

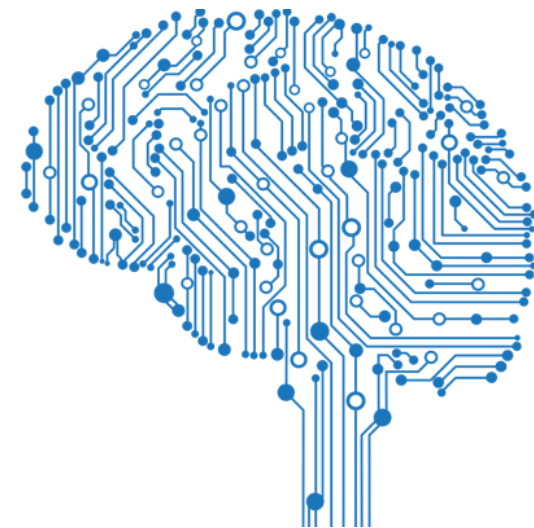
Introduction to Deep Learning

Neural networks, computation, optimization

Yordan Darakchiev

Technical Trainer

iordan93@gmail.com





sli.do

#DeepLearning

Table of Contents

- Computational graphs
- Simple models with tensorflow
 - Low-level API
- Building neural networks
- Regularization

Computational Graphs

Performing simple calculations...
just harder

Installing tensorflow

- Use Anaconda
 - It's easier, and arguably much faster
 - Run as administrator
- If possible, try installing the GPU version
 - The package will try to install the GPU version if available
 - Otherwise, it'll fall back to the CPU (standard) version

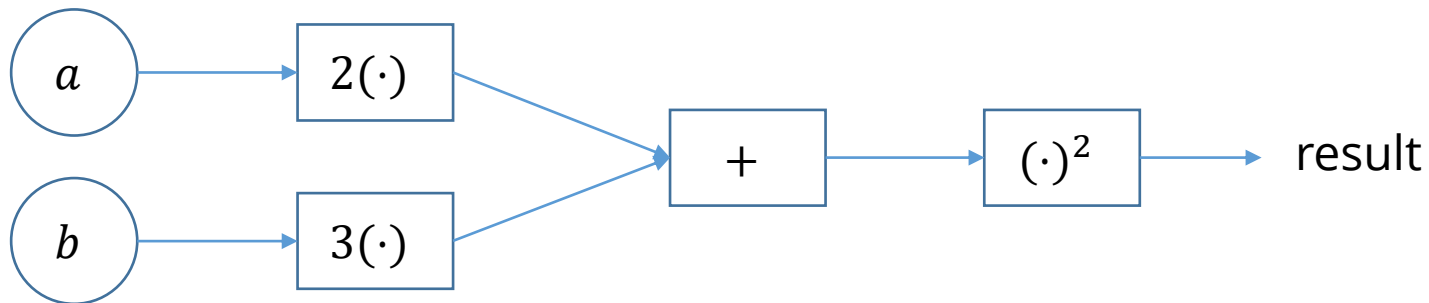
```
conda install tensorflow
```

```
pip install tensorflow
```

- There's no difference in the API
 - GPUs perform computations much faster
 - Sometimes $\sim 10^2 - 10^4$ times faster

Computational Graphs in tensorflow

- "Flow of tensors (multidimensional matrices)"
- Computational graph (DAG)
 - A useful representation of computation sequences
 - Contains data and operations
 - Data "flows" through and gets transformed
- A simple example
 - $(2a + 3b)^2$



Computational Graphs in tensorflow (2)

- Import the library

```
import tensorflow as tf
```

- Create two constants for a and b
 - We will provide their values later to get the result

```
a = tf.constant(2)
b = tf.constant(3)
```

- Create the function
 - Using tensorflow operations

```
def compute(a, b):
    return tf.pow(
        tf.add(
            tf.multiply(2, a),
            tf.multiply(3, b)
        ),
        2)
```

Computational Graphs in tensorflow (3)

- However, it can be much, much simpler!

```
def compute(a, b):  
    return (2 * a + 3 * b) ** 2
```

- This works in regular Python, numpy, and tensorflow
 - Depends on the types of a and b

```
compute(2, 3) # 169  
compute(2.0, 3.0) # 169.0  
  
compute(np.array([2, 3, 4]), np.array([3, 4, 5]))  
# array([169, 324, 529], dtype=int32)  
  
compute(tf.constant(2), tf.constant(3))  
# <tf.Tensor: id=53, shape=(), dtype=int32, numpy=169>  
compute(tf.constant([2, 3, 4]), tf.constant([3, 4, 5]))  
# <tf.Tensor: id=62, shape=(3,), dtype=int32,  
#     numpy=array([169, 324, 529])>
```


Visualizing the Graph

- Decorate the function with `@tf.function`

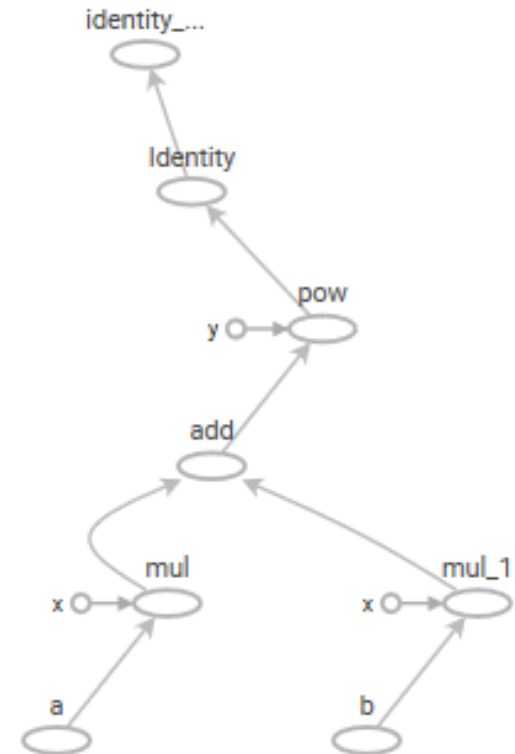
- Trace the execution

```
a, b = tf.constant([2, 3, 4]), tf.constant([3, 4, 5])

tf.summary.trace_on(graph = True, profiler = True)
result = compute(a, b)
print(result.numpy())
with writer.as_default():
    tf.summary.trace_export(
        name = "compute_trace",
        step = 0,
        profiler_outdir = "logs")
```

- Run tensorboard to visualize the graph and function trace

```
tensorboard --logdir logs
```



Linear Models

Logistic regression using
tensorflow

Logistic Regression

- Data: Iris dataset

```
from sklearn.datasets import load_iris
iris = load_iris()
attributes, labels = iris.data, iris.target
```

- Prepare the output – one-hot encoding

```
labels_onehot = tf.one_hot(labels, depth = 3).numpy()
```

- Define the model

- I'm doing the 3 LR's at once (with the same inputs)
 - Equivalent results to scikit-learn's one-vs-rest strategy
- We can inspect it after that

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape = (4,)), # Not required
    tf.keras.layers.Dense(3, activation = "sigmoid")
])
```

Logistic Regression (2)

- Compile the model
 - Specify the loss function

```
model.compile(optimizer = "adam", loss = "categorical_crossentropy")
```

- Fit the model
 - Of course, you can use train / test splits, etc.

```
model.fit(attributes, labels_onehot, epochs = 50)
```

- View the results and see some metrics

```
predictions = model.predict(attributes)

acc = tf.metrics.categorical_accuracy(
    labels_onehot,
    model.predict(attributes))
print(tf.math.reduce_mean(acc).numpy())
# or add metrics = ["accuracy"] to model.compile()
```

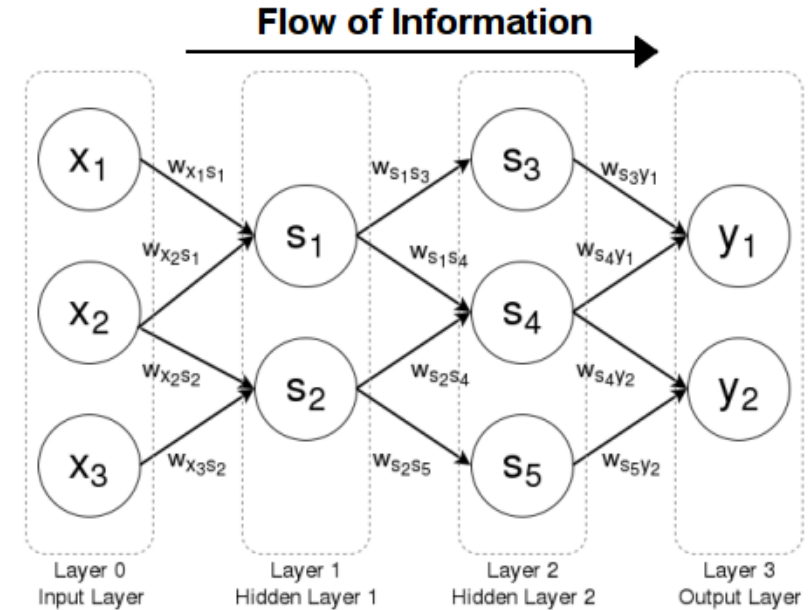
Neural Networks

Going deeper

Deep Feed-Forward Neural Network

- Many **perceptrons** arranged in **layers**

- Input** layer
- Hidden** layers
- Output** layer



- Computing output: forward propagation
- Training: backpropagation
- We can do this using the low-level API
 - If you want to implement this, don't forget
 - Bias term for each layer
 - Activation function
 - Random weight initialization (small numbers with $\mu = 0$)

Example: MNIST

- Load and normalize the data

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

- Create and evaluate the model

```
model = Sequential([
    Flatten(),
    Dense(512, activation = "relu"),
    Dropout(0.2),
    Dense(10, activation = "softmax")])
model.compile(
    optimizer = "adam",
    loss = "sparse_categorical_crossentropy",
    metrics = ["accuracy"])
```

```
model.fit(x_train, y_train, epochs = 5)
model.evaluate(x_test, y_test)
# or add validation_data = (x_test, y_test) to model.fit()
```

Summary

- Computational graphs
- Simple models with tensorflow
 - Low-level API
- Building neural networks
- Regularization

The image features a white background with two blue decorative bars. The top bar is a solid blue strip. Below it is a white space containing the text. At the bottom, there is another white space, followed by a thin dark blue line and a final solid blue strip at the very bottom.

Questions?