Mnacho Echenim
small evolutions : Patrick Reignier

Grenoble INP-Ensimag

2025-2026

Outline

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives
- Closer to what may be found in textbooks

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives
- Closer to what may be found in textbooks
- We want to compute $\frac{\partial \mathcal{C}}{\partial w_k^j}$ and $\frac{\partial \mathcal{C}}{\partial b^j}$

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives
- Closer to what may be found in textbooks
- We want to compute $\frac{\partial \mathcal{C}}{\partial w_k^j}$ and $\frac{\partial \mathcal{C}}{\partial b^j}$
- We have the following equalities:

$$\frac{\partial \mathcal{C}}{\partial w_k^j} = \frac{\partial \mathcal{C}}{\partial a^L} \cdot \frac{\partial a^L}{\partial w_k^j}$$

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives
- Closer to what may be found in textbooks
- We want to compute $\frac{\partial \mathcal{C}}{\partial w_k^j}$ and $\frac{\partial \mathcal{C}}{\partial b^j}$
- We have the following equalities:

$$\begin{aligned}
\frac{\partial \mathcal{C}}{\partial w_k^j} &= \frac{\partial \mathcal{C}}{\partial a^L} \cdot \frac{\partial a^L}{\partial w_k^j} \\
&= \frac{\partial \mathcal{C}}{\partial a^L} \cdot \left[\prod_{i=L}^{j+1} \frac{\partial a^i}{\partial a^{i-1}} \right] \cdot \frac{\partial a^j}{\partial w_k^j}
\end{aligned}$$

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives
- Closer to what may be found in textbooks
- We want to compute $\frac{\partial \mathcal{C}}{\partial w_k^j}$ and $\frac{\partial \mathcal{C}}{\partial b^j}$
- We have the following equalities:

$$
\begin{aligned}
\frac{\partial \mathcal{C}}{\partial w_k^j} &= \frac{\partial \mathcal{C}}{\partial a^L} \cdot \frac{\partial a^L}{\partial w_k^j} \\
&= \frac{\partial \mathcal{C}}{\partial a^L} \cdot \left[\prod_{i=L}^{j+1} \frac{\partial a^i}{\partial a^{i-1}} \right] \cdot \frac{\partial a^j}{\partial w_k^j} \\
&= \mathcal{P}^{j+1} \cdot \frac{\partial a^j}{\partial w_k^j}
\end{aligned}
$$

Derivation of the backpropagation rules (simplified notations)

- **Goal:** get rid of cumbersome notations to represent partial derivatives
- Closer to what may be found in textbooks
- We want to compute $\frac{\partial \mathcal{C}}{\partial w_k^j}$ and $\frac{\partial \mathcal{C}}{\partial b^j}$
- We have the following equalities:

$$
\begin{aligned}
\frac{\partial \mathcal{C}}{\partial w_k^j} &= \frac{\partial \mathcal{C}}{\partial a^L} \cdot \frac{\partial a^L}{\partial w_k^j} \\
&= \frac{\partial \mathcal{C}}{\partial a^L} \cdot \left[\prod_{i=L}^{j+1} \frac{\partial a^i}{\partial a^{i-1}} \right] \cdot \frac{\partial a^j}{\partial w_k^j} \\
&= \mathcal{P}^{j+1} \cdot \frac{\partial a^j}{\partial w_k^j} \\
\frac{\partial \mathcal{C}}{\partial b^j} &= \mathcal{P}^{j+1} \cdot \frac{\partial a^j}{\partial b^j}
\end{aligned}
$$

Derivation of the backpropagation rules (part 2)

$$\frac{\partial a^j}{\partial a^{j-1}} = \begin{pmatrix} \Phi'(\zeta_1^j).\left[\omega_1^j\right]^{\mathrm{T}} \\ \vdots \\ \Phi'(\zeta_{n_j}^j).\left[\omega_{n_j}^j\right]^{\mathrm{T}} \end{pmatrix}$$

Derivation of the backpropagation rules (part 2)

$$\frac{\partial a^j}{\partial a^{j-1}} = \begin{pmatrix} \Phi'(\zeta_1^j).\left[\omega_1^j\right]^{\mathrm{T}} \\ \vdots \\ \Phi'(\zeta_{n_j}^j).\left[\omega_{n_j}^j\right]^{\mathrm{T}} \end{pmatrix}$$

$$\frac{\partial a^j}{\partial w_k^j} = \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k^j).\left[\alpha^{j-1}\right]^{\mathrm{T}} \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} \\ \\ \leftarrow \text{ line } k \\ \\ \\ \end{matrix}$$

Derivation of the backpropagation rules (part 2)

$$\frac{\partial a^j}{\partial a^{j-1}} = \begin{pmatrix} \Phi'(\zeta_1^j).\left[\omega_1^j\right]^{\mathrm{T}} \\ \vdots \\ \Phi'(\zeta_{n_j}^j).\left[\omega_{n_j}^j\right]^{\mathrm{T}} \end{pmatrix}$$

$$\frac{\partial a^j}{\partial w_k^j} = \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k^j).\left[\alpha^{j-1}\right]^{\mathrm{T}} \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{ line } k$$

$$\frac{\partial a^j}{\partial b^j} = \mathrm{diag}\left(\Phi'(\zeta_1^j), \ldots, \Phi'(\zeta_{n_j}^j)\right)$$

Derivation of the backpropagation rules (part 2)

$$
\begin{aligned}
\frac{\partial a^j}{\partial a^{j-1}} &= \begin{pmatrix} \Phi'(\zeta_1^j).\left[\omega_1^j\right]^{\mathrm{T}} \\ \vdots \\ \Phi'(\zeta_{n_j}^j).\left[\omega_{n_j}^j\right]^{\mathrm{T}} \end{pmatrix} \\
\frac{\partial a^j}{\partial w_k^j} &= \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k^j).\left[\alpha^{j-1}\right]^{\mathrm{T}} \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{ line } k \\
\frac{\partial a^j}{\partial b^j} &= \operatorname{diag}\left(\Phi'(\zeta_1^j), \ldots, \Phi'(\zeta_{n_j}^j)\right)
\end{aligned}
$$

The backpropagation equations are then derived as previously

Mini-batch gradient descent

- We are given M inputs $\overline{\alpha^0} \stackrel{\text{def}}{=} (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ and outputs $\overline{\rho} \stackrel{\text{def}}{=} (\rho_{(1)}, \ldots, \rho_{(M)})$

Mini-batch gradient descent

- We are given M inputs $\overline{\alpha^0} \stackrel{\text{def}}{=} (\alpha^0_{(1)}, \dots, \alpha^0_{(M)})$ and outputs $\overline{\rho} \stackrel{\text{def}}{=} (\rho_{(1)}, \dots, \rho_{(M)})$
- The parameters updates become

$$\Omega^j \quad \leftarrow \quad \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{E}(\alpha^0_{(k)}, \dots, \Omega^L, \beta^L, \rho_{(k)})$$

Mini-batch gradient descent

- We are given M inputs $\overline{\alpha^0} \stackrel{\text{def}}{=} (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ and outputs $\overline{\rho} \stackrel{\text{def}}{=} (\rho_{(1)}, \ldots, \rho_{(M)})$
- The parameters updates become

$$\Omega^j \quad \leftarrow \quad \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

$$\Omega^j \quad \leftarrow \quad \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

Mini-batch gradient descent

- We are given M inputs $\overline{\alpha^0} \stackrel{\text{def}}{=} (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ and outputs $\overline{\rho} \stackrel{\text{def}}{=} (\rho_{(1)}, \ldots, \rho_{(M)})$
- The parameters updates become

$$\Omega^j \leftarrow \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

$$\Omega^j \leftarrow \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

$$\beta^j \leftarrow \beta^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{b^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

Mini-batch gradient descent

- We are given M inputs $\overline{\alpha^0} \stackrel{\text{def}}{=} (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ and outputs $\overline{\rho} \stackrel{\text{def}}{=} (\rho_{(1)}, \ldots, \rho_{(M)})$
- The parameters updates become

$$\Omega^j \leftarrow \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

$$\Omega^j \leftarrow \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

$$\beta^j \leftarrow \beta^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{b^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

$$\beta^j \leftarrow \beta^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{b^j} \mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

Mini-batch gradient descent

- We are given M inputs $\overline{\alpha^0} \overset{\text{def}}{=} (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ and outputs $\overline{\rho} \overset{\text{def}}{=} (\rho_{(1)}, \ldots, \rho_{(M)})$
- The parameters updates become

$$\Omega^j \quad \leftarrow \quad \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

$$\Omega^j \quad \leftarrow \quad \Omega^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j} \mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

$$\beta^j \quad \leftarrow \quad \beta^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{b^j} \mathcal{E}(\alpha^0_{(k)}, \ldots, \Omega^L, \beta^L, \rho_{(k)})$$

$$\beta^j \quad \leftarrow \quad \beta^j - \frac{\eta}{M} \cdot \sum_{k=1}^{M} \nabla_{b^j} \mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

- **Goal:** compute these updates efficiently using matrix operations

Extensions to mini-batches

Inputs to the network: $\overline{\alpha^0}, \Omega_1, \beta^1, \ldots, \Omega^L, \beta^L$. We let:

$$
\begin{aligned}
\overline{\beta^i} &\stackrel{\text{def}}{=} (\beta^i, \ldots, \beta^i) \quad (M \text{ columns}) \\
\overline{\alpha^i} &\stackrel{\text{def}}{=} (\alpha^i_{(1)}, \ldots, \alpha^i_{(M)}) = f_i(\overline{\alpha^{i-1}}, \Omega^i, \beta^i) \\
\overline{\zeta^i} &\stackrel{\text{def}}{=} \left(\zeta^i_{(1)}, \ldots, \zeta^i_{(M)}\right) \\
\Phi'(\overline{\zeta^i}) &\stackrel{\text{def}}{=} (\Phi'(\zeta^i_{(1)}), \ldots, \Phi'(\zeta^i_{(M)})) \\
\overline{\mathcal{B}^i} &\stackrel{\text{def}}{=} (\mathcal{B}^i_{(1)}, \ldots, \mathcal{B}^i_{(M)}) \\
\nabla_{a^L}\mathcal{C}(\overline{\alpha^L}, \overline{\rho}) &= \left(\nabla_{a^L}\mathcal{C}(\alpha^L_{(1)}, \rho_{(1)}), \ldots, \nabla_{a^L}\mathcal{C}(\alpha^L_{(M)}, \rho_{(M)})\right)
\end{aligned}
$$

Extensions to mini-batches

Inputs to the network: $\overline{\alpha^0}, \Omega_1, \beta^1, \ldots, \Omega^L, \beta^L$. We let:

$$\begin{aligned}
\overline{\beta^i} &\stackrel{\text{def}}{=} (\beta^i, \ldots, \beta^i) \quad (M \text{ columns}) \\
\overline{\alpha^i} &\stackrel{\text{def}}{=} (\alpha^i_{(1)}, \ldots, \alpha^i_{(M)}) = f_i(\overline{\alpha^{i-1}}, \Omega^i, \beta^i) \\
\overline{\zeta^i} &\stackrel{\text{def}}{=} \left(\zeta^i_{(1)}, \ldots, \zeta^i_{(M)}\right) \\
\Phi'(\overline{\zeta^i}) &\stackrel{\text{def}}{=} (\Phi'(\zeta^i_{(1)}), \ldots, \Phi'(\zeta^i_{(M)})) \\
\overline{\mathcal{B}^i} &\stackrel{\text{def}}{=} (\mathcal{B}^i_{(1)}, \ldots, \mathcal{B}^i_{(M)}) \\
\nabla_{a^L}\mathcal{C}(\overline{\alpha^L}, \overline{\rho}) &= \left(\nabla_{a^L}\mathcal{C}(\alpha^L_{(1)}, \rho_{(1)}), \ldots, \nabla_{a^L}\mathcal{C}(\alpha^L_{(M)}, \rho_{(M)})\right)
\end{aligned}$$

We have the following equalities:

- $\overline{\zeta^i} = \Psi(\overline{\alpha^{i-1}}, \Omega^i, \overline{\beta^i}) = \left[\Omega^i\right]^{\mathrm{T}} \cdot \overline{\alpha^{i-1}} + \overline{\beta^i}$ *for* $i = 1, \ldots, L$
- $\overline{\mathcal{B}^L} = \Phi'(\overline{\zeta^L}) \odot \nabla_{a^L}\mathcal{C}(\overline{\alpha^L}, \overline{\rho})$
- $\overline{\mathcal{B}^j} = \Phi'(\overline{\zeta^j}) \odot \left(\Omega^{j+1} \cdot \overline{\mathcal{B}^{j+1}}\right)$

Other computations

$$\nabla_{W^j}\mathcal{C} \quad = \quad \frac{1}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j}\mathcal{C}(\alpha^L_{(k)}, \rho_{(k)})$$

Other computations

$$
\begin{aligned}
\nabla_{W^j}\mathcal{C} &= \frac{1}{M}\cdot\sum_{k=1}^{M}\nabla_{W^j}\mathcal{C}(\alpha^L_{(k)},\rho_{(k)}) \\
&= \frac{1}{M}\cdot\sum_{k=1}^{M}\alpha^{j-1}_{(k)}\cdot\left[\mathcal{B}^j_{(k)}\right]^{\mathrm{T}}
\end{aligned}
$$

Other computations

$$
\begin{aligned}
\nabla_{W^j}\mathcal{C} &= \frac{1}{M} \cdot \sum_{k=1}^{M} \nabla_{W^j}\mathcal{C}(\alpha^L_{(k)}, \rho_{(k)}) \\
&= \frac{1}{M} \cdot \sum_{k=1}^{M} \alpha^{j-1}_{(k)} \cdot \left[\mathcal{B}^j_{(k)}\right]^{\mathrm{T}} = \frac{1}{M} \cdot \overline{\alpha^{j-1}} \cdot \left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}}
\end{aligned}
$$

Other computations

$$\nabla_{W^j}\mathcal{C} = \frac{1}{M}\cdot\sum_{k=1}^{M}\nabla_{W^j}\mathcal{C}(\alpha^L_{(k)},\rho_{(k)})$$

$$= \frac{1}{M}\cdot\sum_{k=1}^{M}\alpha^{j-1}_{(k)}\cdot\left[\mathcal{B}^j_{(k)}\right]^{\mathrm{T}} = \frac{1}{M}\cdot\overline{\alpha^{j-1}}\cdot\left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}}$$

$$\nabla_{b^j}\mathcal{C} = \frac{1}{M}\cdot\sum_{k=1}^{M}\mathcal{B}^j_{(k)}$$

Other computations

$$
\begin{aligned}
\nabla_{W^j}\mathcal{C} &= \frac{1}{M}\cdot\sum_{k=1}^{M}\nabla_{W^j}\mathcal{C}(\alpha^L_{(k)},\rho_{(k)}) \\
&= \frac{1}{M}\cdot\sum_{k=1}^{M}\alpha^{j-1}_{(k)}\cdot\left[\mathcal{B}^j_{(k)}\right]^{\mathrm{T}} = \frac{1}{M}\cdot\overline{\alpha^{j-1}}\cdot\left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}} \\
\nabla_{b^j}\mathcal{C} &= \frac{1}{M}\cdot\sum_{k=1}^{M}\mathcal{B}^j_{(k)} = \frac{1}{M}\cdot\left(\overline{\mathcal{B}^j}\cdot\mathbf{1}_M\right)
\end{aligned}
$$

Other computations

$$
\begin{aligned}
\nabla_{W^j}\mathcal{C} &= \frac{1}{M}\cdot\sum_{k=1}^{M}\nabla_{W^j}\mathcal{C}(\alpha^L_{(k)},\rho_{(k)}) \\
&= \frac{1}{M}\cdot\sum_{k=1}^{M}\alpha^{j-1}_{(k)}\cdot\left[\mathcal{B}^j_{(k)}\right]^{\mathrm{T}} = \frac{1}{M}\cdot\overline{\alpha^{j-1}}\cdot\left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}} \\
\nabla_{b^j}\mathcal{C} &= \frac{1}{M}\cdot\sum_{k=1}^{M}\mathcal{B}^j_{(k)} = \frac{1}{M}\cdot\left(\overline{\mathcal{B}^j}\cdot\mathbf{1}_M\right)
\end{aligned}
$$

Let $\mathbf{1}_M \stackrel{\text{def}}{=} (1,\ldots,1)^{\mathrm{T}} \in \mathbb{R}^M$. For M inputs and outputs, parameter updates become:

$$
\begin{aligned}
\Omega^j &\leftarrow \Omega^j - \frac{\eta}{M}\cdot\left(\overline{\alpha^{j-1}}\cdot\left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}}\right) \\
\beta_j &\leftarrow \beta_j - \frac{\eta}{M}\cdot\left(\overline{\mathcal{B}^j}\cdot\mathbf{1}_M\right)
\end{aligned}
$$

Mini-batch forward and backpropagation algorithms

Input: A network with L layers
Input: $\overline{\alpha^0} = (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$

for $i \leftarrow 1$ **to** L **do**
  $\overline{\zeta^i} \leftarrow \left[\Omega^i\right]^{\mathrm{T}} \cdot \overline{\alpha^{i-1}} + \overline{\beta^i}$;
  $\overline{\alpha^i} \leftarrow \Phi(\overline{\zeta^i})$;
end

Algorithm 1: Forward propagation

Mini-batch forward and backpropagation algorithms

Input: A network with L layers
Input: $\overline{\alpha^0} = (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$

for $i \leftarrow 1$ **to** L **do**
  $\overline{\zeta^i} \leftarrow \left[\Omega^i\right]^{\mathrm{T}} \cdot \overline{\alpha^{i-1}} + \overline{\beta^i}$;
  $\overline{\alpha^i} \leftarrow \Phi(\overline{\zeta^i})$;
end

Algorithm 3: Forward propagation

Input: A network with L layers
Input: $\overline{\alpha^0} = (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ that has been forward propagated
Input: $\overline{\rho} = (\rho_{(1)}, \ldots, \rho_{(M)})$ the expected outputs

$\overline{\mathcal{B}^L} \leftarrow \Phi'(\overline{\zeta^L}) \odot \nabla_{a^L}\mathcal{C}(\overline{\alpha^L}, \overline{\rho})$;
$\overline{[\mathcal{P}^L]^{\mathrm{T}}} \leftarrow \Omega^L \cdot \overline{\mathcal{B}^L}$;
for $j \leftarrow L-1$ **to** 1 **do**
  $\overline{\mathcal{B}^j} \leftarrow \Phi'(\overline{\zeta^j}) \odot \overline{[\mathcal{P}^{j+1}]^{\mathrm{T}}}$;
  $\overline{[\mathcal{P}^j]^{\mathrm{T}}} \leftarrow \Omega^j \cdot \overline{\mathcal{B}^j}$;
end

Algorithm 4: Backpropagation

Mini-batch gradient computation

Input: A network with L layers
Input: $\overline{\alpha^0} = (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ that has been forward propagated
Input: $(\overline{\mathcal{B}^1}, \ldots, \overline{\mathcal{B}^L})$ that have been updated by backpropagation

for $j \in \{1, \ldots, L\}$ **do**
  $\mathrm{Gradient}(\Omega^j) \leftarrow \frac{1}{M} \cdot \left(\overline{\alpha^{j-1}} \cdot \left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}}\right)$;
  $\mathrm{Gradient}(\beta^j) \leftarrow \frac{1}{M} \cdot \left(\overline{\mathcal{B}^j} \cdot \mathbf{1}_M\right)$;
end

Algorithm 5: Mini-batch gradient computation

Mini-batch gradient computation

Input: A network with L layers
Input: $\overline{\alpha^0} = (\alpha^0_{(1)}, \ldots, \alpha^0_{(M)})$ that has been forward propagated
Input: $(\overline{\mathcal{B}^1}, \ldots, \overline{\mathcal{B}^L})$ that have been updated by backpropagation
for $j \in \{1, \ldots, L\}$ **do**
  $\mathrm{Gradient}(\Omega^j) \leftarrow \frac{1}{M} \cdot \left(\overline{\alpha^{j-1}} \cdot \left[\overline{\mathcal{B}^j}\right]^{\mathrm{T}} \right)$;
  $\mathrm{Gradient}(\beta^j) \leftarrow \frac{1}{M} \cdot \left(\overline{\mathcal{B}^j} \cdot \mathbf{1}_M \right)$;
end

Algorithm 6: Mini-batch gradient computation

Note

By storing the necessary information, backpropagation and mini-batch gradient computations can be carried out in the same loop

Computed quantities in backpropagation

- At layer $j \in [\![1, L]\!]$, two central quantities are computed
  - $\mathcal{P}^j \in \mathbb{R}^{1 \times n_{j-1}}$ and $\mathcal{B}^j \in \mathbb{R}^{n_j}$

Computed quantities in backpropagation

- At layer $j \in [\![1, L]\!]$, two central quantities are computed
  - $\mathcal{P}^j \in \mathbb{R}^{1 \times n_{j-1}}$ and $\mathcal{B}^j \in \mathbb{R}^{n_j}$
- $\mathcal{P}^j$ represents the information that is transmitted to layer $j-1$

Computed quantities in backpropagation

- At layer $j \in [\![1, L]\!]$, two central quantities are computed
  - $\mathcal{P}^j \in \mathbb{R}^{1\times n_{j-1}}$ and $\mathcal{B}^j \in \mathbb{R}^{n_j}$
- $\mathcal{P}^j$ represents the information that is transmitted to layer $j-1$
- $\mathcal{B}^j$ represents the information necessary to compute gradients at layer j

Computed quantities in backpropagation

- At layer $j \in [\![1, L]\!]$, two central quantities are computed
  - $\mathcal{P}^j \in \mathbb{R}^{1\times n_{j-1}}$ and $\mathcal{B}^j \in \mathbb{R}^{n_j}$
- $\mathcal{P}^j$ represents the information that is transmitted to layer $j-1$
- $\mathcal{B}^j$ represents the information necessary to compute gradients at layer j

---

- We have $\mathcal{P}^j = \frac{\partial \mathcal{C}}{\partial a^{j-1}}$
- If z^j is a formal parameter representing the net input of layer j, then $\mathcal{B}^j = \nabla_{z^j}\mathcal{C}$

Outline

Foreword

- What follows is a list of techniques that are often used to improve gradient descent

Foreword

- What follows is a list of techniques that are often used to improve gradient descent
- There are (currently) no formal proofs that these methods improve anything on neural networks
  - They may be viewed by some as nothing more than recipes

Foreword

- What follows is a list of techniques that are often used to improve gradient descent
- There are (currently) no formal proofs that these methods improve anything on neural networks
  - They may be viewed by some as nothing more than recipes
- **But**
  - Several originate from research on improving gradient descent on convex optimization problems
  - E.g., the Nesterov momentum method permits to obtain optimal convergence rates for convex optimization problems
  - It is not far-fetched to try them on non-convex optimization problems

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: "denoise" the estimate by increasing the batch sizes

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: “denoise” the estimate by increasing the batch sizes
- Another idea: decrease the learning rate over time with decrease factor δ

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: “denoise” the estimate by increasing the batch sizes
- Another idea: decrease the learning rate over time with decrease factor δ
  - Inverse decay: $\eta_k = \frac{\eta_0}{1+k\delta}$

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: "denoise" the estimate by increasing the batch sizes
- Another idea: decrease the learning rate over time with decrease factor δ
  - Inverse decay: $\eta_k = \frac{\eta_0}{1+k\delta}$
  - Exponential decay: $\eta_k = \eta_0 \exp(-k\delta)$

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: "denoise" the estimate by increasing the batch sizes
- Another idea: decrease the learning rate over time with decrease factor δ
  - Inverse decay: $\eta_k = \frac{\eta_0}{1+k\delta}$
  - Exponential decay: $\eta_k = \eta_0 \exp(-k\delta)$
  - ...

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: "denoise" the estimate by increasing the batch sizes
- Another idea: decrease the learning rate over time with decrease factor δ
  - Inverse decay: $\eta_k = \frac{\eta_0}{1+k\delta}$
  - Exponential decay: $\eta_k = \eta_0 \exp(-k\delta)$
  - ...
- Demo 2

Reducing noise for mini-batch gradient descent

Issues with mini-batch gradient descent

- As we get closer to an optimum, the noise in the gradient estimate can become a problem
- Demo 1

How can this issue be handled?

- First idea: "denoise" the estimate by increasing the batch sizes
- Another idea: decrease the learning rate over time with decrease factor δ
  - Inverse decay: $\eta_k = \frac{\eta_0}{1+k\delta}$
  - Exponential decay: $\eta_k = \eta_0 \exp(-k\delta)$
  - ...
- Demo 2

Note

It is still difficult to choose the initial rate, decrease factor, decrease schedule...

Is it guaranteed gradient descent will work?

- Problem: are we sure gradient descent will lead to a global minimum of the function?

Is it guaranteed gradient descent will work?

- Problem: are we sure gradient descent will lead to a global minimum of the function?
- In general no: most of the time, the function to optimize is not convex

Is it guaranteed gradient descent will work?

- Problem: are we sure gradient descent will lead to a global minimum of the function?
- In general no: most of the time, the function to optimize is not convex
- Gradient descent could get stuck on
  - Local minima
  - Stationary points

Is it guaranteed gradient descent will work?

- Problem: are we sure gradient descent will lead to a global minimum of the function?
- In general no: most of the time, the function to optimize is not convex
- Gradient descent could get stuck on
  - Local minima
  - Stationary points
- Can the algorithm be adapted to produce better results?

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component
- First approach: delta-bar-delta (Jacobs, 88)

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component
- First approach: delta-bar-delta (Jacobs, 88)
  - Principle: check the sign of each partial derivative

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component
- First approach: delta-bar-delta (Jacobs, 88)
  - Principle: check the sign of each partial derivative
    - If it stays the same, then the optimization direction is correct: **increase the learning rate**

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component
- First approach: delta-bar-delta (Jacobs, 88)
  - Principle: check the sign of each partial derivative
    - If it stays the same, then the optimization direction is correct: **increase the learning rate**
    - Otherwise **decrease the learning rate**

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component
- First approach: delta-bar-delta (Jacobs, 88)
  - Principle: check the sign of each partial derivative
    - If it stays the same, then the optimization direction is correct: **increase the learning rate**
    - Otherwise **decrease the learning rate**
  - **Nb:** This approach can only be applied to full gradient descent
    - The noise in SGD can produce wrong signals

Adaptive learning rates

- Problem: fixing the right learning rate is difficult, even using decay strategies
- It is not clear whether the same learning rate should be used on every component
- Principle: use information about computed gradients to set learning rates individually for each component
- First approach: delta-bar-delta (Jacobs, 88)
  - Principle: check the sign of each partial derivative
    - If it stays the same, then the optimization direction is correct: **increase the learning rate**
    - Otherwise **decrease the learning rate**
  - **Nb:** This approach can only be applied to full gradient descent
    - The noise in SGD can produce wrong signals
- Upcoming approaches: all operations on vectors and matrices are **componentwise**

AdaGrad (Duchi et al, 2011)

- Principle: keep track of aggregated squared magnitude of gradients; use this to scale the individual learning rates
  - Large partial derivatives lead to a fast decrease, and vice-versa

AdaGrad (Duchi et al, 2011)

- Principle: keep track of aggregated squared magnitude of gradients; use this to scale the individual learning rates
  - Large partial derivatives lead to a fast decrease, and vice-versa
- This technique has good properties in a convex setting
  - But the fact that **all** gradients are accumulated may lead to an excessive decrease of the learning rate

AdaGrad (Duchi et al, 2011)

- Principle: keep track of aggregated squared magnitude of gradients; use this to scale the individual learning rates
  - Large partial derivatives lead to a fast decrease, and vice-versa
- This technique has good properties in a convex setting
  - But the fact that **all** gradients are accumulated may lead to an excessive decrease of the learning rate

Input: Step size η, numerical stabilizer δ
Input: Initial parameters θ

set gradient accumulation variable r to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ //*full gradient, mini-batch...*;
    $r \leftarrow r + (g \odot g)$;
    $v \leftarrow -\frac{\eta}{\delta + \sqrt{r}} \odot g$;
    $\theta \leftarrow \theta + v$;
end

AdaGrad (Duchi et al, 2011)

- Principle: keep track of aggregated squared magnitude of gradients; use this to scale the individual learning rates
  - Large partial derivatives lead to a fast decrease, and vice-versa
- This technique has good properties in a convex setting
  - But the fact that **all** gradients are accumulated may lead to an excessive decrease of the learning rate

Input: Step size η, numerical stabilizer δ
Input: Initial parameters θ
set gradient accumulation variable r to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ //*full gradient, mini-batch...*;
    $r \leftarrow r + (g \odot g)$;
    $v \leftarrow -\frac{\eta}{\delta + \sqrt{r}} \odot g$;
    $\theta \leftarrow \theta + v$;
end

- Demo 3

RMSProp (Hinton, 2012)

- Principle: accumulate squared magnitude of gradients, with an exponentially weighted moving average
  - Ancient history is discarded, as old gradients decay exponentially with time

RMSProp (Hinton, 2012)

- Principle: accumulate squared magnitude of gradients, with an exponentially weighted moving average
  - Ancient history is discarded, as old gradients decay exponentially with time
- Example: on a slope with a locally convex bowl:
  - AdaGrad will likely get stuck
  - RMSProp will forget about the past and get out of the bowl

RMSProp (Hinton, 2012)

- Principle: accumulate squared magnitude of gradients, with an exponentially weighted moving average
  - Ancient history is discarded, as old gradients decay exponentially with time
- Example: on a slope with a locally convex bowl:
  - AdaGrad will likely get stuck
  - RMSProp will forget about the past and get out of the bowl

Input: Step size η, numerical stabilizer δ
Input: Exponential decay rate $\rho \in [0, 1[$
Input: Initial parameters θ

set gradient accumulation variable r to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
  $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ *//full gradient, mini-batch...*;
  $r \leftarrow \rho \cdot r + (1-\rho) \cdot (g \odot g)$;
  $v \leftarrow -\frac{\eta}{\sqrt{\delta + r}} \odot g$;
  $\theta \leftarrow \theta + v$;
end

RMSProp (Hinton, 2012)

- Principle: accumulate squared magnitude of gradients, with an exponentially weighted moving average
  - Ancient history is discarded, as old gradients decay exponentially with time
- Example: on a slope with a locally convex bowl:
  - AdaGrad will likely get stuck
  - RMSProp will forget about the past and get out of the bowl

Input: Step size η, numerical stabilizer δ
Input: Exponential decay rate $\rho \in [0, 1[$
Input: Initial parameters θ

set gradient accumulation variable r to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
  $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ *//full gradient, mini-batch...*;
  $r \leftarrow \rho \cdot r + (1 - \rho) \cdot (g \odot g)$;
  $v \leftarrow -\frac{\eta}{\sqrt{\delta + r}} \odot g$;
  $\theta \leftarrow \theta + v$;
end

- Demo 4

AdaDelta (Zeiler, 2012)

- Principle: moving average, similarly to RMSProp

AdaDelta (Zeiler, 2012)

- Principle: moving average, similarly to RMSProp
  - Main difference: **no input step size**
  - Step size is updated depending on previous changes

AdaDelta (Zeiler, 2012)

- Principle: moving average, similarly to RMSProp
  - Main difference: **no input step size**
  - Step size is updated depending on previous changes

Input: Numerical stabilizer δ
Input: Exponential decay rate $\rho \in [0, 1[$
Input: Initial parameters θ

set gradient accumulation variable r to 0;
set parameter update accumulation variable s to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ *//full gradient, mini-batch...*;
    $r \leftarrow \rho \cdot r + (1 - \rho) \cdot (g \odot g)$;
    $v \leftarrow -\frac{\sqrt{\delta + s}}{\sqrt{\delta + r}} \odot g$;
    $s \leftarrow \rho \cdot s + (1 - \rho) \cdot (v \odot v)$;
    $\theta \leftarrow \theta + v$;
end

AdaDelta (Zeiler, 2012)

- Principle: moving average, similarly to RMSProp
  - Main difference: **no input step size**
  - Step size is updated depending on previous changes

Input: Numerical stabilizer δ
Input: Exponential decay rate $\rho \in [0, 1[$
Input: Initial parameters θ

set gradient accumulation variable r to 0;
set parameter update accumulation variable s to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ *//full gradient, mini-batch...*;
    $r \leftarrow \rho \cdot r + (1-\rho) \cdot (g \odot g)$;
    $v \leftarrow -\frac{\sqrt{\delta+s}}{\sqrt{\delta+r}} \odot g$;
    $s \leftarrow \rho \cdot s + (1-\rho) \cdot (v \odot v)$;
    $\theta \leftarrow \theta + v$;
end

- Demo 5

Adam (Kingma & Ba, 2014)

- Can be viewed as a combination of RMSProp and Momentum (coming up later)
  - Momentum is added by considering an exponentially weighted moving average of gradients

Adam (Kingma & Ba, 2014)

- Can be viewed as a combination of RMSProp and Momentum (coming up later)
  - Momentum is added by considering an exponentially weighted moving average of gradients
- Main difference: a bias correction term is applied to the first and second moment variables

Adam (Kingma & Ba, 2014)

- Can be viewed as a combination of RMSProp and Momentum (coming up later)
  - Momentum is added by considering an exponentially weighted moving average of gradients
- Main difference: a bias correction term is applied to the first and second moment variables

Input: Step size η, numerical stabilizer δ
Input: Exponential decay rates $\rho_1, \rho_2 \in [0, 1[$
Input: Initial parameters θ

set first, second moment variable s, r to 0;
for $i \leftarrow 1$ **to** *number of training steps* **do**
  $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ //*full gradient, mini-batch...*;
  $s \leftarrow \rho_1 \cdot s + (1 - \rho_1) \cdot g$;
  $r \leftarrow \rho_2 \cdot r + (1 - \rho_2) \cdot (g \odot g)$;
  $s' \leftarrow \frac{s}{1-\rho_1^i}$;
  $r' \leftarrow \frac{r}{1-\rho_2^i}$;
  $v \leftarrow -\eta \cdot \frac{s'}{\delta + \sqrt{r'}}$;
  $\theta \leftarrow \theta + v$;
end

Adam (Kingma & Ba, 2014)

- Can be viewed as a combination of RMSProp and Momentum (coming up later)
  - Momentum is added by considering an exponentially weighted moving average of gradients
- Main difference: a bias correction term is applied to the first and second moment variables

  Input: Step size η, numerical stabilizer δ
  Input: Exponential decay rates $\rho_1, \rho_2 \in [0, 1[$
  Input: Initial parameters θ

  set first, second moment variable s, r to 0;
  for $i \leftarrow 1$ **to** *number of training steps* **do**
  $\quad g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta)$ //*full gradient, mini-batch...*;
  $\quad s \leftarrow \rho_1 \cdot s + (1 - \rho_1) \cdot g$;
  $\quad r \leftarrow \rho_2 \cdot r + (1 - \rho_2) \cdot (g \odot g)$;
  $\quad s' \leftarrow \frac{s}{1-\rho_1^i}$;
  $\quad r' \leftarrow \frac{r}{1-\rho_2^i}$;
  $\quad v \leftarrow -\eta \cdot \frac{s'}{\delta + \sqrt{r'}}$;
  $\quad \theta \leftarrow \theta + v$;
  end

- Demo 6

Momentum

- Principle: consider a *velocity* variable v

Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum

Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations

Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta)$

Momentum

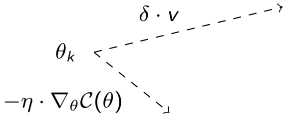
- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta)$

θ_k

$-\eta \cdot \nabla_\theta \mathcal{C}(\theta)$

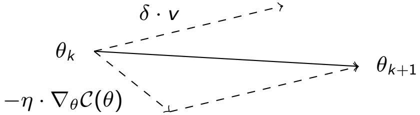
Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta)$



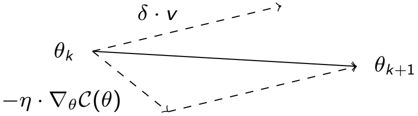
Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta)$



Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta)$



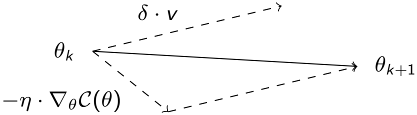
- Algorithm:

Input: Learning rate η, momentum parameter δ
Input: Initial parameters θ, initial velocity $v = 0$

for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta_k)$ //*full gradient, mini-batch...*;
    $v \leftarrow \delta \cdot v - \eta \cdot g$;
    $\theta \leftarrow \theta + v$
end

Momentum

- Principle: consider a *velocity* variable v
  - **Intuition:** v provides the direction and speed at which θ should move toward the optimum
  - **How:** by using information from previous gradient computations
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta)$



- Algorithm:

Input: Learning rate η, momentum parameter δ
Input: Initial parameters θ, initial velocity $v = 0$

for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\theta_k)$ *//full gradient, mini-batch...*;
    $v \leftarrow \delta \cdot v - \eta \cdot g$;
    $\theta \leftarrow \theta + v$
end

- Demo 7

Nesterov momentum

- Principle: use a *modified* velocity variable v

Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient

Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v)$

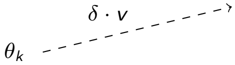
Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v)$

θ_k

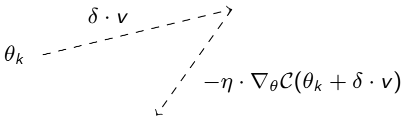
Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v)$



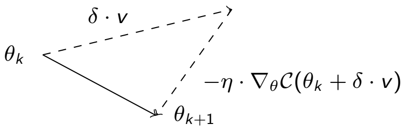
Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v)$



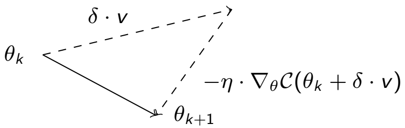
Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v)$



Nesterov momentum

- Principle: use a *modified* velocity variable v
  - Intuition: look ahead with current velocity before computing gradient
  - For $\delta \in [0, 1[$, at each epoch, v is updated by $v \leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v)$



- Algorithm:

Input: Learning rate η, momentum parameter δ
Input: Initial parameters θ, initial velocity $v = 0$

for $i \leftarrow 1$ **to** *number of training steps* **do**
  $\tilde{\theta} \leftarrow \theta + \delta \cdot v$;
  $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\tilde{\theta})$ //*full gradient, mini-batch...*;
  $v \leftarrow \delta \cdot v - \eta \cdot g$;
  $\theta \leftarrow \theta + v$;
end

Implementing Nesterov Accelerated Gradient

- Update rules:

$$
\begin{aligned}
v &\leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v) \\
\theta &\leftarrow \theta + v
\end{aligned}
$$

Implementing Nesterov Accelerated Gradient

- Update rules:

$$\begin{aligned} v &\leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v) \\ \theta &\leftarrow \theta + v \end{aligned}$$

- The gradient computation is *forward looking*

Implementing Nesterov Accelerated Gradient

- Update rules:

$$\begin{aligned} v &\leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v) \\ \theta &\leftarrow \theta + v \end{aligned}$$

- The gradient computation is *forward looking*
- This does not fit well in a generic implementation of forward and backpropagation algorithms

Implementing Nesterov Accelerated Gradient

- Update rules:

$$\begin{aligned} v &\leftarrow \delta \cdot v - \eta \nabla_\theta \mathcal{C}(\theta + \delta \cdot v) \\ \theta &\leftarrow \theta + v \end{aligned}$$

- The gradient computation is *forward looking*
- This does not fit well in a generic implementation of forward and backpropagation algorithms
- **A solution:** “simplified” Nesterov momentum (Bengio et al.: *Advances in optimizing recurrent networks*. 2012)

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$v_{k+1} = \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k)$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$\begin{aligned} v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\ \Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \end{aligned}$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$\begin{aligned} v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\ \Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \;=\; (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \end{aligned}$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$
\begin{aligned}
v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\
\Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \ = \ (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= (\Theta_k - \delta \cdot v_k + v_{k+1}) + \delta \cdot v_{k+1}
\end{aligned}
$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$
\begin{aligned}
v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\
\Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \; = \; (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= (\Theta_k - \delta \cdot v_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot v_{k+1}
\end{aligned}
$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$\begin{aligned}
v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\
\Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \; = \; (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= (\Theta_k - \delta \cdot v_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot (\delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k))
\end{aligned}$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$
\begin{aligned}
v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\
\Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \;=\; (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= (\Theta_k - \delta \cdot v_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot (\delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k)) \\
&= \Theta_k + \delta^2 \cdot v_k - \eta \cdot (1+\delta) \cdot \nabla_\theta \mathcal{C}(\Theta_k)
\end{aligned}
$$

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$
\begin{aligned}
v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\
\Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \;=\; (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= (\Theta_k - \delta \cdot v_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot (\delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k)) \\
&= \Theta_k + \delta^2 \cdot v_k - \eta \cdot (1+\delta) \cdot \nabla_\theta \mathcal{C}(\Theta_k)
\end{aligned}
$$

- Algorithm:

Input: Learning rate η, momentum parameter δ
Input: Initial parameters θ, initial velocity $v = 0$

$\Theta \leftarrow \theta$;
for $i \leftarrow 1$ **to** *number of training steps* **do**
    $g \leftarrow$ computation of $\nabla_\theta \mathcal{C}(\Theta)$ *//full gradient, mini-batch...*;
    $\Theta \leftarrow \Theta + \delta^2 \cdot v - \eta \cdot (1+\delta) \cdot g$;
    $v \leftarrow \delta \cdot v - \eta \cdot g$;
end

Simplified Nesterov momentum

- Perform updates on the lookahead variable $\Theta \stackrel{\text{def}}{=} \theta + \delta \cdot v$
- At iteration $k+1$ we have:

$$\begin{aligned}
v_{k+1} &= \delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k) \\
\Theta_{k+1} &= \theta_{k+1} + \delta \cdot v_{k+1} \;=\; (\theta_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= (\Theta_k - \delta \cdot v_k + v_{k+1}) + \delta \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot v_{k+1} \\
&= \Theta_k - \delta \cdot v_k + (1+\delta) \cdot (\delta \cdot v_k - \eta \cdot \nabla_\theta \mathcal{C}(\Theta_k)) \\
&= \Theta_k + \delta^2 \cdot v_k - \eta \cdot (1+\delta) \cdot \nabla_\theta \mathcal{C}(\Theta_k)
\end{aligned}$$

- Algorithm:

Input: Learning rate η, momentum parameter δ
Input: Initial parameters θ, initial velocity $v = 0$

```
Θ ← θ;
for i ← 1 to number of training steps do
    g ← computation of ∇_θ C(Θ) //full gradient, mini-batch...;
    Θ ← Θ + δ² · v − η · (1 + δ) · g;
    v ← δ · v − η · g;
end
```

- If the optimum is reached at step n ($v_n = 0$) then $\Theta_n = \theta_n$

Which optimization to choose?

- No clear answer

Which optimization to choose?

- No clear answer
- Momentum and Adam are quite popular

Which optimization to choose?

- No clear answer
- Momentum and Adam are quite popular
- Recommendation: choose an optimizer and practice tuning its hyperparameters

Which optimization to choose?

- No clear answer
- Momentum and Adam are quite popular
- Recommendation: choose an optimizer and practice tuning its hyperparameters
- Some reading material
  - Momentum: `https://distill.pub/2017/momentum/`
  - Adam: `https://arxiv.org/abs/1412.6980`