Deep learning for image analysis

Linus Falk

March 23, 2023

Contents

1 A linear classifier

This lecture will describe the linear classifier and ways to train it.

Recap deep learning: "end to end"

Working with deep learning can reduce the number of steps that are involved in image analysis comparing it to traditional approaches that are more human driven (engineering and prior knowledge based). The advantages with the "end to end" approach in deep learning is that we remove the prior knowledge needed for the specific case, that also removes any bias when starting to solve the problem also, since there is no pre-formed idea of how the result should look like. This is in the case were the data set can be consider non biased and non skewed.

The disadvantage of this is that it is typically very data hungry and needs large data sets in order to train well. The deep learning method is data driven and suffers if the data is not satisfying. Traditional image analysis methods c.

Problem formulation

Given a image we have an array of X $\times$ Y $\times$ 3 values representing the pixels in the image (3 colors). Images poses many challenges since the image is a 2D representation of a 3D object in a environment that can change.

- **viewpoint variations** change of camera angle
- **Illumination variations**, environment
- **Deformations**, the object may deform/
- **Occlusions**
- **Background clutter**
- **Intraclass variations**, many different looking cats...

So in contrast to sorting numbers, there is no simple solution to solve this problem with hard-coding.

```
def predict(image):
        # no clue....
        return class_label
```

Data-driven approach

One way to solve it is to collect a lot of images and corresponding labels. Use a machine learning method to train a classifier and then evaluate the result on a test set of images.

The task is to design a classifier: f(x,**W**) that can tell us which class $y_i \in \{1, 2, ..., N\}$ an input image x_i belongs to.

1. Select a classifier type,
   - a linear affine classifier for example: y = Wx+b
2. select a performance measure, loss function
   - Hinge loss
   - Negative Log-likelihood
3. Learning: find the parameters $\mathbf{W} = \{W, b\}$ which maximizes the performance, minimizes the overall loss

Multiclass SVM loss

If given an example image with label (x_i,y_i), where x_i is the image and y_i is the label. The shorthand for the score vector: s = $f(x_i, W)$. Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1) \tag{1}$$

The full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i \tag{2}$$

Softmax classifier (Multinomal logistic regression)

The scores are unnormalized log probabilities of the classes: $s = f(x_i, W)$.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \tag{3}$$

We want to maximize the log likelihood (or minimize the loss function, the negative log likelihood)

$$L_i = -\log P(Y = y_i|X = x_i) \tag{4}$$

and in summary we have:

$$L_i = -\log(\frac{e^{s_k}}{\sum_j e^{s_j}}) \tag{5}$$

Learning

How do we minimize the loss over the training data then:

- arg min loss(training data)
- Follow the slope, gradient descent

With the second alternative we want to follow the slope like a blind person finding its way down from the hills. In the 1D case this is done with the derivative, but in the multidimensional case it's done with gradients (partial derivatives). **Gradient descent** can be used to minimize the loss L by:

1. Initiliaze the weights $\mathbf{W}_0$
2. Compute the gradient with respect to W, ∇L/($\mathbf{W}_k; x$) = ($\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, ...$)
3. Take a small step in the negative direction of the gradient: $\mathbf{W}_{k+1} = \mathbf{W}_k -$ stepsize $\cdot \nabla L$
4. Iterate from (2) until convergence

Linear regression and classification