1 Convolutional Neural Networks

1.1 Sequential Models

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='sgd', metrics=['ac
```

1.1.1 Specifying Input Shape

The model needs to know what input shape it should expect. For this reason, the first layer in a `Sequential` model (and only the first, because following layers can do automatic shape inference) needs to receive information about its input shape. There are several possible ways to do this:

- Pass an `input_shape` argument to the first layer. This is a shape tuple (a tuple of integers or `None` entries, where `None` indicates that any positive integer may be expected). In `input_shape` , the batch dimension is not included.
- Some 2D layers, such as `Dense` , support the specification of their input shape via the argument `input_dim` , and some 3D temporal layers support the arguments `input_dim` and `input_length` .

In [ ]:
```
model = Sequential()
model.add(Dense(32, input_dim=784))
```

1.1.2 Compilation

Before training a model, you need to configure the learning process, which is done via the `compile` method. It receives three arguments:

- An optimizer. This could be the string identifier of an existing optimizer (such as `rmsprop` or `adagrad`), or an instance of the `Optimizer` class.
- A loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as `categorical_crossentropy` or `mse`), or it can be an objective function. See: losses.
- A list of metrics. For any classification problem you will want to set this to `metrics=['accuracy']` . A metric could be the string identifier of an existing metric or a custom metric function.

In [ ]:
```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')
```

1.1.3 Training

Keras models are trained on Numpy arrays of input data and labels. For training a model, you will typically use the `fit` function.

1.1.3.1 Single-Input, 2 Classes (Binary Classification)

In [ ]:
```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

1.1.3.2 Single-Input, 10 Classes (Categorical Classification)

In [ ]:

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(10, size=(1000, 1))

# Convert labels to categorical one-hot encoding
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

1.2 Optimizers

An optimizer is one of the two arguments required for compiling a Keras model:

- SGD
- RMSprop
- Adam

In [ ]:

```
from keras import optimizers

model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

1.2.1 SGD (Stoachastic Gradient Descent)

In [ ]:

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

1.2.1.1 Arguments

- `lr` : float >= 0. Learning rate.
- `momentum` : float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- `decay` : float >= 0. Learning rate decay over each update.
- `nesterov` : boolean. Whether to apply Nesterov momentum.

1.2.2 RMSProp

In [ ]:

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

Arguments

- `lr` : float >= 0. Learning rate.
- `rho` : float >= 0.
- `epsilon` : float >= 0. Fuzz factor. If None, defaults to K.epsilon().
- `decay` : float >= 0. Learning rate decay over each update.

1.2.3 Adam

In [ ]:

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, dec
```

1.2.3.1 Arguments

- `lr` : float >= 0. Learning rate.
- `beta_1` : float, 0 < beta < 1. Generally close to 1.
- `beta_2` : float, 0 < beta < 1. Generally close to 1.
- `epsilon` : float >= 0. Fuzz factor. If None, defaults to K.epsilon().
- `decay` : float >= 0. Learning rate decay over each update.
- `amsgrad` : boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

1.3 Activations

Activations can either be used through an `Activation` layer, or through the `activation` argument supported by all forward layers:

In [ ]:

```
from keras.layers import Activation, Dense

model.add(Dense(64))
model.add(Activation('tanh'))

# EQUIVALENT
model.add(Dense(64, activation='tanh'))
```

1.3.1 softmax

In [ ]:

```
keras.activations.softmax(x, axis=-1)
```

1.3.1.1 Arguments

- `x` : Input tensor.
- `axis` : Integer, axis along which the softmax normalization is applied.

1.3.2 relu (Rectified Linear Unit)

In [ ]:

```
keras.activations.relu(x, alpha=0.0, max_value=None, threshold=0.0)
```

With default values, it returns element-wise `max(x, 0)` .

Otherwise, it follows: `f(x) = max_value` for `x >= max_value` , `f(x) = x` for `threshold <= x < max_value` , `f(x) = alpha * (x - threshold)` otherwise.

1.3.2.1 Arguments

- `x` : Input tensor.
- `alpha` : float. Slope of the negative part. Defaults to zero.
- `max_value` : float. Saturation threshold.
- `threshold` : float. Threshold value for thresholded activation.

1.3.3 tanh

In [ ]:

```
keras.activations.tanh(x)
```

1.3.4 sigmoid

In [ ]:

```
keras.activations.sigmoid(x)
```