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

$$C^{(i)} = \frac{1}{2} \left(\gamma_{i}^{(i)} - \alpha_{i}^{i} \right)$$

$$j=1 \qquad \text{model on put}$$

using the maximum likelihood method under the assumption that the pudicted output at has a Gaussin distribution. This is an acceptable assumption for regussion problems.

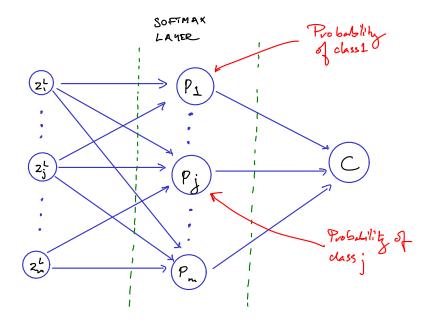
However, for <u>classification</u> problems with m discute class labels 1,..., m, more appropriate to have one ontput unit p; per class j, where p; is interpreted as the probability of class j. These units should there fore satisfy

$$p_{j} = 1$$

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

De replace the last laye by a "softmax" layer

$$P_{j} = \frac{e^{2j}}{Q} \qquad \text{where} \qquad Q := \sum_{k=1}^{m} e^{2k}$$



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

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

De can define a cost function using maximum likelihood principle

$$C = - |o_{0}| \mathcal{L} \left(\omega, b \mid \left(x^{(2)}, y^{(2)} \right), \dots, \left(x^{(n)}, y^{(n)} \right) \right)$$

$$= \frac{1}{m} \sum_{i=1}^{7} C^{(i)}$$

when

$$C^{(i)} := -\log P_{Wb} \left(n_{g}^{(i)} \mid \chi^{(i)} \right)$$

$$= -\log P_{m_{g}^{(i)}}$$

$$= \log Q - \lambda_{m_{g}^{(i)}}$$

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

$$\frac{\partial C}{\partial \omega_{jk}} = \frac{1}{\sqrt{2}} \frac{\partial C^{(n)}}{\partial \omega_{jk}} \quad \text{and} \quad \frac{\partial C}{\partial \omega_{jk}}$$

$$\frac{\partial C}{\partial \omega_{jk}} = \frac{1}{\sqrt{2}} \frac{\partial C^{(n)}}{\partial \omega_{jk}} \quad \frac{\partial C}{\partial \omega_{jk}} = \frac{1}{\sqrt{2}} \frac{\partial C^{(n)}}{\partial \omega_{jk}} = \frac{1}{\sqrt{2}} \frac{\partial$$

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

$$\int_{i}^{L} \frac{\partial C^{(i)}}{\partial x^{i} \dot{y}}.$$

$$S_{j}^{L} = \frac{\partial C^{(i)}}{\partial z_{j}^{L}} = P_{j} - S_{y}^{(i)} j \quad \text{where } S_{ab}^{i} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

Proof

Kvonecker della function

$$\frac{\partial C^{(i)}}{\partial z_{j}^{i}} = -\frac{\partial \log P(x^{(i)} | x^{(i)})}{\partial z_{j}}$$

$$= -\frac{\partial \log P(x^{(i)})}{\partial z_{j}^{i}}$$

$$= -\frac{\partial}{\partial z_{j}^{i}} \left(2 \operatorname{sgn} - \log Q \right)$$

$$= -\left(6 \operatorname{sgn} - \frac{\partial \log Q}{\partial z_{j}^{i}} \right)$$

$$= -\left(6 \operatorname{sgn} - \frac{\partial \log Q}{\partial Q} \cdot \frac{\partial Q}{\partial z_{j}^{i}} \right)$$

$$= -\left(6 \operatorname{sgn} - \frac{1}{Q} e^{2i} \right)$$

$$= -\frac{\partial}{\partial y_{j}^{i}} - \frac{1}{Q} e^{2i}$$

Numerical losnes with Softman

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 rathe that the float69 data type in Numby can represent 15 approximately 1.8.308.

When computing the numerator in 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 expussion above with the constant

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