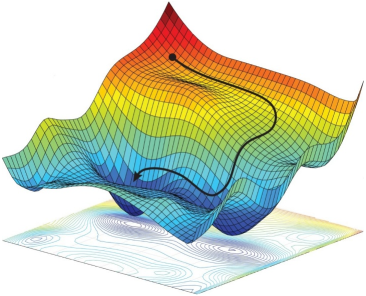
Neural network optimization

Victor Kitov

v.v.kitov@yandex.ru



Basic gradient methods

- **Batch gradient descent**: gradient descent using all objects
  - slow for big data
  - not applicable for dynamic data
  - gets stuck in local optima and inflection points (as all gradient based methods)

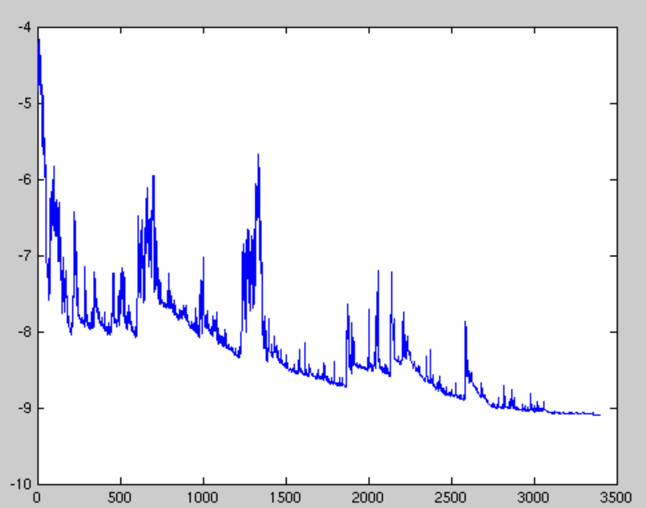
$$w_{t+1} := w_t - \eta \nabla L(\theta; X, Y)$$

- **Stochastic gradient descent**: stochastic descent with sampling one object

$$w_{t+1} := w_t - \eta_t \nabla L(\theta; x_i, y_i)$$

  - requires $\eta_t \to 0$
  - unstable gradient estimate

SGD convergence example



Basic gradient methods

- **Minibatch stochastic gradient descent**: stochastic descent with sampling a set of objects

$$w_{t+1} := w_t - \eta_t L(\theta; x_{i+1:i+K}, y_{i+1:i+K})$$

- more accurate gradient estimates
  - faster: computations parallelization over objects in the minibatch
- Difficulties:
  - requires $\eta_t \to 0$
  - the same step for different weights
    - better to take less weight, where the function changes sharply.

Momentum and Nesterov momentum

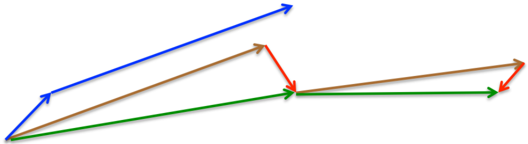
- **Momentum** ($\gamma > 0,\ \eta > 0$ - hyperparameters)

$$v_t := \gamma v_{t-1} + \eta \nabla_\theta L(\theta)$$
$$w_{t+1} := w_t - v_t$$

  - an analogy with a ball rolling down a mountain
  - does not stop in small local optima

- **Nesterov Accelerated Gradient** (Nesterov Momentum)

$$v_t := \gamma v_{t-1} + \eta \nabla_\theta L(\theta - \gamma v_{t-1})$$
$$w_{t+1} := w_t - v_t$$



Modifications of SGD

- Denote $g_t = \nabla L(\theta_t)$; $\theta, g_t \in \mathbb{R}^K$. Vector operations are elementwise
- **AdaGrad** ($\varepsilon = 10^{-6}$)

$$G_t := G_t + g_t^2$$
$$w_{t+1} := w_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} g_t$$

- **RMSprop**

$$E\left[g^2\right]_t := \gamma E\left[g^2\right]_{t-1} + (1-\gamma)\, g_t^2$$
$$w_{t+1} := w_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \varepsilon}} g_t$$

Modifications of SGD

- **Adam**=RMSprop+momentum
  ($\beta_1 = 0.9,\ \beta_2 = 0.999,\ \varepsilon = 10^{-8}$):

$$m_t := \beta_1 m_{t-1} + (1-\beta_1)\, g_1$$
$$v_t := \beta_2 v_{t-1} + (1-\beta_2)\, g_1^2$$
$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$w_{t+1} := w_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \varepsilon}\widehat{m}_t$$

- **Nadam**: Adam+Nesterov Accelerated Gradient.

Modifications of SGD

- **AMSGrad**: remembers gradients without exponential forgetting.
  - renormalization m_t, v_t is not applied and needed.

$$m_t := \beta_1 m_{t-1} + (1-\beta_1)\, g_1$$
$$v_t := \beta_2 v_{t-1} + (1-\beta_2)\, g_1^2$$
$$\widehat{v}_t = \max\left(\widehat{v}_{t-1}, v_t\right)$$
$$w_{t+1} := w_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \varepsilon} \widehat{m}_t$$

Additional improvements[1]

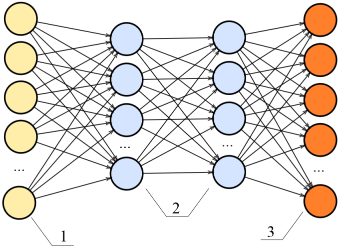
- **Early stopping** - combats overfitting.
- Adding noise to the gradient allows finding the optimum with a larger neighborhood:

$$g_t := g_t + \mathcal{N}(0, \sigma_t^2)$$
$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

- **Curriculum learning**
    - first learn on simple objects, then on complex ones
- Improving speed of convergence: batch normalization.

[1] More info: https://ruder.io/deep-learning-optimization-2017/

BatchNorm: motivation



- SGD $w := w - \varepsilon \nabla_w \mathcal{L}(x, y)$ updates all weights on all layers simultaneously.
- Distribution of outputs changes and later layers have to relearn from scratch.
- Also input can shift to saturation region of non-linearity.

BatchNorm: intro

- Idea: standardize outputs at intermediate layers

$$\tilde{x}_k = \frac{x_k - \mu_k}{\sigma_k}, \quad \mu_k = \mathbb{E}x_k, \sigma_k = \sqrt{Var(x_k)}$$

  - guarantees $\mathbb{E}\tilde{x}_k = 0, \ \mathrm{Var}\,\tilde{x}_k = 1$ after weight updates on previous layers.
    - training is faster for later layers
- Training:
  - problem: don't know μ_k, σ_k
    - they change dynamically with weight updates
  - solution: estimate them on current minibatch (should be large enough)
- Inference:
  - since now distribution of x_k is fixed, can estimate μ_k, σ_k from the whole training set.
  - more efficient: average estimates of μ_k, σ_k from final minibatches of training.

BatchNorm: actual version

$$\tilde{x}_k = \alpha_k \frac{x_k - \mu_k}{\sqrt{\sigma_k^2 + \varepsilon}} + \beta_k, \quad \mu_k = \bar{x}_k,\ \sigma_k = \sqrt{Var(x_k)},\ \varepsilon = 10^{-6}.$$

- Training:
  - μ_k, σ_k estimated from the minibatch
  - α_k, β_k - output std.dev. and mean.
    - learned during backpropagation
  - motivation:
    - may cancel normalization effect (e.g. in image->time prediction)
    - flexibility to adjust better to non-linearity
- Inference:
  - μ_k, σ_k estimated fixed to be mean, std.dev. from a wide set of objects.
  - α_k, β_k fixed.