Neural Networks: Learning Parameters

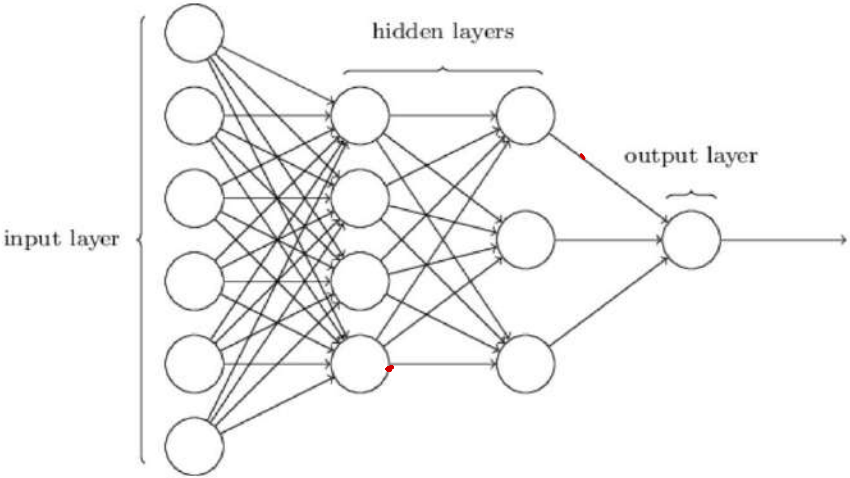
Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Data Mining and Machine Learning
August–December 2020

Neural networks

- Acyclic network of perceptrons with non-linear activation functions



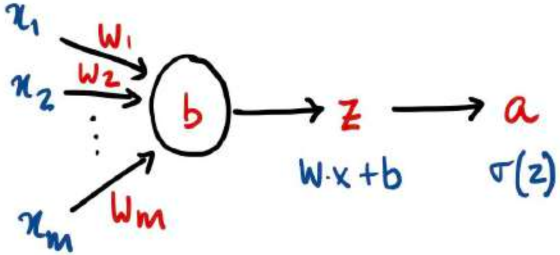
Neural networks

- Without loss of generality,
  - Assume the network is layered
    - All paths from input to output have the same length
  - Each layer is fully connected to the previous one
    - Set weight to 0 if connection is not needed

Neural networks

- Without loss of generality,
  - Assume the network is layered
    - All paths from input to output have the same length
  - Each layer is fully connected to the previous one
    - Set weight to 0 if connection is not needed

- Structure of an individual neuron
  - Input weights $w_1, \ldots, w_m$, bias b, output z, activation value a

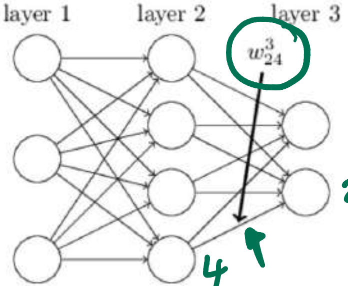


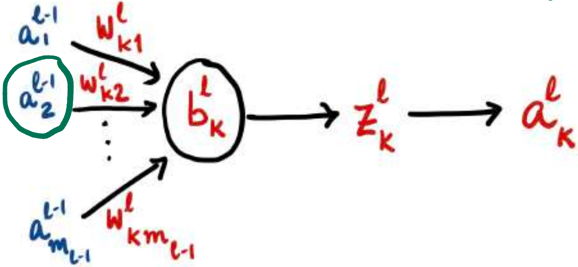
Notation

- Layers $\ell \in \{1, 2, \ldots, L\}$
  - Inputs are connected first hidden layer, layer 1
  - Layer L is the output layer
- Layer ℓ has m_ℓ nodes $1, 2, \ldots, m_\ell$

Notation

- Layers $\ell \in \{1, 2, \ldots, L\}$
  - Inputs are connected first hidden layer, layer 1
  - Layer L is the output layer
- Layer ℓ has m_ℓ nodes $1, 2, \ldots, m_\ell$
- Node k in layer ℓ has bias b^ℓ_k, output z^ℓ_k and activation value a^ℓ_k
- Weight on edge from node j in level $\ell-1$ to node k in level ℓ is w^ℓ_{kj}





Notation

- Why the inversion of indices in the subscript w^ℓ_{kj}?
  - $z^\ell_k = w^\ell_{k1} a^{\ell-1}_1 + w^\ell_{k2} a^{\ell-1}_2 + \cdots + w^\ell_{km_{\ell-1}} a^{\ell-1}_{m_{\ell-1}}$
  - Let $\overline{w}^\ell_k = (w^\ell_{k1}, w^\ell_{k2}, \ldots, w^\ell_{km_{\ell-1}})$
    and $\overline{a}^{\ell-1} = (a^{\ell-1}_1, a^{\ell-1}_2, \ldots, a^{\ell-1}_{m_{\ell-1}})$
  - Then $z^\ell_k = \overline{w}^\ell_k \cdot \overline{a}^{\ell-1}$

Notation

- Why the inversion of indices in the subscript w^ℓ_{kj}?
  - $z^\ell_k = w^\ell_{k1} a^{\ell-1}_1 + w^\ell_{k2} a^{\ell-1}_2 + \cdots + w^\ell_{km_{\ell-1}} a^{\ell-1}_{m_{\ell-1}}$
  - Let $\overline{w}^\ell_k = (w^\ell_{k1}, w^\ell_{k2}, \ldots, w^\ell_{km_{\ell-1}})$
    and $\overline{a}^{\ell-1} = (a^{\ell-1}_1, a^{\ell-1}_2, \ldots, a^{\ell-1}_{m_{\ell-1}})$
  - Then $z^\ell_k = \overline{w}^\ell_k \cdot \overline{a}^{\ell-1}$
- Assume all layers have same number of nodes
  - Let $m = \max_{\ell \in \{1.2,\ldots,L\}} m_\ell$
  - For any layer i, for $k > m_i$, we set all of $w^\ell_{kj}, b^\ell_k, z^\ell_k, a^\ell_k$ to 0
- Matrix formulation

$$\begin{bmatrix} \overline{z}^\ell_1 \\ \overline{z}^\ell_2 \\ \cdots \\ \overline{z}^\ell_m \end{bmatrix} = \begin{bmatrix} \overline{w}^\ell_1 \\ \overline{w}^\ell_2 \\ \cdots \\ \overline{w}^\ell_m \end{bmatrix} \begin{bmatrix} a^{\ell-1}_1 \\ a^{\ell-1}_2 \\ \cdots \\ a^{\ell-1}_m \end{bmatrix}$$

Learning the parameters

- Need to find optimum values for all weights w^{ℓ}_{kj} , b^{ℓ}_{k}
- Use gradient descent
  - Cost function C, partial derivatives $\frac{\partial C}{\partial w^{\ell}_{kj}}, \frac{\partial C}{\partial b^{\ell}_{k}}$

Learning the parameters

- Need to find optimum values for all weights w^{ℓ}_{kj}
- Use gradient descent
  - Cost function C, partial derivatives $\dfrac{\partial C}{\partial w^{\ell}_{kj}}, \dfrac{\partial C}{\partial b^{\ell}_{k}}$
- Assumptions about the cost function

Learning the parameters

- Need to find optimum values for all weights w_{kj}^{ℓ}
- Use gradient descent
  - Cost function C, partial derivatives $\frac{\partial C}{\partial w_{kj}^{\ell}}, \frac{\partial C}{\partial b_k^{\ell}}$
- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function of only the output layer activation, a^L
     - For instance, for training input $(\boldsymbol{x}_i, y_i)$, sum-squared error is $(y_i - a_i^L)^2$
     - Note that $\boldsymbol{x}_i$, y_i are fixed values, only a_i^L is a variable

Learning the parameters

- Need to find optimum values for all weights w_{kj}^{ℓ}
- Use gradient descent
  - Cost function C, partial derivatives $\dfrac{\partial C}{\partial w_{kj}^{\ell}}$, $\dfrac{\partial C}{\partial b_{k}^{\ell}}$
- Assumptions about the cost function
  1. For input $\mathbf{x}$, $C(\mathbf{x})$ is a function of only the output layer activation, a^L
     - For instance, for training input $(\mathbf{x}_i, y_i)$, sum-squared error is $(y_i - a_i^L)^2$
     - Note that $\mathbf{x}_i$, y_i are fixed values, only a_i^L is a variable
  2. Total cost is average of individual input costs
     - Each input $\mathbf{x}_i$ incurs cost $C(\mathbf{x}_i)$, total cost is $\dfrac{1}{n}\sum_{i=1}^{n} C(\mathbf{x}_i)$
     - For instance, mean sum-squared error $\dfrac{1}{n}\sum_{i=1}^{n}(y_i - a_i^L)^2$

Learning the parameters

- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function of only the output layer activation, a^L
  2. Total cost is average of individual input costs
- With these assumptions:
  - We can write $\dfrac{\partial C}{\partial w^\ell_{kj}}, \dfrac{\partial C}{\partial b^\ell_k}$ in terms of individual $\dfrac{\partial a^L_i}{\partial w^\ell_{kj}}, \dfrac{\partial a^L_i}{\partial b^\ell_k}$
  - Can extrapolate change in individual cost $C(x)$ to change in overall cost C — stochastic gradient descent

$X = [x_1, x_2, x_3, \ldots, x_n]$ $\quad \mu = \text{mean}(X)$

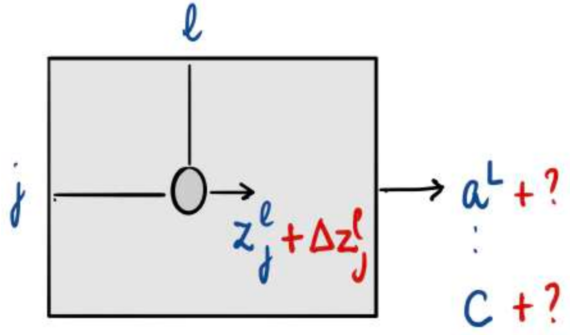
$Y = [x_1, x_3, x_7] \rightarrow \mu_Y \rightsquigarrow \mu$

Learning the parameters

- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function of only the output layer activation, a^L
  2. Total cost is average of individual input costs
- With these assumptions:
  - We can write $\frac{\partial C}{\partial w_{kj}^{\ell}}$, $\frac{\partial C}{\partial b_k^{\ell}}$ in terms of individual $\frac{\partial a_i^L}{\partial w_{kj}^{\ell}}$, $\frac{\partial a_i^L}{\partial b_k^{\ell}}$
  - Can extrapolate change in individual cost $C(x)$ to change in overall cost C — stochastic gradient descent
- Complex dependency of C on w_{kj}^{ℓ}, b_k^{ℓ}
  - Many intermediate layers
  - Many paths through these layers
- Use chain rule to decompose into local dependencies
  - $y = g(f(x)) \Rightarrow \frac{\partial g}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$

Calculating dependencies

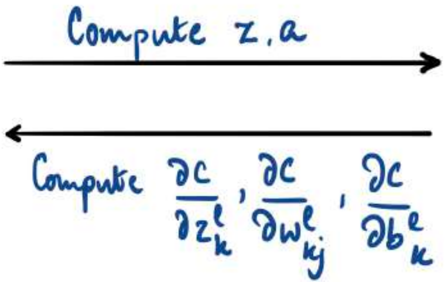
- If we perturb the output z_j^ℓ at node j in layer ℓ, what is the impact on final output, overall cost?



- Focus on $\frac{\partial C}{\partial z_j^\ell}$ — from these, we can compute $\frac{\partial C}{\partial w_{kj}^\ell}, \frac{\partial C}{\partial b_k^\ell}$

Computing partial derivatives

- Use chain rule to run backpropagation algorithm
  - Given an input, execute the network from left to right to compute all outputs
  - Using the chain rule, work backwards from right to left to compute all values of $\dfrac{\partial C}{\partial z_j^\ell}$



Applying the chain rule

Let δ_j^ℓ denote $\dfrac{\partial C}{\partial z_j^\ell}$

Applying the chain rule

Let δ_j^ℓ denote $\frac{\partial C}{\partial z_j^\ell}$

Induction : $\ell = L, L-1, \ldots, 1$

Base Case

$\ell = L$, δ_j^L

- Chain rule: $\frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}$

Applying the chain rule

Let δ_j^ℓ denote $\dfrac{\partial C}{\partial z_j^\ell}$

Base Case

$\ell = L$, δ_j^L

- Chain rule: $\dfrac{\partial C}{\partial z_j^L} = \dfrac{\partial C}{\partial a_j^L} \dfrac{\partial a_j^L}{\partial z_j^L}$
- $C = \dfrac{1}{n}\sum_{i=1}^{n}(y_i - a_i^L)^2$, so $\dfrac{\partial C}{\partial a_j^L} = 2(y_j - a_j^L)(-1) = 2(a_j^L - y_j)$

MSE

Applying the chain rule

Let δ_j^ℓ denote $\dfrac{\partial C}{\partial z_j^\ell}$

Base Case

$\ell = L$, δ_j^L

- Chain rule: $\dfrac{\partial C}{\partial z_j^L} = \dfrac{\partial C}{\partial a_j^L}\dfrac{\partial a_j^L}{\partial z_j^L}$
- $C = \dfrac{1}{n}\sum_{i=1}^{n}(y_i - a_i^L)^2$, so $\dfrac{\partial C}{\partial a_j^L} = 2(y_j - a_j^L)(-1) = 2(a_j^L - y_j)$
- $a_j^L = \sigma(z_j^L)$, so $\dfrac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L)$

Applying the chain rule

Let δ_j^ℓ denote $\dfrac{\partial C}{\partial z_j^\ell}$

Base Case

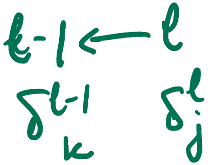
$\ell = L$, δ_j^L

- Chain rule: $\dfrac{\partial C}{\partial z_j^L} = \dfrac{\partial C}{\partial a_j^L} \dfrac{\partial a_j^L}{\partial z_j^L}$
- $C = \dfrac{1}{n}\sum_{i=1}^{n}(y_i - a_i^L)^2$, so $\dfrac{\partial C}{\partial a_j^L} = 2(y_j - a_j^L)(-1) = 2(a_j^L - y_j)$
- $a_j^L = \sigma(z_j^L)$, so $\dfrac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L)$
  - $\sigma(u) = \dfrac{1}{1+e^{-u}}$, $\sigma'(u) = \dfrac{\partial \sigma(u)}{\partial u} = \sigma(u)(1-\sigma(u))$ Work this out!

Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

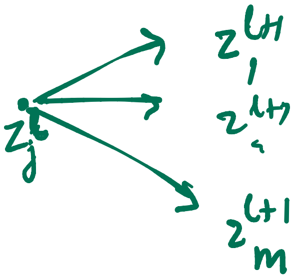


Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \frac{\partial C}{\partial z_j^{\ell}} = \sum_{k=1}^{m} \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$



Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \frac{\partial C}{\partial z_j^{\ell}} = \sum_{k=1}^{m} \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$
- First term inside summation: $\frac{\partial C}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$

Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \dfrac{\partial C}{\partial z_j^{\ell}} = \displaystyle\sum_{k=1}^{m} \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$
- First term inside summation: $\dfrac{\partial C}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$
- Second term: $z_k^{\ell+1} = \displaystyle\sum_{i=1}^{m} w_{ki}^{\ell+1} a_i^{\ell} + b_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}$

Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \dfrac{\partial C}{\partial z_j^{\ell}} = \displaystyle\sum_{k=1}^{m} \dfrac{\partial C}{\partial z_k^{\ell+1}} \dfrac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$
- First term inside summation: $\dfrac{\partial C}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$
- Second term: $z_k^{\ell+1} = \displaystyle\sum_{i=1}^{m} w_{ki}^{\ell+1} a_i^{\ell} + b_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}$
  - For $i \neq j$, $\dfrac{\partial}{\partial z_j^{\ell}} [w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}] = 0$

↑ z_j^{ℓ}

Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \frac{\partial C}{\partial z_j^{\ell}} = \sum_{k=1}^{m} \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$
- First term inside summation: $\frac{\partial C}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$
- Second term: $z_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} a_i^{\ell} + b_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}$
    - For $i \neq j$, $\frac{\partial}{\partial z_j^{\ell}} [w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}] = 0$
    - For $i = j$, $\frac{\partial}{\partial z_j^{\ell}} [w_{kj}^{\ell+1} \sigma(z_j^{\ell}) + b_k^{\ell+1}] = w_{kj}^{\ell+1} \sigma'(z_j^{\ell})$

Applying the chain rule

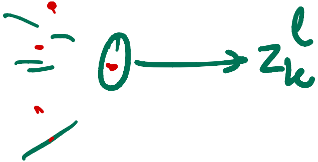
Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \frac{\partial C}{\partial z_j^{\ell}} = \sum_{k=1}^{m} \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$
- First term inside summation: $\frac{\partial C}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$
- Second term: $z_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} a_i^{\ell} + b_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}$
  - For $i \neq j$, $\frac{\partial}{\partial z_j^{\ell}} [w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}] = 0$
  - For $i = j$, $\frac{\partial}{\partial z_j^{\ell}} [w_{kj}^{\ell+1} \sigma(z_j^{\ell}) + b_k^{\ell+1}] = w_{kj}^{\ell+1} \sigma'(z_j^{\ell})$
  - So $\frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}} = w_{kj}^{\ell+1} \sigma'(z_j^{\ell})$

Finishing touches

What we actually need to compute are $\frac{\partial C}{\partial w^\ell_{kj}}$, $\frac{\partial C}{\partial b^\ell_k}$



Finishing touches

What we actually need to compute are $\frac{\partial C}{\partial w_{kj}^{\ell}}, \frac{\partial C}{\partial b_k^{\ell}}$

- $\frac{\partial C}{\partial w_{kj}^{\ell}} = \frac{\partial C}{\partial z_k^{\ell}} \frac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}} = \delta_k^{\ell} \frac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}}$
- $\frac{\partial C}{\partial b_k^{\ell}} = \frac{\partial C}{\partial z_k^{\ell}} \frac{\partial z_k^{\ell}}{\partial b_k^{\ell}} = \delta_k^{\ell} \frac{\partial z_k^{\ell}}{\partial b_k^{\ell}}$

Finishing touches

What we actually need to compute are $\dfrac{\partial C}{\partial w_{kj}^{\ell}}, \dfrac{\partial C}{\partial b_k^{\ell}}$

- $\dfrac{\partial C}{\partial w_{kj}^{\ell}} = \dfrac{\partial C}{\partial z_k^{\ell}} \dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}} = \delta_k^{\ell} \dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}}$
- $\dfrac{\partial C}{\partial b_k^{\ell}} = \dfrac{\partial C}{\partial z_k^{\ell}} \dfrac{\partial z_k^{\ell}}{\partial b_k^{\ell}} = \delta_k^{\ell} \dfrac{\partial z_k^{\ell}}{\partial b_k^{\ell}}$

We have already computed δ_k^{ℓ}, so what remains is $\dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}}, \dfrac{\partial z_k^{\ell}}{\partial b_k^{\ell}}$

Finishing touches

What we actually need to compute are $\dfrac{\partial C}{\partial w_{kj}^{\ell}}, \dfrac{\partial C}{\partial b_k^{\ell}}$

- $\dfrac{\partial C}{\partial w_{kj}^{\ell}} = \dfrac{\partial C}{\partial z_k^{\ell}} \dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}} = \delta_k^{\ell} \dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}}$
- $\dfrac{\partial C}{\partial b_k^{\ell}} = \dfrac{\partial C}{\partial z_k^{\ell}} \dfrac{\partial z_k^{\ell}}{\partial b_k^{\ell}} = \delta_k^{\ell} \dfrac{\partial z_k^{\ell}}{\partial b_k^{\ell}}$

We have already computed δ_k^{ℓ}, so what remains is $\dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}}, \dfrac{\partial z_k^{\ell}}{\partial b_k^{\ell}}$

- Since $z_k^{\ell} = \displaystyle\sum_{i=1}^{m} w_{ki}^{\ell} a_i^{\ell-1} + b_k^{\ell}$, it follows that
  - $\dfrac{\partial z_k^{\ell}}{\partial w_{kj}^{\ell}} = a_j^{\ell-1}$ — terms with $i \neq j$ vanish

Finishing touches

What we actually need to compute are $\dfrac{\partial C}{\partial w^\ell_{kj}}, \dfrac{\partial C}{\partial b^\ell_k}$

- $\dfrac{\partial C}{\partial w^\ell_{kj}} = \dfrac{\partial C}{\partial z^\ell_k}\dfrac{\partial z^\ell_k}{\partial w^\ell_{kj}} = \delta^\ell_k \dfrac{\partial z^\ell_k}{\partial w^\ell_{kj}}$
- $\dfrac{\partial C}{\partial b^\ell_k} = \dfrac{\partial C}{\partial z^\ell_k}\dfrac{\partial z^\ell_k}{\partial b^\ell_k} = \delta^\ell_k \dfrac{\partial z^\ell_k}{\partial b^\ell_k}$

$$\frac{\partial C}{\partial b^\ell_k} = \delta^\ell_k \cdot 1$$

We have already computed δ^ℓ_k, so what remains is $\dfrac{\partial z^\ell_k}{\partial w^\ell_{kj}}, \dfrac{\partial z^\ell_k}{\partial b^\ell_k}$

- Since $z^\ell_k = \sum_{i=1}^{m} w^\ell_{ki} a^{\ell-1}_i + b^\ell_k$, it follows that
  - $\dfrac{\partial z^\ell_k}{\partial w^\ell_{kj}} = a^{\ell-1}_j$ — terms with $i \neq j$ vanish
  - $\dfrac{\partial z^\ell_k}{\partial b^\ell_k} = 1$ — terms with $i \neq j$ vanish

$$\frac{\partial C}{\partial w^\ell_{kj}} = \delta^\ell_k \cdot a^{\ell-1}_j$$

Backpropagation

- In the forward pass, compute all z_k^ℓ, a_k^ℓ
- In the backward pass, compute all δ_k^ℓ, from which we can get all $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$
- Increment each parameter by a step Δ in the direction opposite the gradient

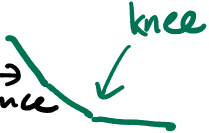
Backpropagation

- In the forward pass, compute all z_k^ℓ, a_k^ℓ
- In the backward pass, compute all δ_k^ℓ, from which we can get all $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$
- Increment each parameter by a step Δ in the direction opposite the gradient

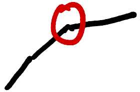
Typically, partition the training data into groups (mini batches)

- Update parameters after each mini batch — stochastic gradient descent
- Epoch — one pass through the entire training data

intra cluster distance →



accuracy



Train for fixed no. of epochs & check

Challenges

- Backpropagation dates from mid-1980's

  Learning representations by back-propagating errors
  David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams
  Nature, **323**, 533–536 (1986)

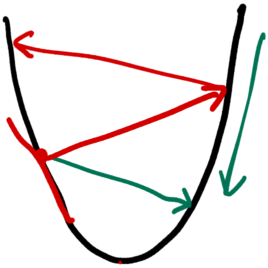
- Computationally infeasible till advent of modern parallel hardware, GPUs for vector (tensor) calculations

- Vanishing gradient problem — cascading derivatives make gradients in initial layers very small, convergence is slow

  - In rare cases, exploding gradient also occurs

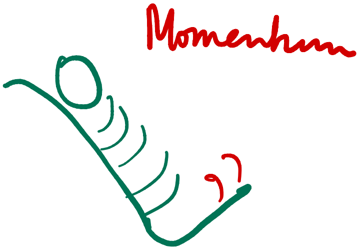
Pragmatics

- Many heuristics to speed up gradient descent
  - Dynamically vary step size
  - Dampen positive-negative oscillations . . .



Pragmatics

- Many heuristics to speed up gradient descent
  - Dynamically vary step size
  - Dampen positive-negative oscillations . . .
- Libraries implementing neural networks have several hyperparameters that can be tuned
  - Network structure: Number of layers, type of activation function
  - Training: Mini-batch size, number of epochs
  - Heuristics: Choice of optimizer for gradient descent



Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

- For gradient descent, we compute $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$

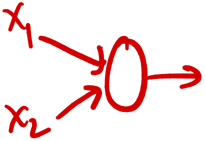
Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

- For gradient descent, we compute $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$
  - For $j = 1, 2$,

$$\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(y_i - a_i) \cdot -\frac{\partial a_i}{\partial w_j}$$



Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

- For gradient descent, we compute $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$
  - For $j = 1, 2$,

$$\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(y_i - a_i) \cdot -\frac{\partial a_i}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\,\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial w_j}$$

Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

- For gradient descent, we compute $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$
  - For $j = 1, 2$,

$$\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(y_i - a_i) \cdot -\frac{\partial a_i}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial w_j}$$

$$= \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)x_j^i$$

Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

- For gradient descent, we compute $\dfrac{\partial C}{\partial w_1}, \dfrac{\partial C}{\partial w_2}, \dfrac{\partial C}{\partial b}$
  - For $j = 1, 2$,

$$\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(y_i - a_i) \cdot -\frac{\partial a_i}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial w_j}$$

$$= \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)x_j^i$$

  - $\dfrac{\partial C}{\partial b} = \dfrac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\dfrac{\partial a_i}{\partial z_i}\dfrac{\partial z_i}{\partial b} = \dfrac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)$

Loss functions . . .

- $\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)x_j^i$, $\frac{\partial C}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)$
- Each term in $\frac{\partial C}{\partial w_1}$, $\frac{\partial C}{\partial w_2}$, $\frac{\partial C}{\partial b}$ is proportional to $\sigma'(z_i)$

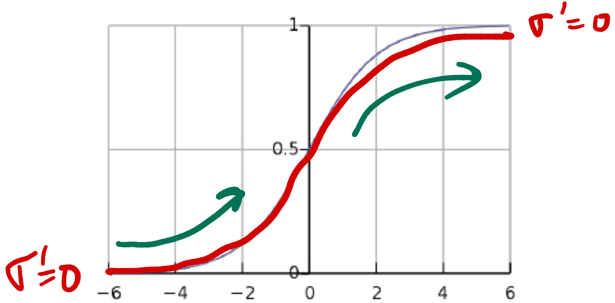
Loss functions . . .

- $\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)x_j^i$, $\frac{\partial C}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)$
- Each term in $\frac{\partial C}{\partial w_1}$, $\frac{\partial C}{\partial w_2}$, $\frac{\partial C}{\partial b}$ is proportional to $\sigma'(z_i)$
- Ideally, gradient descent should take large steps when $a - y$ is large

Loss functions ...

- $\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)x_j^i$, $\frac{\partial C}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)$
- Each term in $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$ is proportional to $\sigma'(z_i)$
- Ideally, gradient descent should take large steps when $a - y$ is large

  - $\sigma(z)$ is flat at both extremes
  - If a is completely wrong, $a \approx (1 - y)$, we still have $\sigma'(z) \approx 0$
  - Learning is slow even when current model is far from optimal



Cross entropy

- A better loss function

$$C(a,y) = \begin{cases} -\ln(a), & \text{if } y = 1 \\ -\ln(1-a), & \text{if } y = 0 \end{cases}$$

  - If $a \approx y$, $C(a,y) \approx 0$ in both cases
  - If $a \approx 1-y$, $C(a,y) \rightarrow \infty$ in both cases

Cross entropy

- A better loss function

$$C(a,y) = \begin{cases} -\ln(a), & \text{if } y = 1 \\ -\ln(1-a), & \text{if } y = 0 \end{cases}$$

  - If $a \approx y$, $C(a,y) \approx 0$ in both cases
  - If $a \approx 1-y$, $C(a,y) \to \infty$ in both cases

- Combine into a single equation

$$C(a,y) = -[y\ln(a) + (1-y)\ln(1-a)]$$

  - $y = 1 \Rightarrow$ second term vanishes, $C = -\ln(a)$
  - $y = 0 \Rightarrow$ first term vanishes, $C = -\ln(1-a)$

Cross entropy

- A better loss function

$$C(a,y) = \begin{cases} -\ln(a), & \text{if } y = 1 \\ -\ln(1-a), & \text{if } y = 0 \end{cases}$$

  - If $a \approx y$, $C(a,y) \approx 0$ in both cases
  - If $a \approx 1-y$, $C(a,y) \to \infty$ in both cases

- Combine into a single equation

$$C(a,y) = -[y \ln(a) + (1-y)\ln(1-a)]$$

  - $y = 1 \Rightarrow$ second term vanishes, $C = -\ln(a)$
  - $y = 0 \Rightarrow$ first term vanishes, $C = -\ln(1-a)$

- This is called cross entropy

Informt: p_i
Coding: q_i

$- \sum p_i \log p_i$

Min # bits for encoding values distrib by p_i

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y) \ln(1-\sigma(z))]$

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$
- $\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_j}$

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$
- $\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_j} = -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial w_j}$

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$

- $$\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_j} = -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial w_j}$$
  $$= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial w_j}$$

σ'

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$

- $$\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_j} = -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial w_j}$$
$$= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial w_j}$$
$$= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\sigma'(z)x_j$$

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$

- $$\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_j} = -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial w_j}$$
$$= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial w_j}$$
$$= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\sigma'(z)x_j$$
$$= -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$$

Cross entropy and gradient descent . . .

- $\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$
- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

Cross entropy and gradient descent . . .

- $\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$
- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$
- Therefore, $\frac{\partial C}{\partial w_j} = -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j$

Cross entropy and gradient descent . . .

- $\dfrac{\partial C}{\partial w_j} = -\left[\dfrac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$

- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

- Therefore, $\dfrac{\partial C}{\partial w_j} = -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j$

  $= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j$

Cross entropy and gradient descent . . .

- $\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$

- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

- Therefore, $\frac{\partial C}{\partial w_j} = -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j$

$$= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j$$

$$= (\sigma(z) - y)x_j$$

Cross entropy and gradient descent . . .

- $\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$

- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

- Therefore, $$\begin{aligned}\frac{\partial C}{\partial w_j} &= -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\ &= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j \\ &= (\sigma(z) - y)x_j \\ &= (a-y)x_j\end{aligned}$$

Cross entropy and gradient descent ...

- $$\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$$
- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$
- Therefore, $$\begin{aligned}\frac{\partial C}{\partial w_j} &= -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\ &= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j \\ &= (\sigma(z) - y)x_j \\ &= (a-y)x_j\end{aligned}$$
- Similarly, $\dfrac{\partial C}{\partial b} = (a-y)$

Cross entropy and gradient descent . . .

- $$\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$$
- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$
- Therefore, $$\begin{aligned}\frac{\partial C}{\partial w_j} &= -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\ &= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j \\ &= (\sigma(z) - y)x_j \\ &= (a-y)x_j\end{aligned}$$
- Similarly, $\dfrac{\partial C}{\partial b} = (a-y)$
- Thus, as we wanted, the gradient is proportional to $a - y$

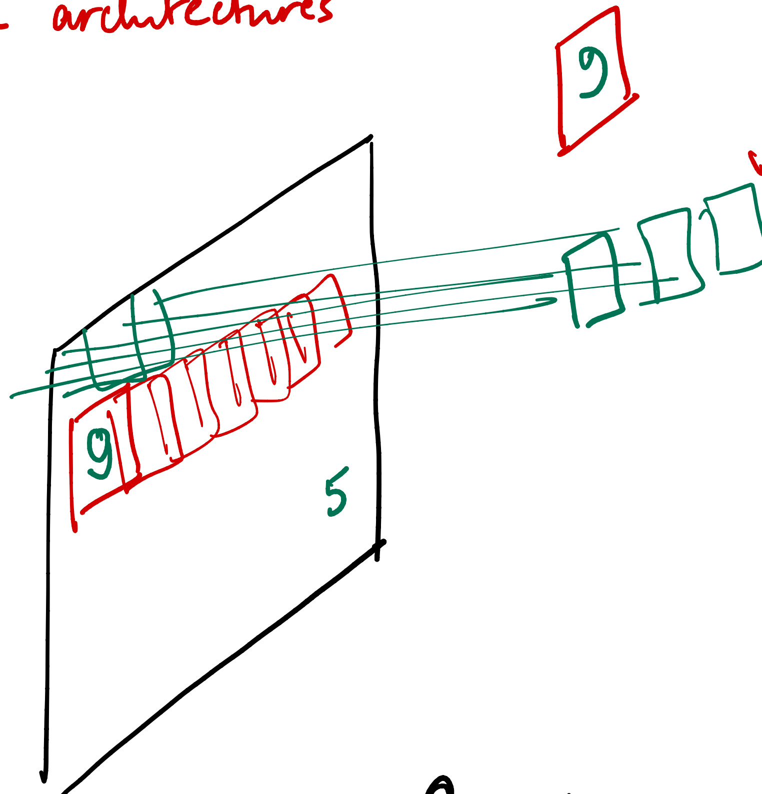
Cross entropy and gradient descent . . .

- $\frac{\partial C}{\partial w_j} = -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$
- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$
- Therefore, $$\begin{aligned}\frac{\partial C}{\partial w_j} &= -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\ &= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j \\ &= (\sigma(z) - y)x_j \\ &= (a-y)x_j\end{aligned}$$
- Similarly, $\frac{\partial C}{\partial b} = (a-y)$
- Thus, as we wanted, the gradient is proportional to $a - y$
- The greater the error, the faster the learning rate

Cross entropy . . .

- Overall,
  - $\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(a_i - y_i)x_j^i$
  - $\frac{\partial C}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(a_i - y_i)$
- Cross entropy allows the network to learn faster when the model is far from the true one
- Other theoretical justifications to justify using cross entropy
  - Derive from goal of maximizing log-likelihood of model

Specialized architectures



many copies, each connected to a different window

Same weights & biases

Shared parameter

Convolution Neural Network - CNN

2012 - AlexNet