Transforming Auto-encoders

Anonymous Author(s)
Affiliation
Address
email

Abstract

In recent years, the demand and possible applications for good computer vision algorithms has been staggering. However, we still lack robust systems that can recognize general objects regardless of pose and lighting conditions. This year, a deterministic neural network model was introduced that is not only capable of recognizing features in different poses and lighting, but is also capable of outputing pose-specific variables to be used by higher visual layers rather than discarding them [1]. This paper will provide some motivation for the model, describe it in detail, and introduce ideas for extensions to the model.

1 Introduction

Recent advances in the cognitive sciences [2] and in computer vision [3] have shown the importance of invariant representations in vision and other cognitive tasks. Indeed, humans are capable of very quickly recognizing features regardless of position, orientation, scale, and lighting conditions. This fact suggests that we have invariant representations of such features or objects. Moreover, with the configuration information that we obtain visually, our brains are capable of infering larger features, e.g. seeing two eyes and a nose in the correct relative configuration allows us to infer the existence (and relative position, orientation, etc.) of a mouth, even though the mouth may be hidden.

It has become clear that if a machine is to perform a similar task in a robust way, its vision algorithm must meet some key requirements: locality, invariance, and hierarchy. Such a system must be capable of recognizing local features (one of the reasons for the success of the SIFT algorithm [3]). Meanwhile, it must represent them in an invariant way that is independent of the configuration of the particular instance of the feature. Indeed, we shall henceforth think of the presence of a feature and its configuration in a particular instance as two separate notions. Concerning configuration parameters, an artificial visual system should have an implicitly defined prototype of a feature against which it can compare particular instances of said feature to generate transformation variables. These variables are called *instantiation parameters* and can range from affine transformation parameters to lighting conditions. Finally, the algorithm must have a hierarchical architecture in order to capture the intrinsic part-whole structure of the objects it attempts to recognize. In our previous example, our brains were capable of infering the existence of the mouth because they have been trained to know that a pair of eyes and a nose are usually seen with a mouth as *part* of a face (*whole*). The neuron responsible for firing when a face is in our visual scene, weights its inputs in favour of features such as eyes, nose, and mouth. Such hierarchical structure is believed to be an essential ingredient to our intelligence [4].

Recognizing a whole from its parts is as much an exercise in establishing the presence of features as it is quantifying relative distances and orientations of said features. Indeed, if we saw a pair of eyes and an upside down nose under it, even our advanced brains would be less certain of the existence of a mouth, since the relative orientations no longer match our trained expectation. Therefore, it is important that lower-level feature recognizers relay both the features present and their corresponding instantiation parameters to higher-level recognizers. While the feature presence should be an in-

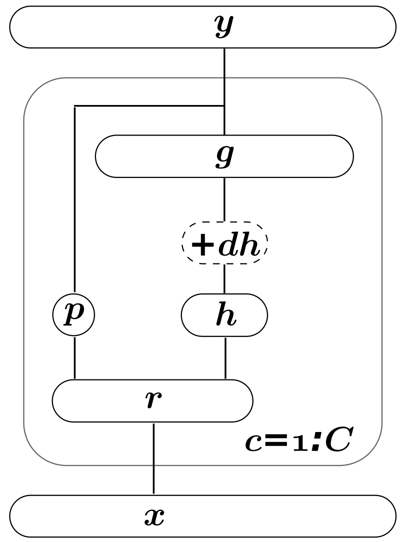

Figure 1: Training the transforming auto-encoder.

variant quantity, independent of transformation, the instantiation parameters should be *equivariant*, meaning they should change with the transformation of the feature in each instance.

Based on these ideas, Hinton et al. have proposed a model [1] which is the subject of this paper. We will start by giving a detailed description of the model in Section 2. In Section 3, we discuss techincal considerations and difficulties that can arise, e.g. activation functions, parameter initialization, etc. Finally, we discuss the merits and limitations of the model, as well as suggest possible future work.

2 Model description

The model proposed by Hinton et al. assumes the existence of a higher-level recognition unit that takes as input the presence and instantiation parameters of all features to recognize the presence of high-level features. Also, for simplicity, we will assume that we are only interested in the location of the features – extension to more complicated transformations is straightforward. Therefore, we are henceforth only concerned with detecting features and their position on the image.

The model is composed of C elements called *capsules*. As a "black box", each capsule is trained to take an input image x, recognize a single feature, and return its probability p of being present in the image, and position vector h in some implicit coordinate system. The value of h is not what is important here, so we never need to know the coordinate system in which each capsule measures h. Recall that h can be any other equivariant quantity we wish to measure: orientation, scale, brightness, etc. We will refer to units belonging to a specific capsule with a subscript c, as in h_c. However, we will omit the subscript when it is superfluous.

Considering our earlier requirements for a good computer vision system, these capsules seem like ideal building blocks. Each capsule's p_c acts as an invariant identifier, while each h_c acts as an equivariant quantifier of transformation.

2.1 Capsule feed-forward

In order to understand how a capsule learns, it is useful to consider a single capsule's feed-forward operation. Figure 1 will help follow the inner workings of a capsule described below.

When capsule c receives an input x, it first computes a dimensionally reduced *representation*, r_c, and from it, computes the desired quantities p_c and h_c. The hidden units, h_c, can then be used to compute a set of *generation* units, g_c, which are in turn used to compute a capsule-specific reconstruction, y_c (not depicted in Figure 1). All the y_c are then modulated by the corresponding p_c and summed to produce a network reconstruction, y. (Note that x and y are deliberately not subscripted because they are the entire network's input and output.)

In principle, when the capsules are fully trained, performing, $h'_c = h_c + \Delta h,\ \forall c$, before the capsule-specific reconstruction (as shown in Figure 1) should shift the entire input image x by Δh, since *all* features are shifted by that amount. With this particular idea in mind, let us now consider the task of training the entire network.

2.2 Training the network

All the capsules are trained together as a network. Recall that we have restricted our attention to h representing a $2D$ position, but an extension to orientation, lighting conditions, or other transformations is straightforward.

The idea is to pick an x and a translation Δh, resulting in a shifted image x'. The network is then given x and Δh as inputs and returns an output y as described above. The reconstruction, y, will be compared to x' in some cost function, $J(y, x'|\theta)$, where θ represents all the parameters in the network.

Therefore, training this neural network is no different from a generic neural network; it consists of feeding forward, computing gradients by back-propagation, and updating by stochastic gradient descent, as outlined in Algorithm 1.

Algorithm 1 Training a transforming auto-encoder, using back-propagation and stochastic gradient descent.

```
for mini-batch in training data do
  for all capsule do
    Compute presence p_c and instantiation h_c
    Transform h_c
    Compute capsule-specific transformed reconstruction y_c and other activations
    return All activations
  end for
  Compute transformed reconstruction: ∑_c p_c y_c
  Compute true transformed batch
  for all capsule do
    Compute gradients evaluated at activations and current weights given reconstructed and
    true batch
    Average gradients over mini-batch
    Update weights
  end for
end for
```

2.3 Activation functions

Finally, we provide a more explicit description of the model, with expressions for the various activations involved. They are provided in feed-forward order.

$$r_c = \sigma\left(xW^c_{\text{rep}} + b^c_{\text{rep}}\right) \tag{1}$$

$$p_c = \sigma\left(r_c W^c_{\text{act}} + b^c_{\text{act}}\right) \tag{2}$$

$$h_c = r_c W^c_{\text{trans}} + b^c_{\text{trans}} \tag{3}$$

$$h'_c = h_c + \Delta h \tag{4}$$

$$g_c = \sigma\left(h'_c W^c_{\text{gen}} + b^c_{\text{gen}}\right) \tag{5}$$

$$y_c = \sigma\left(g_c W^c_{\text{out}} + b^c_{\text{out}}\right) \tag{6}$$

$$J(y, x'|\theta) = \left|\left|\sum_c p_c y_c - x'\right|\right|^2 \tag{7}$$

where $\sigma(\cdot)$ is an activation function. The units $r_c, p_c, \ldots$ are called *activations* because they are activated by the units of the previous layer.

The cost function will be minimized when h_c corresponds to the position of the feature that capsule c is responsible for recognizing. Therefore, if the weights are made to follow a gradient descent, they should converge to the weights that will provide the desired h_c. Notice that since h_c is to represent a real valued scalar (not an activation or probability), it is obtained by a linear regression alone. There is still some freedom in the choice of σ, however, the most popular being the sigmoid, $\tanh$, and softsign functions [5].

3 Technical Considerations

Though the model is simple to understand and implement, when training the network, one encounters a few degrees of freedom, that are not addressed in Hinton, Kryzkevsky, and Wang's paper [1]. As we will see these choices can have dramatic effects on training.

3.1 Activation functions revisited

One choice we have, that was already mentioned, is the activation function. Possibly the simplest and most commonly used is the sigmoid activation function. However, without pre-training, we have no choice but to randomly initialize the weights. In this setting, the sigmoid is known to slow down learning in deep networks [6]. This effect is due to the saturation of activations, and indeed we have observed this phenomenon in our experiments. The hyperbolic tangent and softsign ($\sigma(x) = \frac{x}{1+|x|}$) activation functions suffer a similar fate. In experiments conducted by Bengio et al. $\tanh$ activations would saturate one layer at a time, in feed-forward order, while the same experiment with softsign activations yielded a gradual saturation of all layers in unison.

3.2 Normalized random initialization

Bengio et al. also studied the effect of initialization on activations and gradients. They showed that, using a previous heuristic random initialization, the activations and back-propagated gradients would undesireably approach zero for deeper layers.

In constraining the variance of the gradients, they have established a new so-called *normalized initialization* [6] as follows:

$$W \sim U\left[-\sqrt{\frac{6}{n_{l_1}+n_{l_2}}}, \sqrt{\frac{6}{n_{l_1}+n_{l_2}}}\right] \tag{8}$$

where $U[a, b]$ represents a uniform distribution in the interval $[a, b]$, and n_{l_1}, n_{l_2} correspond to the number of rows and columns of W, respectively.

Indeed, with this new initialization and the hyperbolic tangent activation function, Bengio et al. showed that the activations and back-propagated gradients have the same distribution across all layers.

4 Discussion and Future work

The model's capacity to produce invariant representations as well as equivariant instantiation parameters is a considerable advantage over other computer vision algorithms. We suspect that extracting information on the pose of particular instances of features will be instrumental in building the next generation of vision systems.

On the topic of extracting pose information, it is worth mentioning similar work by Hinton and Memisevic. Using a probabilistic higher-order factored Restricted Boltzmann Machines (RBM) they learned filters that encode transformations (translations, rotations, scaling, and even random kernel transformations) [7]. However, their paper only mentions transforming entire images and needs to be extended to feature-based transformations, for a proper comparison to the deterministic model described in the present paper.

Nevertheless, one idea from the Memisevic paper is worth attempting with the transforming auto-encoder model. The idea is to use randomly generated images to train the auto-encoder. If this works, the capsules would be trained to recognize general features that are not due to a particular data set. However, we remain skeptical that this would work. In any case, in principle this would require many more capsules with many more representation and generation units to handle the increased domain of possible features, which could render training prohibitively slow.

On the other hand, the idea of transfer learning is a task that can be readily tried given a working auto-encoder model, and we believe this experiment has a high probability of success. For instance, the transforming auto-encoder could be trained to detect features and their positions on MNIST. (MNIST is a famous data set of handwritten digits 0 to 9.) Those same capsules can then be given a handwritten character to recognize. The domain of features should be very similar for both handwritten digits and characters, so this task should be relatively easy for the auto-encode.

Furthermore, one could try analogy learning, which is also demonstrated in [7]. For this exeperiment, the MNIST-trained auto-encoder would be given a handwritten digit and its translated version, recording the shifts in position for each capsule. Then, a handwritten character can be fed to the auto-encoder with the recorded shifts; the output should be the same character shifted in the same way the digit was. As with transfer learning, we are optimistic that this very simple extension would succeed.

Finally, transforming auto-encoders have potential application in video compressing, and semantic hashing. Indeed, if one were able to extract information on how features transform in a sequence of images, in principle, one could compress video by eliminating redundant frames. Similarly, one could extract spatio-temporal features that carry more meaning for semantic hashing. For instance, if one feature is 'dog' and another is 'cat', having information on how they transform from frame to frame can help categorize the video as either 'dog chasing cat' or 'dog and cat cuddling'. Moreover, if this can be done quickly, with streaming data in real-time, then there are straightforward applications to MicroSoft's Kinect and other such gaming devices.

Acknowledgements

Thanks to Misha and Mareija for help, support, and proofreading.

References

[1] G. E. Hinton & A. Krizhevsky & S. D. Wang. Transforming Auto-encoders, 2011.

[2] R. Quian Quiroga & L. Reddy & G. Kreiman & C. Koch & I. Fried. Invariant visual representation by single neurons in the human brain, 2005.

[3] D. G. Lowe. Object recognition from local scale-invariant features, 1999.

[4] J. Hawkins & S. Blakeslee. On Intelligence: How a New Understanding of the Brain will Lead to the Creation of Truly Intelligent Machines, 2004.

[5] J. Bergstra & G. Desjardins & P. Lamblin & Y. Bengio. Quadratic polynomials learn better image features, 2009.

[6] X. Glorot & Y. Bengio. Understanding the difficulty of training deep feedforward neural networks, 2010.

[7] R. Memisevic & G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines, 2010.