[**https://keras.io/optimizers/**](https://keras.io/optimizers/)

|  |
| --- |
| """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. |
|  | """ |

[**https://github.com/keras-team/keras/blob/master/keras/optimizers.py#L157**](https://github.com/keras-team/keras/blob/master/keras/optimizers.py#L157)

**Nesterov Momentum**

There's a good description of Nesterov Momentum (aka Nesterov Accelerated Gradient) properties in, for example, [Sutskever, Martens et al."On the importance of initialization and momentum in deep learning" 2013](http://proceedings.mlr.press/v28/sutskever13.pdf).

The main difference is in classical momentum you first correct your velocity and then make a big step according to that velocity (and then repeat), but in Nesterov momentum you first making a step into velocity direction and then make a correction to a velocity vector based on new location (then repeat).

i.e. Classical momentum:

vW(t+1) = momentum.\*Vw(t) - scaling .\* gradient\_F( W(t) )

W(t+1) = W(t) + vW(t+1)

While Nesterov momentum is this:

vW(t+1) = momentum.\*Vw(t) - scaling .\* gradient\_F( W(t) + momentum.\*vW(t) )

W(t+1) = W(t) + vW(t+1)

Actually, this makes a huge difference in practice...

Link: <https://stats.stackexchange.com/questions/179915/whats-the-difference-between-momentum-based-gradient-descent-and-nesterovs-acc>

[**https://keras.io/losses/**](https://keras.io/losses/)

**Note**: when using the categorical\_crossentropy loss, your targets should be in categorical format (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is all-zeros except for a 1 at the index corresponding to the class of the sample). In order to convert integer targets into categorical targets, you can use the Keras utility to\_categorical:

**from** keras.utils **import** to\_categorical

categorical\_labels = to\_categorical(int\_labels, num\_classes=**None**)