Tricks and optimization in deep learning

Master’s Computer Vision

Sergey Nikolenko, Alex Davydow

Harbour Space University, Barcelona

May 20, 2020

Random facts:

- May 20 is celebrated as the World Bee Day; Anton Janša, the pioneer of beekeeping, was born on May 20, 1734
- On May 20, 1498, Vasco da Gama arrived at Kozhikode (Calicut), India
- on May 20, 1609, Thomas Thorpe first published Shakespeare’s sonnets in London; it was most probably an illicit publication, as was the custom at the time
- On May 20, 1873, Levi Strauss and Jacob Davis patented blue jeans with copper rivets
- On May 20, 2019, the international system of units (SI) was restructured, and the international prototype of the kilogram (IPK), stored in an underground vault near Paris, lost its importance; the IPK had been a problem because no one could explain why its mass and the mass of its copies deviated by tens of micrograms over the last century, and the whole SI was built upon the IPK

Weight initialization

Weight initialization

- The deep learning revolution began with *unsupervised pretraining*.
- Main idea: get to a good region of the search space, then fine-tune with gradient descent.
- Turns out by now we don’t need unsupervised pretraining with complex models like RBM to get to a good region.
- Weight initialization is an important part of why.

Weight initialization

- *Xavier initialization* (Glorot, Bengio, 2010).
- Let's consider a single linear unit:

$$y = \mathbf{w}^\top \mathbf{x} + b = \sum_i w_i x_i + b.$$

- The variance is

$$\begin{aligned}\mathrm{Var}\left[y_i\right] = \mathrm{Var}\left[w_i x_i\right] = \mathbb{E}\left[X^2 Y^2\right] - \left(\mathbb{E}\left[XY\right]\right)^2 = \\ = \mathbb{E}\left[x_i\right]^2 \mathrm{Var}\left[w_i\right] + \mathbb{E}\left[w_i\right]^2 \mathrm{Var}\left[x_i\right] + \mathrm{Var}\left[w_i\right] \mathrm{Var}\left[x_i\right].\end{aligned}$$

Weight initialization

- The variance is

$$\begin{aligned}\mathrm{Var}\left[y_i\right] = \mathrm{Var}\left[w_i x_i\right] = \mathbb{E}\left[X^2 Y^2\right] - \left(\mathbb{E}\left[XY\right]\right)^2 = \\ = \mathbb{E}\left[x_i\right]^2 \mathrm{Var}\left[w_i\right] + \mathbb{E}\left[w_i\right]^2 \mathrm{Var}\left[x_i\right] + \mathrm{Var}\left[w_i\right] \mathrm{Var}\left[x_i\right].\end{aligned}$$

- For symmetric activation functions and zero mean of the weights

$$\mathrm{Var}\left[y_i\right] = \mathrm{Var}\left[w_i\right] \mathrm{Var}\left[x_i\right].$$

- And if w_i and x_i are initialized independently from the same distribution,

$$\mathrm{Var}\left[y\right] = \mathrm{Var}\left[\sum_{i=1}^{n_{\mathrm{out}}} y_i\right] = \sum_{i=1}^{n_{\mathrm{out}}} \mathrm{Var}\left[w_i x_i\right] = n_{\mathrm{out}} \mathrm{Var}\left[w_i\right] \mathrm{Var}\left[x_i\right].$$

- In other words, the output variance is proportional to the input variance with coefficient $n_{\mathrm{out}} \mathrm{Var}\left[w_i\right]$.

Weight initialization

- Before (Glorot, Bengio, 2010), the standard way to initialize was (it's all over older literature)

$$w_i \sim U\left[-\frac{1}{\sqrt{n_{\mathrm{out}}}}, \frac{1}{\sqrt{n_{\mathrm{out}}}}\right].$$

- So in this case we get

$$\mathrm{Var}\left[w_i\right] = \frac{1}{12}\left(\frac{1}{\sqrt{n_{\mathrm{out}}}} + \frac{1}{\sqrt{n_{\mathrm{out}}}}\right)^2 = \frac{1}{3n_{\mathrm{out}}}, \text{ so}$$

$$n_{\mathrm{out}}\mathrm{Var}\left[w_i\right] = \frac{1}{3},$$

and after a few layers the signal dies down; the same happens in backprop.

Weight initialization

- Xavier initialization tries to reduce the change in variance, so we take
$$\mathrm{Var}\left[w_i\right] = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}},$$
which for uniform distribution means
$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{\mathrm{in}} + n_{\mathrm{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\mathrm{in}} + n_{\mathrm{out}}}}\right].$$
- But it only works for symmetric activations, i.e., not for ReLU...

Weight initialization

- …until (He et al., 2015)! Let's go back to

$$\mathrm{Var}\left[w_i x_i\right] = \mathbb{E}\left[x_i\right]^2 \mathrm{Var}\left[w_i\right] + \mathbb{E}\left[w_i\right]^2 \mathrm{Var}\left[x_i\right] + \mathrm{Var}\left[w_i\right] \mathrm{Var}\left[x_i\right]$$

- We now can only make the second term zero:

$$\mathrm{Var}\left[w_i x_i\right] = \mathbb{E}\left[x_i\right]^2 \mathrm{Var}\left[w_i\right] + \mathrm{Var}\left[w_i\right] \mathrm{Var}\left[x_i\right] = \mathrm{Var}\left[w_i\right] \mathbb{E}\left[x_i^2\right], \text{ so}$$

$$\mathrm{Var}\left[y^{(l)}\right] = n_{\mathrm{in}}^{(l)} \mathrm{Var}\left[w^{(l)}\right] \mathbb{E}\left[\left(x^{(l)}\right)^2\right].$$

Weight initialization

- We now can only make the second term zero:

$$\mathrm{Var}\left[y^{(l)}\right] = n_{\mathrm{in}}^{(l)} \mathrm{Var}\left[w^{(l)}\right] \mathbb{E}\left[\left(x^{(l)}\right)^2\right].$$

- Suppose now that $x^{(l)} = \max(0, y^{(l-1)})$, and $y^{(l-1)}$ has a symmetric distribution around zero. Then

$$\mathbb{E}\left[\left(x^{(l)}\right)^2\right] = \frac{1}{2}\mathrm{Var}\left[y^{(l-1)}\right], \quad \mathrm{Var}\left[y^{(l)}\right] = \frac{n_{\mathrm{in}}^{(l)}}{2}\mathrm{Var}\left[w^{(l)}\right]\mathrm{Var}\left[y^{(l-1)}\right].$$

- And this leads to the variance for ReLU init; there is no n_{out} now:

$$\mathrm{Var}\left[w_i\right] = 2/n_{\mathrm{in}}^{(l)}.$$

- You don't have to make it uniform, btw; e.g., a normal distribution is fine: $w_i \sim \mathcal{N}\left(0, \sqrt{2/n_{\mathrm{in}}^{(l)}}\right)$.

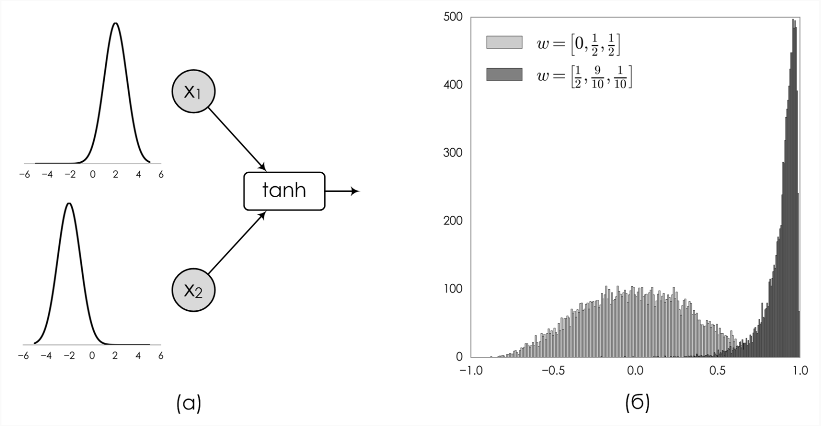
Batch normalization

Batch normalization

- Important problem in deep neural networks: *internal covariate shift*.
- When we change the weights of a layer, the distribution of its outputs changes.
- This means that the next layer has to re-train almost from scratch, it did not expect these outputs.
- Moreover, these neurons might have already reached saturation, so they can't re-train quickly.
- This seriously impedes learning.

Batch normalization

- A characteristic example; note how different the distributions are:



- What can we do?

Batch normalization

- We could try to normalize (whiten) after every layer.
- Does not work: consider a layer that simply adds a bias b to its inputs $\boldsymbol{u}$:
$$\hat{\mathrm{x}} = \mathrm{x} - \mathbb{E}[\mathrm{x}], \text{ where } \mathrm{x} = \boldsymbol{u} + b.$$
- On the next gradient descent step, we'll have $b := b + \Delta b$...
- ...but $\hat{\mathrm{x}}$ will not change:
$$\boldsymbol{u} + b + \Delta b - \mathbb{E}[\boldsymbol{u} + b + \Delta b] = \boldsymbol{u} + b - \mathbb{E}[\boldsymbol{u} + b].$$
- So the biases will simply increase unboundedly, and that's all the training we'll get; not a good thing.

Batch normalization

- We can try to add normalization as a layer:

$$\hat{\mathsf{x}} = \mathrm{Norm}(\mathsf{x}, \mathcal{X}).$$

- But note that the entire dataset $\mathcal{X}$ is required here.
- So on the gradient descent step we'll need to compute $\frac{\partial \mathrm{Norm}}{\partial \mathsf{x}}$ and $\frac{\partial \mathrm{Norm}}{\partial \mathcal{X}}$, and also the covariance matrix

$$\mathrm{Cov}[\mathsf{x}] = \mathbb{E}_{\mathsf{x} \in \mathcal{X}}\left[\mathsf{x}\mathsf{x}^\top\right] - \mathbb{E}\left[\mathsf{x}\right]\mathbb{E}\left[\mathsf{x}\right]^\top.$$

- Definitely won't work.

Batch normalization

- The solution is to normalize each component separately, and not over the whole dataset but over the current mini-batch; hence *batch normalization*.
- After batch normalization we get

$$\hat{x}_k = \frac{x_k - \mathbb{E}\left[x_k\right]}{\sqrt{\mathrm{Var}\left[x_k\right]}},$$

  where the statistics are computed over the current mini-batch.
- However, one more problem: now nonlinearities disappear!
- E.g., we will almost always get into the region where σ is very close to linear.

Batch normalization

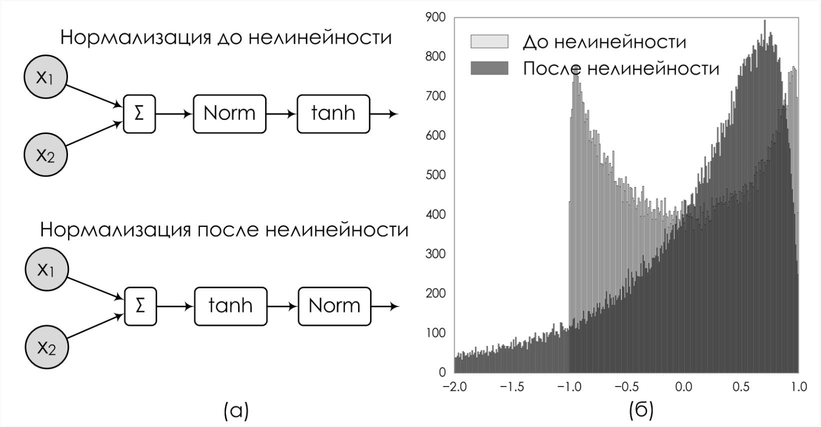
- To fix this, we have to allow the batchnorm layer enough flexibility to sometimes do *nothing* with the inputs.
- So we introduce additional shift and scale parameters:

$$y_k = \gamma_k \hat{x}_k + \beta_k = \gamma_k \frac{x_k - \mathbb{E}\left[x_k\right]}{\sqrt{\mathrm{Var}[x_k]}} + \beta_k.$$

- γ_k and β_k are new variables and will be trained just like the weights.

Batch normalization

- Last remark: it matters where to put the batchnorm.
- You can put it either before or after the nonlinearity.



Variations of gradient descent

Gradient descent

- “Vanilla” gradient descent:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha \nabla f(\mathbf{x}_k).$$

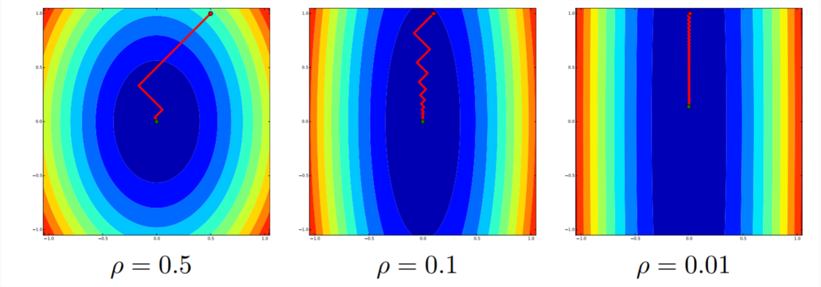
- So much depends on the learning rate α.
- First idea — let α decrease with time:
    - linear decay:
    $$\alpha = \alpha_0 \left(1 - \frac{t}{T}\right);$$
    - exponential decay:
    $$\alpha = \alpha_0 e^{-\frac{t}{T}}.$$

Gradient descent

- There're a lot of results here. Wolfe conditions: if we are solving $\min_{\mathbf{x}} f(\mathbf{x})$, and on step k we already know the direction $\mathbf{p}_k$ where to go (e.g., $\mathbf{p}_k = \nabla_{\mathbf{x}} f(\mathbf{x}_k)$), i.e., we need to solve $\min_\alpha f(\mathbf{x}_k + \alpha \mathbf{p}_k)$, then:
    - for $\phi_k(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ we have $\phi'_k(\alpha) = \nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^\top \mathbf{p}_k$, and if $\mathbf{p}_k$ is the descent direction then $\phi'_k(0) < 0$;
    - the step size α must satisfy the Armijo rule:
    $$\phi_k(\alpha) \leq \phi_k(0) + c_1 \alpha \phi'_k(0) \text{ for some } c_1 \in (0, \frac{1}{2});$$
    - or even stronger Wolfe's rule: Armijo rule plus
    $$|\phi'_k(\alpha)| \leq c_2 |\phi'_k(0)|,$$
    i.e., we'd like to decrease the projection of the gradient.
- We stop the process when $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k)\|^2 \leq \epsilon$ or $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k)\|^2 \leq \epsilon \|\nabla_{\mathbf{x}} f(\mathbf{x}_0)\|^2$ (why square, btw?).

Gradient descent

- Let's see what happens if the scale is different: for a function $f(x, y) = \frac{1}{2}x^2 + \frac{\rho}{2}y^2 \to \min_{x,y}$



- For elongated "valleys" (variables with different scale) we get a lot of superfluous iterations, everything is very slow.
- Much better to be *adaptive*; but how?

Gradient descent

- The best thing in life is, of course, *Newton's method*: let's scale back with the Hessian

$$\mathbf{g}_k = \nabla_{\mathbf{x}} f(\mathbf{x}_k),\ H_k = \nabla^2_{\mathbf{x}} f(\mathbf{x}_k),\ \text{и}\ \mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k^{-1}\mathbf{g}_k.$$

- Armijo rule is applicable here as well:

$$\alpha_k : \quad f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - c_1 \alpha_k g_k^\top H_k^{-1}\mathbf{g}_k,\ c_1 \approx 10^{-4}.$$

- Would be very nice but, alas, you can't just compute H_k.

Gradient descent

- There are approximations.
- Conjugate gradients, quasi-Newtonian methods...
- L-BFGS (limited memory Broyden–Fletcher–Goldfarb–Shanno):
    - construct an approximation to H^{-1};
    - to do that, save updates of the arguments and then express H^{-1} through them.
- Important open question: can we make L-BFGS or something similar work for deep learning?
- Doesn't work so far, mostly because you do need to be able to compute the gradient.
- And we aren't. Wait, what?..

Stochastic gradient descent

- We are usually dealing with stochastic gradient descent:

$$\mathsf{x}_t = \mathsf{x}_{t-1} - \alpha \nabla f(\mathsf{x}_t, \mathsf{x}_{t-1}, y_t).$$

- With mini-batches, too. How do we understand this formally?
- We are usually dealing with *stochastic optimization* problems:

$$F(\mathsf{x}) = \mathbb{E}_{q(\mathsf{y})} f(\mathsf{x}, \mathsf{y}) \to \min_{\mathsf{x}} :$$

  - empirical risk minimization:

$$F(\mathsf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathsf{x}) = \mathbb{E}_{i\sim \mathrm{U}(1,\dots,N)} f_i(\mathsf{x}) \to \min_{\mathsf{x}};$$

  - variational lower bound (ELBO) minimization... but later about that (if ever).
- So what are mini-batches, then?

Stochastic gradient descent

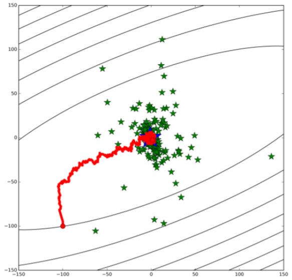
- Simply empirical estimates of a random function from a sample:

$$\hat{F}(\mathsf{x}) = \frac{1}{m}\sum_{i=1}^{m} f(\mathsf{x}, \mathsf{y}_i), \quad \hat{\mathsf{g}}(\mathsf{x}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\mathsf{x}} f(\mathsf{x}, \mathsf{y}_i).$$

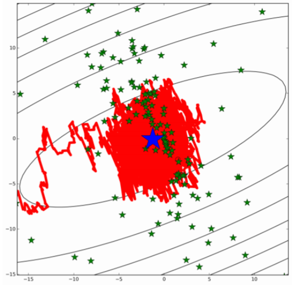
- Very good estimates: unbiased, converging (albeit slowly), easy to compute.
- In general, that's how you motivate stochastic gradient descent (SGD), it's a Monte-Carlo approach. But there are problems...

Stochastic gradient descent

- SGD problems:
  - never goes in the right direction,
  - even at the exact optimum of $F(\mathrm{x})$ the step is nonzero, i.e., it cannot converge with a constant step size,
  - we know neither $F(\mathrm{x})$ nor $\nabla F(\mathrm{x})$, i.e., we cannot use Armijo and Wolfe's rules.



SGD trajectory



optimum vicinity

Stochastic gradient descent

- Nevertheless, we can try to analyze an SGD iteration for $F(\mathsf{x}) = \mathbb{E}_{q(\mathsf{y})} f(\mathsf{x}, \mathsf{y}) \to \min_{\mathsf{x}}$:

$$\mathsf{x}_{k+1} = \mathsf{x}_k - \alpha_k \hat{\mathsf{g}}_k, \quad \mathbb{E}\hat{\mathsf{g}}_k = \mathsf{g}_k = \nabla F(\mathsf{x}_k).$$

- Let's estimate the residue of a point on some iteration:

$$\begin{aligned}\|\mathsf{x}_{k+1} - \mathsf{x}_{\mathrm{opt}}\|^2 = \|\mathsf{x}_k - \alpha_k \hat{\mathsf{g}}_k - \mathsf{x}_{\mathrm{opt}}\|^2 = \\ = \|\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}\|^2 - 2\alpha_k \hat{\mathsf{g}}_k^\top (\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}) + \alpha_k^2 \|\hat{\mathsf{g}}_k\|^2.\end{aligned}$$

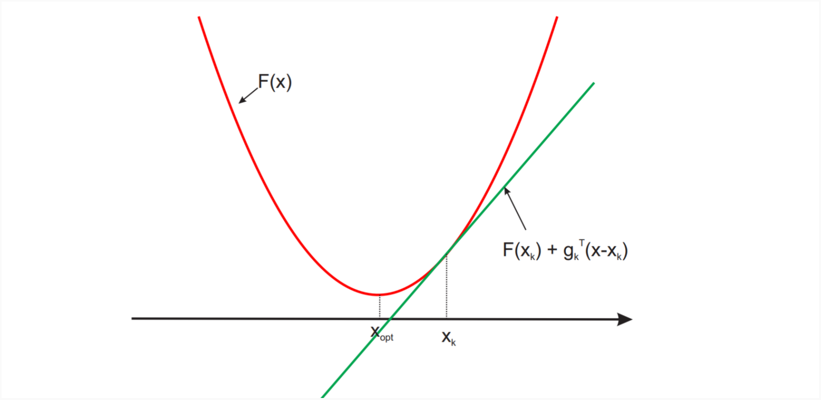
- Take expectation with respect to $q(\mathsf{y})$ at time moment k:

$$\mathbb{E}\|\mathsf{x}_{k+1} - \mathsf{x}_{\mathrm{opt}}\|^2 = \|\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}\|^2 - 2\alpha_k \mathsf{g}_k^\top (\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}) + \alpha_k^2 \mathbb{E}\|\hat{\mathsf{g}}_k\|^2.$$

Stochastic gradient descent

- Assume for simplicity that F is convex:

$$F(\mathsf{x}_{\mathrm{opt}}) \geq F(\mathsf{x}_k) + \mathsf{g}_k^\top(\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}})$$



Stochastic gradient descent

- We had

$$\mathbb{E}\|\mathsf{x}_{k+1} - \mathsf{x}_{\mathrm{opt}}\|^2 = \|\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}\|^2 - 2\alpha_k \mathsf{g}_k^\top(\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}) + \alpha_k^2 \mathbb{E}\|\hat{\mathsf{g}}_k\|^2,$$
$$F(\mathsf{x}_{\mathrm{opt}}) \geq F(\mathsf{x}_k) + \mathsf{g}_k^\top(\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}).$$

- Therefore,

$$\begin{aligned}\alpha_k(F(\mathsf{x}_k) - F(\mathsf{x}_{\mathrm{opt}})) \leq \alpha_k \mathsf{g}_k^\top(\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}) = \\ = \frac{1}{2}\|\mathsf{x}_k - \mathsf{x}_{\mathrm{opt}}\|^2 + \frac{1}{2}\alpha_k^2 \mathbb{E}\|\hat{\mathsf{g}}_k\|^2 - \frac{1}{2}\mathbb{E}\|\mathsf{x}_{k+1} - \mathsf{x}_{\mathrm{opt}}\|^2.\end{aligned}$$

Stochastic gradient descent

- Take the expectation of the left-hand side and sum up:

$$\sum_{i=0}^{k} \alpha_i (\mathbb{E}F(x_i) - F(x_{\mathrm{opt}})) \leq$$

$$\leq \frac{1}{2}\|x_0 - x_{\mathrm{opt}}\|^2 + \frac{1}{2}\sum_{i=0}^{k} \alpha_i^2 \mathbb{E}\|\hat{g}_i\|^2 - \frac{1}{2}\mathbb{E}\|x_{k+1} - x_{\mathrm{opt}}\|^2 \leq$$

$$\leq \frac{1}{2}\|x_0 - x_{\mathrm{opt}}\|^2 + \frac{1}{2}\sum_{i=0}^{k} \alpha_i^2 \mathbb{E}\|\hat{g}_i\|^2.$$

- We got the sum of function values at different points with weights α_i. What do we do now?

Stochastic gradient descent

- Use convexity:

$$\mathbb{E}F\left(\frac{\sum_i \alpha_i \mathsf{x}_i}{\sum_i \alpha_i}\right) - F(\mathsf{x}_{\mathrm{opt}}) \leq$$
$$\leq \frac{\sum_i \alpha_i(\mathbb{E}F(\mathsf{x}_i) - F(\mathsf{x}_{\mathrm{opt}}))}{\sum_i \alpha_i} \leq \frac{\frac{1}{2}\|\mathsf{x}_0 - \mathsf{x}_{\mathrm{opt}}\|^2 + \frac{1}{2}\sum_{i=0}^k \alpha_i^2 \mathbb{E}\|\hat{\mathsf{g}}_i\|^2}{\sum_i \alpha_i}.$$

- The estimate is on the value in a linear combination of points (in practice there is no difference or it's even better to take the last point).
- If $\|\mathsf{x}_0 - \mathsf{x}_{\mathrm{opt}}\| \leq R$ and $\mathbb{E}\|\hat{\mathsf{g}}_k\|^2 \leq G^2$ then

$$\mathbb{E}F(\hat{\mathsf{x}}_k) - F(\mathsf{x}_{\mathrm{opt}}) \leq \frac{R^2 + G^2 \sum_{i=0}^k \alpha_i^2}{2\sum_{i=0}^k \alpha_i}.$$

Stochastic gradient descent

- The main bound regarding SGD:

$$\mathbb{E}F(\hat{\mathsf{x}}_k) - F(\mathsf{x}_{\mathrm{opt}}) \leq \frac{R^2 + G^2 \sum_{i=0}^{k} \alpha_i^2}{2 \sum_{i=0}^{k} \alpha_i}.$$

- R is an estimate for the initial residue, and G is an estimate of something like the variance of the stochastic gradient.
- For instance, for constant step size $\alpha_i = h$

$$\mathbb{E}F(\hat{\mathsf{x}}_k) - F(\mathsf{x}_{\mathrm{opt}}) \leq \frac{R^2}{2h(k+1)} + \frac{G^2 h}{2} \to_{k\to\infty} \frac{G^2 h}{2}.$$

Stochastic gradient descent

- SGD summary:
    - SGD comes to an “uncertainty region” of radius $\frac{1}{2}G^2h$, and it is proportional to step size;
    - the faster we go, the faster we arrive there but the larger the uncertainty region will be, i.e., we need to reduce the learning rate with time;
    - SGD converges slowly: full GD for convex functions has residue $O(1/k)$, while SGD has only $O(1/\sqrt{k})$;
    - but far from the uncertainty region we still have $O(1/k)$ residue for constant learning rate, i.e., it slows down only near the optimum, and generally our goal is to reach the uncertainty region;
    - but it still depends on G, and this will be especially important for neurobayesian approaches.

Thank you!

Thank you for your attention!