Deep Learning - lab 6

Keras callbacks and CNNs

Prof. Stefano Carrazza

University of Milan and INFN Milan

Keras Callbacks

Keras callbacks

A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

You can use callbacks to:

- Periodically save your model to disk
- Do early stopping
- Write TensorBoard logs after every batch of training to monitor your metrics
- Get a view on internal states and statistics of a model during training
- ...and more

Keras callback API

Example: storing model parameters during training

```
# save the weights manually
model = create_model()
model.fit(...)
model.save_weights('./checkpoints/my_checkpoint')

# restore weights manually
model = create_model()
model.load_weights('./checkpoints/my_checkpoint')
```

Keras callback API

Example: storing model parameters during training

```
from tensorflow import keras

model = ... # define your model as usual

# declare callbacks
my_callbacks = [
  keras.callbacks.ModelCheckpoint(filepath='model.{epoch}-{val_loss:.2f}.h5'),
]

model.fit(data, epochs=10, callbacks=my_callbacks)
```

ModelCheckpoint

`ModelCheckpoint` class

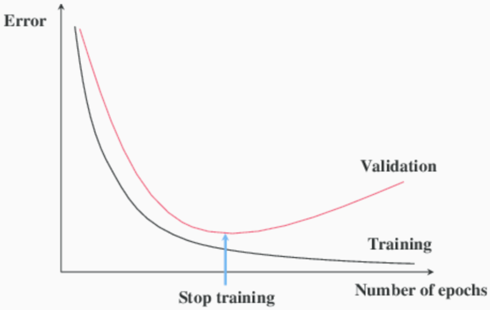
```
tf.keras.callbacks.ModelCheckpoint(
    filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=False,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
    options=None,
    initial_value_threshold=None,
    **kwargs
)
```

Callback to save the Keras model or model weights at some frequency.

`ModelCheckpoint` callback is used in conjunction with training using `model.fit()` to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

Keras callback API

Example: early stopping




```python
from tensorflow import keras

model = ... # define your model

# declare callbacks
my_callbacks = [
  keras.callbacks.EarlyStopping(patience=2),
]

model.fit(data, epochs=10,
          callbacks=my_callbacks)
```

EarlyStopping

`EarlyStopping` class

```
tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=0,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

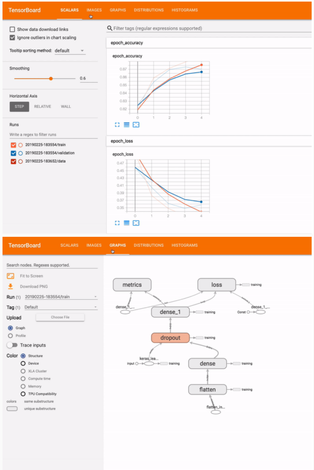
Stop training when a monitored metric has stopped improving.

Assuming the goal of a training is to minimize the loss. With this, the metric to be monitored would be `'loss'`, and mode would be `'min'`. A `model.fit()` training loop will check at end of every epoch whether the loss is no longer decreasing, considering the `min_delta` and `patience` if applicable. Once it's found no longer decreasing, `model.stop_training` is marked True and the training terminates.

Keras callback API

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs



Keras callback API

Example: storing logs for TensorBoard

```
from tensorflow import keras

model = ... # define your model

# declare callbacks
my_callbacks = [
  keras.callbacks.TensorBoard(log_dir='./logs'),
  keras.callbacks.EarlyStopping(patience=2),
]

model.fit(data, epochs=10, callbacks=my_callbacks)

# open tensorboard with
# tensorboard --logdir=path_to_your_logs
```

TensorBoard

`TensorBoard` class

```
tf.keras.callbacks.TensorBoard(
    log_dir="logs",
    histogram_freq=0,
    write_graph=True,
    write_images=False,
    write_steps_per_second=False,
    update_freq="epoch",
    profile_batch=0,
    embeddings_freq=0,
    embeddings_metadata=None,
    **kwargs
)
```

Enable visualizations for TensorBoard.

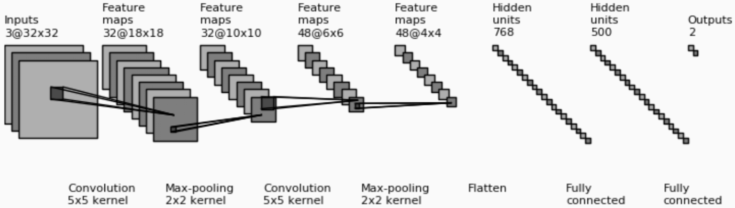
TensorBoard is a visualization tool provided with TensorFlow.

This callback logs events for TensorBoard, including:

- Metrics summary plots
- Training graph visualization
- Weight histograms
- Sampled profiling

Convolutional Neural Nets

Convolutional base




```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(...)
```

Conv2D layer

`Conv2D` class

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

MaxPooling2D layer

`MaxPooling2D` class

```
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid", data_format=None, **kwargs
)
```

Max pooling operation for 2D spatial data.

Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by `pool_size`) for each channel of the input. The window is shifted by `strides` along each dimension.

The resulting output, when using the `"valid"` padding option, has a spatial shape (number of rows or columns) of: `output_shape = math.floor((input_shape - pool_size) / strides) + 1` (when `input_shape >= pool_size`)

The resulting output shape when using the `"same"` padding option is: `output_shape = math.floor((input_shape - 1) / strides) + 1`

Example: CNN classifier

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```