Optimizers

Based on slides by Gilles Louppe
Gradient Descent

To minimize a loss $\mathcal{L}(\theta)$ of the form

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n; \theta)),$$

standard **batch gradient descent** (GD) consists in applying the update rule

$$g_t = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\theta} \ell(y_n, f(x_n; \theta_t))$$

$$\theta_{t+1} = \theta_t - \gamma g_t,$$

where $\gamma$ is the learning rate.
Gradient Descent
To reduce the computational complexity, stochastic gradient descent (SGD) consists in updating the parameters after every sample

\[ g_t = \nabla_{\theta} \ell(y_n(t), f(x_n(t); \theta_t)) \]

\[ \theta_{t+1} = \theta_t - \gamma g_t. \]
Stochastic Gradient Descent

The stochastic behavior of SGD helps evade local minima.

While being computationally faster than batch gradient descent,

- gradient estimates used by SGD can be very noisy,
- SGD does not benefit from the speed-up of batch-processing.

Thus, mini-batching is used to reduce the variance of the gradient estimate and to enables the speed-up of batch processing.
Momentum

In the situation of small but consistent gradients, as through valley floor, gradient descent moves very slowly.
Momentum

An improvement to gradient descent is to use momentum to add inertia in the choice of the step direction, that is

\[ u_t = \alpha u_{t-1} - \gamma g_t \]
\[ \theta_{t+1} = \theta_t + u_t. \]

- The new variable \( u_t \) is the velocity. It corresponds to the direction and speed by which the parameters move as the learning dynamics progresses, modeled as an exponentially decaying moving average of negative gradients.
- Gradient descent with momentum has three nice properties:
  - it can go through local barriers,
  - it accelerates if the gradient does not change much,
  - it dampens oscillations in narrow valleys.
Momentum

The hyper-parameter $\alpha$ controls how recent gradients affect the current update.

- Usually, $\alpha = 0.9$, with $\alpha > \gamma$.
- If at each update we observed $g$, the step would (eventually) be

$$u = -\frac{\gamma}{1 - \alpha} g.$$

- Therefore, for $\alpha = 0.9$, it is like multiplying the maximum speed by 10 relative to the current direction.
Momentum
Nesterov Momentum

An alternative consists in simulating a step in the direction of the velocity, then calculate the gradient and make a correction.

\[
g_t = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\theta} \ell(y_n, f(x_n; \theta_t + \alpha u_{t-1}))
\]

\[
u_t = \alpha u_{t-1} - \gamma g_t
\]

\[
\theta_{t+1} = \theta_t + u_t
\]
Nesterov Momentum
Limitations of Gradient Descent

- Gradient Descent makes a strong assumption about the magnitude of the local curvature to set up the step size.

- The isotropy of the curvature, so that the same step size makes sense in all directions.
Limitations of Gradient Descent

Thus, Adaptive Learning Rates
AdaGrad

Per-parameter downscale by square-root of sum of squares of all its historical values.

\[
\begin{align*}
    r_t &= r_{t-1} + g_t \odot g_t \\
    \theta_{t+1} &= \theta_t - \frac{\gamma}{\delta + \sqrt{r_t}} \odot g_t.
\end{align*}
\]

- AdaGrad eliminates the need to manually tune the learning rate. Most implementation use $\gamma = 0.01$ as default.
- It is good when the objective is convex.
- $r_t$ grows unboundedly during training, which may cause the step size to shrink and eventually become infinitesimally small.
RMSProp

Same as AdaGrad but accumulate an exponentially decaying average of the gradient.

\[
\begin{align*}
r_t &= \rho r_{t-1} + (1 - \rho) g_t \odot g_t \\
\theta_{t+1} &= \theta_t - \frac{\gamma}{\delta + \sqrt{r_t}} \odot g_t.
\end{align*}
\]

- Perform better in non-convex settings.
- Does not grow unboundedly.
Adam

Similar to RMSProp with momentum, but with bias correction terms for the first and second moments.

\[ s_t = \rho_1 s_{t-1} + (1 - \rho_1) g_t \]
\[ \hat{s}_t = \frac{s_t}{1 - \rho_1^t} \]
\[ r_t = \rho_2 r_{t-1} + (1 - \rho_2) g_t \odot g_t \]
\[ \hat{r}_t = \frac{s_t}{1 - \rho_2^t} \]
\[ \theta_{t+1} = \theta_t - \gamma \frac{\hat{s}_t}{\delta + \sqrt{\hat{r}_t}} \]

- Good defaults are \( \rho_1 = 0.9 \) and \( \rho_2 = 0.999 \).
- Adam is one of the default optimizers in deep learning, along with SGD with momentum.
Adaptive Learning Rates
Initialization

- In convex problems, provided a good learning rate $\gamma$, convergence is guaranteed regardless of the initial parameter values.
- In the non-convex regime, initialization is much more important!
- Little is known on the mathematics of initialization strategies of neural networks.
  - What is known: initialization should break symmetry.
  - What is known: the scale of weights is important.
Initialization: the forward pass

Controlling for the variance in the forward pass

A first strategy is to initialize the network parameters such that activations preserve the same variance across layers.

Intuitively, this ensures that the information keeps flowing during the forward pass, without reducing or magnifying the magnitude of input signals exponentially.
Initialization: the forward pass

Let us assume that

- we are in a linear regime at initialization (e.g., the positive part of a ReLU or the middle of a sigmoid),
- weights $w_{ij}^l$ are initialized independently,
- biases $b_l$ are initialized to be 0,
- input feature variances are the same, which we denote as $\nabla [x]$.

Then, the variance of the activation $h_i^l$ of unit $i$ in layer $l$ is

$$\nabla [h_i^l] = \nabla \left[ \sum_{j=0}^{q_{l-1} - 1} w_{ij}^l h_j^{l-1} \right]$$

$$= \sum_{j=0}^{q_{l-1} - 1} \nabla [w_{ij}^l] \nabla [h_j^{l-1}]$$

where $q_l$ is the width of layer $l$ and $h_j^0 = x_j$ for all $j = 0, \ldots, p - 1$. 
Initialization: the forward pass

If we further assume that weights $w_{ij}^l$ at layer $l$ share the same variance $\nabla [w^l]$ and that the variance of the activations in the previous layer are the same, then we can drop the indices and write

$$\nabla [h^l] = q_{l-1} \nabla [w^l] \nabla [h^{l-1}].$$

Therefore, the variance of the activations is preserved across layers when

$$\nabla [w^l] = \frac{1}{q_{l-1}} \quad \forall l.$$

This condition is enforced in LeCun's uniform initialization, which is defined as

$$w_{ij}^l \sim \mathcal{U} \left[ -\sqrt{\frac{3}{q_{l-1}}}, \sqrt{\frac{3}{q_{l-1}}} \right].$$
Initialization: the backward pass

Controlling for the variance in the backward pass

A similar idea can be applied to ensure that the gradients flow in the **backward pass** (without vanishing nor exploding), by maintaining the variance of the gradient with respect to the activations fixed across layers.

Under the same assumptions as before,

\[
\nabla \left[ \frac{d\hat{y}}{dh_i^l} \right] = \nabla \left[ \sum_{j=0}^{q_{l+1}-1} \frac{d\hat{y}}{dh_j^{l+1}} \frac{\partial h_j^{l+1}}{\partial h_i^l} \right] \\
= \nabla \left[ \sum_{j=0}^{q_{l+1}-1} \frac{d\hat{y}}{dh_j^{l+1}} w_{j,i}^{l+1} \right] \\
= \sum_{j=0}^{q_{l+1}-1} \nabla \left[ \frac{d\hat{y}}{dh_j^{l+1}} \right] \nabla [w_{ji}^{l+1}] 
\]
Initialization: the backward pass

If we further assume that

- the gradients of the activations at layer $l$ share the same variance
- the weights at layer $l + 1$ share the same variance $\nabla \left[ w^{l+1} \right]$,

then we can drop the indices and write

$$\nabla \left[ \frac{d\hat{y}}{dh^l} \right] = q_{l+1} \nabla \left[ \frac{d\hat{y}}{dh^{l+1}} \right] \nabla \left[ w^{l+1} \right].$$

Therefore, the variance of the gradients with respect to the activations is preserved across layers when

$$\nabla \left[ w^l \right] = \frac{1}{q_l} \quad \forall l.$$
Xavier Initialization

We have derived two different conditions on the variance of $w^l$,

- $\text{Var}[w^l] = \frac{1}{q_{l-1}}$
- $\text{Var}[w^l] = \frac{1}{q_l}$.

A compromise is the **Xavier initialization**, which initializes $w^l$ randomly from a distribution with variance

$$
\text{Var}[w^l] = \frac{1}{\frac{q_{l-1} + q_l}{2}} = \frac{2}{q_{l-1} + q_l}.
$$

For example, **normalized initialization** is defined as

$$
\omega^l_{ij} \sim \mathcal{U} \left[ -\sqrt{\frac{6}{q_{l-1} + q_l}}, \sqrt{\frac{6}{q_{l-1} + q_l}} \right].
$$
Xavier Initialization

Figure 6: Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.