Error of a Neural Network

\[ E_{in}(W^{(1)}, ..., W^{(L)}) = \frac{1}{n} \sum_{i=1}^{n} e(h(x_i | W^{(1)}, ..., W^{(L)}), y_i) \]

*awkward silence*
• Input: \( \mathcal{D} = \{(\overrightarrow{x_1}, y_1), \ldots, (\overrightarrow{x_n}, y_n)\}, \eta_0 \)

• Initialize all weights \( W^{(1)}(0), \ldots, W^{(L)}(0) \) to small, random numbers and set \( t = 0 \)

• While some termination condition is not satisfied
  • For \( l = 1, \ldots, L \)
    • Compute \( G^{(l)} = \nabla_{W^{(l)}} E_{in}(W^{(1)}(t), \ldots, W^{(L)}(t)) \)
    • Update \( W^{(l)}: W^{(l)}(t+1) = W^{(l)}(t) - \eta_0 G^{(l)} \)
  • Increment \( t: t = t + 1 \)

• Output: \( W^{(1)}(t), \ldots, W^{(L)}(t) \)
Computing Gradients

\[ E_{in} (W^{(1)}_t, ..., W^{(L)}_t) = \frac{1}{n} \sum_{i=1}^{n} e \left( h \left( \vec{x}_i \mid W^{(1)}_t, ..., W^{(L)}_t \right), y_i \right) \]

\[ \nabla_{w^{(l)}} E_{in} \left( W^{(1)}_t, ..., W^{(L)}_t \right) = \begin{bmatrix}
\frac{\partial E_{in}}{\partial w^{(l)}_{01}} & \frac{\partial E_{in}}{\partial w^{(l)}_{02}} & \cdots & \frac{\partial E_{in}}{\partial w^{(l)}_{0d^{(l)}}} \\
\frac{\partial E_{in}}{\partial w^{(l)}_{11}} & \frac{\partial E_{in}}{\partial w^{(l)}_{12}} & \cdots & \frac{\partial E_{in}}{\partial w^{(l)}_{1d^{(l)}}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial E_{in}}{\partial w^{(l)}_{d^{(l-1)}1}} & \frac{\partial E_{in}}{\partial w^{(l)}_{d^{(l-1)}2}} & \cdots & \frac{\partial E_{in}}{\partial w^{(l)}_{d^{(l-1)}d^{(l)}}}
\end{bmatrix} \]
Computing Gradients

\[ E_{in} \left( W^{(1)}(t), ..., W^{(L)}(t) \right) = \frac{1}{n} \sum_{i=1}^{n} e \left( h \left( \vec{x}_i \mid W^{(1)}(t), ..., W^{(L)}(t) \right), y_i \right) \]

\[ \frac{\partial E_{in} \left( W^{(1)}(t), ..., W^{(L)}(t) \right)}{\partial w_{ab}^{(l)}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial e \left( h \left( \vec{x}_i \mid W^{(1)}(t), ..., W^{(L)}(t) \right), y_i \right)}{\partial w_{ab}^{(l)}} \]

\[ = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial e \left( \vec{x}^{(L)}, y_i \right)}{\partial w_{ab}^{(l)}} \]
Computing \( \nabla_{W^{(t)}} E_{in} \left( W^{(1)}_t, ..., W^{(L)}_t \right) \) reduces to computing

\[
\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial w^{(l)}_{ab}}
\]

Insight: \( w^{(l)}_{ab} \) only affects \( e \left( x^{(L)}, y_i \right) \) via \( s^{(l)}_b \)
Computing $\nabla_{W^{(l)}} E_{in} \left( W_{(1)}^{(t)}, ..., W_{(t)}^{(L)} \right)$ reduces to computing

$$\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial w_{ab}^{(l)}}$$

Insight: $w_{ab}^{(l)}$ only affects $e \left( x^{(L)}, y_i \right)$ via $s_b^{(l)}$

Use the chain rule:

$$\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial w_{ab}^{(l)}} = \frac{\partial e \left( x^{(L)}, y_i \right)}{\partial s_b^{(l)}} \left( \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} \right)$$
Computing Partial Derivatives

To compute \( \frac{\partial e(x^{(L)}, y_i)}{\partial s_b^{(l)}} \left( \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} \right) \), recall:

\[
s_b^{(l)} = \sum_{a=0}^{d^{(l-1)}} w_{ab}^{(l)} x_a^{(l-1)} \rightarrow \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} = x_a^{(l-1)}
\]

Can compute all outputs \( \hat{x}^{(l)} \) \( \forall \ l \in \{0, \ldots, L\} \) using forward propagation.

Let \( \delta_b^{(l)} = \frac{\partial e(x^{(L)}, y_i)}{\partial s_b^{(l)}} \)

\( \delta^{(l)} \) is called the sensitivity vector for layer \( l \).
Insight: $s_b^{(l)}$ only affects $e \left( x^{(L)}, y_i \right)$ via $x_b^{(l)}$
Computing Sensitivities

Insight: $s_b^{(l)}$ only affects $e \left( x^{(l)}, y_i \right)$ via $x_b^{(l)}$

Use the chain rule: $\delta_b^{(l)} = \frac{\partial e \left( x^{(l)}, y_i \right)}{\partial x_b^{(l)}} \left( \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} \right)$

Recall: $x_b^{(l)} = \theta \left( s_b^{(l)} \right) \rightarrow \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta \left( s_b^{(l)} \right)}{\partial s_b^{(l)}}$

$= 1 - \left( x_b^{(l)} \right)^2$

when $\theta(\cdot) = \tanh(\cdot)$
Insight: $x_b^{(l)}$ affects $e(x^{(l)}, y_i)$ via $s_{1}^{(l+1)}, ..., s_{d}^{(l+1)}$.
Computing Sensitivities

Insight: $x_b^{(l)}$ affects $e \left( x^{(L)}, y_i \right)$ via $s_1^{(l+1)}, \ldots, s_{d(l+1)}^{(l+1)}$

Use the chain rule:

$$\frac{\partial e \left( x^{(L)}, y_i \right)}{\partial x_b^{(l)}} = \sum_{c=1}^{d(l+1)} \frac{\partial e \left( x^{(L)}, y_i \right)}{\partial s_c^{(l+1)}} \left( \frac{\partial s_c^{(l+1)}}{\partial x_b^{(l)}} \right)$$

$$= \sum_{c=1}^{d(l+1)} \delta_c^{(l+1)} \left( w_{bc}^{(l+1)} \right)$$
Computing Sensitivities

\[
\delta^{(l)}_b = \frac{\partial e(x^{(l)}, y_i)}{\partial x^{(l)}_b} \left( \frac{\partial x^{(l)}_b}{\partial s^{(l)}_b} \right)
\]

\[
= \left( \sum_{c=1}^{d^{(l+1)}} \delta^{(l+1)}_c \left( w^{(l+1)}_{bc} \right) \right) \left( 1 - (x^{(l)}_b)^2 \right)
\]

\[
\delta^{(l)} = W^{(l+1)} \delta^{(l+1)} \otimes \left( 1 - x^{(l)} \otimes x^{(l)} \right)
\]

where \( \otimes \) is the element-wise product operation.

Sanity check: \( W^{(l+1)} \in \mathbb{R}^{d^{(l+1)+1} \times d^{(l+1)}} \) and

\[
\delta^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times 1}
\]

so

\[
W^{(l+1)} \delta^{(l+1)} \in \mathbb{R}^{(d^{(l)+1}) \times 1}, \text{ the same size as } x^{(l)}
\]
Computing $\nabla_{W^{(l)}} E_{in} (W^{(1)}_{(t)}, ..., W^{(L)}_{(t)})$ reduces to computing

$$\frac{\partial e (x^{(L)}, y_i)}{\partial w_{ab}^{(l)}} = \frac{\partial e (x^{(L)}, y_i)}{\partial s_b^{(l)}} \left( \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} \right)$$

$$= \frac{\partial e (x^{(L)}, y_i)}{\partial x_b^{(l)}} \left( \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} \right) \left( \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} \right)$$

$$= \sum_{c=1}^{d^{(l+1)}} \frac{\partial e (x^{(L)}, y_i)}{\partial s_c^{(l+1)}} \left( \frac{\partial s_c^{(l+1)}}{\partial x_b^{(l)}} \right) \left( \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} \right) \left( \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} \right)$$

$$\delta_b^{(l)}$$
Computing Partial Derivatives

\[
\frac{\partial e(x^{(L)}, y_i)}{\partial w^{(l)}_{ab}} = \delta^{(l)}_b \left( \frac{\partial s^{(l)}_b}{\partial w^{(l)}_{ab}} \right) = \delta^{(l)}_b \left( x^{(l-1)}_a \right)
\]

\[
\nabla_{W^{(l)}} e(x^{(L)}, y_i) = x^{(l-1)} \left( \delta^{(l)} \right)^T
\]

Sanity check: \(x^{(l-1)} \in \mathbb{R}^{(d^{(l-1)}+1) \times 1}\) and \(\delta^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}\) so

\[
x^{(l-1)} \left( \delta^{(l)} \right)^T \in \mathbb{R}^{(d^{(l-1)}+1) \times d^{(l)}}, \text{ the same size as } W^{(l)}
\]
Can recursively compute $\delta^{(l)}$ using $\delta^{(l+1)}$; need to compute the base case: $\delta^{(L)}$.

Assume the output layer is a single node and the error function is the squared error:

$$\delta^{(L)} = \delta_1^{(L)}, x^{(L)} = x_1^{(L)} \text{ and } e\left(x_1^{(L)}, y_i\right) = \left(x_1^{(L)} - y_i\right)^2$$

$$\delta_1^{(L)} = \frac{\partial e\left(x_1^{(L)}, y_i\right)}{\partial s_1^{(L)}} = \frac{\partial}{\partial s_1^{(L)}} \left(x_1^{(L)} - y_i\right)^2$$

$$= 2 \left(x_1^{(L)} - y_i\right) \frac{\partial x_1^{(L)}}{\partial s_1^{(L)}} = 2 \left(x_1^{(L)} - y_i\right) \left(1 - \left(x_1^{(L)}\right)^2\right)$$

when $\theta(\cdot) = \tanh(\cdot)$
Back-propagation

- Input: weights $W^{(1)}, \ldots, W^{(L)}$ and a query point $\mathbf{x}$
- Run forward propagation to get $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}$
- Initialize $\delta^{(L)}_1 = 2 \left( x^{(L)}_1 - y_i \right) \left( 1 - (x^{(L)}_1)^2 \right)$
- For $l = L - 1, \ldots, 1$
  - Compute $\delta^{(l)} = W^{(l+1)} \delta^{(l+1)} \otimes \left( 1 - x^{(l)} \otimes x^{(l)} \right)$
- Output: $\delta^{(1)}, \ldots, \delta^{(L)}$
Computing Gradients

- **Input:** \( W^{(1)}, \ldots, W^{(L)} \) and \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \)
- **Initialize** \( E_{in} = 0 \) and \( G^{(l)} = 0 \) \( \forall l = 1, \ldots, L \)
- **For** \( i = 1, \ldots, n \)
  - Run forward propagation to get \( x^{(1)}, \ldots, x^{(L)} \)
  - Run backpropagation to get \( \delta^{(1)}, \ldots, \delta^{(L)} \)
  - Increment \( E_{in} \): \( E_{in} = E_{in} + \frac{1}{n}(x^{(L)} - y_i)^2 \)
- **For** \( l = 1, \ldots, L \)
  - Compute \( G^{(l)} = x^{(l-1)}(\delta^{(l)})^T \)
  - Increment \( G^{(l)} \): \( G^{(l)} = G^{(l)} + \frac{1}{n}G^{(l)}_i \)
- **Output:** \( G^{(1)}, \ldots, G^{(L)} \), the gradients of \( E_{in} \) w.r.t \( W^{(1)}, \ldots, W^{(L)} \)
Complexity

• Both forward and backpropagation contain matrix multiplications involving $W^{(1)}, ..., W^{(L)} \rightarrow$ both take time $O(|W^{(1)}| + \cdots + |W^{(L)}|)$ ...

• Computing $G^{(1)}, ..., G^{(L)}$ requires running forward and backpropagation for each training point $(\vec{x}, y) \in \mathcal{D}$ ...

• Each iteration of gradient descent for a neural network takes time $O\left(n(|W^{(1)}| + \cdots + |W^{(L)}|)\right)$

• Can use parallelization via GPUs / TPUs
• Gradient descent is not guaranteed to find a global minimum on non-convex surfaces
Which of the following strategies can help gradient descent address non-convexity?

- Use a random subset of training points when computing gradients
- Randomly terminate the algorithm before satisfying the termination criteria
- Take completely random step sizes
- Run the algorithm many times with different initializations
• Input: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}, \eta_0$

• Initialize all weights $W^{(1)}_0, \ldots, W^{(L)}_0$ to small, random numbers and set $t = 0$

• While some termination condition is not satisfied
  • For $l = 1, \ldots, L$
    • Randomly select a point $(\mathbf{x}_i, y_i)$
    • Compute $G^{(l)'} = \nabla_{W^{(l)}} \mathbb{E} \left( h \left( \mathbf{x}_i | W^{(1)}_t, \ldots, W^{(L)}_t \right), y_i \right)$
    • Update $W^{(l)}: W^{(l)}_{(t+1)} = W^{(l)}_{(t)} - \eta_0 G^{(l)'}$
  • Increment $t: t = t + 1$

• Output: $W^{(1)}_t, \ldots, W^{(L)}_t$
Mini-batch Gradient Descent for Neural Networks

- **Input:** \( D = \{(x_1, y_1), \ldots, (x_n, y_n)\}, \eta_0, m \)

- **Initialize all weights** \( W^{(1)}_0, \ldots, W^{(L)}_0 \) to small, random numbers and set \( t = 0 \)

- **While some termination condition is not satisfied**
  - **For** \( l = 1, \ldots, L \)
    - Randomly select \( m \) points \( \{(x_{i_1}, y_{i_1}), \ldots, (x_{i_m}, y_{i_m})\} \)
    - Compute \( G^{(l)}' = \frac{1}{m} \sum_{j=1}^{m} \nabla_{W^{(l)}} e \left( h \left( x_{i_j} \mid W^{(1)}_t, \ldots, W^{(L)}_t \right), y_{i_j} \right) \)
    - Update \( W^{(l)}: W^{(l)}_{(t+1)} = W^{(l)}_t - \eta_0 G^{(l)}' \)
  - Increment \( t: t = t + 1 \)

- **Output:** \( W^{(1)}_t, \ldots, W^{(L)}_t \)
**Mini-batch Gradient Descent with Momentum for Neural Networks**

**Input:** \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}, \eta_0, m, \beta \)

**Initialize all weights** \( W^{(1)}_0, \ldots, W^{(L)}_0 \) to small, random numbers and set \( t = 0, G^{(l)}_{-1} = 0 \ast W^{(l)} \) \( \forall l = 1, \ldots, L \)

**While some termination condition is not satisfied**
- For \( l = 1, \ldots, L \)
  - Randomly select \( m \) points \( \{(x_{i_1}, y_{i_1}), \ldots, (x_{i_m}, y_{i_m})\} \)
  - Compute \( G^{(l)}_t = \frac{1}{m} \sum_{j=1}^{m} \nabla_{W^{(l)}} e \left( h \left( x_{i_j} | W^{(1)}_t, \ldots, W^{(L)}_t \right), y_{i_j} \right) \)
  - Update \( W^{(l)}_t : W^{(l)}_{(t+1)} = W^{(l)}_{(t)} - \eta_0 \left( \beta G^{(l)}_{t-1} + G^{(l)}_t \right) \)
  - Increment \( t : t = t + 1 \)

**Output:** \( W^{(1)}_t, \ldots, W^{(L)}_t \)
Mini-batch Gradient Descent with Momentum for Neural Networks
Mini-batch Gradient Descent with Momentum for Neural Networks
Mini-batch Gradient Descent with Momentum for Neural Networks
Random Restarts

- Run (mini-batch) gradient descent (with momentum) multiple times, each time starting with a different, random initialization for the weights.
- Compute the in-sample error of each run at termination and return the set of weights corresponding to the run with the lowest in-sample error.
Random Restarts
Random Restarts
For non-convex surfaces, the gradient’s magnitude is often not a good metric for proximity to a minimum.
For non-convex surfaces, the gradient’s magnitude is often not a good metric for proximity to a minimum.

Combine multiple termination criteria e.g. only stop if enough iterations have passed and the improvement in error is small.

Alternatively, terminate early by using a validation data set: if the validation error starts to increase, just stop!

- Early stopping asks like regularization by \textbf{limiting how much of the hypothesis set} is explored by gradient descent.
• Minimize $E_{aug}(W^{(1)}, ..., W^{(L)}, \lambda_c)$
  
  $= E_{in} (W^{(1)}, ..., W^{(L)}) + \frac{\lambda_c}{n} \Omega(W^{(1)}, ..., W^{(L)})$

• Weight decay:

  $\Omega(W^{(1)}, ..., W^{(L)}) = \sum_{l=1}^{L} \sum_{i=0}^{d^{(l-1)}} \sum_{j=1}^{d^{(l)}} (w_{ij}^{(l)})^2$

• Weight elimination:

  $\Omega(W^{(1)}, ..., W^{(L)}) = \sum_{l=1}^{L} \sum_{i=0}^{d^{(l-1)}} \sum_{j=1}^{d^{(l)}} \frac{(w_{ij}^{(l)})^2}{1 + (w_{ij}^{(l)})^2}$
• **Dropout**
  - In each iteration of gradient descent, randomly remove some of the nodes in the network
  - Compute the gradient using only the remaining nodes
  - The weights on edges going into and out of “dropped out” nodes are not updated

Neural Networks and “Strange” Regularization

- **Jitter**
  - In each iteration of gradient descent, add some random noise or “jitter” to each training data point
  - Instead of computing the gradient w.r.t. \((\vec{x}_i, y_i)\), use \((\vec{x}_i + \vec{\varepsilon}, y_i)\) where \(\vec{\varepsilon} \sim N(\vec{0}, \sigma^2 I)\)
  - Makes neural networks resilient to input noise, like SVMs