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

06 – Softmax

Per Kristian Lehre

Last week

- Implementation of a fully connected feedforward neural network
  - backpropagation algorithm
  - mini-batch gradient descent
  - Training on Fashion MNIST classification problem using mean squared error (MSE) as cost function

Outline

- Redesign network to make it more appropriate as a probabilistic model for classification
- Replace the output layer with a “softmax” layer
- Define a new cost function based on maximum likelihood
- Compute local gradients for the softmax layer

Earlier, we obtained the per-example cost function

$$C^{(i)} = \sum_{j=1}^{m} \frac{1}{2} \left(y_j^{(i)} - a_j^L \right)^2$$

model output

using the maximum likelihood method under the assumption that the predicted output a_j^L has a Gaussian distribution. This is an acceptable assumption for <u>regression</u> problems.

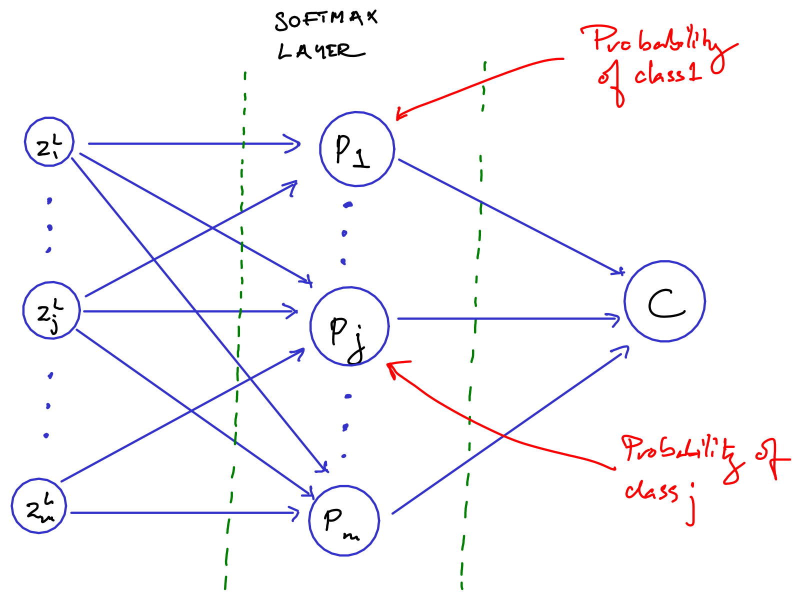
However, for <u>classification</u> problems with m discrete class labels $1, \dots, m$, more appropriate to have one output unit p_j per class j, where p_j is interpreted as the probability of class j.
These units should therefore satisfy

$$p_j \geq 0 \quad \text{for all } j \in \{1, \dots, m\}, \text{ and}$$

$$\sum_{j=1}^{m} p_j = 1$$

We replace the last layer by a "softmax" layer

$$p_j := \frac{e^{z_j^L}}{Q} \quad \text{where} \quad Q := \sum_{k=1}^{m} e^{z_k^L}$$



Note that since $p_k \geq 0$ and $\sum_{k=1}^{m} p_k = 1$,
we can interpret the output of the network
as a probabilistic model where

$$p_y = P_{wb}(y \mid x),$$

i.e., the probability of class y given input x.

Given n independent observations $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$,
where $y^{(i)} \in \{1, \dots, m\}$ is the class corresponding to input $x^{(i)}$,
the likelihood of weight and bias parameters w, b is

$$\mathcal{L}\left(w, b \mid (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\right) = \prod_{i=1}^{n} P_{wb}\left(y^{(i)} \mid x^{(i)}\right) = \prod_{i=1}^{n} p_{y^{(i)}}$$

We can define a cost function using maximum likelihood principle

$$C = -\log \mathcal{L}\left(w, b \mid (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right)$$

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

where

$$C^{(i)} := -\log P_{wb}\left(y^{(i)} \mid x^{(i)}\right)$$

$$= -\log p_{y^{(i)}}$$

$$= \log Q - z^{L}_{y^{(i)}}$$

To apply gradient descent to minimise the cost function, we need to compute the gradients, i.e.,

$$\frac{\partial C}{\partial w^{l}_{jk}} = \sum_{i=1}^{n} \frac{\partial C^{(i)}}{\partial w^{l}_{jk}} \quad \text{and}$$

$$\frac{\partial C}{\partial b^{l}_{j}} = \sum_{i=1}^{n} \frac{\partial C^{(i)}}{\partial b^{l}_{j}} .$$

To compute these with backpropagation, we need to compute the local gradient for the softmax layer

$$\delta^{L}_{j} := \frac{\partial C^{(i)}}{\partial z_j} .$$

Theorem

$$\delta_j^L = \frac{\partial C^{(i)}}{\partial z_j^L} = p_j - \delta_{y^{(i)}j} \quad \text{where} \quad \delta_{ab} := \begin{cases} 1 & \text{if } a=b \\ 0 & \text{otherwise} \end{cases}$$

Kronecker-delta function

Proof

$$\frac{\partial C^{(i)}}{\partial z_j^L} = -\frac{\partial \log p(y^{(i)} \mid x^{(i)})}{\partial z_j}$$

$$= -\frac{\partial \log p_{y^{(i)}}}{\partial z_j}$$

$$\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$$

$$p_{y^{(i)}} = \frac{e^{a_{y^{(i)}}}}{Q}$$

$$= -\frac{\partial}{\partial z_j}\left(z_{y^{(i)}} - \log Q\right)$$

$$= -\left(\delta_{y^{(i)}j} - \frac{\partial \log Q}{\partial z_j}\right)$$

$$= -\left(\delta_{y^{(i)}j} - \frac{\partial \log Q}{\partial Q} \cdot \frac{\partial Q}{\partial z_j}\right)$$

$$= -\left(\delta_{y^{(i)}j} - \frac{1}{Q} e^{z_j}\right)$$

$$= p_j - \delta_{y^{(i)}j}$$

$\square$

Numerical Issues with Softmax

Neural networks are usually implemented with fixed length representations of real numbers. Clearly, such representations can only represent a finite number of real values.

E.g., the maximal value that the float64 data type in NumPy can represent is approximately $1.8 \cdot 10^{308}$.
When computing the numerator in $\frac{e^{z_j^L}}{Q}$, we can easily exceed this value.

Note that for any constant r

$$P_j = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} = \frac{e^r \cdot e^{z_j^L}}{e^r \cdot \sum_k e^{z_k^L}} = \frac{e^{z_j^L + r}}{\sum_k e^{z_k^L + r}}$$

To avoid too large exponents, it is common to implement the softmax function as the rightmost expression above with the constant

$$r := -\max_k z_k^L$$

Summary

- A softmax output layer allows output nodes to be interpreted as probabilities
- The probabilities indicate the likelihood of a class, given the input and the network
- A naive implementation of the softmax function can be numerically unstable

Next lecture

- Improvements of gradient descent