Facial Keypoints Detection Using Convolutional Neutral Network

Qinyao He
CST34
Tsinghua University
2012010548

Xiaocheng Yang
CST34
Tsinghua University
2013XXXXXX

Abstract

In this report we building an neural network for carrying out facial keypoints detection problem. We have tried several ways, but a problem of non convergence still need to be resolved, which till now we have no idea about that.

1. Introduction

This task comes originally from Kaggle[1], a online website holding many data science competitions. It's aim is to determine the location of several certain key points on face, given a photograph contains a human face.

This key points detection can form the building block for many applications. Such as:

* tracking faces in images and video
* analysing facial expressions
* detecting dysmorphic facial signs for medical diagnosis
* biometrics / face recognition

Detecing facial keypoints is a very challenging problem. Facial features vary greatly from one individual to another, and even for a single individual, there is a large amount of variation due to 3D pose, size, position, viewing angle, and illumination conditions. Computer vision research has come a long way in addressing these difficulties, but there remain many opportunities for improvement.

In this specific competition[2], we are required to localize 15 facial key points given a images. Each predicted point is specified as (x, y) real-valued pair in the space of pixel indices. The 15 keypoints is:

left_eye_center, right_eye_center, left_eye_inner_corner, left_eye_outer_corner, right_eye_inner_corner, right_eye_outer_corner, left_eyebrow_inner_end, left_eyebrow_outer_end, right_eyebrow_inner_end, right_eyebrow_outer_end, nose_tip, mouth_left_corner, mouth_right_corner, mouth_center_top_lip, mouth_center_bottom_lip

so 30 real value is going to be predicted for each input image.

Kaggle given a training data set of 7049 image with size of $96 * 96$, color is gray scale and represented as an integer from 0 to 255 for each pixel. And each training images is labeled with 30 real value for position of the 15 key points. In some examples, some of the target keypoint positions are misssing.

Also, a test set of 1783 images without label is given, for competitors to submit their predicted result to Kaggle for score.

Performance is graded by MSE(mean square error) for those 30 real values. A lower MSE indicate a better prediction.

In our work, we have carried out experiment on several models, include basic multilayer perceptron, convolutional neural network and cascaded convolutional network.

2. Related Work

3. Technical Approach

In this section we demonstrate some technical approach which, based on previous work, can improve performance of the network, accelerate convergence, and reduce overfitting.

3.1. Data Normalization

In the original training data, each pixel is represented by an integer from 0 to 255, and the real-value label range from 0 to 96. This large range of input and target data may force the neuron to saturate, which slow down convergence. Also, the non-error-centered target value make the full connected layer hard to get a ideal output.

In our implementation, gray scale from 0 to 255 are divided by 256, rescaled to range 0 to 1, and target value are rescaled and zero centered to range from -1 to 1. Let train_x,

[1] https://www.kaggle.com/

[2] https://www.kaggle.com/c/facial-keypoints-detection

train_y stand for the image data and label, the normalize preprocess looks like:

$$train\_x = \frac{train\_x}{256.0}$$
$$train\_y = \frac{train\_y - 48.0}{48.0}$$

3.2. ReLU Nonlinearity

When training deep feedforward neural network, gradient diminishing when back propagation often occurs and make it very slow to converge. This issue is common when using sigmoid-like activation function, since they have a so-called saturation regime where gradient almost vanished. According to experiment by Glorot [2], for a feedforward neural net with 4 layers and activate with sigmoid function, the last hidden layer quickly saturate with output value of 0, which slow down the progress of learning. and after a hundred of epochs, it escaped, and the followed layer began to saturate.

Glorot hypothesize that this behavior is the result of combination of random initialization and the fact that an output of zero correspond to a saturated sigmoid. Unsupervised pre-training can successfully solve this issue.

While with another kind of activation function, known as ReLU(rectify linear unit), the problem of saturation and gradient diminishing is also solved. ReLU activation act as follow:

$$rectify(x) = max(x, 0)$$

It can achieve the performance as better as a pre-trained sigmoid network[3] without any unsupervised pre-training. It behavior is also more plausible at the viewpoint of biological neural science, which claim a neuron to be closed below zero, and behave linearly when input is above zero with in some range, and this range is satisfied most of time.

Also rectifier activation function allow a network to easily obtain sparse representations, which is biologically plausible and have mathematical advantages.

The non-linearity when using rectify comes from path-selection. For different input, different set of neurons are active. This model can be viewed as an *exponential number of linear models that share parameters*[6].

Experiment shows both improvement on learning speed and performance [5, 3].

In our model, both the convolution layer and the full connected layer is activate by ReLU function except for the last layer, which followed by the finally loss layer. Because we want the output result to have both positive and negative value, while ReLU only result in positive and zero.

Also this approach can be used in classification problem with softmax and cross entropy loss. Allow result to be negative increase the difference when doing softmax, which result in better classification result.

3.3. Weight Initialization

In a deep enough neural network, initialization of weight parameter sometimes determine whether the training progress converge. A too large initialization cause saturation in case of sigmoid or hyperbolic activation function, or layer-by-layer amplification of output which finally result in infinity for NaN when deep enough in case of ReLU. Both case put a obvious obstacle on gradient descent progress. While too small initialization result in small gradient to propagate (since this gradient is proportional to the value of the weights).

Typical method is to initialize weights from zero-centered normal or uniform distribution with specified standard deviation. The choose of this variance, as a hyper-parameter, becomes a problem. Glorot[2] propose a method, which commonly called "Xavier" initialization, based on the assumption that the activations are linear. It take the form:

$$Var(w) = \frac{2}{n_{in} + n_{out}}$$

where n_{in} and n_{out} are the number of units in the previous layer and the next layer. Or equivalently, n_in is the number of input for each neuron, and n_out is the total number of neuron in this layer.

A more recently research by He[4] provide a more refined analysis specifically on ReLU activation, and it has form:

$$Var(x) = \frac{2}{n}$$

where n is the number of input for each neuron, e.g. the number of unit in previous layer for full connected layer.

The above discussion target full connected layer. Convolution layer, share the same principle, since convolution is also a linear combination equivalent to full connected.

For a convolution filter with size $w \times h$, input channel size n_{in}, and output channel size n_{out}, each output number is associated with $w \times h$ input numbers each channel. So we can have the formula for convolution layer:

$$Var(w) = \frac{2}{(n_{in} + n_{out}) \times (w \times h)}$$

for Glorot's method and:

$$Var(w) = \frac{2}{n \times (w \times h)}$$

for He's method.

Since we have this variance, can easily drop an implementation for this random initialization. Normal distribution can easily multiply $\sqrt{Var(w)}$ to the standard normal distribution. And uniform distribution, can be generate from interval:

$$[-\sqrt{3 * Var(w)}, \sqrt{3 * Var(w)}]$$

In our model, we use both the above two method of initialization, while without any noticeable difference.

When initialize for bias, typical way is set to zero. While some have other ways[5] to initialize to a small positive number to avoid ReLU unit from diminish at the beginning. However, it is not clear if this provides a consistent improvement and it is more common to simply use 0 bias initialization[1].

3.4. Dropout

4. Network Structure

5. Experiment

6. Conclusion

References

[1] CS231n Convolutional Neural Networks for Visual Recognition. `http://http://cs231n.github.io/neural-networks-2/#init`. Accessed: 2015-12-21.

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[3] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.