Lecture 11: Neural networks

Spatial Statistics and Image Analysis

David Bolin
University of Gothenburg

Gothenburg
May 13, 2019

Neural nets

- A problem with the methods for image classification from last time is the need for feature selection.
- Neural networks is a class of methods that can be used to design classifiers without the need to select features.
- Let us start with the binary classification problem: We have an image $\mathbf{x}$ with pixels $x_1, \ldots, x_p$, which can belong to one of two classes.
- Model:

$$y_1 = \mathsf{P}(z = 0|\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}),$$
$$y_2 = \mathsf{P}(z = 1|\mathbf{x}) = 1 - f(\mathbf{x}; \boldsymbol{\theta})$$

  for some non-linear function of the pixel values.
- Likelihood for training the model from M images:

$$\ell(\boldsymbol{\theta}) = \prod_{i=1}^{M} f(\mathbf{x}_i; \boldsymbol{\theta})^{z_i} (1 - f(\mathbf{x}_i; \boldsymbol{\theta})^{z_i})^{1-z_i}$$

A single-layer neural net

- The idea of neural nets is to approximate $f(\mathbf{x})$ as a sequence of "'simple" non-linear functions.
- Let's look at a single-layer model first.
- Start by forming p_1 different linear combinations of the data:

$$\mathbf{W}_1^{(1)} \cdot \mathbf{x}, \mathbf{W}_2^{(1)} \cdot \mathbf{x}, \ldots, \mathbf{W}_{p_1}^{(1)} \cdot \mathbf{x}$$

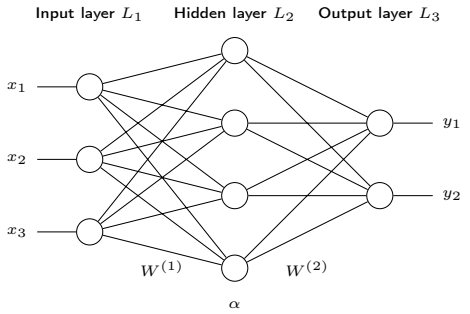
  where $\mathbf{W}_k^{(1)}$ are weights and $\mathbf{W}_k^{(1)} \cdot \mathbf{x} = w_{k0}^{(1)} + \sum_{i=1}^{p} w_{ki}^{(1)} x_i$.
- To each linear combination, apply a non-linear function $g^{(1)}$:

$$\alpha_1 = g^{(1)}(\mathbf{W}_1^{(1)} \cdot \mathbf{x}), \alpha_2 = g^{(1)}(\mathbf{W}_2^{(1)} \cdot \mathbf{x}), \ldots, \alpha_{p_1} = g^{(1)}(\mathbf{W}_{p_1}^{(1)} \cdot \mathbf{x})$$

- Finally approximate y_1 and y_2 as a transformed linear combinations of these values:

$$y_1 = g_1^{(2)}(\mathbf{W}_1^{(2)} \cdot \boldsymbol{\alpha}, \mathbf{W}_2^{(2)} \cdot \boldsymbol{\alpha}), y_2 = g_2^{(2)}(\mathbf{W}_1^{(2)} \cdot \boldsymbol{\alpha}, \mathbf{W}_2^{(2)} \cdot \boldsymbol{\alpha})$$

- To make sure that y_1, y_2 are probabilities, we take $g^{(2)}(x)$ as the softmax function: $g_k^{(2)}(x_1, x_2) = \frac{e^{x_k}}{\sum_{l=1}^{2} e^{x_l}}$.
- We can represent this model as a network:



- This is a feed-forward network since information only flows forward in the network.
- The nodes in the hidden layer are called neurons.
- The functions $g^{(1)}$ and $g^{(2)}$ are called activation functions.

A single-layer neural net

- Our model for $y_1 = f(\mathbf{x})$ is thus

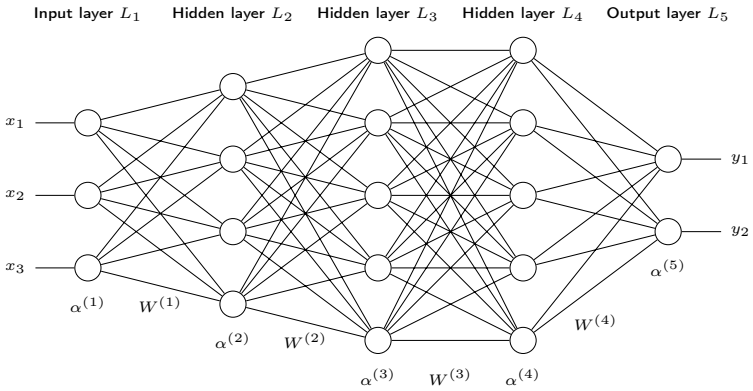
$$f(\mathbf{x}) = \frac{e^{z_1}}{e^{z_1} + e^{z_2}},\ z_k = w_{k0}^{(2)} + \sum_{i=1}^{p} w_{ki}^{(2)} g^{(1)} \left(w_{i0}^{(1)} + \sum_{j=1}^{p} w_{ij}^{(1)} x_i \right)$$

  where all the weights w should be estimated to give a good fit.
- The main idea of neural networks is that we should be able to approximate any function $f(\mathbf{x})$ in this way:

The universal approximation theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.

Example for binary classification



General feed-forward neural nets for classification

- Input data $x_1, \ldots, x_p$. Output: probabilities for K classes.
- In total $L-1$ hidden layers in the model.
- We can allow for a non-linear transformation of the input data in the Input layer, giving $\alpha_k^{(1)} = g^{(0)}(x_k)$.
- Usually we set $g^{(0)}$ to the identity function but keep the notation $\alpha_k^{(1)} = x_k$ to simplify the formulas.
- At layer k in the model, define linear combination of the neurons in previous layers, and new neuron values:

$$z_l^{(l)} = w_{k0}^{(l-1)} + \sum_{j=1}^{p_{l-1}} w_{kj}^{(l-1)} \alpha_j^{(l-1)} := W^{(l-1)} \alpha^{(l-1)}$$
$$\alpha^{(l)} = g^{(l)}(z^{(l)})$$

  for $l = 1, \ldots, L$, where $p_1 = p$, and $\alpha^{(1)} = x$.

Comments

- The output probabilities are given by $\alpha^{(L)}$.
- Common activation functions for the internal layers:
  - Rectified linear: $g(v) = \max(0, v)$. Sometimes called Rectified Linear Unit (RELU).
  - Sigmoid function: $g(v) = \frac{1}{1+e^{-v}}$. Sometimes called a radial basis function (RBF network).
  - $g(v) = \tanh(v)$.
- Common activation function for the output layer for classification:
  - Softmax $g_i(v_1, \ldots, v_K) = \frac{\exp(v_i)}{\sum_{k=1}^{K} \exp(v_k)}$.
  - Symmetric version of the logit link used for logistic regression.
- The neural network is nothing else than a hierarchically specified non-linear regression. Compare with logistic regression.

Parameter estimation

- The neural network defines a nonlinear function $f(x, W)$ of the input variables x, depending on the unknown weights $W = \{W^{(1)}, W^{(2)}, \ldots, W^{(L)}\}$.
- To estimate W, for some input data $\{\mathbf{x}_i, y_i\}_{i=1}^M$, we can define a loss-function $R(y, f(\mathbf{x}, W))$ and compute

$$\hat{W} = \arg\min_W \sum_{i=1}^M R(y_i, f(\mathbf{x}_i, W))$$

Simple examples of L:

- For regression: Squared loss $R(y, f(\mathbf{x}, W)) = \frac{1}{2}\|y - f(\mathbf{x}, W)\|^2$.
- For classification: Cross-entropy loss $R(y, f(\mathbf{x}, W)) = \sum_{k=1}^K 1(y = k) \log f_k(\mathbf{x}, W)$.

- Estimate W using gradient-descent.

Backpropagation

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

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

$$\delta_k^{(L)} = \frac{\partial R}{\partial z_k^{(L)}} = \frac{\partial R}{\partial \alpha_k^{(L)}} \frac{\partial \alpha_k^{(L)}}{\partial z_k^{(L)}} = \frac{\partial R}{\partial \alpha_k^{(L)}} \dot{g}^{(L)}(z_k^{(L)})$$

3. For $l = L - 1, \ldots, 2$, compute

$$\delta_k^{(l)} = \left(\sum_{j=1}^{p_{l+1}} w_{kj}^{(l)} \delta_j^{(l+1)} \right) \dot{g}^{(l)}(z_k^{(l)})$$

4. Compute $\frac{\partial R}{\partial w_{kj}^{(l)}} = \alpha_j^{(l)} \delta_k^{(l+1)}$

Regularization

- Neural networks in general have too many parameters and will overfit the data.
- An early solution to this problem was to stop the gradient-based estimation before convergence.
- A validation dataset can be used to determine when to stop.
- A more explicit method for regularization is to include a penalty on the weights in the loss-function:
- The neural network defines a nonlinear function $f(\mathbf{x}, W)$ of the input variables $\mathbf{x}$, depending on the unknown weights

$$\hat{W} = \arg\min_W \sum_{i=1}^M R(y_i, f(\mathbf{x}_i, W)) + \lambda J(W)$$

- A common example is the weight-decay penalty: $J(W) = \sum_{j,l,k} (w_{k,j}^{(l)})^2$ which will pull the weights to zero.
- λ is a tuning parameter: estimate using cross-validation.

Gradient descent

- Update $w_{kj}^{(l)}$ using a gradient descent step. Assuming weight-decay penalty:

$$w_{kj}^{(l)} \leftarrow w_{kj}^{(l)} - \gamma \left(\frac{\partial R}{\partial w_{kj}^{(l)}} + \lambda w_{kj}^{(l)} \right)$$

- We need a lot of data to estimate these models, and for large datasets, the computation of $\frac{\partial R}{\partial w_{kj}^{(l)}}$ is expensive: For M training images with p pixels, and a network with N hidden units, $\mathcal{O}(pMN)$ operations are needed.

Stochastic gradient descent

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

- Option 1: Define $G(W) = \frac{1}{s}\sum_{i=1}^{M} J_i \frac{\partial R}{\partial W^{(l)}}$, where J_i are independent $\mathrm{Be}(s)$ random variables. Thus, we are randomly selecting (on average) $100s\%$ of the images in each iteration. Then

$$\mathrm{E}(G(W)) = \frac{1}{s}\sum_{i=1}^{M} \mathrm{E}(J_i)\frac{\partial R}{\partial W^{(l)}} = \sum_{i=1}^{M} \frac{\partial R}{\partial W^{(l)}}$$

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

There are several other tricks to speed up convergence, such as momentum updates.

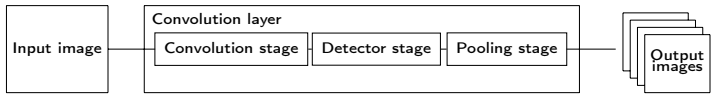
Convolutional Neural networks

- Often, a fully connected network simply has too many parameters: For the first single-layer network for binary classification, we have $pp_1 + 2p_1$ unknown weights. $p = p_1 = 1000$ thus gives 1002000 unknown parameters!
- The problem is that we have a separate weight between each pixel and each hidden node.
- The idea of Convolutional neural networks is to reduce the number of parameters by assuming that most of the weights are zero, and that the non-zero weights have a common structure.
- 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 are based on filtering the image with a kernel.

Convolution layers

A convolution layer has three stages:

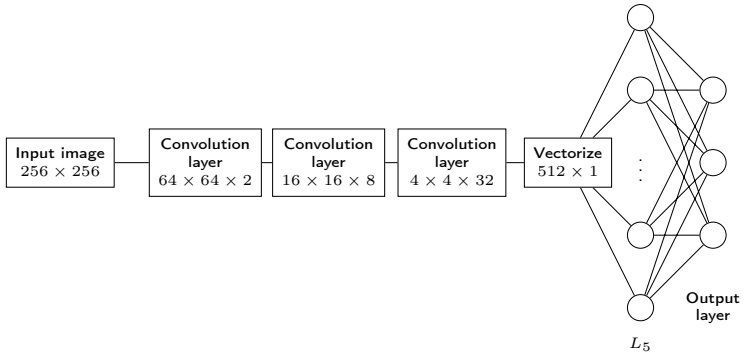
1. Convolution stage: Convolve each input image with f different linear filters, with kernels of size $q \times q$, producing f output images.
2. Detector stage: Apply a non-linear function to each image. Typically the rectified linear function $g(v) = \max(0, v)$.
3. Pooling stage: For each image, reduce each non-overlapping block of $r \times r$ pixels to one single value, by for example taking the largest value in the block.



Comments

- 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 fq^2 parameters, which is much less than a corresponding fully connected layer with $(pp_l)^2$ 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 $f = 2$ filters, the second has $f = 4$, the third has $f = 4$.
- Each pooling stage uses $r = 4$.
- The final hidden layer is a usual fully connected layer.

Comments

- A CNN is a method for image classification using filtered images as features, but where we do not need to specify features manually.
- Using CNNs for image classification re-popularized neural networks around 2010, and "Deep learning" was coined as a flashy name for using "deep" neural networks with more than one hidden layer.
- For further details on neural networks, for example see:
  1. Computer age statistical inference by Efron and Hastie
  2. `deeplearningbook.org`
  3. Matlab guides: Create simple deep learning network for classification