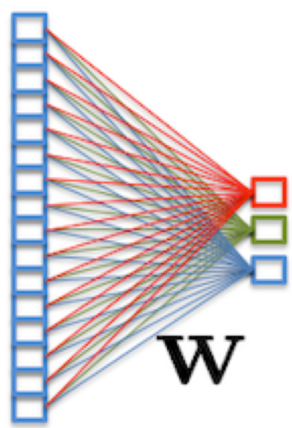
5IF - Deep Learning et Programmation Différentielle

2.6 Stochastic Gradient Descent



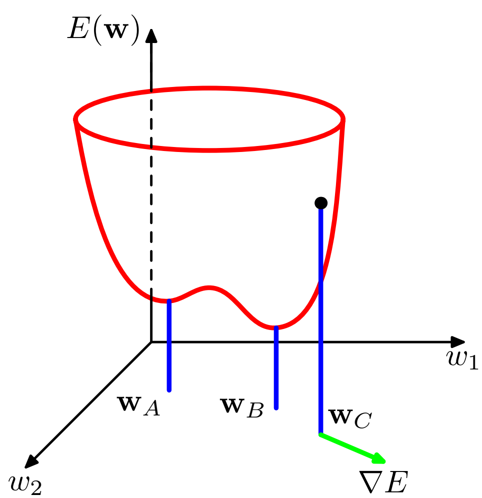
Christian Wolf

Learning by gradient descent

Iterative minimisation through **gradient descent:**

$$\theta^{[t+1]} = \theta^{[t]} + \boxed{\nu} \nabla \mathcal{L}(h(x,\theta), y^*)$$

Learning rate



[Figure: C. Bishop, 2006]

Can be blocked in a local minimum (not that it matters much …)

Stochastic Gradient Descent

The vanilla version of SGD:

$$\theta^{[t+1]} = \theta^{[t]} - \nu \nabla$$

where ν is the learning rate (a hyper-parameter).

The learning rate has a big impact on convergence and convergence speed:

- Low ν: slow convergence
- High ν: overshoot target

$\Rightarrow$ decay learning rate during learning, e.g. divide by two every X epochs.

The following slides are partially inspired by

`http://cs231n.github.io/neural-networks-3/`

SGD with Momentum

Momentum tends to maintains gradient direction between updates:

$$\begin{aligned}\mathbf{v}^{[t+1]} &= \mu \mathbf{v}^{[t]} - \nu \nabla \\ \theta^{[t+1]} &= \theta^{[t]} + \mathbf{v}^{[t+1]}\end{aligned}$$

where μ is a new hyper-parameter.
Typical values: $\mu = 0.5, 0.9, 0.95, 0.99$.

Nesterov's Accelerated Momentum

Nesterov's Accelerated Momentum calculates the gradient ∇ at the position $\tilde{\theta}^{[t+1]}$ at which momentum alone would have brought it:

$$\begin{aligned}
\tilde{\theta}^{[t+1]} &= \theta^{[t+1]} + \mu \mathbf{v}^{[t]} \\
\mathbf{v}^{[t+1]} &= \mu \mathbf{v}^{[t]} - \nu \nabla \tilde{\theta}^{[t+1]} \\
\theta^{[t+1]} &= \theta^{[t]} + \mathbf{v}^{[t+1]}
\end{aligned}$$

`Y. Nesterov. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). Soviet Mathematics Doklady, 1983`

Adaptive learning rates: Adagrad

Adagrad keeps a variable vector $\boldsymbol{c}$ holding sums of squared derivatives, per gradient element:

$$
\begin{aligned}
\boldsymbol{c}^{[t+1]} &= \boldsymbol{c}^{[t]} + \nabla^2 \\
\theta^{[t+1]} &= \theta^{[t]} - \nu \frac{\nabla}{\sqrt{\boldsymbol{c}^{[t+1]}} + \epsilon}
\end{aligned}
$$

where ϵ is small and ensures numerical stability.

Effect: a large gradient value will lead to lower effective learning rate for a given parameter.

J. Duchi, E. Hazan, Y. Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, JMLR, 2011.

Adaptive learning rates: RMSProp

RMSProp keeps a running average instead of accumulated gradients:

$$
\begin{aligned}
\boldsymbol{c}^{[t+1]} &= \beta \boldsymbol{c}^{[t]} + (1-\beta)\nabla^2 \\
\theta^{[t+1]} &= \theta^{[t]} - \nu \frac{\nabla}{\sqrt{\boldsymbol{c}^{[t+1]}} + \epsilon}
\end{aligned}
$$

`G. Hinton, unpublished.`

Adaptive learning rates: ADAM

The ADAM update rule is similar to RMSProp, but smoothes the momentum term:

$$
\begin{aligned}
\boldsymbol{m}^{[t+1]} &= \beta_1 * \boldsymbol{m}^{[t]} + (1-\beta_1)\nabla \\
\boldsymbol{v}^{[t+1]} &= \beta_2 * \boldsymbol{v}^{[t]} + (1-\beta_2)\nabla^2 \\
\theta^{[t+1]} &= \theta^{[t]} - \nu \frac{\boldsymbol{m}^{[t+1]}}{\sqrt{\boldsymbol{v}^{[t+1]}}+\epsilon}
\end{aligned}
$$

Typical values of the hyper-parameters:
$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$

`D.P. Kingma, J. Ba, Adam: A method for stochastic optimization. Machine Learning, 2014`

Adaptive learning rates: ADAM

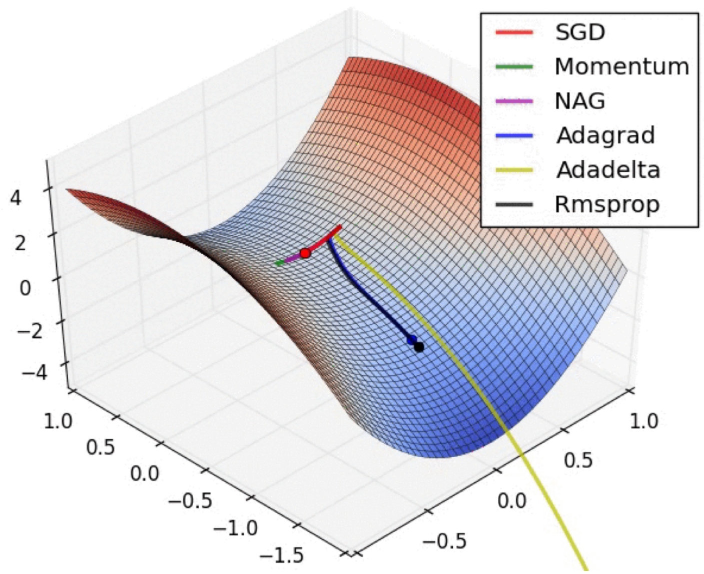
The ADAM update rule with bias correction decreases the effect of initialization to zero (bias):

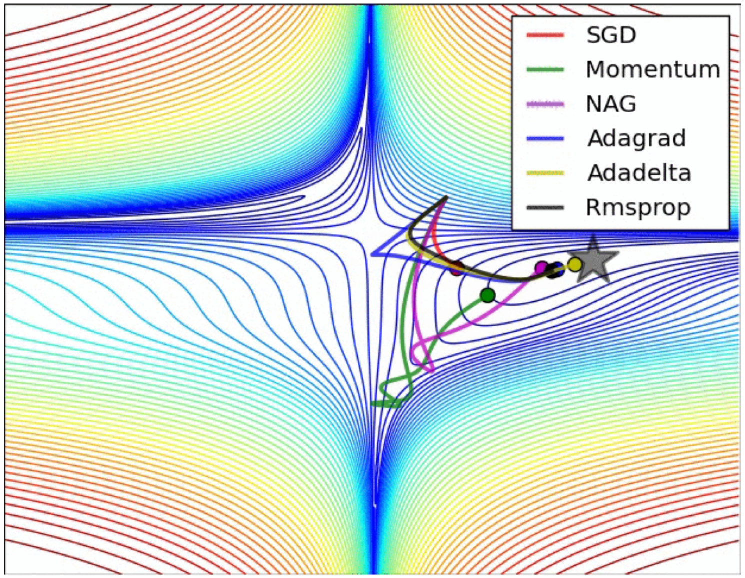
$$\begin{aligned}
\tilde{\boldsymbol{m}}^{[t+1]} &= \beta_1 * \boldsymbol{m}^{[t]} + (1-\beta_1)\nabla \\
\boldsymbol{m}^{[t+1]} &= \frac{\tilde{\boldsymbol{m}}^{[t+1]}}{1-\beta_1^t} \\
\tilde{\boldsymbol{v}}^{[t+1]} &= \beta_2 * \boldsymbol{v}^{[t]} + (1-\beta_2)\nabla^2 \\
\boldsymbol{v}^{[t+1]} &= \frac{\tilde{\boldsymbol{v}}^{[t+1]}}{1-\beta_2^t} \\
\theta^{[t+1]} &= \theta^{[t]} - \nu \frac{\boldsymbol{m}^{[t+1]}}{\sqrt{\boldsymbol{c}^{[t+1]}} + \epsilon}
\end{aligned}$$

Remark: $x^{[t]}$ indexes iteration t; x^t denotes x to the power of t.

`D.P. Kingma, J. Ba, Adam: A method for stochastic optimization. Machine Learning, 2014`

Visualization





[Animations: Alex Radford, Open-AI]

http://cs231n.github.io/neural-networks-3/

Learning rates

If you are unsure, use ADAM but also try SGD.

Even the adaptive methods use global learning rates, which need to be set.

Recall:

- Low ν: slow convergence
- High ν: overshoot target

$\Rightarrow$ decay learning rate during learning, e.g. divide by two every X epochs.

Experiments

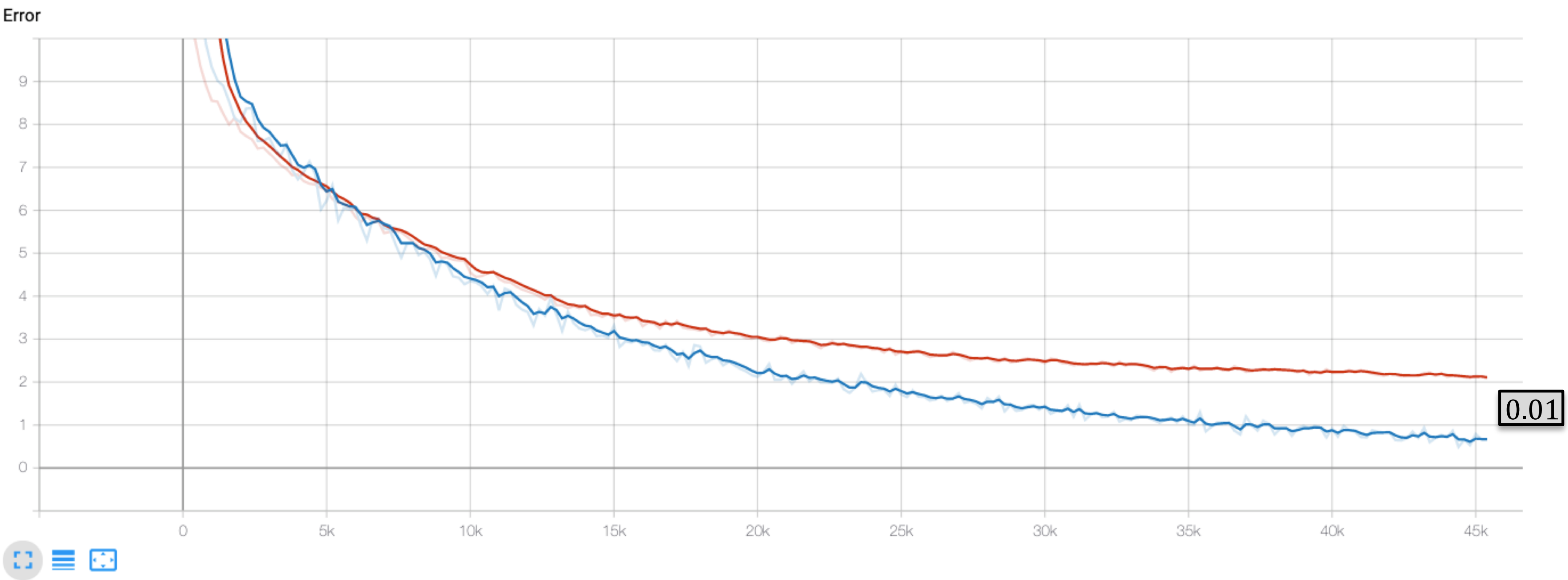
We will reuse our 2 layer MLP with 2000 hidden units and ReLU activation and optimize it with SGD and different learning rates on MNIST:

```
class MLP(torch.nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = torch.nn.Linear(28*28, 200)
        self.fc2 = torch.nn.Linear(200, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        return self.fc2(x)

model = MLP()
crossentropy = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(),
    lr=0.01)
```

Learning rate

The impact of learning rates

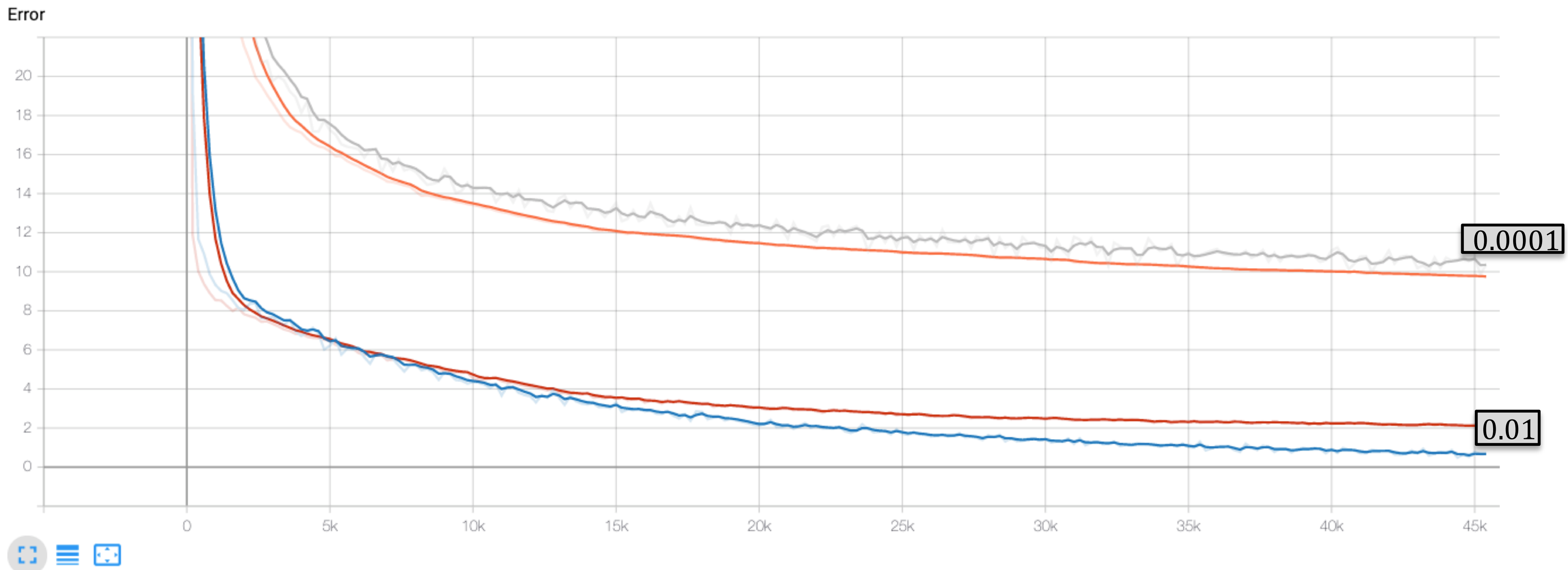
A well chosen learning rate of 0.01:



The impact of learning rates

We add the curves for a low learning rate 0.0001:

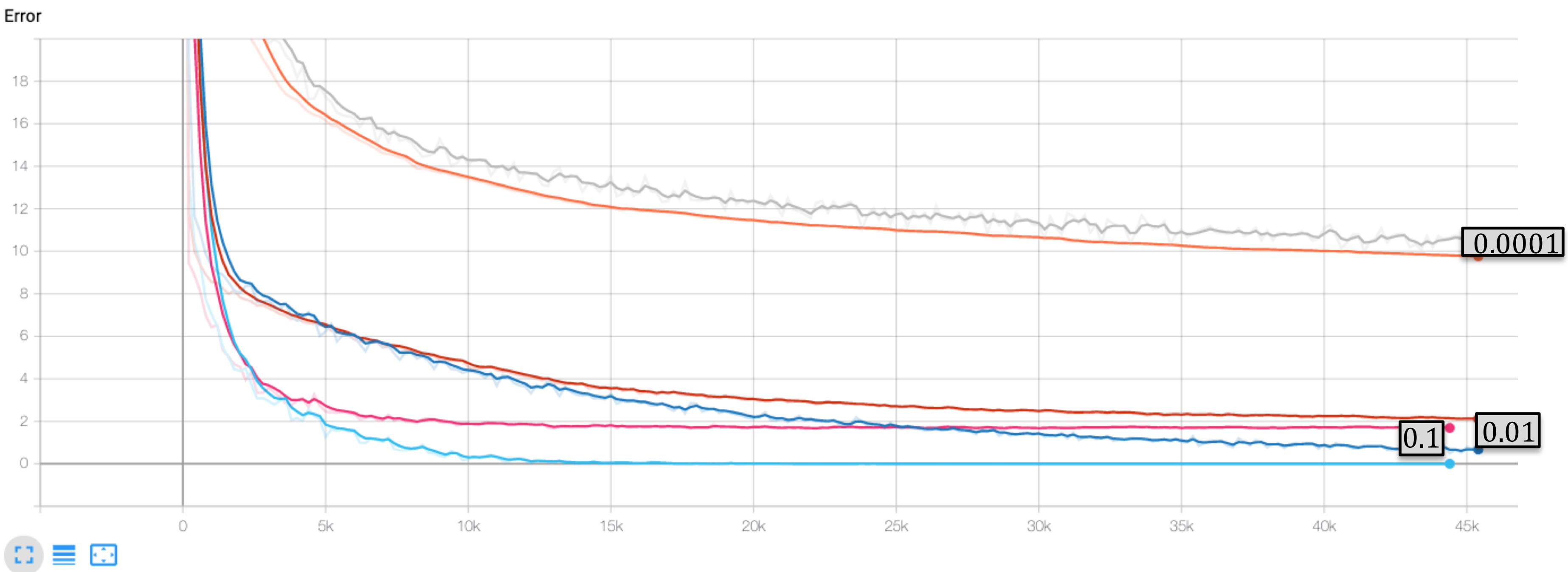
Slow convergence.



The impact of learning rates

We add the curves for a high learning rate 0.1:

Convergence is fast at the beginning but fails to find a good optimum at the end.



The impact of learning rates

We add the curves for a ridiculously high learning rate 1:
Oscillations start to appear (convergence problems).