

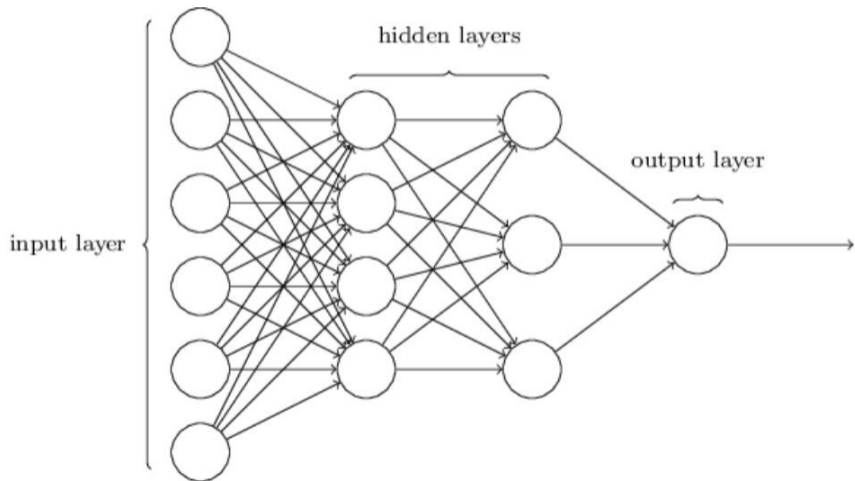
Lecture 20: 11 April, 2022

Madhavan Mukund

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

Data Mining and Machine Learning
January–May 2022

- Acyclic network of perceptrons with non-linear activation functions

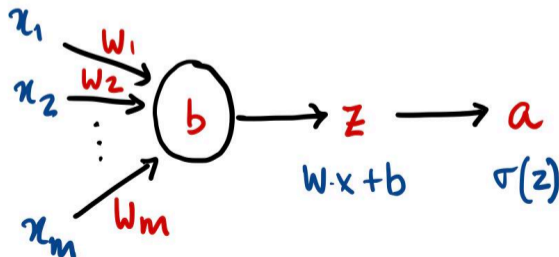


Neural networks

- 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, \dots, w_m , bias b , output z , activation value a

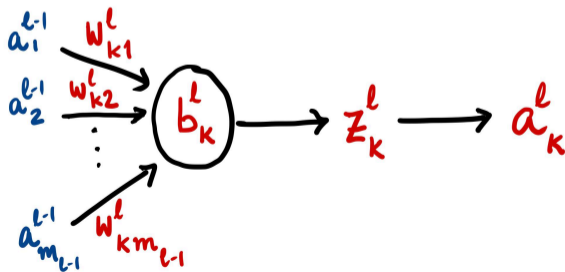
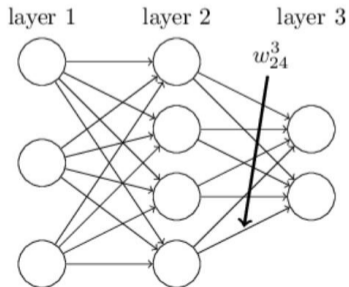


Notation

- 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 $1, 2, \dots, m_\ell$

Notation

- Layers $l \in \{1, 2, \dots, L\}$
 - Inputs are connected first hidden layer, layer 1
 - Layer L is the output layer
- Layer l has m_l nodes $1, 2, \dots, m_l$
- 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 $l-1$ to node k in level l is w_{kj}^l



- Why the inversion of indices in the subscript w_{kj}^l ?
 - $z_k^l = w_{k1}^l a_1^{\ell-1} + w_{k2}^l a_2^{\ell-1} + \dots + w_{km_{\ell-1}}^l a_{m_{\ell-1}}^{\ell-1}$
 - Let $\bar{w}_k^l = (w_{k1}^l, w_{k2}^l, \dots, w_{km_{\ell-1}}^l)$
and $\bar{a}^{\ell-1} = (a_1^{\ell-1}, a_2^{\ell-1}, \dots, a_{m_{\ell-1}}^{\ell-1})$
 - Then $z_k^l = \bar{w}_k^l \cdot \bar{a}^{\ell-1}$

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

- $z_k^l = w_{k1}^l a_1^{l-1} + w_{k2}^l a_2^{l-1} + \dots + w_{km_{l-1}}^l a_{m_{l-1}}^{l-1}$

- Let $\bar{w}_k^l = (w_{k1}^l, w_{k2}^l, \dots, w_{km_{l-1}}^l)$
and $\bar{a}^{l-1} = (a_1^{l-1}, a_2^{l-1}, \dots, a_{m_{l-1}}^{l-1})$

- Then $z_k^l = \bar{w}_k^l \cdot \bar{a}^{l-1}$

- 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 $w_{kj}^l, b_k^l, z_k^l, a_k^l$ to 0

- Matrix formulation

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

Learning the parameters

- Need to find optimum values for all weights w_{kj}^l
- Use gradient descent
 - Cost function C , partial derivatives $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$

Learning the parameters

- Need to find optimum values for all weights w_{kj}^l
- Use gradient descent
 - Cost function C , partial derivatives $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$
- 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 $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$
- 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

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 $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$ in terms of individual $\frac{\partial a_i^l}{\partial w_{kj}^l}$, $\frac{\partial a_i^l}{\partial b_k^l}$
- Can extrapolate change in individual cost $C(x)$ to change in overall cost C — **stochastic gradient descent**

Learning the parameters

- Assumptions about the cost function

- For input \mathbf{x} , $C(\mathbf{x})$ is a function of only the output layer activation, a^L
- Total cost is average of individual input costs

- With these assumptions:

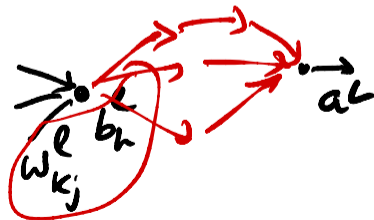
- We can write $\frac{\partial C}{\partial w_{kj}^l}$, $\frac{\partial C}{\partial b_k^l}$ in terms of individual $\frac{\partial a_i^l}{\partial w_{kj}^l}$, $\frac{\partial a_i^l}{\partial b_k^l}$
- Can extrapolate change in individual cost $C(\mathbf{x})$ to change in overall cost C — **stochastic gradient descent**

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

- Many intermediate layers
- Many paths through these layers

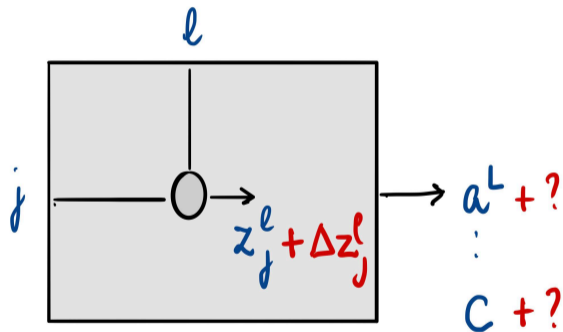
- Use **chain rule** to decompose into local dependencies

- $y = g(f(x)) \Rightarrow \frac{\partial g}{\partial x} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x}$



Calculating dependencies

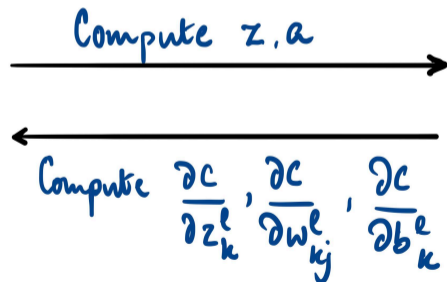
- If we perturb the output z_j^l at node j in layer l , what is the impact on final output, overall cost?



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

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

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

Cost
fn

$\frac{\partial C}{\partial z_1^L}$

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 $\frac{\partial C}{\partial a_j^L} = 2(y_j - a_j^L)(-1) = 2(a_j^L - y_j)$

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 $\frac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L)$

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

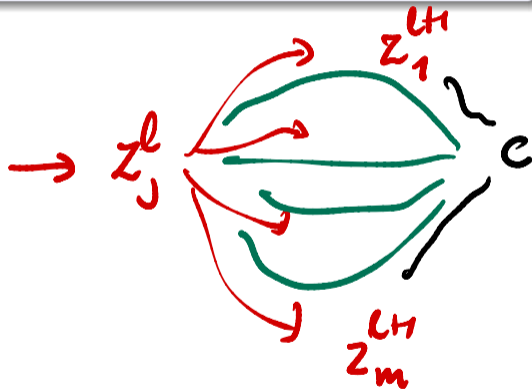
From $\delta_j^{\ell+1}$ to δ_j^ℓ

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

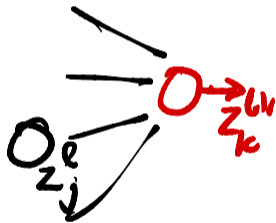
- $\delta_j^\ell = \frac{\partial \mathcal{C}}{\partial z_j^\ell} = \sum_{k=1}^m \frac{\partial \mathcal{C}}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell}$
- First term inside summation: $\frac{\partial \mathcal{C}}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$

Applying the chain rule

Induction step

From δ_j^{l+1} to δ_j^l

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



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 $i \neq j$, $\frac{\partial}{\partial z_j^\ell} [w_{ki}^{\ell+1} \sigma(z_i^\ell) + b_k^{\ell+1}] = 0$

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 $i = j$, $\frac{\partial}{\partial z_j^\ell} [w_{kj}^{\ell+1} \sigma(z_j^\ell) + b_k^{\ell+1}] = w_{kj}^{\ell+1} \sigma'(z_j^\ell)$

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 $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$

Finishing touches

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

$$\blacksquare \frac{\partial C}{\partial w_{kj}^l} = \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial w_{kj}^l} = \delta_k^l \frac{\partial z_k^l}{\partial w_{kj}^l}$$

$$\blacksquare \frac{\partial C}{\partial b_k^l} = \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_k^l} = \delta_k^l \frac{\partial z_k^l}{\partial b_k^l}$$

Finishing touches

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

$$\blacksquare \frac{\partial C}{\partial w_{kj}^l} = \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial w_{kj}^l} = \delta_k^l \frac{\partial z_k^l}{\partial w_{kj}^l}$$

$$\blacksquare \frac{\partial C}{\partial b_k^l} = \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_k^l} = \delta_k^l \frac{\partial z_k^l}{\partial b_k^l}$$

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

Finishing touches

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

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

- $\frac{\partial C}{\partial b_k^l} = \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_k^l} = \delta_k^l \frac{\partial z_k^l}{\partial b_k^l}$

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

- Since $z_k^l = \sum_{i=1}^m w_{ki}^l a_i^{l-1} + b_k^l$, it follows that

- $\frac{\partial z_k^l}{\partial w_{kj}^l} = a_j^{l-1}$ — terms with $i \neq j$ vanish

Finishing touches

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

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

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

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

Handwritten red annotations showing the chain rule for the partial derivatives of the cost function C with respect to weights and biases. The top expression is $\frac{\partial C}{\partial z_j^l}$. An arrow points down to two expressions: $\frac{\partial z_j^l}{\partial w_{jk}^l}$ and $\frac{\partial z_j^l}{\partial b_j^l}$.

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 $\frac{\partial C}{\partial w_{kj}^l}, \frac{\partial C}{\partial b_k^l}$
- 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 $\frac{\partial C}{\partial w_{kj}^l}, \frac{\partial C}{\partial b_k^l}$
- 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

- 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 . . .

- 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

- 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

