Combating Adversarial Inputs Using a Predictive-Estimator Network

Jeff Orchard(✉) and Louis Castricato

Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
{jorchard,lcastric}@uwaterloo.ca
http://cs.uwaterloo.ca/∼jorchard/

Abstract. Deep classification networks have shown great accuracy in classifying inputs. However, they fall prey to adversarial inputs, random inputs chosen to yield a classification with a high confidence. But perception is a two-way process, involving the interplay between feedforward sensory input and feedback expectations. In this paper, we construct a predictive estimator (PE) network, incorporating generative (predictive) feedback, and show that the PE network is less susceptible to adversarial inputs. We also demonstrate some other properties of the PE network.

Keywords: Neural network · Predictive estimator · Autoencoder · Adversarial

1 Introduction

Adversarial input is input chosen specifically to yield a classification with high confidence, and yet not resemble any typical members of that class. For example, a search process found random images (as well as generic patterns) that, when fed into AlexNet [1], were classified with greater than 99.99% confidence [2]. Recent work has looked into overcoming the issue of adversarial input [3,4].

Our hypothesis is that a network with built-in generative capabilities might be able to overcome such adversarial input. Most perceptual networks are feedforward, taking in sensory input and generating a classification as its output [1]. But some networks are also generative; Hinton et al. used RBMs to model the two-way process of recognizing and generating images of hand-written digits, trained on the MNIST dataset [5].

Perception is a two-way process, where sensory inputs interact with expectations, attempting to find a network state that is consistent with both. For example, when you are looking at a raisin, and you are told it looks like a person's face, it shifts your perception; your expectations impinge on the process and your network shifts into a state in which the sensory input of the raisin is meshed with your expectation of a face. You try to see the raisin as a face.

The anatomy of the sensory cortices supports the notion that feedback plays an important role in perception, since most connections between cortical regions are reciprocal (two-way) [6].

D. Liu et al. (Eds.): ICONIP 2017, Part II, LNCS 10635, pp. 118–125, 2017.
https://doi.org/10.1007/978-3-319-70096-0_13

A predictive estimator (PE) is an architecture that has built-in feedback [7]. The higher layers in the perceptual hierarchy send down predictions of what the lower layer should be experiencing, and the lower layers send up the error between that predication and their actual state. However, previous work on PEs have used copied connection weights, where the feedforward connection weights are also used as the feedback connection weights.

In this paper, we aim to generate a predictive estimator network (without weight copying) and see whether the back-and-forth operation of the PE network could be used to combat adversarial inputs. After all, the feedback projections in a PE network contain predictions. These predictions might increase the classification specificity.

2 Methods

Our approach to creating a deep predictive estimator (PE) network is to train a bidirectional network and then use its connections in our PE network.

2.1 Training a Bidirectional Network

Consider the bidirectional network shown in Fig. 1. The network has feedforward connection weights, $\mathbf{W}$ and $\mathbf{P}$, and corresponding biases, $\boldsymbol{a}$ and $\boldsymbol{b}$, as well as feedback connection weights, $\mathbf{R}$ and $\mathbf{M}$, and biases, $\boldsymbol{c}$ and $\boldsymbol{d}$.

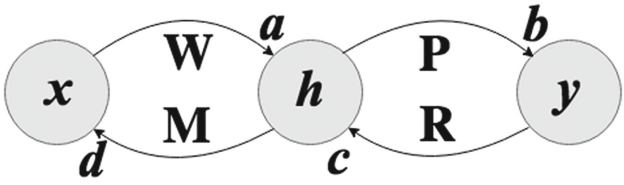

Fig. 1. Simple bidirectional network

If we set the feedback connections to zero (i.e. $\mathbf{M}$, $\mathbf{R}$, $\boldsymbol{c}$ and $\boldsymbol{d}$ are all zeros), then the network simply behaves as a feedforward network. We can train that network using a dataset of $(\boldsymbol{x}, \boldsymbol{y})$ samples, so that

$$\boldsymbol{y} = \sigma\left(\mathbf{P}\boldsymbol{h}_{\text{ff}} + \boldsymbol{b}\right) \text{ , where} \tag{1}$$

$$\boldsymbol{h}_{\text{ff}} = \sigma\left(\mathbf{W}\boldsymbol{x} + \boldsymbol{a}\right) , \tag{2}$$

where σ is the logistic activation function. Likewise, if all of the feedforward connections are zero (i.e. $\mathbf{W}$, $\mathbf{P}$, $\boldsymbol{a}$ and $\boldsymbol{b}$ are all zero), then the network behaves in a purely feedback manner. Such a network can be trained so that

$$\boldsymbol{x} = \sigma\left(\mathbf{M}\boldsymbol{h}_{\text{fb}} + \boldsymbol{d}\right) \text{ , where} \tag{3}$$

$$\boldsymbol{h}_{\text{fb}} = \sigma\left(\mathbf{R}\boldsymbol{y} + \boldsymbol{c}\right) . \tag{4}$$

However, notice that $\boldsymbol{h}_{\text{ff}}$ is not necessarily the same as $\boldsymbol{h}_{\text{fb}}$. The intermediate (hidden) representation for the two directions does not have to be the same, especially if the feedforward and feedback connections are learned independently of each other.

Let us suppose that the feedforward and feedback networks were trained using backpropagation so that

$$\boldsymbol{h} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{a}) \tag{5}$$
$$\boldsymbol{y} = \sigma(\mathbf{P}\boldsymbol{h} + \boldsymbol{b}) \tag{6}$$
$$\boldsymbol{h} = \sigma(\mathbf{R}\boldsymbol{y} + \boldsymbol{c}) \tag{7}$$
$$\boldsymbol{x} = \sigma(\mathbf{M}\boldsymbol{h} + \boldsymbol{d}). \tag{8}$$

It is not sufficient to train $\mathbf{W}$ and $\mathbf{P}$ using simple backpropagation, followed by training $\mathbf{M}$ and $\mathbf{R}$ using backpropagation in the opposite direction. The problem is that the activity in the intermediate layers need to be approximately the same for the feedforward and feedback modes of operation.

Instead of training one direction at a time, we trained one layer at a time as an autoencoder [8]. This is now a fairly common practice. According to the simple network depicted in Fig. 1, we first learn the connection weights $(\mathbf{W}, \boldsymbol{a})$ and $(\mathbf{M}, \boldsymbol{d})$ by minimizing the reconstruction error,

$$C\Big(\boldsymbol{x}, \sigma\big(\mathbf{M}\sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{a}) + \boldsymbol{d}\big)\Big),$$

where C is a cost function such as sum-of-squares, or cross entropy. That is, we project the input $\boldsymbol{x}$ up to the hidden layer, and then project the activity in the hidden layer back down to the input layer. The difference between the input and the reconstruction is used to update the up and down connections between those two layers.

After the first layer is trained as an autoencoder, we train the next layer. If the next layer reaches the output layer of the network, then we train it as an associative memory. This method for learning bidirectional connection weights and biases is similar to the method outlined in [9].

2.2 Predictive Estimator Network

We want to use the learned bidirectional network to generate a continuous-time predictive-estimator network, like the one shown in Fig. 2. The figure shows three PE units. Each PE unit contains a *state* node, denoted $\boldsymbol{x}$, $\boldsymbol{h}$ and $\boldsymbol{y}$ in the figure, and an *error* node, denoted δ, ε and γ in the figure. Within each PE unit (shown as a shaded box), the state node and the error node are reciprocally connected.

The parameter β can be used to adjust the relative weight of sensory input (coming from the lower levels of the hierarchy on the left) and expectation input (coming from the higher levels on the right). When β is 0, the input to the state nodes comes solely from the feedforward projection. When β is 1, the opposite is true, and the state nodes only receive input from the error node. Hence, when

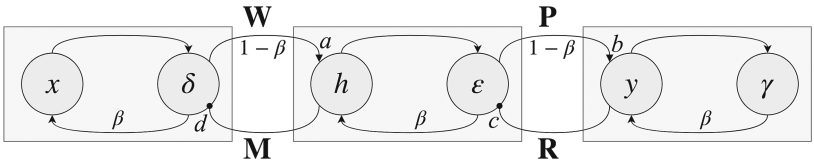

Fig. 2. Diagram of part of a network

$\beta = 1$, the PE network operates in a feedback mode where the state nodes only receive input from higher levels in the hierarchy.

The equations that govern the dynamics of the central PE unit in Fig. 2 are,

$$\tau \frac{d\boldsymbol{z}}{dt} = \mathbf{W}\boldsymbol{\delta} - (\rho + \beta)\boldsymbol{z} \tag{9}$$

$$\tau \frac{d\boldsymbol{h}}{dt} = (1 - \beta)\Big(\sigma(\boldsymbol{z} + \boldsymbol{a}) - \boldsymbol{h}\Big) - \beta\varepsilon \tag{10}$$

$$\tau \frac{d\varepsilon}{dt} = \boldsymbol{h} - \sigma(\mathbf{R}\boldsymbol{y} + \boldsymbol{c}) - \varepsilon \tag{11}$$

where τ represents a time constant. The variable $\boldsymbol{z}$ represents the input current coming from the error node below, and ρ is its default decay coefficient. Roughly speaking, (9) integrates the unit's input current $\boldsymbol{z}$, (10) converts the input current to an activation $\boldsymbol{h}$, and (11) tracks the unit's error ε, which is the difference between the unit's state, $\boldsymbol{h}$, and the predication being sent down from above.

Note that if β is large (close to 1), then the network is operating in a primarily feedback mode, and the last term in (9) causes $\boldsymbol{z}$ to decay quickly and have little influence on $\boldsymbol{h}$. On the other hand, if β is small (close to 0), then the network is operating in a primarily feedforward mode, and (9) behaves more like an integrator that accumulates the errors being sent from below.

Perhaps the best way to understand the functioning of the PE unit is to study the equilibrium solutions of Eqs. (9)-(11). Consider the equilibrium state of the feedforward system (i.e. set all three derivatives on the left to zero, and let $\beta = 0$). Then Eq. (10) can be written

$$\boldsymbol{h} = \sigma(\boldsymbol{z} + \boldsymbol{b})\,.$$

This would match (5) if we could show that $\boldsymbol{z}$ must be $\mathbf{W}\boldsymbol{x}$. Suppose, for now, that $\rho = 0$, and consider what happens if we let $\boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \Delta\boldsymbol{z}$. A perturbation analysis (not shown here) seems to suggest that the feedback loop through δ (see Fig. 2) would result in $\frac{d\boldsymbol{z}}{dt} \propto -\Delta\boldsymbol{z}$. Thus, $\boldsymbol{z}$ would be pushed back towards $\mathbf{W}\boldsymbol{x}$.[1]

Equation (11) implies that $\varepsilon \to 0$ since we know that $\boldsymbol{h} = \sigma(\mathbf{R}\boldsymbol{y}+\boldsymbol{c})$ from (7). Finally, δ would also have to be zero.

If β is equal to 1 instead, we can follow a similar argument to show that $\varepsilon = 0$ (from Eq. (10)), and $\boldsymbol{h} = \sigma(\mathbf{R}\boldsymbol{y} + \boldsymbol{c})$, and $\boldsymbol{z} = \frac{\mathbf{W}\delta}{\beta+\rho}$. But δ must be zero for the same reason ε is zero. Hence, $\boldsymbol{z} = 0$.

[1] The default decay rate of ρ would eventually drive $\boldsymbol{z} \to 0$, but our analysis is relevant as long as ρ is small.

This equilibrium is the goal state of the PE network, in which the error nodes are zero, and the state nodes encode the data to translate between the input and the output (bottom and top) layers.

3 Experiments

3.1 PE Network Behaviour

To get a feel for how the PE network behaves, we ran some experiments using the MNIST dataset. A bidirectional network with five layers (three hidden layers) was trained using the method outlined in Sect. 2.1. The input layer had 784 nodes (the number of pixels in a 28×28 image), and the hidden layers had 100, 80, and 80 nodes. The output layer had 10 nodes. We trained for 30 epochs using stochastic gradient descent with a batch size of 10. Our learning rate was 0.05. We used cross entropy as our cost, and we regularized the connection weights using a decay of 0.001. The parameter τ was set to 0.05 s, and ρ was set to 0.1. After the training, the feedforward part of the network achieved a test accuracy of 76%. We know that other neural networks can do better after more training, but this accuracy was sufficient to serve our purposes.

After training the bidirectional network, we used the connection weights and biases to create our PE network, as outlined in Sect. 2.2. Then we fed a chosen digit as input and simulated the PE network. Note that the PE network receives input at both the bottom layer and the top layer. However, if $\beta = 0$, the network operates in feedforward mode and the top-layer input is ignored. Likewise, if $\beta = 1$, the network operates in feedback mode and the bottom-layer input is ignored.

In these experiments, we ran our network for 1 simulation second, after which we cut off the inputs and simulated some more. The inputs were removed by setting $\beta = 1$ for the bottom layer, and setting $\beta = 0$ for the top layer (each layer can have a different β). The purpose of this sequence is to first allow the input to set the network state, but then remove the inputs and let the PE network "deliberate". After this process, we looked at the top-layer state to see how it compared to the ideal in the test set.

Our PE network yielded a 71% accuracy with $\beta = 0$, using a deliberation time of 0.

Figure 3 shows one example. The true digit (on the left) is a "4". The bidirectional network incorrectly classified this input as a "6" with 41.6% confidence. However, the PE network correctly classified it as a "4" with 23.9% confidence.

Figure 4 shows another example. The bidirectional network classified it as a "3" with 43.8% confidence. In feedforward mode ($\beta = 0$), the PE network classified it as either a "2" or "3" with confidence around 25%. However, we can allow expectation to influence the perception. If β is set to 0.2, then it weights the feedforward/feedback with 80/20. Then the PE network favours the classification of "2" with confidence 28.8%. The corresponding generative images are shown in Fig. 4.

Fig. 3. MNIST example. The image on the left is the original MNIST sample. The middle image was generated using the bidirectional network, (setting the top layer to class 4). The image on the right is the one generated by the PE network.

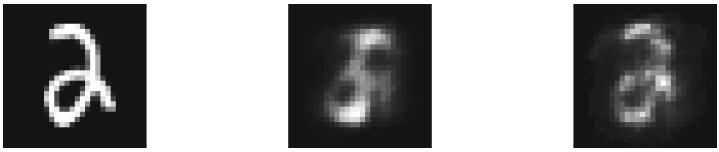

Fig. 4. MNIST example. The image on the left is the original MNIST sample. The middle image was generated using the bidirectional network (setting the top layer to class 2). The image on the right is the one generated by the PE network with $\beta = 0.2$.

3.2 Susceptibility to Adversarial Input

One of our hypotheses is that the generative, feedback nature of the PE networks will prevent it from being fooled by many adversarial inputs. To test this hypothesis, we ran an experiment on a small dataset, in which binary strings of length 8 are classified into 5 categories.

$$[1,0,1,0,0,1,1,0] \leftrightarrow [1,0,0,0,0]$$
$$[0,1,0,1,0,1,0,1] \leftrightarrow [0,1,0,0,0]$$
$$[0,1,1,0,1,0,0,1] \leftrightarrow [0,0,1,0,0]$$
$$[1,0,0,0,1,0,1,1] \leftrightarrow [0,0,0,1,0]$$
$$[1,0,0,1,0,1,0,1] \leftrightarrow [0,0,0,0,1]$$

We trained a bidirectional network on that data, and then built the corresponding PE network. The bidirectional network was trained on 300,000 samples, using stochastic gradient descent with a batch size of 10. We used cross entropy as our cost function, and the learning rate was 0.01.

Adversarial input was generated by simply choosing random inputs of 8 values, where each value was chosen randomly from a uniform distribution between 0 and 1. Each sample was fed into the bidirectional network and the PE network. The output of each network was assessed to see if either one yielded a "confident" classification (ie. if the soft-max output gave a value greater than 0.95).

For this experiment, the PE network was run for 1 simulation second with $\beta = 0$ (feedforward mode).

In the adversarial input experiment, the feedforward network yielded a 95% confidence on the random inputs 5% of the time (50 times out of 1000 trials). The PE network did not show a confidence level of 95% on any of the 1000 random strings. However, both the feedforward network and the PE network exhibited 100% accuracy in classifying the true binary strings.

4 Conclusions

The predictive encoder network we created from the connection weights of the bidirectional network exhibited some interesting properties. The feedback inherent in the PE network allowed, in some cases, the network to deliberate and change its mind on an incorrect classification. Moreover, in some cases, allowing an expectation (by setting β to a non-zero value) helped the network to converge to the correct class.

As hypothesized, the predictive estimator network was far less susceptible to adversarial (random) inputs. While the feedforward part of the bidirectional network confidently (mis)classified random inputs 5% of the time, we never observed the PE network confidently misclassify any random inputs. This is despite the fact that both networks exhibited 100% accuracy on the binary dataset.

More study is needed to try to get the accuracy of the bidirectional and PE networks up to contemporary levels on the MNIST dataset (i.e. greater than 98% accuracy). However, we note that the MNIST dataset is not invertible; knowing a digit class is not enough to generate the input image. We plan to investigate stochastic extensions to our method to allow us to generate a variety of inputs, similar to RBMs.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105 (2012)
2. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: Computer Vision and Pattern Recognition, pp. 427–436 (2015)
3. Goodfellow, I., Pouget-Abadie, J., Mirza, M.: Generative adversarial networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
4. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proceedings of ICLR. arXiv:1412.6572v3 (2015)
5. Hinton, G.E.: A Practical guide to training restricted boltzmann machines. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, pp. 599–619. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35289-8_32
6. Bastos, A.M., Usrey, W.M., Adams, R., Mangun, G.R., Fries, P., Friston, K.J.: Canonical Microcircuits for Predictive Coding. Neuron **76**(4), 695–711 (2012)

7. Rao, R.P.N., Ballard, D.H.: Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nat. Neurosci. **2**(1), 79–87 (1999)
8. Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in Neural Information Processing Systems, pp. 396–404 (1990)
9. Luo, H., Fu, J., Glass, J.: Bidirectional Backpropagation: Towards Biologically Plausible Error Signal Transmission in Neural Networks. arXiv:1702.07097v3 (2017)