Usage of optimizers

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

```
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)
```

You can either instantiate an optimizer before passing it to `model.compile()`, as in the above example, or you can call it by its name. In the latter case, the default parameters for the optimizer will be used.

```
# pass optimizer by name: default parameters will be used
model.compile(loss='mean_squared_error', optimizer='sgd')
```

Parameters common to all Keras optimizers

The parameters `clipnorm` and `clipvalue` can be used with all optimizers to control gradient clipping:

```
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum norm of 1.
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
```

```
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum value of 0.5 and
# a minimum value of -0.5.
sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)
```

SGD [source]

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

Stochastic gradient descent optimizer.

Includes support for momentum, learning rate decay, and Nesterov momentum.

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.

RMSprop [source]

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

RMSProp optimizer.

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.

References

- rmsprop: Divide the gradient by a running average of its recent magnitude

---

Adagrad [source]

```
keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
```

Adagrad optimizer.

Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

It is recommended to leave the parameters of this optimizer at their default values.

Arguments

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

References

- Adaptive Subgradient Methods for Online Learning and Stochastic Optimization

---

Adadelta [source]

```
keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
```

Adadelta optimizer.

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. Compared to Adagrad, in the original version of Adadelta you don't have to set an initial learning rate. In this version, initial learning rate and decay factor can be set, as in most other Keras optimizers.

It is recommended to leave the parameters of this optimizer at their default values.

Arguments

- **lr**: float >= 0. Initial learning rate, defaults to 1. It is recommended to leave it at the default value.
- **rho**: float >= 0. Adadelta decay factor, corresponding to fraction of gradient to keep at each time step.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Initial learning rate decay.

References

- Adadelta - an adaptive learning rate method

Adam [source]

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=Fals
```

Adam optimizer.

Default parameters follow those provided in the original paper.

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".

References

- Adam - A Method for Stochastic Optimization
- On the Convergence of Adam and Beyond

Adamax [source]

```
keras.optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)
```

Adamax optimizer from Adam paper's Section 7.

It is a variant of Adam based on the infinity norm. Default parameters follow those provided in the paper.

Arguments

- **lr**: float >= 0. Learning rate.
- **beta_1/beta_2**: floats, 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.

References

- Adam - A Method for Stochastic Optimization

Nadam [source]

```
keras.optimizers.Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, schedule_decay=0.004)
```

Nesterov Adam optimizer.

Much like Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum.

Default parameters follow those provided in the paper. It is recommended to leave the parameters of this optimizer at their default values.

Arguments

- **lr**: float >= 0. Learning rate.
- **beta_1/beta_2**: floats, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.

References

- Nadam report
- On the importance of initialization and momentum in deep learning

---

TFOptimizer [source]

```
keras.optimizers.TFOptimizer(optimizer)
```

Wrapper class for native TensorFlow optimizers.