Machine Learning

Pawel Wocjan

University of Central Florida

Spring 2019

Layers: the building blocks of deep learning

- The fundamental data structure in neural networks is the **layer**.
- A layer is a data-processing module that takes as input one or more tensors and that outputs one or more tensors.
- Some layers are stateless, but more frequently layers have a state: the layers **weights**, one or several tensors learned with stochastic gradient descent, which together contain the network's **knowledge**.

Layers: the building blocks of deep learning

- Different layers are appropriate for different tensor formats and different types of data processing.
- Simple vector data, stored in 2D tensors of shape `(samples, features)` is often processed by **densely connected layers**, also called **fully connected layers** (the `Dense` class in Keras).
- Sequence data, stored in 3D tensors of shape `(samples, timesteps, features)`, is typically processed by **recurrent** layers such as **long-short term memory (LSTM)** layer.
- Image data, stored in 4D tensors, is usually processed by 2D **convolutional** layers (`Con2D`).

https://keras.io/layers/core/

https://keras.io/layers/convolutional/

https://keras.io/layers/recurrent/

Layers: the building blocks of deep learning

- You can think of layers as LEGO bricks of deep learning.
- Building deep-learning models in Keras is done by combining compatible layers to form useful data-processing pipelines.
- Layer compatibility means that every layer will only accept input tensors of a certain shape and will return output tensors of a certain shape.
- When using Keras, you don't have to worry about compatibility, because the layers you add to your model are dynamically built to match the shape of the incoming layer.

Layers: the building blocks of deep learning

```
from keras import models
from keras import layers

network = models.Sequential()

network.add(layers.Dense(512,
                         activation='relu',
                         input_shape=(28 * 28,)))

# no need to specify input_shape for second layer
network.add(layers.Dense(10,
                         activation='softmax'))
```

The second layer didn't receive an input shape argument – instead, it automatically inferred its input shape as being the output shape of the first layer.

Models: networks of layers

- A deep-learning model is a directed, acyclic graph of layers.
- The most common topology is a linear stack of layers, mapping a single input to a single output. These can be implemented using `models.Sequential()`.
- Initially, we will only work with linear stacks of layers.
- Later, we will also look at other network topologies such as two-branch networks, multi-head networks, and inception blocks.

https://keras.io/getting-started/sequential-model-guide/

Models: networks of layers

- The topology of a network defines a **hypothesis space**.
- By choosing a network topology, you constrain your **space of possibilities** (hypothesis space) to a specific series of tensor operations, mapping input data to output data.
- You'll be then searching for a good set of values for the weight tensor involved in these tensor operations using stochastic a variant of gradient descent.
- Picking the right network architecture is more art than a science. We will study explicit principles for building neural networks and develop intuition as to what works or doesn't for specific problems.

Loss functions & optimizers: keys to configuring the learning process

- Once the network architecture is defined, you still need to do two things:
  - **Loss function (objective function)**
    The quantity that will be minimized during training. It represents a measure of success for that task at hand.
    `https://keras.io/losses/`
  - **Optimizer** Determines how the network will be updated based on the loss function. Implements a specific variant of stochastic gradient descent (SGD).
    `https://keras.io/optimizers/`

Loss functions & optimizers: keys to configuring the learning process

- Choosing the right objective function for the right problem is extremely important: your network will take any shortcut it can, to minimize the loss.
- Fortunately, there are simple guidelines you can use to choose the correct loss for common problems such as classification, regression, and sequence prediction.

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multi-label classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	`None`	`mse`
Regression to values in $[0, 1]$	`sigmoid`	`mse` or `binary_crossentropy`