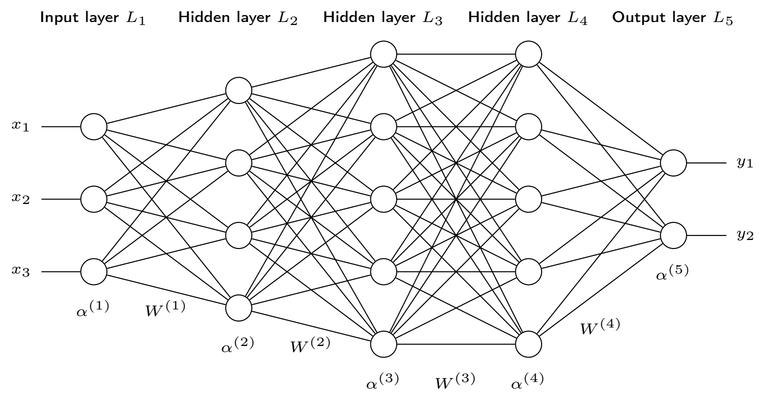
Statistical Image Analysis

Lecture 11: Neural nets

David Bolin
University of Gothenburg

Gothenburg
May 14, 2018

Example for binary classification



Feedforward neural nets

Input data $x_1, \ldots, x_p$. Output probabilities for M classes. Model

$$z_l^{(k)} = w_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} w_{lj}^{(k-1)} \alpha_j^{(k-1)} := W^{(k-1)} \alpha^{(k-1)}$$

$$\alpha^{(k)} = g^{(k)}(z^{(k)})$$

for $k = 1, \ldots, K$, where $p_1 = p$, $\alpha^{(1)} = x$, and α^K defines the probabilities.

- Where $W^{(1)}, \ldots, W^{(k)}$ are weights.
- $g^{(1)}, \ldots, g^{(K)}$ are non-linear functions.

Common non-linear functions:

- Rectified linear: $g(v) = \max(0, v)$.
- Softmax $g(v_i, v) = \frac{\exp(v_i)}{\sum_j \exp(v_j)}$.

Parameter estimation

- The neural network defines a nonlinear function $f(x, W)$ of the input variables x, depending on the unknown weights $W = \{W^{(1)}, W^{(2)}, \ldots, W^{(K)}\}$.
- To estimate W, for some input data $\{x_i, y_i\}_{i=1}^n$, we define a loss-function $L(y, f(x, W))$ as well as a regularization factor $J(W)$ and estimate

$$\hat{W} = \arg\min_W \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i, W)) + \lambda J(W)$$

Simple examples of L and J are

- Squared loss: $L(y, f(x, W)) = \frac{1}{2}\|y - f(x, W)\|^2$.
- Weight-decay penalty: $J(W) = \sum_{j,l,k} (w_{l,j}^{(k)})^2$.

- We estimate W using gradient-descent.

Backpropagation

The gradient of L can be estimated using the chain rule.

1. Compute $\alpha_l^{(k)}$ for each layer k and each node l based on the current estimate of W.
2. For the output layer, compute

$$\delta_l^{(K)} = \frac{\partial L}{\partial z_l^{(K)}} = \frac{\partial L}{\partial \alpha_l^{(K)}}\frac{\partial \alpha_l^{(K)}}{\partial z_l^{(K)}} = \frac{\partial L}{\partial \alpha_l^{(K)}}\dot{g}^{(K)}(K)(z_l^{(K)})$$

3. For $k = K-1, \dots, 2$, compute

$$\delta_l^{(k)} = \left(\sum_{j=1}^{p_{k+1}} w_{lj}^{(k)}\delta_j^{(k+1)}\right)\dot{g}^k(K)(z_l^{(k)})$$

4. Compute $\frac{\partial L}{\partial w_{lj}^{(k)}} = \alpha_j^{(k)}\delta_l^{(k+1)}$

Stochastic gradient descent

To speed up the estimation, it is common to replace the exact gradient by a stochastic estimate:

- Option 1: Define $G(W) = \frac{1}{ns}\sum_{i=1}^n J_i \frac{\partial L}{\partial W^{(k)}}$, where J_i are independent $\mathsf{Be}(s)$ random variables. Then

$$\mathsf{E}(G(W)) = \frac{1}{ns}\sum_{i=1}^n \mathsf{E}(J_i)\frac{\partial L}{\partial W^{(k)}} = \frac{1}{n}\sum_{i=1}^n \frac{\partial L}{\partial W^{(k)}}$$

- Option 2: Divide the training data into k batches and randomly sample one of the batches in each iteration.

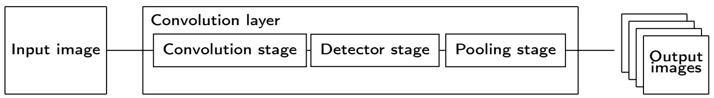
Verification for the output layer

Assume squared-loss:

$$\begin{aligned}
\frac{\partial L}{\partial w_{lj}^{(K-1)}} &= \frac{\partial}{\partial w_{lj}^{(K-1)}}\frac{1}{2}\sum_{j=1}^M (y_j - \alpha_j)^2 \\
&= (y_l - \alpha_l)\frac{\partial \alpha_l}{\partial w_{lj}^{(K-1)}} \\
&= (y_l - \alpha_l)\frac{\partial g^{(K)}(z_l^{(K)})}{\partial w_{lj}^{(K-1)}} \\
&= (y_l - \alpha_l)\frac{\partial g^{(K)}(z_l^{(K)})}{\partial z_l^{(K)}}\frac{\partial z_l^{(K)}}{\partial w_{lj}^{(K-1)}} \\
&= \underbrace{(y_l - \alpha_l)\dot{g}^k(K)(z_l^{(k)})}_{\delta_l^{(K)}}\alpha_l^{(K-1)}
\end{aligned}$$

Convolutional Neural networks

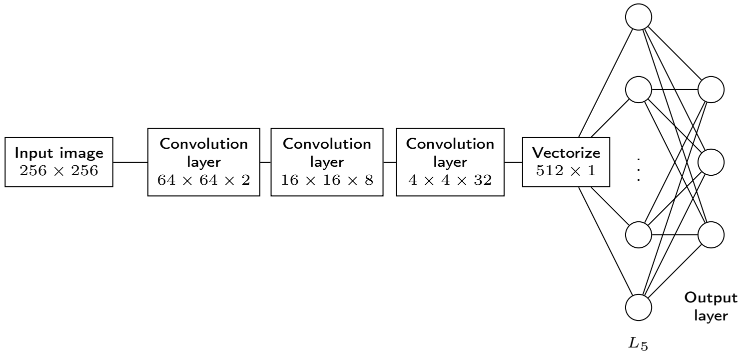
- A CNN assumes that the input data has a lattice structure, like an image.
- Consists of a special type of layers called convolution layers, which has three stages:
  1. Convolution stage: Convolve each input image with p different linear filters, with kernels of size $q \times q$, producing p output images.
  2. Detector stage: Apply a non-linear function to each image. Typically the rectified linear function $g(v) = \max(0, v)$.
  3. Pooling stage: For each image, reduce each non-overlapping block of $r \times r$ pixels to one single value, by for example taking the largest value in the block.



Comments

- Using a CNN, we do not need to specify features manually.
- One could view the convolution stage as a regular layer where most of the weights are zero: A pixel in the output image only depends on the $q \times q$ nearest pixels in the input image.
- The different nodes share parameters, since we use the same convolution kernel across the entire image.
- As a result, a convolution layer has pq^2 parameters, which is much less than a corresponding fully connected layer with $(mn)^2$ parameters.
- Since pooling reduces the image size, we can in the next stage use more filters without increasing the total number of nodes.
- Pooling makes the output less sensitive to small translations of the input.
- Another variant of pooling is to take the max across different learned features. This can make the output invariant to other things, such as rotations.

Example of a CNN



- The first layer has $p = 2$ filters, the second has $p = 4$, the third has $p = 4$.
- Each pooling stage uses $r = 4$.
- The final hidden layer is a usual fully connected layer.