opt vs Training.

SGD:

\[
\text{\(\theta = \text{get\_initial\_params()}; \) }
\]
\[
\epsilon = 0.001
\]

while stopping criterion not met
\[
\text{\(x, y = \text{get\_mini\_batch()};\) }
\]
compute \(\nabla \theta J\)
\[
\theta \leftarrow \theta - \epsilon \nabla \theta J
\]

- Momentum
- Adagrad
Parameter Initialization

Algorithms are strongly affected by the choice of initialization.
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights

Xavier initialization:

$$W_{ij} = \mathcal{U}(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}})$$
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

\[
W_{ij} = \begin{cases} 
\frac{-\sqrt{6}}{\sqrt{m+n}}, & \text{if } i \leq \frac{m}{2} \\
\frac{\sqrt{6}}{\sqrt{m+n}}, & \text{if } i > \frac{m}{2}
\end{cases}
\]
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

  Computationally cheaper
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

  Computationally cheaper

  Unlikely to assign any units to compute the same function as each other

Scale of the distribution?

\[
\begin{array}{l}
W_{ij} = \mathcal{U} \left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)
\end{array}
\]
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

- Computationally cheaper
- Unlikely to assign any units to compute the same function as each other

Scale of the distribution?
Smaller initial weights
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

- Computationally cheaper
- Unlikely to assign any units to compute the same function as each other

Scale of the distribution?
Smaller initial weights $\rightarrow$ vanishing gradient
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

  Computationally cheaper
  Unlikely to assign any units to compute the same function as each other

Scale of the distribution?
Smaller initial weights $\rightarrow$ vanishing gradient
Larger initial weights

$W_{i,j} = \mathcal{U}\left(\frac{-\sqrt{6}}{\sqrt{m+n}}, \frac{\sqrt{6}}{\sqrt{m+n}}\right)$
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

  Computationally cheaper

  Unlikely to assign any units to compute the same function as each other

Scale of the distribution?
Smaller initial weights $\rightarrow$ vanishing gradient
Larger initial weights $\rightarrow$ exploding values during forward/backward propagation

Xavier initialization:
$$W_{ij} = \mathcal{U}(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}})$$
Parameter Initialization

Algorithms are strongly affected by the choice of initialization
No two units should have the same weights
Random initialization from a high-entropy distribution

- Computationally cheaper
- Unlikely to assign any units to compute the same function as each other

Scale of the distribution?
Smaller initial weights $\rightarrow$ vanishing gradient
Larger initial weights $\rightarrow$ exploding values during forward/backward propagation

Xavier initialization:

$$W_{i,j} = U\left(-\frac{\sqrt{6}}{\sqrt{m+n}}, \frac{\sqrt{6}}{\sqrt{m+n}}\right)$$
Batch Normalization

Gradient based learning:

\[ \theta \leftarrow \theta - \epsilon g \]

This is problematic for deep neural networks.
Gradient based learning:

Parameter updates under the assumption that the other layers do not change
Batch Normalization

Gradient based learning:

Parameter updates under the assumption that the other layers do not change

Gradient updates are computed and applied simultaneously

$$\theta \leftarrow \theta - \epsilon g$$
Batch Normalization

Gradient based learning:

Parameter updates under the assumption that the other layers do not change
Gradient updates are computed and applied simultaneously

$$\theta \leftarrow \theta - \epsilon g$$

This is problematic for deep neural networks
Problem

Taylor approx: \( f(x+\varepsilon g) \approx f(x) - \varepsilon \frac{\partial f}{\partial x} + \varepsilon^2 \frac{\partial^2 f}{\partial x^2} H_\alpha + \cdots \)

\[ \hat{g} = x, \omega_1, \omega_2, \ldots, \omega_m \]

\[ w \leftarrow w - \varepsilon g \]

\[ \hat{y}(w-\varepsilon g) \approx \hat{y}(w) - \varepsilon \frac{\partial f}{\partial x} \]

First order approx.

\[ \varepsilon \frac{\partial f}{\partial x} = 0.1 \rightarrow \varepsilon = \frac{0.1}{\partial f/\partial x} \]

\[ \hat{y} = x(w_1-\varepsilon g)(w_2-\varepsilon g)\cdots(w_m-\varepsilon g) \]

\[ \varepsilon g^T H \hat{g} = \begin{pmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_m} \end{pmatrix} \begin{pmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_m} \end{pmatrix} \]

For DNNs, \( \frac{\partial f}{\partial w} = \sum_{i=3}^{m} w_i \)
Problem
Solution: Batch Normalization

Let \( \mathbf{H} \) be a mini-batch of activations of the layer
Replace it with

\[
\mathbf{H}' = \frac{\mathbf{H} - \mu}{\sigma}
\]

\[
\mu = \frac{1}{m} \sum_{i=1}^{m} H_i
\]

\[
\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (H_i - \mu)^2}
\]

\[
\hat{y} = \mathbf{h}_{l-1} \omega_l
\]

⇒ if \( x \) is normally distributed

\( \mathbf{h}_{l-1} \): standard normal distribution
Solution: Batch Normalization

Normalizing a unit can reduce the expressive power of the neural network containing that unit.
Solution: Batch Normalization

Normalizing a unit can reduce the expressive power of the neural network containing that unit.

Make the mean and standard deviation a trainable parameter:

\[ \gamma H' + \beta \]

\[ \Theta = [w, b, \gamma, \beta] \]