Modul - Introduction to AI - part II (AI2)

Bachelor Programme AAI

06 - Neural Networks with Keras/Tensorflow

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Agenda

On the menu for today:

- Short Introduction in Tensorflow/Keras
- Tensors
- Neural Networks (again!)
    - This time with Tensorflow/Keras

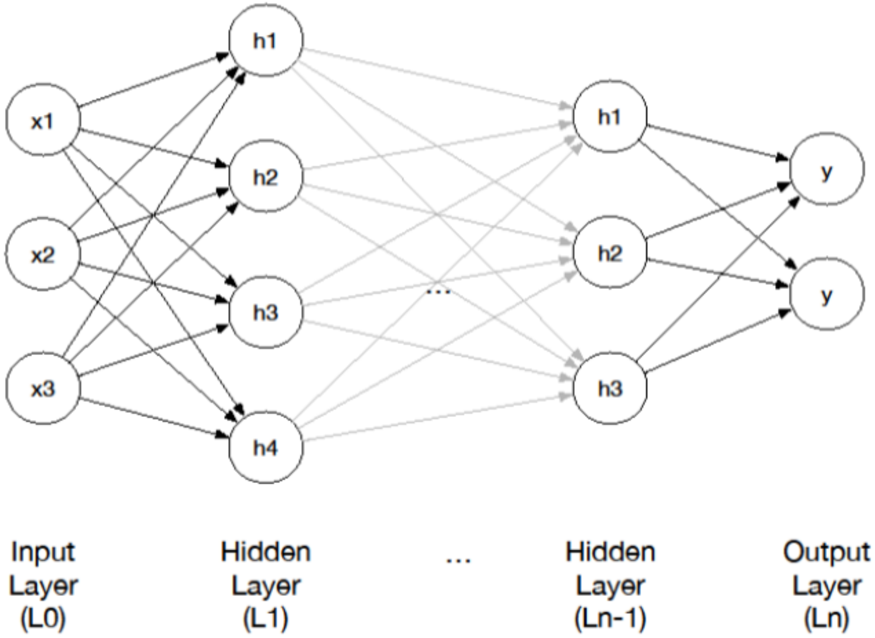
Multi-Layer Neural Network

Activation functions
- Sigmoid, tanh, ReLU,…

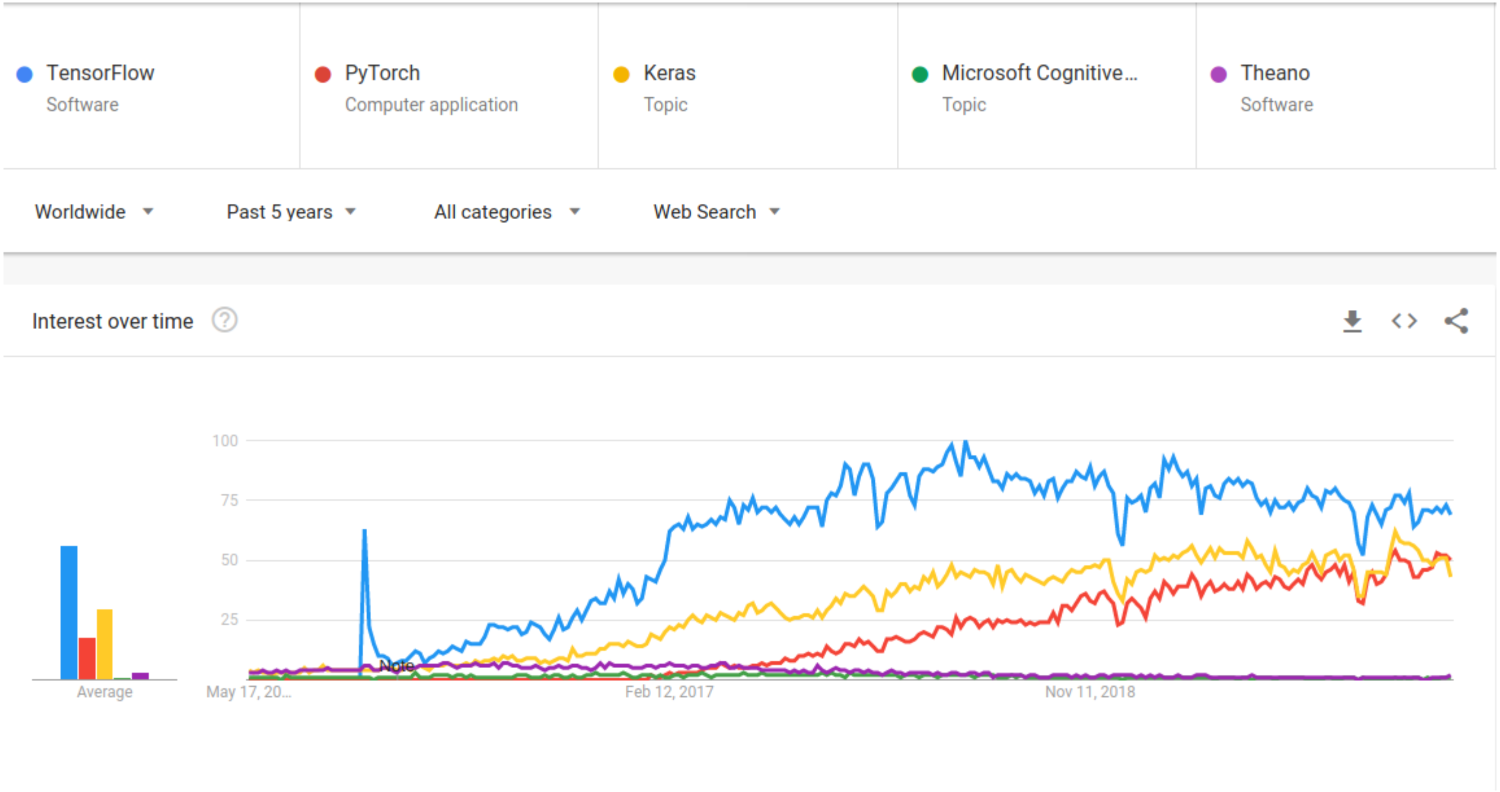
Loss
- Gradient Descent, …

Forward-, Backpropagation

Learn weights!!!



DNN Frameworks



● TensorFlow Software

● PyTorch Computer application

● Keras Topic

● Microsoft Cognitive... Topic

● Theano Software

Worldwide ▾ Past 5 years ▾ All categories ▾ Web Search ▾

Welcome to TensorFlow

- Open source software library for numerical computation using data flow graphs
- Originally developed by Google **Brain Team** to conduct machine learning and deep neural networks research
- General enough to be applicable in a wide variety of other domains as well
- https://www.tensorflow.org/

Why TensorFlow?

- Python API, TensorFlow.js, TensorFlow Light
- **Portability**: 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
- **Visualization** via TensorBoard
- Checkpoints (for managing experiments)
- Huge Community (https://github.com/tensorflow/tensorflow)
- Succesful projects already using TensorFlow

TensorFlow

- TensorFlow provides an extensive suite of functions and classes that allow users to define models from scratch.
- There are also pretrained models available (**model zoo**)
- *TF Learn (tf.contrib.learn)*: simplified interface that helps users transition from the world of one-liner such as scikit-learn
- *TF Slim (tf.contrib.slim)*: lightweight library for defining, training and evaluating complex models in TensorFlow.
- **High level API**: Keras, TFLearn, Pretty Tensor

```
import tensorflow as tf
```

or do this

- Computes math on tensors (like numpy but with GPU-support!)

What is a Tensor?

- In mathematics, a tensor is an algebraic object that describes a multilinear relationship between sets of algebraic objects related to a vector space.
- In Deep Learning it is common to see a lot of discussion around tensors as the cornerstone data structure.
- Tensor even appears in name of Google's flagship machine learning library: "TensorFlow".
- Tensors are a type of data structure used in linear algebra, and like vectors and matrices, you can calculate arithmetic operations with tensors.

Why Tensor?

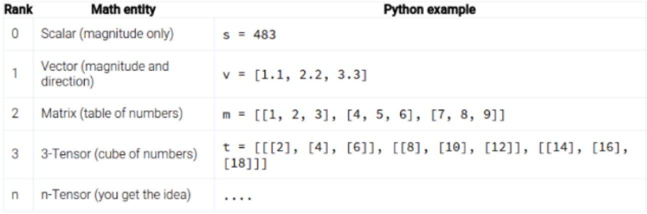
- A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array.
  - An n-dimensional array …
  - multilinear maps from vector spaces to the real numbers

0-d rank tensor: scalar (number)

1-d rank tensor: vector

2-d rank tensor: matrix

and so on



Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

Tensor Basics

Let's create some basic tensors.

Here is a "scalar" or "rank-0" tensor:

```
# This will be an int32 tensor by default; see "dtypes" below.
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)

>tf.Tensor(4, shape=(), dtype=int32)
```

A "vector" or "rank-1" tensor is like a list of values. A vector has one axis:

```
# Let's make this a float tensor.
rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)

>tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
```

More Complex

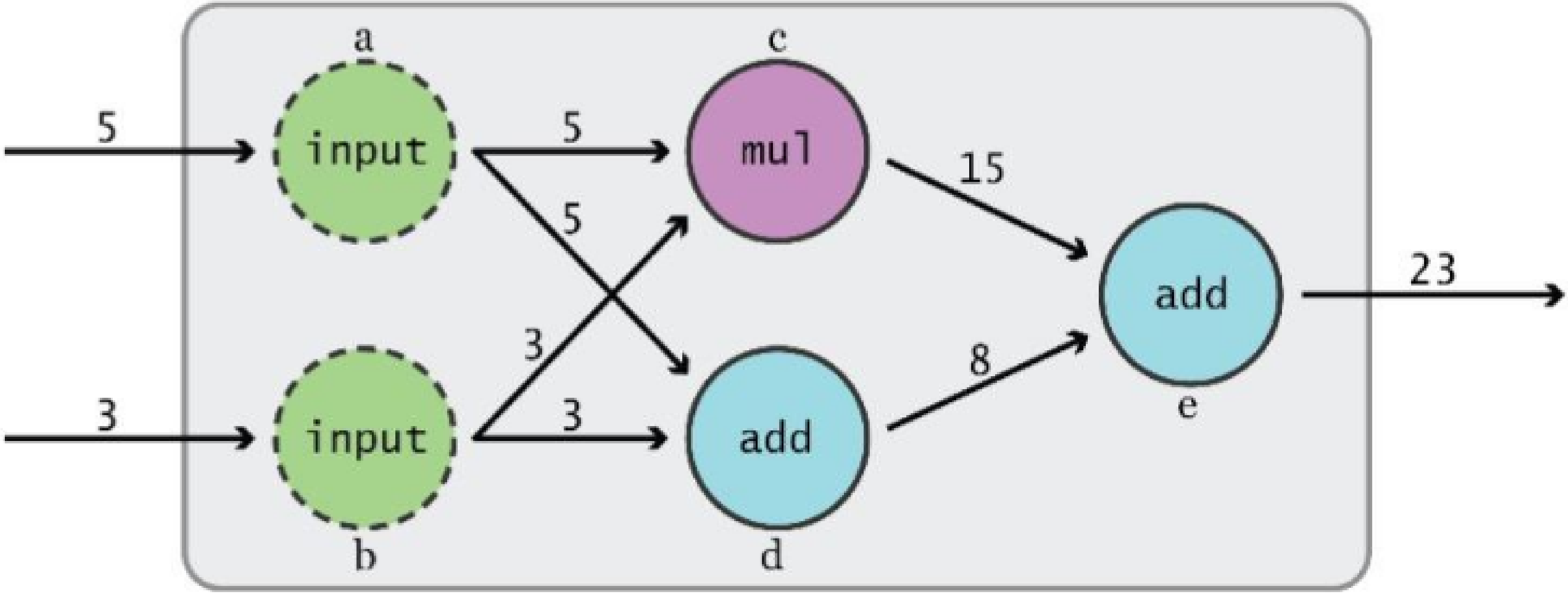
A "matrix" or "rank-2" tensor has two axes:

```
# If you want to be specific, you can set the dtype
rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)

>tf.Tensor(
[[1. 2.]
 [3. 4.]
 [5. 6.]], shape=(3, 2), dtype=float16)
```

Data Flow Graphs

1. Assemble a graph
2. Use a **session** to execute operations in the graph



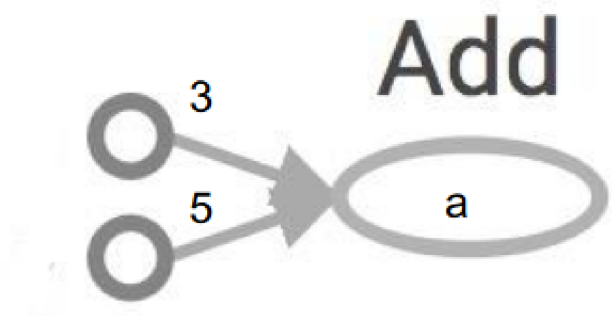
Data Flow Graphs

First attempt:

```
import tensorflow as tf

a = tf.add(3, 5)
print(a)
```

```
tf.Tensor(8, shape=(), dtype=int32)
```



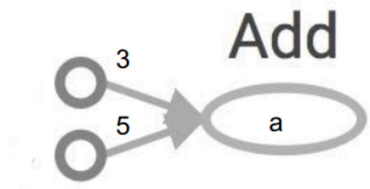
How to get the value?

- TensorFlow computations define a computation graph that has no numerical value until evaluated!
- Before TF2.0: A session `tf.Session()` evaluates the graph with `run()`

```
import tensorflow as tf

a = tf.add(3, 5)
print(a.numpy())
```

```
8
```



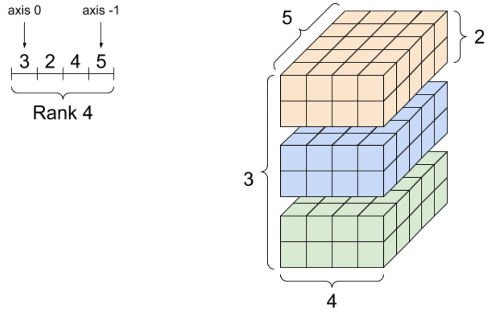
About shapes

Tensors have shapes. Some vocabulary:

- **Shape**: The length (number of elements) of each of the axes of a tensor.
- **Rank**: Number of tensor axes. A scalar has rank 0, a vector has rank 1, a matrix is rank 2.
- **Axis or Dimension**: A particular dimension of a tensor.
- **Size**: The total number of items in the tensor, the product shape vector.

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
```

> A rank-4 tensor, shape: [3, 2, 4, 5]



Manipulating Shapes

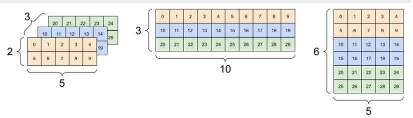
Reshaping tensors:

```
tf.Tensor(
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29], shape=(30,), dtype=int32)
```

Typically the only reasonable use of tf.reshape is to combine or split adjacent axes (or add/remove 1s).

For this 3x2x5 tensor, reshaping to (3x2=6)x5 or 3x(2x5=10) are both reasonable things to do, as the slices do not mix:

```
print(tf.reshape(rank_3_tensor, [3*2, 5]), "\n")
print(tf.reshape(rank_3_tensor, [3, -1]))
```



Quick fun with JavaScript

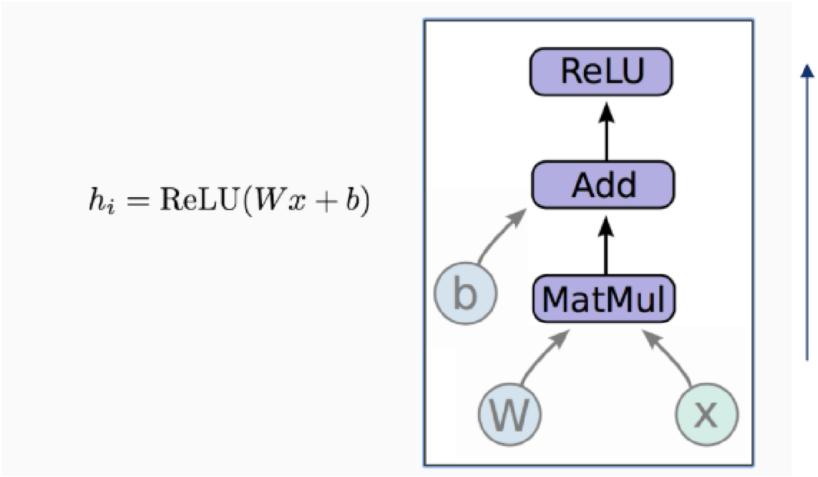
https://js.tensorflow.org/api/

Fun part:

./06-lecture/TFCalc.html

Tensorflow Computation Graph

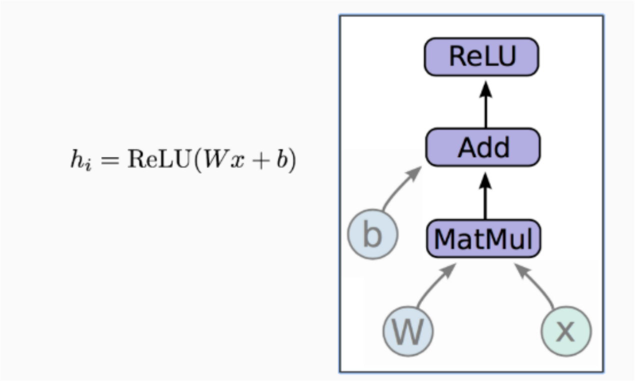
- Express a numeric computation as a **graph**.
    - Graph nodes are **operations** which have any number of inputs and outputs
    - Graph edges are **tensors** which flow between nodes



Mathematical operations

- Computations that will act on tensors
  - **MatMul**: Multiply two matrix values
  - **Add** : Add elementwise(with broadcasting)
  - **ReLu** : Activate with elementwiserectified linear function
  - etc.

```python
import tensorflow as tf
w = tf.constant(3.0)
x = tf.constant(2.0)
b = tf.constant(5.0)
mul= tf.multiply(w, x)
intermed= tf.add(mul, b)
result= tf.nn.relu(intermed)
print(result.numpy())
```



Placeholder and Variables

- **tf.Variable**: When you train a model you use **variables** to hold and update parameters. Variables are in-memory buffers containing tensors State is retrained across multiple executions
- TF < 2.0: **tf.placholder**: Placeholder are dummy nodes that provide entry points for data to a computational graph
    - value is fed in at execution time

```
x = tf.constant([[3, 2],
                 [5, 2],
                 [9, 5],
                 [1, 3]])
w = tf.Variable(tf.ones(shape=[2,4], dtype=tf.int32))
b = tf.Variable(tf.zeros(shape=[4,4], dtype=tf.int32))

result= tf.nn.relu(tf.matmul(x,w) + b)

print(result.numpy())
```

But how to build a *DNN* with it?

A Model with Tensorflow

Simply straight forward:

```
model = tf.sequential()
model.add(tf.layers.dense({units: 1, inputShape: [1]}))

# Prepare the model for training: Specify the loss and the optimizer.
model.compile({loss: 'meanSquaredError', optimizer: 'sgd'})

# Generate some synthetic data for training.
xs = tf.tensor2d([1, 2, 3, 4, 10], [5, 1])
ys = tf.tensor2d([1, 4, 6, 8, 20], [5, 1])

# Train the model using the data.
model.fit(xs, ys, {epochs: 10})

testData = [5,9,11,19,21]
result = model.predict(tf.tensor2d(testData, [testData.length, 1]))
```

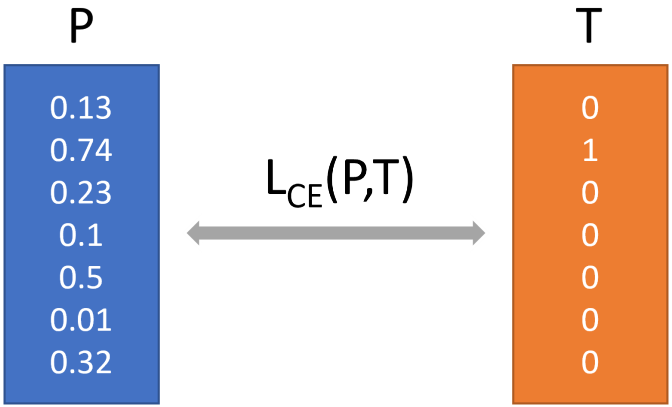
./06-lecture/HelloWorld.html

A word about Loss-Functions

- When working on a Machine Learning or a Deep Learning Problem, loss/cost functions are used to optimize the model during training.
- The objective is almost always to minimize the loss function.
- The lower the loss the better the model.
- So far, I have used *Mean Squared Error*!
- Loss Functions
    - MSE
    - MAE
    - ...
    - **Cross-Entropy**
    - https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Cross-Entropy (1/2)

- Cross-Entropy loss is a most important cost function.
- The purpose of the Cross-Entropy is to take the output probabilities (P) and measure the distance from the truth values (T).



P

0.13
0.74
0.23
0.1
0.5
0.01
0.32

$L_{CE}(P,T)$

T

0
1
0
0
0
0
0

Cross-Entropy (2/2)

> Reminder:Entropy of a random variable X is the level of uncertainty inherent in the variables possible outcome.
>
> $$H(x) = -\sum p(x) log_2(p(x))$$

Cross-entropy is defined as

$$Lce = -\sum t_i \cdot log_2(p_i)$$

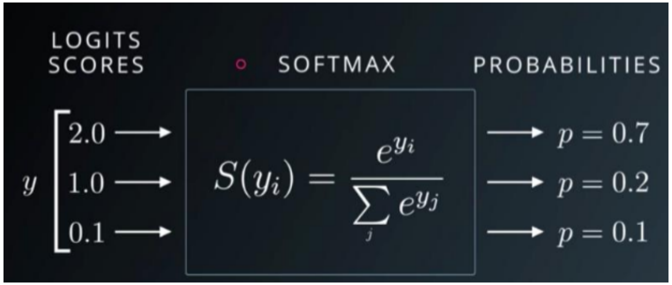
where t_i is the truth label and p_i is the propability (Softmax) of th i^{th} class.

```
def ce(S,T):
    return (-1) * sum([(t* log(s,2))for s,t in zip(S,T)])
```

Softmax-Function

The **softmax function** is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities.

- The output vector contains scores (*logits*_); after applying **softmax**, each component will be in the interval (0, 1) and the components will add up to 1



```python
def softmax(X):
    s = sum([exp(x) for x in X])
    return [exp(x) / s  for x in X]
```

… ok, now in Python

- There is no `sequential` in tf?
- How to stack layers?

Lets have a look at Keras

- https://keras.io/
- Is a simplified API on top of **TensorFlow**
- Provides easy access to some datasets (e.g. MNIST)

Keras Models

in `tensorflow.keras.models`

- The **Sequential model**, which is very straightforward (a simple list of layers), but is limited to single-input, single-output stacks of layers (as the name gives aways).
- The **Functional API**, which is an easy-to-use, fully-featured API that supports arbitrary model architectures. For most people and most use cases, this is what you should be using. This is the Keras "industry strength" model.
- **Model subclassing**, where you implement everything from scratch on your own. Use this if you have complex, out-of-the-box research use cases.

Regression with Tensorflow

- This time in Python

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model
model = Sequential()
model.add(Dense(units = 1, input_shape=[1,]))

# Prepare the model for training: Specify the loss
model.compile(loss='mse', optimizer="sgd")

# Generate some synthetic data for training.
xs = [1, 2, 3, 4, 10]
ys = [1, 4, 6, 8, 20]

# Train the model using the data.
model.fit(xs, ys, epochs=10)

testData = [5,9,11,19,21]
result = model.predict(testData)
```

… more Keras code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
# Stacking layers is as easy as .add():
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

# Once your model looks good, configure its learning process with .compile():
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# If you need to, you can further configure your optimizer. The Keras philosophy is to simple things simple,
# while allowing the user to be # fully in control when they need to (the ultimate control being the easy
# extensibility of the source code # via subclassing).
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.SGD(learning_rate=0.01)

# You can now iterate on your training data in batches:
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)

# Evaluate your test loss and metrics in one line:
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

# Or generate predictions on new data:
classes = model.predict(x_test, batch_size=128)
```

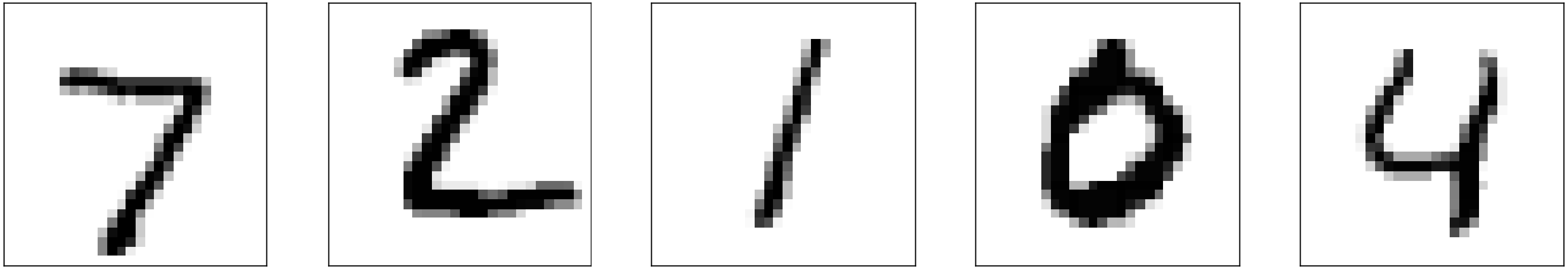
Code 1/2

Let's see how this works in running code!

GitLab

or

Local



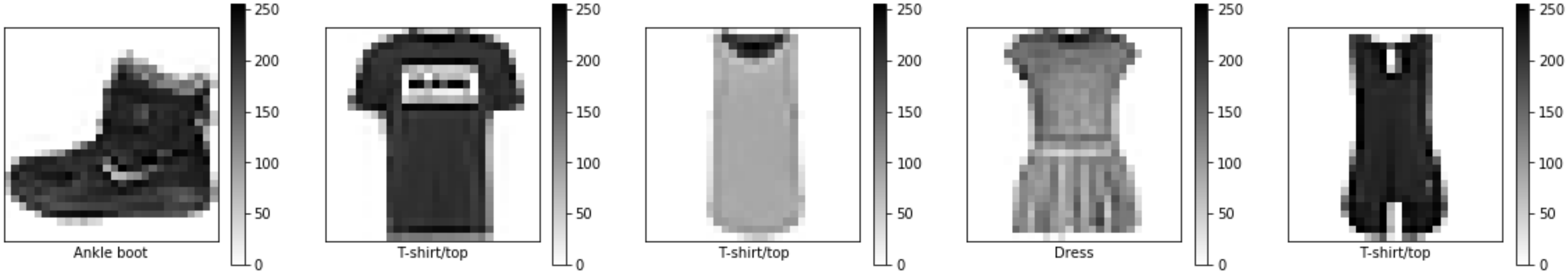
Code 2/2

A bit more fancy for exercise:

[Gitlab]

or

[Local]



Summary

Lessons learned today:

- Tensorflow and Keras
    - How to define DNN?
- Tensorflow.js