Basics of `keras`

Romain Tavenard

Specifying model architecture

The `Sequential` model class

- `Sequential`: easiest way to code a simple model in `keras`
- Requirements
  - Single input (can be multidimensional)
  - Single output (can be multidimensional)
  - model is just a stack of layers (no loop in the model graph)

```
from keras.models import Sequential

model = Sequential(
    [
        # Here, we will put a list of layers
    ]
)
```

Layers used to define an MLP

- `InputLayer` (not mandatory, takes an `input_shape` argument)
- `Dense` (link to docs)
  - Can be used for both internal and output layers
  - `activation` is the activation function (`str`)

```
from keras.models import Sequential
from keras.layers import InputLayer, Dense

model = Sequential(
    [
        # Input layer (16-dimensional)
        InputLayer(input_shape=(16, )),
        # Hidden layer (256 neurons, ReLU activation)
        Dense(units=256, activation="relu"),
        # Output layer (1 neuron, sigmoid activation)
        Dense(units=1, activation="sigmoid")
    ]
)
```

Fitting a model

Setting optimization strategy (1/2)

- `compile` step
  - `optimizer` (required)
    - can be a string (`"sgd"`, `"adam"`, ...)
    - can be a `keras` optimizer instance (allows to set learning rate, ...)
  - `loss` (required) can be a string
    - `"mse"`: Mean Squared Error
    - `"binary_crossentropy"`: Cross-entropy in the binary case (single output neuron)
    - `"categorical_crossentropy"`: Cross-entropy in the multiclass case (multiple output neurons)
  - `metrics` (optional) is a list
    - additional metrics to be reported during model training (*eg.* `"accuracy"` for classification tasks)

Setting optimization strategy (2/2)

```
model = Sequential(
    [
        # Layers here
    ]
)

model.compile(
    optimizer="sgd",
    loss="mse",
    metrics=[]
)
```

Setting `fit` options (1/2)

- `fit` step
  - First 2 arguments: training data (required)
  - `validation_data` (optional) is a tuple
    - validation set organized as a pair (X, y)
  - `epochs`: number of epochs (required)
  - `batch_size`: number of samples per batch (strongly recommended)

Setting `fit` options (2/2)

```
model = Sequential(
    [
        # Layers here
    ]
)

model.compile(
    # Compile options
)
model.fit(X, y,
          validation_data=(X_val, y_val),
          epochs=10,
          batch_size=64)
```

Callbacks

Callbacks in general

- Callbacks are ways to "interact" during training
  - *eg.* report statistics, early stopping, save models
  - at each minibatch or at each epoch
- A list of callbacks can be passed at fit time:

```
model.fit(X, y,
          validation_data=(X_val, y_val),
          epochs=10,
          batch_size=64,
          callbacks=[cb1, cb2, ...])
```

- See docs for usage (link)
  - `EarlyStopping`
  - `ModelCheckpoint`
  - `TensorBoard`

History callback

- Returned by `fit` by default (no need to set up)
- has a `history` attribute
  - `dict`
  - losses and metrics recorded during fit at each epoch

```
h = model.fit(X, y,
              validation_data=(X_val, y_val),
              epochs=10,
              batch_size=64)
print(h.history)
# {
#   "loss": [...],
#   "val_loss": [...],
#   "accuracy": [...],
#   "val_accuracy": [...]
# }
```