06-20416 and 06-12412
(Intro to) Neural Computation

05 – Backpropagation

Per Kristian Lehre

Last lecture

- Functions of multiple variables
- Partial derivatives and the chain rule
- Gradients
  - Direction of steepest ascent
- Gradient descent

This lecture

- Feedforward Neural Networks
  - Model in ML consisting of multiple layers of units
  - Each unit a non-linear transformation of weighted inputs
  - Model parameters are weights and biases
- Backpropagation algorithm (start)
  - How to compute the gradient of the cost of a feedforward neural network wrt its parameters

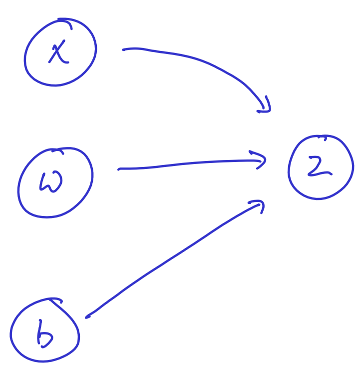
Computation Graphs

We will describe machine learning models using computation graphs where

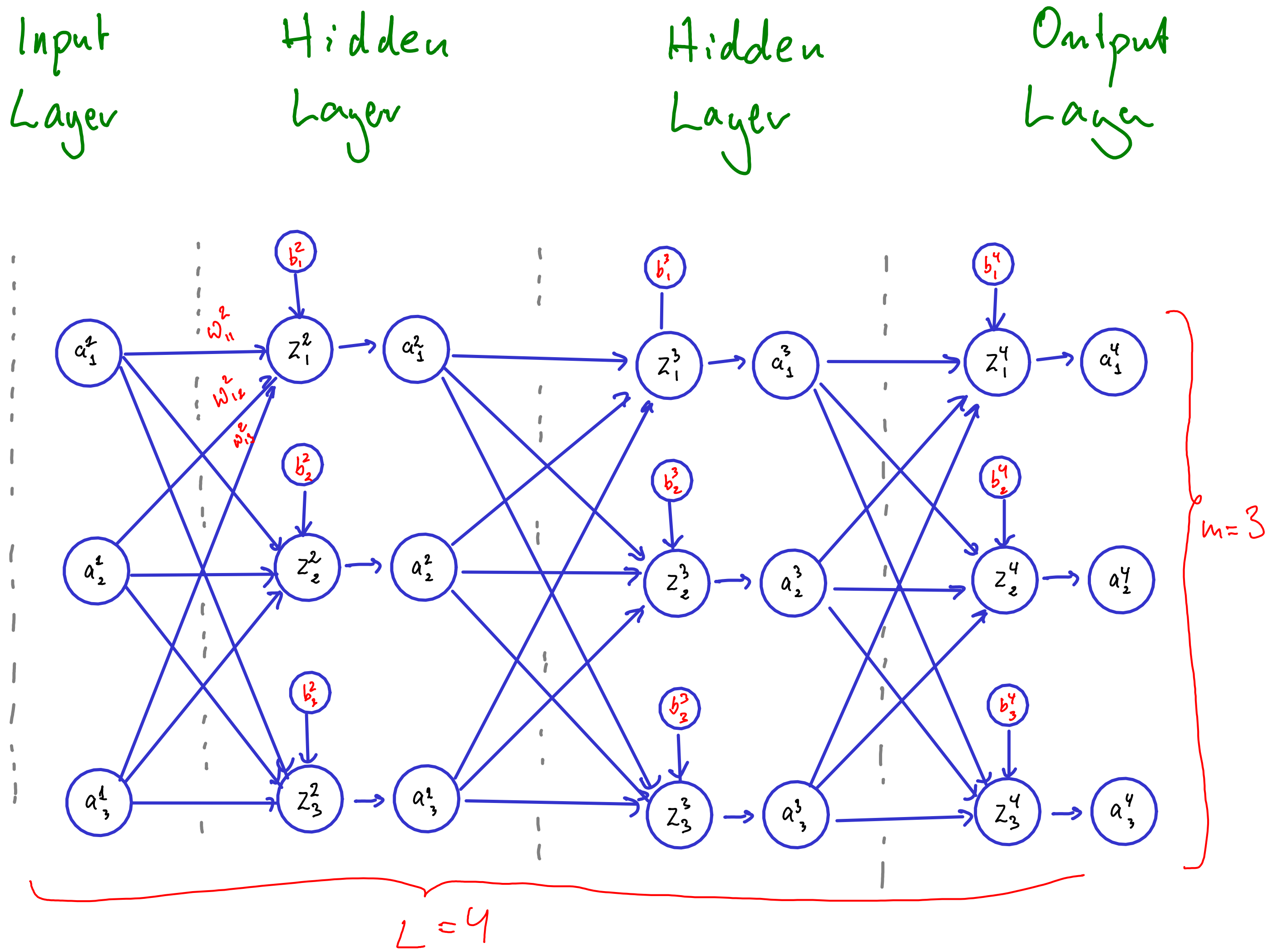
- nodes repuesent variables (values, vectors, matrices),
- edges represent functional dependencies, i.e., an edge from x to y indicates that y is a function of x.

Example

A linear regression model $z = \sum_{i=1}^{m} a_i w_i + b$



Feed Forward Neural Network



L number of layers in the network, where layer 1 is "input layer", and layer L is "output layer"

m "width" of network (can vary between layers)

w_{jk}^{ℓ} "weight" of connection between k-th unit in layer $\ell-1$, to j-th unit in layer ℓ

b_j^{ℓ} "bias" of j-th unit in layer ℓ

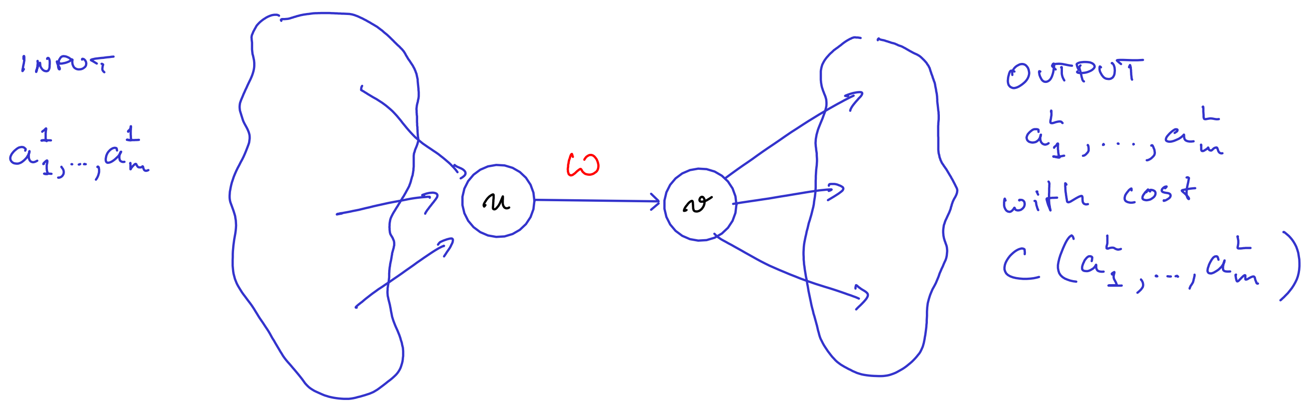
$$z_j^{\ell} = \sum_k w_{jk}^{\ell} a_k^{\ell-1} + b_j^{\ell}$$ weighted input to unit j in layer ℓ

$$a_j^{\ell} = \phi\left(z_j^{\ell}\right)$$ "activation" of unit j in layer ℓ, where ϕ is an "activation function"

Training of Feedforward Neural Networks

The parameters of the network are

- the weights w^{ℓ}_{jk} in each layer
- the biases b^{ℓ}_{j}

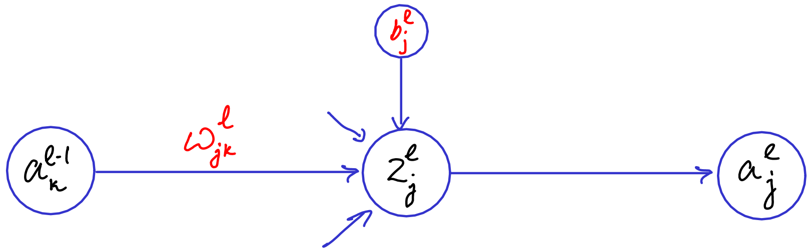


General Idea

To apply gradient descent to optimise a weight w (or bias b) in a network, we apply the chain rule

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial v} \cdot \frac{\partial v}{\partial w}$$

Idea applied to Feed Forward Neural Networks



$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l$$

$$a_j^l = \phi(z_j^l)$$

"Activation" of unit k in layer $l-1$.

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot a_k^{l-1}$$

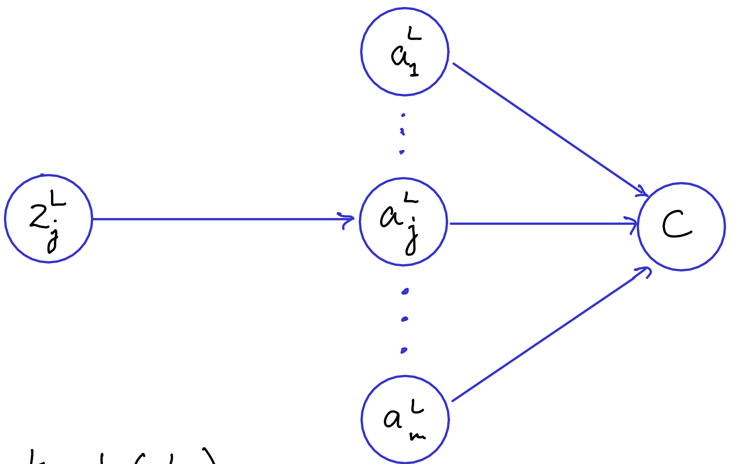
$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot 1$$

Hence, we can compute $\frac{\partial C}{\partial w_{kj}^l}$ and $\frac{\partial C}{\partial b_j^l}$ if we know

$$\delta_j^l := \frac{\partial C}{\partial z_j^l}$$

The vector δ^l is called the local gradient for layer l.

Local Gradient for Output Layer



$$a_j^L = \phi(z_j^L)$$

The local gradient for the output layer is

$$\partial_j^L = \frac{\partial C}{\partial z_j^L} \qquad \text{by definition}$$

$$= \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \qquad \text{by the chain rule}$$

$$= \frac{\partial C}{\partial a_j^L} \cdot \phi'(z_j^L) \qquad \text{because } a_j^L = \phi(z_j^L)$$

The partial derivative $\frac{\partial C}{\partial a_j^L}$ depends on the cost function.

For example, for a regression problem in m dimensions, one could define

$$C(a_1^L, \dots, a_m^L) := \frac{1}{2} \sum_{k=1}^{m} (y_k - a_k^L)^2,$$

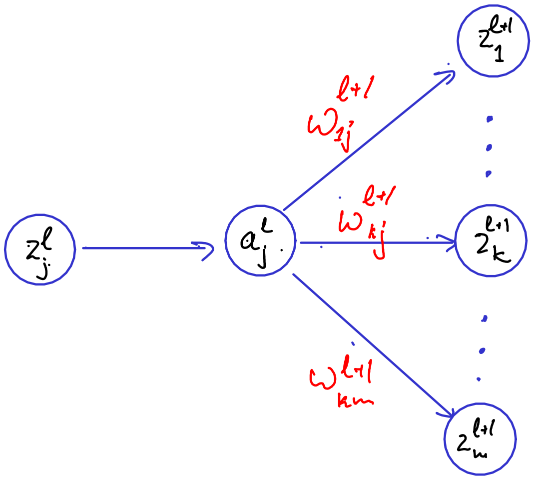
desired output in k-th dimension

predicted output in k-th dimension

in which case

$$\frac{\partial C}{\partial a_j^L} = a_j^L - y_j$$

Local Gradient for Hidden Layers



$$z_k^{l+1} = \sum_r w_{kr}^{l+1} a_r^l$$

$$a_j^l = \phi(z_j^l)$$

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad \text{by def of local gradient } \delta_j^l$$

$$= \frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \qquad \text{by chain rule}$$

$$= \left(\sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial a_j^l} \right) \cdot \phi'(z_j^l) \qquad \text{by chain rule wrt } \frac{\partial C}{\partial a_j^l}$$

$$= \phi'(z_j^l) \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \qquad \text{by def. of local gradient } \delta_k^{l+1}$$

Summary

For all weights and biases

$$\frac{\partial C}{\partial w_{jk}^{l}} = \delta_j^{l} \cdot a_k^{l-1}$$

$$\frac{\partial C}{\partial b_j^{l}} = \delta_j^{l}$$

where the local gradient δ_j^l is

$$\delta_j^{l} = \begin{cases} \phi'(z_j^L) \cdot \dfrac{\partial C}{\partial a_j^L} & \text{if } l = L \text{ (output layer)} \\[2ex] \phi'(z_j^l) \cdot \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} & \text{otherwise (hidden layer)} \end{cases}$$

Matrix Description

The backpropagation algorithm can exploit efficient implementations of matrix operations, e.g. in numerical libraries such as NumPy or on a GPU.

Recall that for a matrix $A \in \mathbb{R}^m \times \mathbb{R}^n$, A_{ij} denotes the element in the i-th row and j-th column.

For two matrices $A \in \mathbb{R}^m \times \mathbb{R}^n$ and $B \in \mathbb{R}^n \times \mathbb{R}^\ell$

$$(A^T)_{ij} = A_{ji}$$ matrix transpose

$$(AB)_{ij} = \sum_k A_{ik} B_{kj}$$ matrix multiplication

For two vectors $u, v \in \mathbb{R}^m$

$$u + v = (u_1 + v_1, \dots, u_m + v_m)$$ vector addition

$$u \cdot v = \sum_{i=1}^m u_i v_i$$ dot product

$$u \odot v = (u_1 v_1, \dots, u_m v_m)$$ Hadamard product

$$(uv^T)_{ij} = u_i v_j$$ outer product

Weighted Inputs and Activations

$$z^l = \left(z_1^l, \dots, z_m^l \right)$$

$$= \left(\sum_{k=1}^{m} w_{1k}^l a_k^{l-1} + b_1^l, \dots, \sum_{k=1}^{m} w_{mk}^l a_k^{l-1} + b_m^l \right)$$

$$= w^l a^{l-1} + b$$

$$a^l = \left(a_1^l, \dots, a_m^l \right)$$

$$= \left(\phi(z_1^l), \dots, \phi(z_m^l) \right)$$

$$= \phi(z^l)$$

Local Gradients

Output Layer

$$\delta^L = \left(\delta_1^L, \dots, \delta_m^L\right)$$

$$= \left(\frac{\partial C}{\partial a_1^L} \cdot \phi'(z_1^L), \dots, \frac{\partial C}{\partial a_m^L} \cdot \phi'(z_m^L)\right)$$

$$= \nabla_{a^L} C \odot \phi'(z^L)$$

Hidden Layer

$$\delta^\ell = \left(\delta_1^\ell, \dots, \delta_m^\ell\right)$$

$$= \left(\phi'(z_1^\ell) \cdot \sum_k \delta_k^{\ell+1} \cdot w_{k1}^{\ell+1}, \dots, \phi'(z_m^\ell) \cdot \sum_k \delta_k^{\ell+1} \cdot w_{km}^{\ell+1}\right)$$

$$= \phi'(z^\ell) \odot \left(\sum_k \left(w^{\ell+1}\right)^T_{1k} \delta_k^{\ell+1}, \dots, \sum_k \left(w^{\ell+1}\right)^T_{mk} \delta_k^{\ell+1}\right)$$

$$= \phi'(z^\ell) \odot \left(w^{\ell+1}\right)^T \delta^{\ell+1}$$

Backpropagation Algorithm

Input: A training example $(x, y) \in \mathbb{R}^m \times \mathbb{R}^{m'}$

1. Set the activation in the input layer

$$a^1 = x$$

2. for each $l = 2$ to L, feed forward

$$z^l = w^l a^{l-1} + b^l$$

$$a^l = \phi(z^l)$$

3. compute local gradient for output layer

$$\delta^L := \nabla_{a^L} C \odot \phi'(z^L)$$

4. backpropagate local gradients for hidden layers, i.e.

for each $l = L-1$ to 2

$$\delta^l := \left((w^{l+1})^T \delta^{l+1} \right) \odot \phi'(z^l)$$

5. <u>return</u> the partial derivatives

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

Training Feed Forward Networks

Assume n training examples

$$\left(x^{(1)}, y^{(1)}\right), \dots, \left(x^{(n)}, y^{(n)}\right),$$

and a cost function

$$C = \frac{1}{n} \sum_{i=1}^{n} C^{(i)},$$

where $C^{(i)}$ is the cost on the i-th example.

For example, with MSE, we can define

$$C^{(i)} = \frac{1}{2}\left(y^{(i)} - a^{L}\right)$$

where a^L is the output of the network when $a^1 = x^{(i)}$

Backpropagation gives us the gradient of the overall cost function as follows

$$\frac{\partial C}{\partial w^{\ell}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C^{(i)}}{\partial w^{\ell}}$$

$$\frac{\partial C}{\partial b^{\ell}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C^{(i)}}{\partial b^{\ell}}$$

NB "Average" gradient per training example.

In principle, we can now use gradient descent to optimise the weights w and biases b.

Mini-batch Gradient Descent

Computing the gradients is expensive when the number of training examples n is large.

We can approximate the gradients

$$\frac{\partial C}{\partial w^l} \approx \frac{1}{b} \sum_{i=1}^{b} \frac{\partial C^{(i)}}{\partial w^l}$$

$$\frac{\partial C}{\partial b^l} \approx \frac{1}{b} \sum_{i=1}^{b} \frac{\partial C^{(i)}}{\partial b^l}$$

using a random "mini-batch" of $b \leq n$ training examples.

$1 < b < n \implies$ Mini-batch Gradient Descent

$b = 1 \implies$ Stochastic Gradient Descent

$b = n \implies$ Batch Gradient Descent

Common to use mini-batch size $b \in (20, 100)$.

Summary

- Computation graphs
- Feedforward Neural Networks
  - Input nodes and biases
  - Activation functions
  - Input, hidden, and output layers
- Backpropagation algorithm (start)
  - The gradient can be computed using local derivatives.
  - Local derivatives are computed backwards, starting from the output layer

Next lecture

- More about backpropagation