor is a multi-dimensional array, usually can be represented graphically as a cube.

The number of dimensions is called the **rank** of the tensor.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

One of the most common operation performed on tensors is the product. The DL frameworks optimize this kind of operation to produce faster results.

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

As we saw in the previous lesson, a huge amount of the underlying operations in a neural net are actually **matrix/tensor operations**

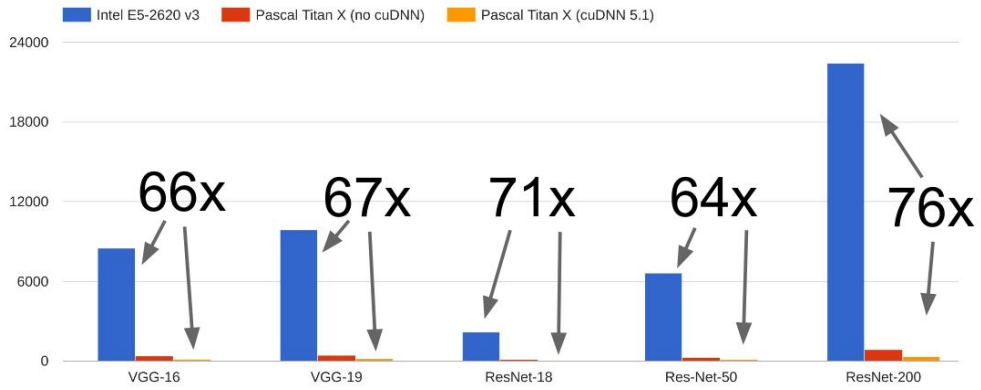
Supot Nitsuwat. *Multilinear Principal Component Analysis for Tensor Data: A Survey*

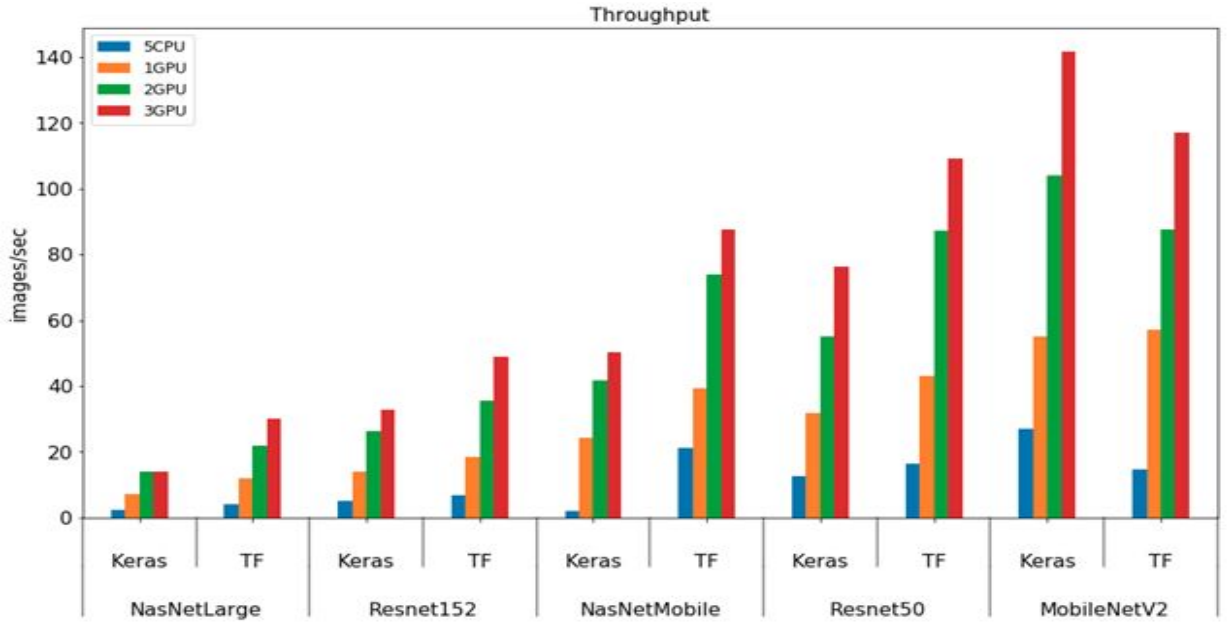
Deep Learning and GPU

A Graphics Processing Unit (GPU) is a microprocessing chip designed to handle Graphics in computing environments. GPUs can have thousands of more cores than a CPU, however, they run at lower speeds.

- Each GPU has a large number of cores, allowing for better computation of multiple parallel processes.
- Deep learning computations need to handle large amounts of data, making the high memory bandwidth in GPUs (which can run at up to 750 GB/s vs only 50 GB/s offered by traditional CPUs) better suited to a deep learning machine.
- Nvidia's CUDA is both a platform designed for GPU parallelism and an API created for its GPUs.
- The Nvidia CUDA Deep Neural Network library (cuDNN), is a library for deep learning frameworks designed to accelerate its GPUs and improve performance.

GPU vs CPU: Some benchmarks





Fei Fei Li, Fidan Boylu Uz

Deep Learning Frameworks (some of them)

We cannot build a neural net from the scratch every time we need one, right? ;)

The most important frameworks need to be/have:

- Optimized for performance.
- Easy to code
- Need to have a good community
- Need to automatically compute gradients

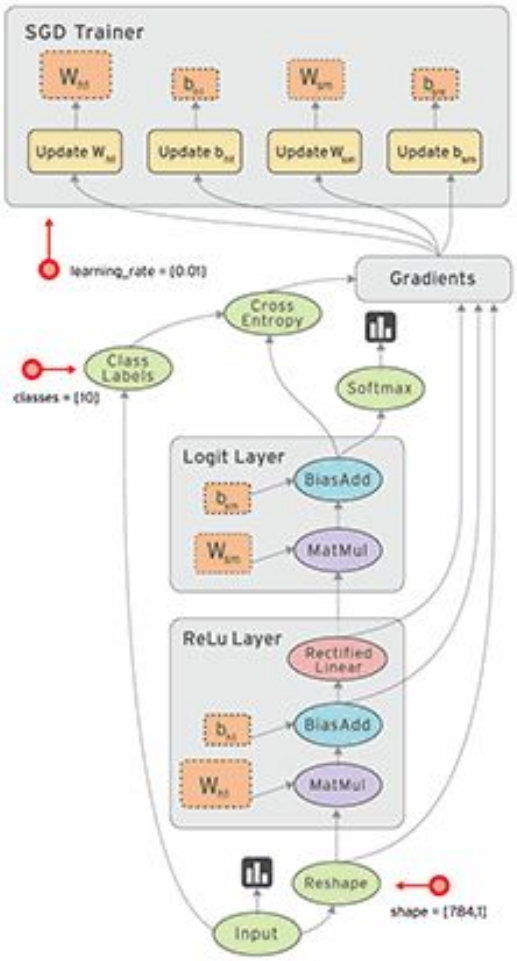
R.I.P.

Contenders

Rockstars

Tensorflow.
Keras

Tensorflow basics



The main TensorFlow entities are:

1. **Graph.** As we mentioned, TensorFlow computations can be represented as data flow graphs. Each graph is built as a set of Operation objects.
2. **Operation.** Each Operation object represents a graph node, which is a unit of computation (addition, multiplication, or something more complex) performed on a tensor flow. It takes tensors as input and produces a tensor as output.
3. **Tensor.** Tensors can be represented as edges of a data flow graph. They do not represent or hold any particular value produced by an application of an operation. Instead, they define the type of this value and the means by which this value should be calculated during the session.
4. **Session.** This is an entity that represents an environment for running calculations on the data flow graph.

TensorFlow is based on the concept of the data flow graph. The nodes of this graph represent operations. The edges are tensors. Each data flow graph computation runs within a session on a CPU or GPU.

TensorFlow exposes a Python-based API, where tensors are NumPy array entities. On the low level, the library relies on highly optimized C++ code and supports a native C and C++ API.

tensorflow.org

Tensorflow pros and cons

Pros

- You can do anything in tensorflow
- Tensorflow can also be used as a general hardware acceleration library
- Multiple language support
- Huuuuuuuuuuge community. Huge resources online
- Tensorflow is compatible with google's TPUs which can run tensor operations much faster than just a GPU
- It is open source
- **Tensorboard**
- Easy production-ready model sharing
- **Keras**

Cons

- It is a very low level with a steep learning curve. Too cluttered code
- Not need any of the super low-level stuff.
- TensorFlow has odd structure, so it's hard to find an error and also difficult to debug.
- Release compatibility is not the best

Tensorflow code sample (I)

```
class Autoencoder(object):

    def __init__(self, n_layers, transfer_function=tf.nn.softplus, optimizer=tf.train.AdamOptimizer()):
        self.n_layers = n_layers
        self.transfer = transfer_function

        network_weights = self._initialize_weights()
        self.weights = network_weights

        # model
        self.x = tf.placeholder(tf.float32, [None, self.n_layers[0]])
        self.hidden_encode = []
        h = self.x
        for layer in range(len(self.n_layers)-1):
            h = self.transfer(
                tf.add(tf.matmul(h, self.weights['encode'][layer]['w']),
                       self.weights['encode'][layer]['b']))
            self.hidden_encode.append(h)

        self.hidden_recon = []
        for layer in range(len(self.n_layers)-1):
            h = self.transfer(
                tf.add(tf.matmul(h, self.weights['recon'][layer]['w']),
                       self.weights['recon'][layer]['b']))
            self.hidden_recon.append(h)
        self.reconstruction = self.hidden_recon[-1]

        # cost
        self.cost = 0.5 * tf.reduce_sum(tf.pow(tf.subtract(self.reconstruction, self.x), 2.0))
        self.optimizer = optimizer.minimize(self.cost)

        init = tf.global_variables_initializer()
        self.sess = tf.Session()
        self.sess.run(init)
```

Tensorflow code sample (II)

```
autoencoder = Autoencoder(n_layers=[784, 200],
                          transfer_function = tf.nn.softplus,
                          optimizer = tf.train.AdamOptimizer(learning_rate = 0.001))

for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(n_samples / batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs = get_random_block_from_data(X_train, batch_size)

        # Fit training using batch data
        cost = autoencoder.partial_fit(batch_xs)
        # Compute average loss
        avg_cost += cost / n_samples * batch_size

    # Display logs per epoch step
    if epoch % display_step == 0:
        print("Epoch:", '%d,' % (epoch + 1),
              "Cost:", "{:.9f}".format(avg_cost))

print("Total cost: " + str(autoencoder.calc_total_cost(X_test)))
```

Keras

Keras is a minimalist Python library for deep learning that can run on top of TensorFlow (and formerly Theano)

Developed to make implementing deep learning models as fast and easy as possible for research and development. It runs on Python 2 or 3 and can seamlessly execute on GPUs and CPUs given the underlying frameworks.

Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

- **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
- **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- **Extensibility**: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- **Python**: No separate model files with custom file formats. Everything is native Python.

Keras Sequential API

```
trainX, trainY, testX, testY = load_dataset()
# prepare pixel data
trainX, testX = prep_pixels(trainX, testX)
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
# compile model
opt = SGD(lr=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
```

The main type of model is called a Sequence which is a linear stack of layers. Basically you create a sequence and add layers to it in the order that you wish for the computation to be performed.

Once defined, you compile the model which makes use of the underlying framework to optimize the computation to be performed by your model. In this you can specify the loss function and the optimizer to be used. Once compiled you can fit to the data.

Once compiled, the model must be fit to data. This can be done one batch of data at a time or by firing off the entire model training regime. This is where all the compute happens.

Once trained, you can use your model to make predictions on new data.

Keras Functional API

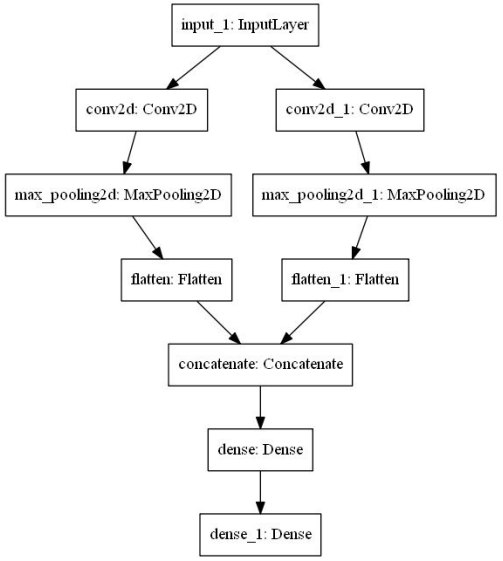
The functional API in Keras is an alternate way of creating models that offers a lot more flexibility, including creating more complex models.

It specifically allows you to define multiple input or output models as well as models that share layers. More than that, it allows you to define ad hoc acyclic network graphs.

Models are defined by creating instances of layers and connecting them directly to each other in pairs, then defining a Model that specifies the layers to act as the input and output to the model.

```
visible = Input(shape=(10,))
hidden1 = Dense(10, activation='relu')(visible)
hidden2 = Dense(20, activation='relu')(hidden1)
hidden3 = Dense(10, activation='relu')(hidden2)
output = Dense(1, activation='sigmoid')(hidden3)
model = Model(inputs=visible, outputs=output)

visible = Input(shape=(64,64,1))
# first feature extractor
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
flat1 = Flatten()(pool1)
# second feature extractor
conv2 = Conv2D(16, kernel_size=8, activation='relu')(visible)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat2 = Flatten()(pool2)
# merge feature extractors
merge = concatenate([flat1, flat2])
# interpretation layer
hidden1 = Dense(10, activation='relu')(merge)
# prediction output
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```



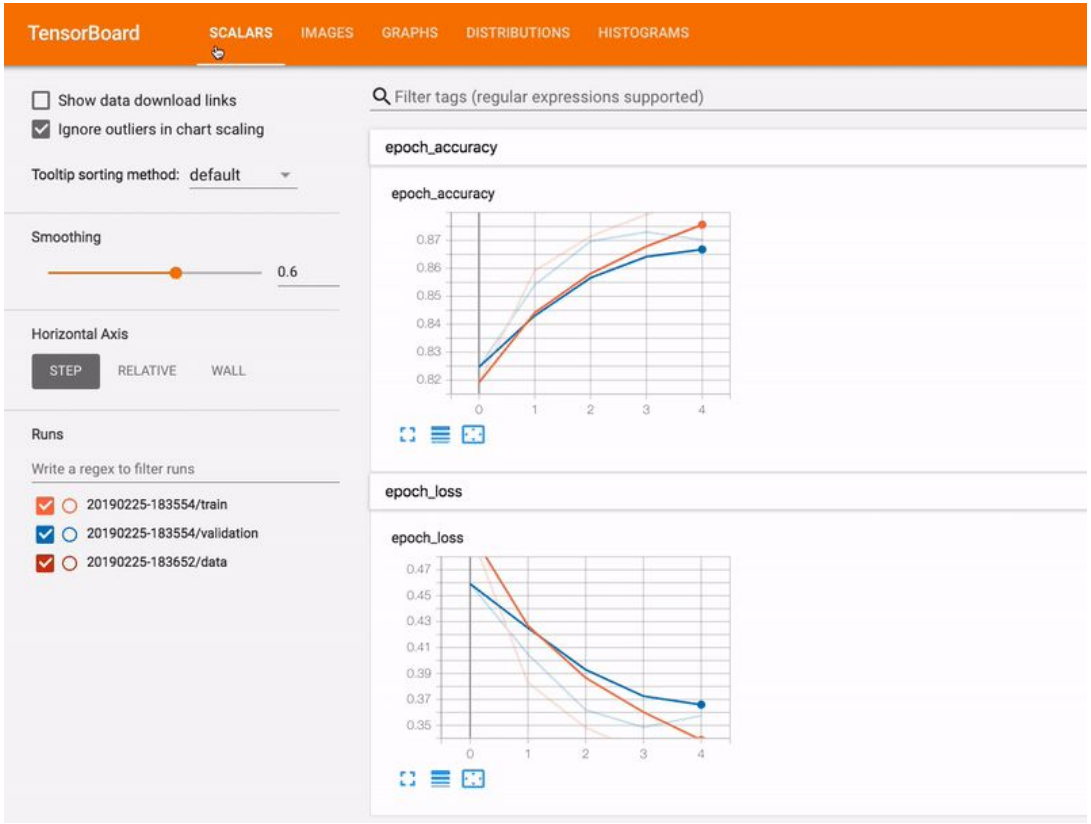
Keras Code example

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=2, activation="softmax"))

opt = Adam(lr=0.001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

Tensorboard



TensorBoard is a visualization software that comes with any standard TensorFlow installation. Its purpose is to make it easier to understand, debug, and optimize TensorFlow programs, in the shape of a suite of visualization tools.

TensorFlow programs can range from very simple to super complex problems, and they all have two basic components, Operations and Tensors.The idea is to create a model that consists of a set of operations, feed data in to the model and the tensors will flow between the operations until you get an output tensor.

Some of the most common dashboards are:

- The **Scalars** dashboard. It shows how the loss and metrics change with every epoch, making it a nice tool to track training speed, learning rate, and other scalar values.
- The **Graphs** dashboard. It helps to visualize your model. In this case, the Keras graph of layers is shown which can help to ensure it is built correctly.
- The **Distributions** and **Histograms** dashboards. They show the distribution of a Tensor over time. This can be useful to visualize weights and biases and verify that they are changing in an expected way.

Keras pros and cons

Pros

- It is the easiest to use Deep Learning framework. It has been created to leave DL practitioners out of code complexity and let them experiment and it is really helpful when learning the basics.
- Empowers Tensorflow. Tf 2.x is basically embedded in keras and this is because users have now a possibility of building powerful code with a few lines.
- There is a lot of developers behind it. As Tensorflow is the most common framework and has a huge community behind, keras has that community also. Most of the issues you find will probably be issues for many other people
- Inherits TF easiness of deployment which makes it a standard in the industry.

Cons

- It is not the most flexible framework in the world. There are ways of building models and they admit no change.
- Little customization compared to other frameworks. As in the previous point, it is too high level for most of the developers. Building for example custom optimizers or loss functions is possible, but not that intuitive due to this feature.
- Debugging is also not that advanced as in other frameworks
- Not the slowest, but not the fastest framework.

Tensorflow.js

- TensorFlow.js is a JavaScript library that allows you to add machine learning capabilities to any web application.
- The idea is to develop machine learning systems from low to high levels.
- The API can be used to build lightweight models that can be directly trained in the browser or on our Node.js server application.
- After that, TensorFlow.js can be used to run the trained models in JavaScript environment.
- JavaScript code cannot run on GPU. To overcome this problem, WebGL was introduced. WebGL is a browser interface for OpenGL that allows to execute JavaScript code on the GPU.

Quick Takeaways

- ❖ It is actually much slower than Tensorflow. When you train huge models, it can be up to 10-15x slow
- ❖ Tf.js makes model deployment easier, but you have more freedom on tf.
- ❖ No Tensorboard, no party

Tensorflow basics

Configuring Tensorflow for GPU usage is not easy. After doing this like 15-20 times I still have some issues from time to time. So, this is a small recopilation of hints in case you want to configure it on **linux**. You will find a lot of help online, but these are the things I learn from failing those installations once and again.

- You are going to need the driver of your gpu, a CUDA installation, CUDNN and finally a GPU-compatible version of tf.
- Install the latest driver. Download it from NVIDIA web page
- When installing cuda, take a look at the version compatibility. 10.0 works well on tf 1.1x, but not that well on tf 2.x. CUDA 10.1 is good on 2.x and not that good on 1.1x…
- Check the post installation tasks in the manual (you should have a direct link when you download the software). Make sure every step is correct before going to the next
- Perhaps you'll find some issues with the Bazel version. No problem, just install the one the compiler asks you for.
- CUDNN should not be an issue, but always make sure to have all the prerequisites
- Please, do use a virtual environment. Regardless if it is conda or pip
- if you want tf 2, it doesn't need that much complicated stuff, but if you want tf1 maybe you need to build it from source (compiling the whole library for generating a wheel pip can install). It is expensive (in time) but not that hard. Try it
- After installing start a python console and make sure you 're actually using GPU. There are some code lines you can try. If that works, we are home!!
- Good luck

I needed to put a picture of these guys somewhere.
Introducing the DL Mafia

Pytorch

Pytorch

Based on the Torch library, PyTorch is an open-source machine learning library. PyTorch is imperative, which means computations run immediately, means user need not wait to write the full code before checking if it works or not. We can efficiently run a part of the code and inspect it in real-time. The library is python based built for providing flexibility as a deep learning development platform.

Key features:

- Easy to use API
- **Python support –** It smoothly integrates with the python data science stack. It was created to be similar to numpy, so it lowers the learning curve
- **Dynamic computation graphs –** PyTorch provides a framework to build computational graphs on the go, and even change them during runtime instead of predefined graphs with specific functionalities. This service is valuable for situations where we don't know the requirement of memory for creating a neural network.

Other key strengths of the machine learning framework include:

- Allows easy debugging integration with any debugging tool
- **Distributed Training:** Distributed backend. Torch enables performance optimization in research and production and scalable distributed training.
- Tensorboard compatible.

Pytorch Pros and Cons

Pros:

- **Simplicity.** It's similar to numpy, very pythonic, and integrates easily with the rest of the Python ecosystem. Pretty easy to learn.
- **Easy to debug.** For example, you can simply throw in a pdb breakpoint anywhere into your PyTorch model and it'll work. In TensorFlow, debugging the model requires an active session and ends up being much trickier.
- **Great API.** Most researchers prefer PyTorch's API to TensorFlow's API. This is partially because PyTorch is better designed and partially because TensorFlow has handicapped itself by switching APIs so many times (e.g. 'layers' -> 'slim' -> 'estimators' -> 'tf.keras').
- **Dynamic**. Both on its approach to gpu and in the graphs it builds that are transparent to the user
- **Performance.** Despite the fact that PyTorch's dynamic graphs give strictly less opportunity for optimization, there have been many anecdotal reports that PyTorch is as fast if not faster than TensorFlow. It's not clear if this is really true, but at the very least, TensorFlow hasn't gained a decisive advantage in this area.
- **Growing community** of focused developers (Small group who solve big issues)

Facebook Research

Cons

- Although it is production-ready, it is n**ot as easy to deploy** as Tensorflow
- Its community is growing, but still doesn't have the level of tf.
- Doesn't have **proprietary monitoring interfaces**, but it integrates well with Tensorboard.

Pytorch code

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```python
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

Bonus track: Fast.ai

```
learn = cnn_learner(data, models.resnet34, metrics=error_rate)
```

```
learn.fit_one_cycle(4)
```

Total time: 01:46

epoch	train_loss	valid_loss	error_rate
1	1.409939	0.357608	0.102165
2	0.539408	0.242496	0.073072
3	0.340212	0.221338	0.066306
4	0.261859	0.216619	0.071042

```
data = ImageDataBunch.from_name_re(path_img, fnames, pat, ds_tfms=get_transforms(), size=224, bs=bs
                                   ).normalize(imagenet_stats)
```

```
data.show_batch(rows=3, figsize=(7,6))
```

fast.ai

Making neural nets uncool again

FastAI is a different kind of metaframework. It provides a single consistent interface to all the most commonly used deep learning applications for vision, text, tabular data, time series, and collaborative filtering. It can be described as a research lab, an easy-to-use Python library with a huge community. Their library wraps popular deep learning and machine learning libraries for common workflows and provides a user-friendly interface. Most importantly, it follows the "top down" approach.

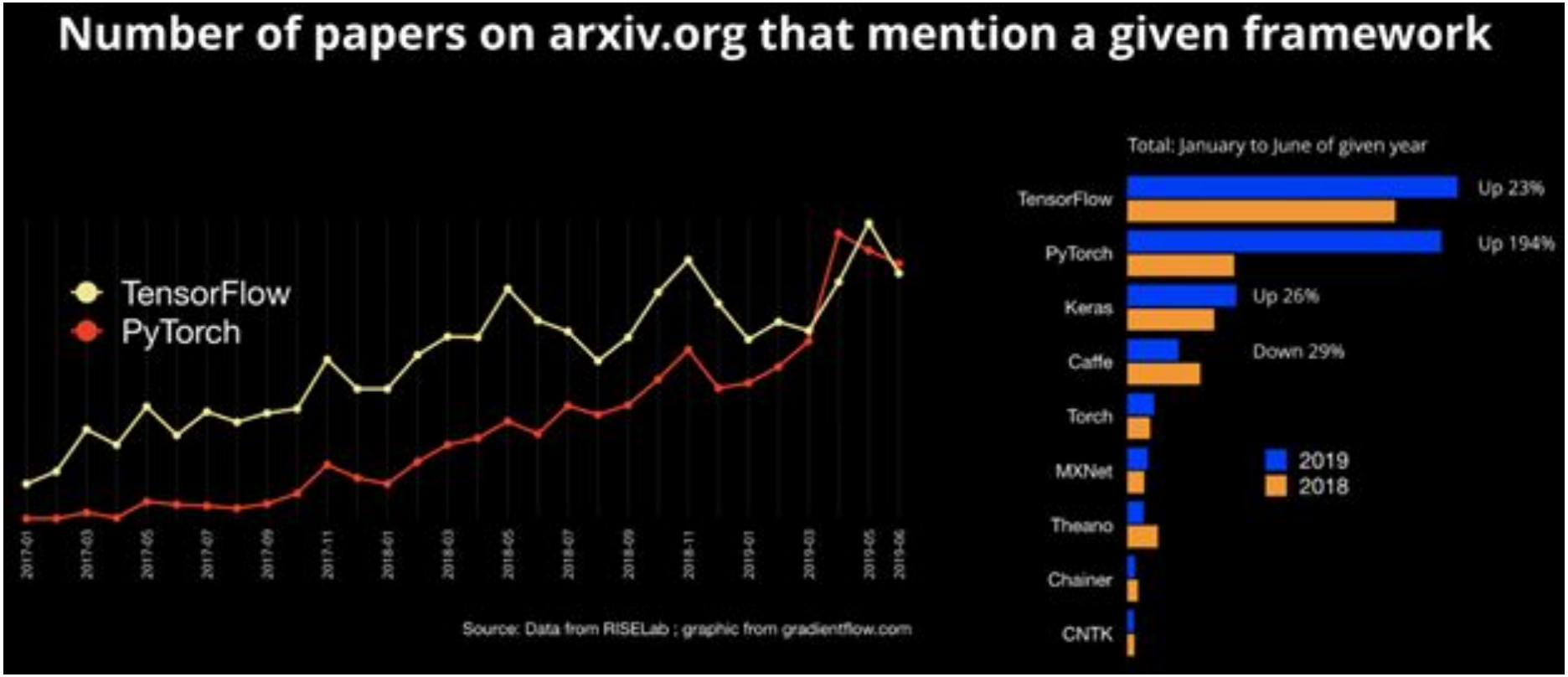
fast.ai course notebooks

So... Then what?

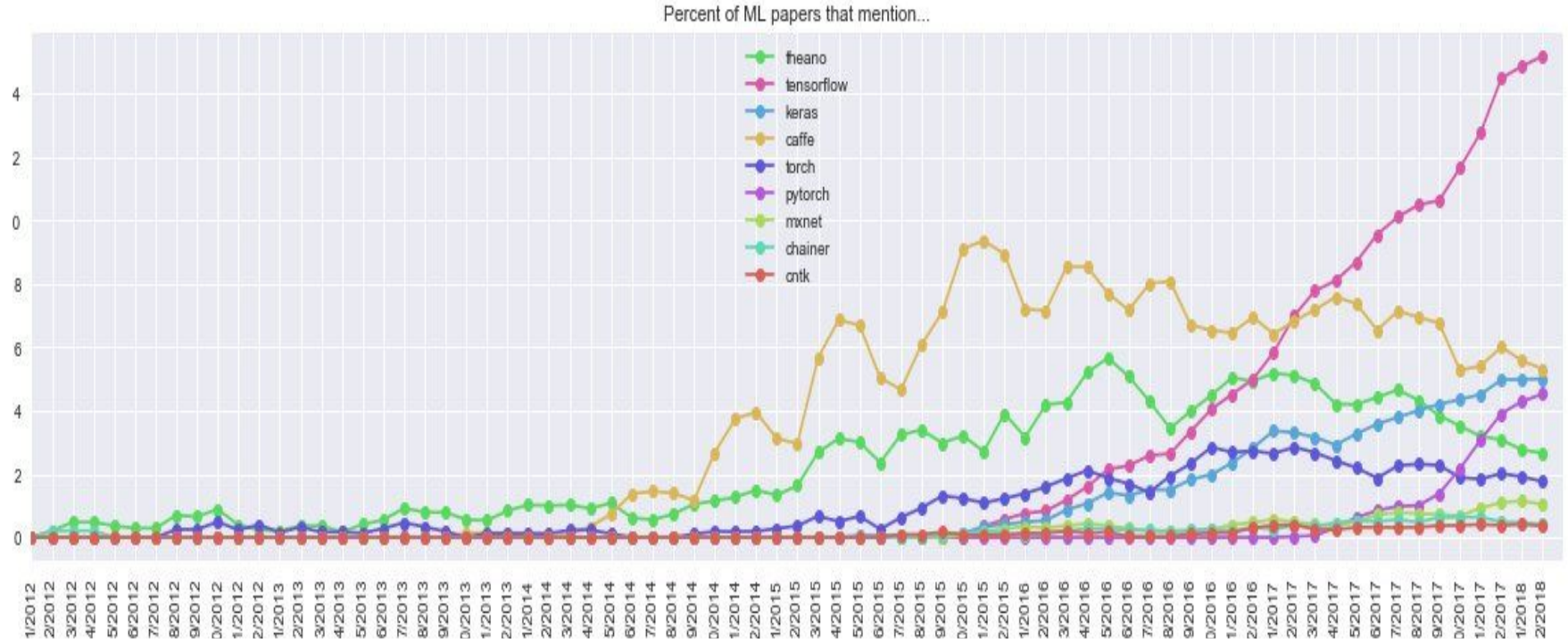
Pytorch AND Tensorflow

- **Creators:** While both Tensorflow and PyTorch are open-source, they have been created by two different wizards. Tensorflow is based on Theano and has been developed by Google, whereas PyTorch is based on Torch and has been developed by Facebook.
- **Dynamic vs Static Graphs:** The most important difference between the two is the way these frameworks define the computational graphs. While Tensorflow creates a static graph, PyTorch believes in a dynamic graph. So what does this mean? In Tensorflow, you first have to define the entire computation graph of the model and then run your ML model. But in PyTorch, you can define/manipulate your graph on-the-go. This is particularly helpful while using variable length inputs in RNNs.
- **Easy to use:** Tensorflow has a more steep learning curve than PyTorch. PyTorch is more *pythonic* and building ML models feels more intuitive. On the other hand, for using Tensorflow, you will have to learn a bit more about it's working (sessions, placeholders etc.) and so it becomes a bit more difficult to learn Tensorflow than PyTorch.
- **Community:** Tensorflow has a much bigger community behind it than PyTorch. This means that it becomes easier to find resources to learn Tensorflow and also, to find solutions to your problems. This is because PyTorch is a relatively new framework as compared to Tensorflow. So, in terms of resources, you will find much more content about Tensorflow than PyTorch.
- **Deployment**: Finally, Tensorflow is much better for production models and scalability. It was built to be production ready. Whereas, PyTorch is easier to learn and lighter to work with, and hence, is relatively better for passion projects and building rapid prototypes. It is considered to be the ideal framework for researchers, and applications of deep learning requiring optimizing custom expressions.

Some numbers

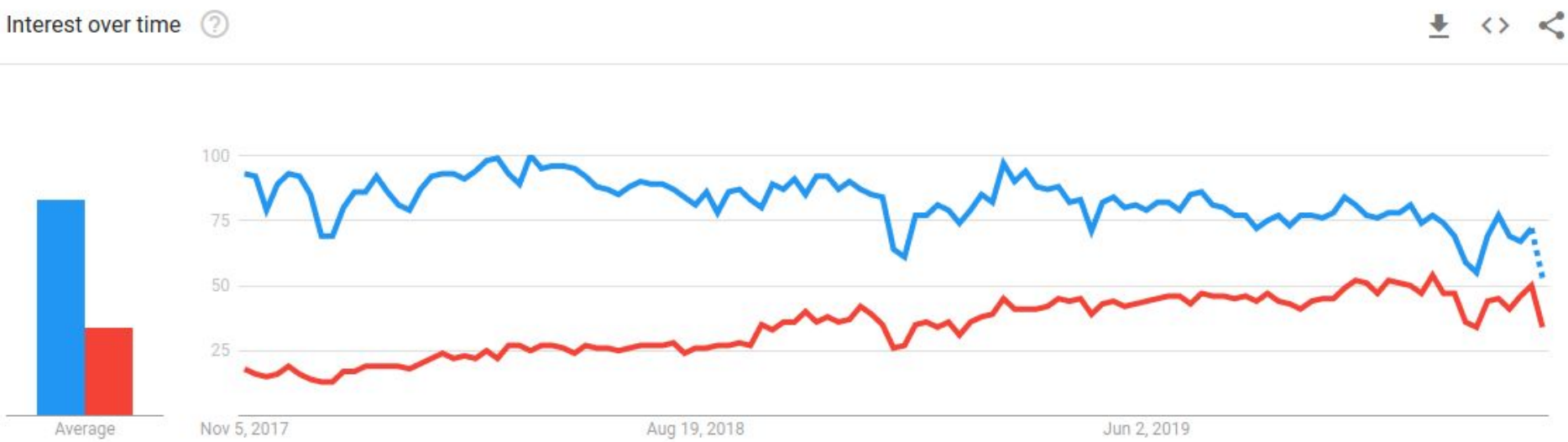


Some numbers



Andrej Karpathy

Some numbers



Andrej Karpathy