Lecture 20: 28 March, 2023

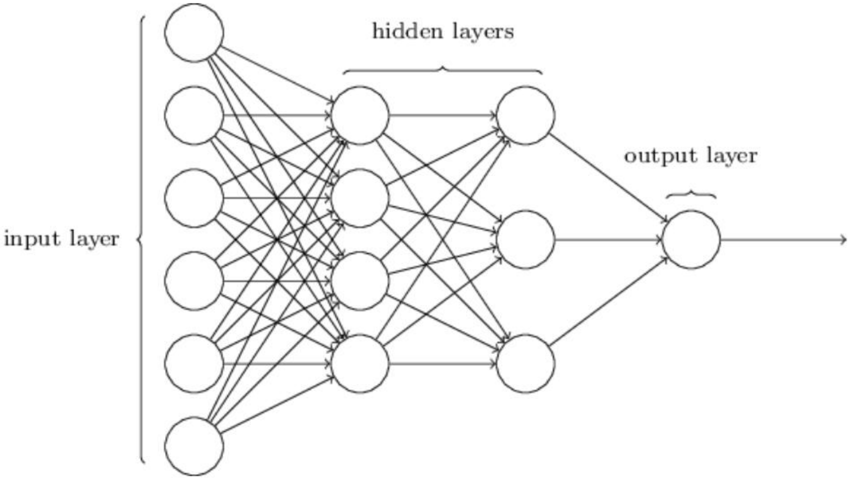
Madhavan Mukund

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

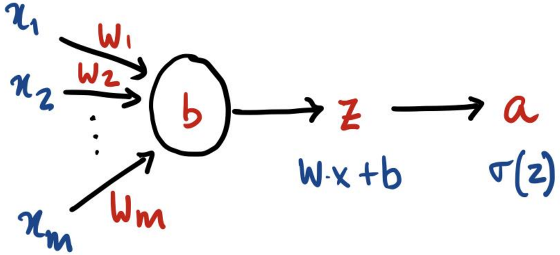
Data Mining and Machine Learning
January–April 2023

Neural networks

- Acyclic network of perceptrons with non-linear activation functions

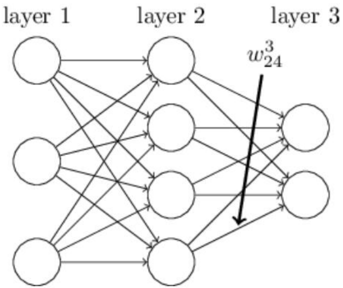


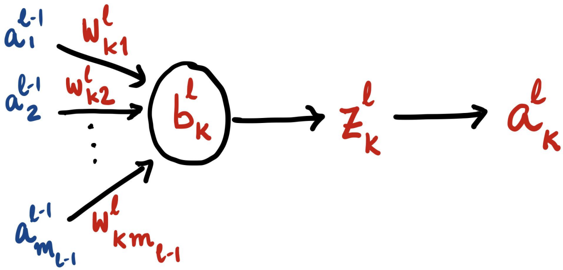
- Without loss of generality,
  - Assume the network is layered
    - All paths from input to output have the same length
  - Each layer is fully connected to the previous one
    - Set weight to 0 if connection is not needed
- Structure of an individual neuron
  - Input weights $w_1, \ldots, w_m$, bias b, output z, activation value a



Notation

- Layers $\ell \in \{1, 2, \dots, L\}$
  - Inputs are connected first hidden layer, layer 1
  - Layer L is the output layer
- Layer ℓ has m_ℓ nodes $1, 2, \dots, m_\ell$
- Node k in layer ℓ has bias b_k^ℓ, output z_k^ℓ and activation value a_k^ℓ
- Weight on edge from node j in level $\ell{-}1$ to node k in level ℓ is w_{kj}^ℓ





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

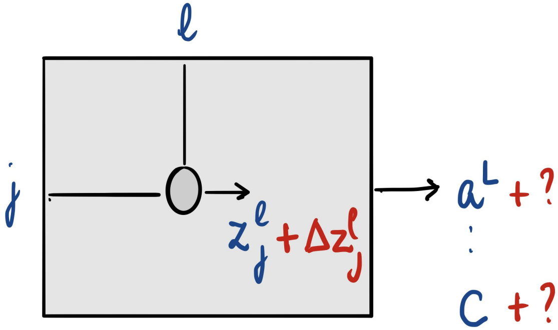
- Need to find optimum values for all weights w^ℓ_{kj}
- Use gradient descent
  - Cost function C, partial derivatives $\dfrac{\partial C}{\partial w^\ell_{kj}}$, $\dfrac{\partial C}{\partial b^\ell_k}$
- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function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

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 $\frac{\partial C}{\partial w^\ell_{kj}}$, $\frac{\partial C}{\partial b^\ell_k}$ in terms of individual $\frac{\partial a^L_i}{\partial w^\ell_{kj}}$, $\frac{\partial a^L_i}{\partial b^\ell_k}$
  - Can extrapolate change in individual cost $C(x)$ to change in overall cost C — stochastic gradient descent
- Complex dependency of C on w^ℓ_{kj}, b^ℓ_k
  - 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

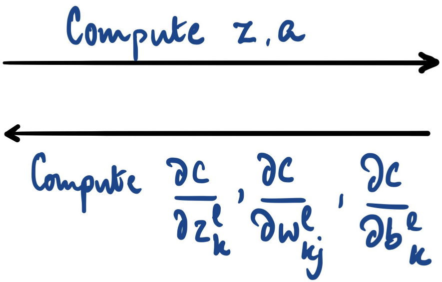
- If we perturb the output z_j^ℓ at node j in layer ℓ, what is the impact on final output, overall cost?



- Focus on $\dfrac{\partial C}{\partial z_j^\ell}$ — from these, we can compute $\dfrac{\partial C}{\partial w_{jk}^\ell}$, $\dfrac{\partial C}{\partial b_j^\ell}$

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 $\frac{\partial C}{\partial z_j^\ell}$



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: $\dfrac{\partial C}{\partial z_j^L} = \dfrac{\partial C}{\partial a_j^L}\dfrac{\partial a_j^L}{\partial z_j^L}$
- For instance, if $C = \dfrac{1}{n}\sum_{i=1}^{n}(y_i - a_i^L)^2$, then $\dfrac{\partial C}{\partial a_j^L} = \dfrac{1}{n}(2(y_j - a_j^L)(-1)) = \dfrac{2}{n}(a_j^L - y_j)$
- $a_j^L = \sigma(z_j^L)$, 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}

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

Finishing touches

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

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

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 $z^\ell_k = \displaystyle\sum_{i=1}^{m} w^\ell_{ki} a^{\ell-1}_i + b^\ell_k$, 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

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

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 ...
- Libraries implementing neural networks have several hyperparameters that can be tuned
  - Network structure: Number of layers, type of activation function — RELU, tanh
  - Training: Mini-batch size, number of epochs
  - Heuristics: Choice of optimizer for gradient descent
- Loss functions
  - As we have seen MSE is not a good choice
  - Cross entropy is better — corresponds to finding MLE