CASI day 5 lecture notes: Neural networks

Tuomo Nieminen

update on: 2019-05-16

1 Introduction

The goal of machine learning is to find a function f to characterize a predictive relationship between the data x and some target y, i.e. to find an f for which

$$f(x) = y.$$

A simple example is linear regression, where

$$f(x) = w_0 + \sum_{j=1,..,p} x_j w_j.$$

A neural network is a highly parametrized model which in theory can learn any smooth predictive relationship* $f : x \to y$.

*See universal approximation theory

2 Structure of a neural network

2.1 Layers

A neural network consists of K layers (L_k), which can be categorized to

- An input layer (L_1)
- Hidden layers ($L_2, .., L_{K-1}$)
- An output layer (L_K)

See fig. 18.1. in Computer age statistical inference

2.2 Units and activations

The layers of a neural network consist of *units*, which have *activation functions.* The outputs of the activation functions are fed to the next layer. The unit l in layer k takes as input the outputs from the previous layer, combines them via a linear function, and then applies a nonlinear transformation g_l to produce outputs $a_l^{(k)}$.

So, for example the output of the units in the second layer are nonlinear transormations of linear combinations of the input vector:

$$z_l^{(2)} = w_{l0}^{(1)} + \sum_{j=1,..,p} w_{lj}^{(1)} x_j$$

$$a_l^{(2)} = g_l^{(2)}(z_l^{(2)}).$$

2.3 Transitions between layers

In general, the transition $L_{k-1} \to L_k$ is given by

$$z^{(k)} = W^{(k-1)} a^{(k-1)}$$
$$a^{(k)} = g^{(k)}(z^{(k)})$$

where W represents the matrix of weights from layer L_{k-1} to layer L_k, $a^{(k)}$ is the entire vector of activations at layer L_k, and $g^{(k)}$ operates elementwise.

2.4 Activation functions

Typical choices for the nonlinear part of the activation functions are

- Rectified linear unit (ReLU): $g(x) = max(0, x)$
- Sigmoid: $g(x) = 1/(1 + e^{-x})$

Without the nonlinearities, the neural network reduces to a linear model.

2.5 Example 1: Simple linear model as a neural network

Say we have the following model

$$y = w_0 + w_1 x_1 + w_2 x_2.$$

Let us present it as a neural network.

Input layer
The input layer has 2 units for inputs x_1 and x_2.

Hidden layers
This network has no hidden layers at all

Output layer
The units in the output layer take as input the outputs of the previous layer, which in this case is the input layer (with two units). Let's call these inputs x_1 and x_2. The output layer has a single node with the following activation:

$$z = w_0 + w_1 x_1 + w_2 x_2$$
$$g = I(z)$$
$$a = g(z),$$

which is the simple linear model.

2.6 Example 2: A neural network for classification

Say we want to predict one of 10 classes, using the data x as input, i.e. we want

$$f(x) = \{0, 1, .., 9\}$$

It is generally more useful to learn the probability function

$$P(y = j|x), \quad j = 0, 1, ..9$$

We will define a neural network which produces probabilities for each of the labels. In this example the output layer of the network has 10 units, one for each of the possible class labels. The final nonlinear transformation $g^{(K)}$ in the **output layer** is the *softmax function*

$$g^{(K)}(z_m^{(K)}; z^{(K)}) = \frac{exp(z_m^{(K)})}{\sum_{l=1,..,M} exp(z_l^{(K)})},$$

which computes a number (probability) between zero and one, and all M of them sum to one. This is a symmetric version of the inverse link function used for multiclass logistic regression: a generalization of the sigmoid.

3 Fitting a neural network

3.1 Train and test data

A machine learning algorithm is usually trained by first splitting the data into train and test sets. The training data is used to fit the model and the test data is used to evaluate the model performance.

3.2 Loss function

A neural network model is a hierarchical function $f(x; W)$ of the feature vector x, and the collection of weights W. A loss function is a function defined on a single data point x_i, a prediction $\hat{y}_i = f(x_i; W)$ and target y_i, and it measures the penalty caused by an error in the prediction. For example the square loss is given by

$$L(f(x_i; W), y_i) = (f(x_i; W) - y_i)^2 = (\hat{y}_i - y_i)^2.$$

The *categorical cross-entropy* is often used as the loss function for classification problems. It is equivalent to the multinomial log likelihood. For a 10-class model it is given by

$$L(f(x_i; W), y_i) = \sum_{j=0,..,9} y_{ij} \cdot log(\hat{p}_{ij}).$$

The negative log likelihood and categorical cross-entropy can be said to be two different interpretations of the same formula.

refs: wikipedia, stackechange

3.3 Cost function

A cost function defines the optimization problem to determine the optimal set of parameters in the model. A typical choice is the average loss over the training set plus some model complexity penalty (regularization):

$$C(W) = \frac{1}{n} \sum_{i=1,..,n} L(f(x_i; W), y_i) + \lambda J(W),$$

where λ is a tuning parameter for the penalty function $J(W)$. For example, using the square loss and no penalty ($\lambda = 0$), the cost becomes

$$\frac{1}{n} \sum_{i=1,..,n} (\hat{y}_i - y_i)^2,$$

also known as Ordinary Least Squares (OLS). A cost function can be derived via maximum likelihood estimation (MLE), possibly adding a penalty term. An MLE is an example of an *objective function*, a general expression for the function to optimize (maximize or minimize).

ref: stats.stackexchange.com

3.4 Gradient Descent

To fit the model, the objective is to minimize the cost function with respect to the parameters (weights) W. There is no analytical solution, so we resort to numerical methods. An algorithm called Gradient Descent can be used to find the W which minimizes the cost function. Given the gradient $\Delta C(W)$ of the cost function, the gradient descent update on the weights is given by

$$W^{(k)} \leftarrow W^{(k)} - \alpha \Delta C(W^{(k)}), \quad k = 1, ..., K,$$

where α is the learning rate. Gradient descent requires (nonzero, typically random) starting values for all the weights W.

3.5 Backpropagation to find the gradient of the cost function

Since $f(x; W)$ is defined hierarchically, the elements of W occur in layers, starting from the input layer. Computing the gradient is also done most naturally in layers. The backpropagation algorithm implements the chain rule for differentiation to find the gradient of the objective function.

The goal is to compute the derivative of $L(f(x; W), y)$ with respect to any of the elements of W, for a generic input–output pair (x, y). Since the objective function is a sum of loss functions, the overall gradient will be the sum of these individual gradient elements over the training pairs (x_i, y_i)

$$\Delta C(W) = \sum_{i=1,..,n} \Delta L(f(x_i; W), y_i) + \Delta \lambda J(W)$$

3.6 Stochastic gradient descent

Rather than process all the observations before making a gradient step, it can be more efficient to process smaller *batches* of data at a time, or even a single data point i. The gradient computed for a batch approximates the true gradient.

$$E[\Delta C_i(W)] = \Delta C(W)$$

Using sub samples to compute the gradient is called stochastic gradient descent. Several passes can be made over the training set until the algorithm converges. Data is usually shuffled in between to avoid stucking to poor local minima (hence the term *stochastic*?). An *epoch* of training means that all n training samples have been used in gradient steps.

4 Types of networks

Convolutional neural networks https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Recurrent neural networks

Autoencoders