

Lecture 22: 24 June, 2021

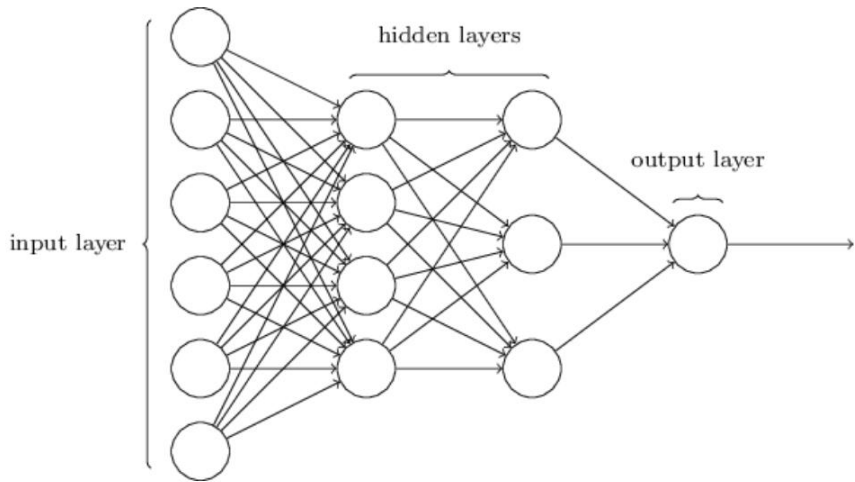
Madhavan Mukund

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

Data Mining and Machine Learning
April–July 2021

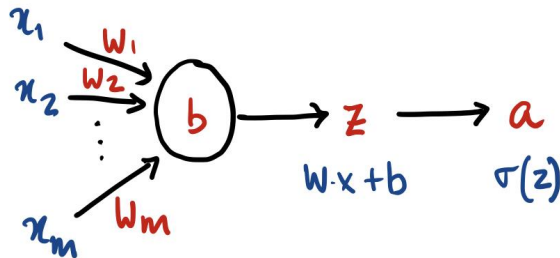
Neural networks

- Acyclic network of perceptrons with non-linear activation functions



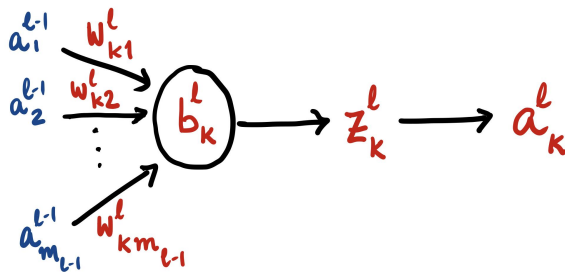
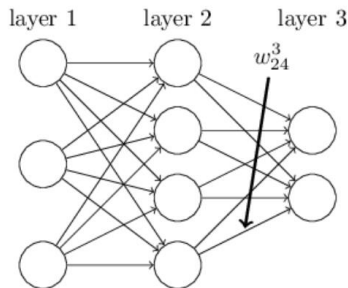
Neural networks

- 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
- Structure of an individual neuron
 - Input weights w_1, \dots, 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



- Why the inversion of indices in the subscript w_{kj}^ℓ ?

- $z_k^\ell = w_{k1}^\ell a_1^{\ell-1} + w_{k2}^\ell a_2^{\ell-1} + \dots + w_{km_{\ell-1}}^\ell a_{m_{\ell-1}}^{\ell-1}$
- Let $\overline{w}_k^\ell = (w_{k1}^\ell, w_{k2}^\ell, \dots, w_{km_{\ell-1}}^\ell)$
and $\overline{a}^{\ell-1} = (a_1^{\ell-1}, a_2^{\ell-1}, \dots, a_{m_{\ell-1}}^{\ell-1})$
- Then $z_k^\ell = \overline{w}_k^\ell \cdot \overline{a}^{\ell-1}$

- Assume all layers have same number of nodes

- Let $m = \max_{\ell \in \{1,2,\dots,L\}} m_\ell$
- For any layer i , for $k > m_i$, we set all of $w_{kj}^\ell, b_k^\ell, z_k^\ell, a_k^\ell$ to 0

- Matrix formulation

$$\begin{bmatrix} \overline{z}_1^\ell \\ \overline{z}_2^\ell \\ \dots \\ \overline{z}_m^\ell \end{bmatrix} = \begin{bmatrix} \overline{w}_1^\ell \\ \overline{w}_2^\ell \\ \dots \\ \overline{w}_m^\ell \end{bmatrix} \begin{bmatrix} a_1^{\ell-1} \\ a_2^{\ell-1} \\ \dots \\ a_m^{\ell-1} \end{bmatrix}$$

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 \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 $\frac{1}{n} \sum_{i=1}^n C(\mathbf{x}_i)$
 - For instance, mean sum-squared error $\frac{1}{n} \sum_{i=1}^n (y_i - a_i^L)^2$

Learning the parameters

- Assumptions about the cost function

- 1 For input \mathbf{x} , $C(\mathbf{x})$ 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 $C(\mathbf{x})$ to change in overall cost C — stochastic gradient descent

- Complex dependency of C on w_{kj}^ℓ , b_k^ℓ

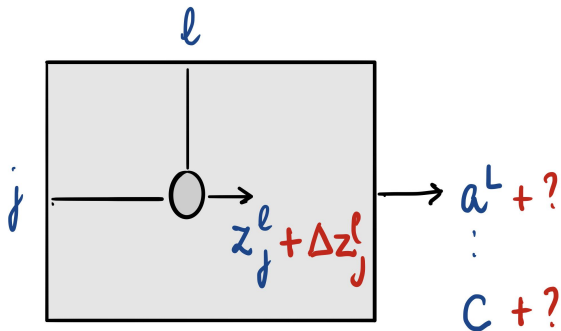
- Many intermediate layers
- Many paths through these layers

- 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

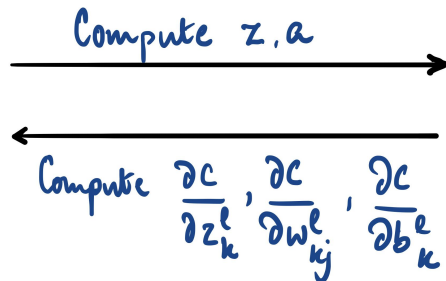
- If we perturb the output z_j^ℓ at node j in layer ℓ , what is the impact on final output, overall cost?



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

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



Applying the chain rule

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

Base Case

$\ell = L, \delta_j^L$

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

Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^ℓ

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

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

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

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

- 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