1 Introduction

This report summarized some experiments about asymmetric backpropagation. The backpropagation algorithm is crucial to neural network training, and it's based on the chain rule of derivatives. However, when backpropagation is viewed in a way of neural signal feedback, it's not biological plausible since the backward signal in real neural system is through a independent path. This limitation is due to the fact that neural cells are all one way cells. Asymmetric backpropagation do not use the information of weights in the forward propagation, which leads to a new form of neural network training.

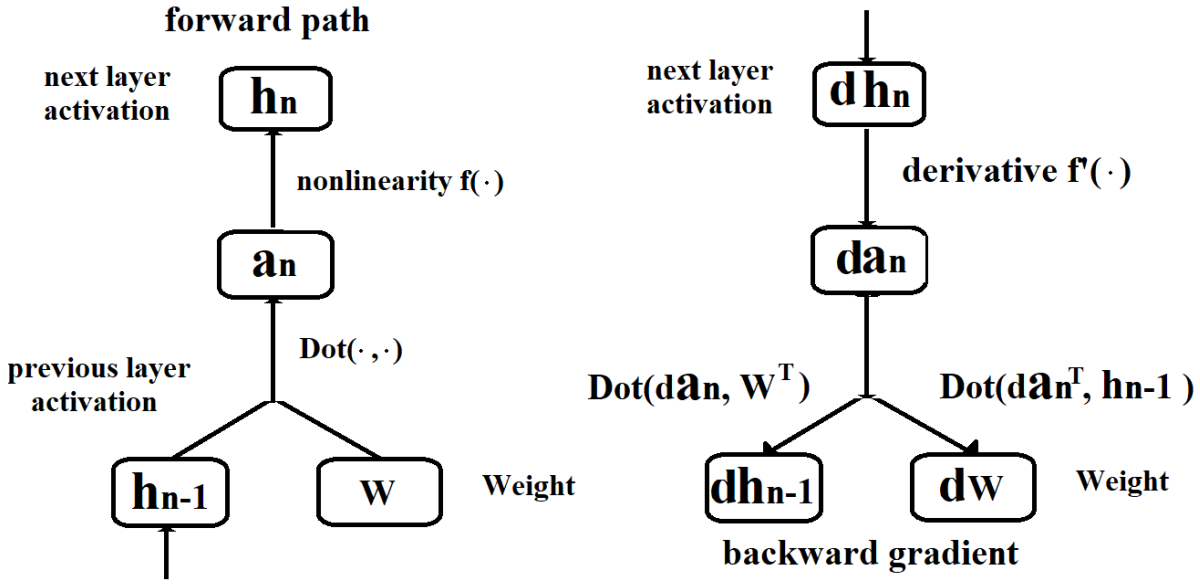

Figure 1: The computation graph of one dense layer, left graph shows the forward path and the right graph shows the backward path.

Figure 1 shows the computation graph in one dense layer, in the neural network implemeted by "graph based" auto differentiation, typical dense layers is:

$$h_{n+1} = f(h_n W) \tag{1}$$

where h_n is the preception state value of layer n and W is the connection weight from layer n to layer n+1. f is the nonlinearity, possibly RELU. In real implemention, h is a matrix with row for batch size and p column for features at this layer, W has p row for feature in the layer n and p' column for feature in layer $n+1$.

In the original backpropagation, let L be our final loss, then according to the chain rule:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h_{n+1}} \frac{\partial h_{n+1}}{\partial h_n W} \frac{\partial h_n W}{\partial W} = h_n^T (\frac{\partial L}{\partial h_{n+1}} * f'(h_n W)) = dot(h_n^T, G) \tag{2}$$

$$\frac{\partial L}{\partial h_n} = \frac{\partial L}{\partial h_{n+1}} \frac{\partial h_{n+1}}{\partial h_n W} \frac{\partial h_n W}{\partial h_n} = (\frac{\partial L}{\partial h_{n+1}} * f'(h_n W)) W^T = dot(G, W^T) \tag{3}$$

$$\tag{4}$$

Here G denote value matrix of $\frac{\partial L}{\partial h_n W}$, operator $*$ denote element-wise product.

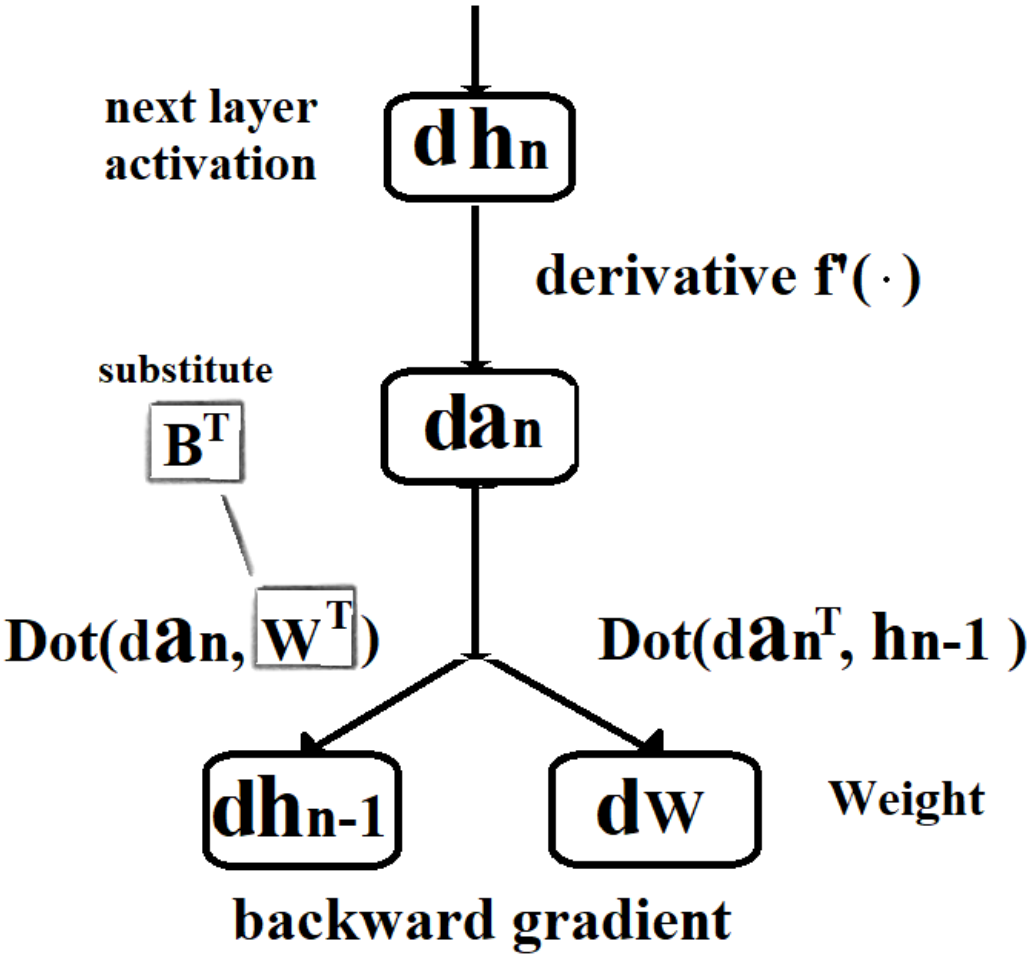

Figure 2: The computation graph for asymmetric backpropagation

In biological analogy, G is the feedback signal, the gradient for W is fine because the it do not involve signal from forward path. It only need the cell state h_n. However, the gradient for h_n is problematic in that a backward neural connection cannot know information from the forward path, say W. Asymmetric backpropagation now assume that

$$\frac{\partial L}{\partial h_n} = dot(G, B^T) \tag{5}$$

where B substitude W as something else. In Figure 2., what we need to change is inserting another matrix in the computation graph.

2 Related Works

In this section some major works for asymmetric backpropagation is summarized.

2.1 Lillicrap 2014,Random feedback weights support learning in deep neural networks

In this paper, the author first brought up the question that algorithm for backpropagation is not suitable for brain computing, since the gradient defined require synaptic weight transfer, which is impossible for neurons in brain. The alternative way is to use a matrix that is randomly initialized to replace weight W in the back propagation. The author argue that even if the matrix involved is not mathematically correct in the gradient computation, the network will adjust itself, more specificly adjusting weight W in the forward path to learn how to use tha substituted matrix B.
To illustrate the learning power of asymmetric backpropagation, the author tried 3 different task, all of which achieved good training result:

- Learning a fixed linear mapping from a 30 dimensional space to a 10 dimensional space.
- Learning to classify MNIST dataset.
- Learning to approximate a nonlinear map(a neural network) ...

To support the arguement that the forward path tried to learn how to use the random feedback substituted in tha backward path, the author plot the angle of feedback gradient from asymmetric backpropagation and the mathematically correct gradient. It can be seen that for proper initialization the asymmetric back propagation gradient is very close to the correct one after some epochs of training.

2.2 Poggio 2016, How Important Is Weight Symmetry in Backpropagation?

In Lillicrap's paper only dense layer is used in the network. One interesting extension is that whether the same things happen for convolutional neural network. This paper tried to use asymmetric back-propagation on the classification layers and possibly dense layers before it and after convolution layers.

From their result it's evident that the asymmetric backpropagation is problematic when the network is deeper and especially with convolutional layers. There are 2 major contributions in the paper: one is that they explore the possible structures that are crucial to ensure that the training is effective.They conducted various experiments to support that the sign of the random feedback matrix B matters, but the magnitude does not. Secondly batch normalization is essential in preventing exploding gradient.
The list below is the summary for the variability they control:

- Sign concordance: the percentage of sign that are concordant with the right feedback.
- Feedback magnitude: they keep the sign and then random the magnitude of the feedback, either fixed or re-initialize every batch.
- Whether to use Batch normalization
- Whether to use batch manhattan(see original paper for detail) ...

The key result they observed is that:

- Batch normalization stabilized training on diverse dataset and produce good accuracy.
- Under batch normalization, keep the sign to be concordant, varying the magnitude do not change accuracy.
- Accuracy drop when large percentage of the sign is altered. ...

3 Asymmetric Backpropagation With Only Dense Layers

To see whether this kind of things works, I tried MNIST for digit classification. But MNIST is so simple that everythings works.Instead I trained some networks on CIFAR10 and then observe the difference.

3.1 Shallow Networks with Asymmetric Backpropatgation

The network here are merely dense layers, without convolutional layers. All nonlinearities is RELU, R denote W substituted by a random matrix when sending gradient to the previous layer, and optionally I can update them as W(marked as R*).
Also, hinge loss can be used instead of crossentropy. Hinge Loss is more simple when backpropagate gradient, which is easier for real biological cell to "process". In the model specification table, HL denote hingle loss objective.
Adam is used for these training and results are collected after 100 epoch.

Table 1: Model Specification, with Softmax and Crossentropy/ Hinge Loss

DS0	DSR1	DSR2	DSR1-U	DSR2-U
Dense512	Dense512	Dense512	Dense512	Dense512
Dense256	Dense256	Dense256(R)	Dense256	Dense256(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Softmax	Softmax	Softmax	Softmax	Softmax
CE	CE	CE	CE	CE
DS0-HL	DSR1-HL	DSR2-HL	DSR1-U-HL	DSR2-U-HL
Dense512	Dense512	Dense512	Dense512	Dense512
Dense256	Dense256	Dense256(R)	Dense256	Dense256(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Identity	Identity	Identity	Identity	Identity
HL	HL	HL	HL	HL

The result are:

Table 2: Accuracy on test set for dense layers, after 100 epoch

DS0	50.06%	DS0-HL	51.54%
DSR1	50.08%	DSR1-HL	48.83%
DSR2	44.19%	DSR2-HL	38.66%
DSR1-U	49.37%	DSR1-U-HL	48.82%
DSR2-U	50.49%	DSR2-U-HL	48.93%

It can be seen from the results that asymmetric backpropagation can achieve reasonable performance in shallow dense layer neural network. Stacking asymmetric layers make things worse.

3.2 Deeper Networks with Asymmetric Backpropatgation

Here more dense layers are used and there is some difference in the result.Again (R) marks the asymmetric random feedback in that layer, and (R*) marks the updating random feedback. As the network become deeper, the random feedback start to have severe impact on traininig process. Typically 100 epoch is not enough, so I also collect results after 200 epoch and put them in the brackect.

Table 3: Model Specification, with Softmax and Crossentropy/ Hinge Loss

DD0	DDR1	DDR2	DDR4	DDR1-U	DDR2-U	DDR4-U
Dense512	Dense512	Dense512	Dense512	Dense512	Dense512	Dense512
Dense512	Dense512	Dense512	Dense512(R)	Dense512	Dense512	Dense512(R*)
Dense256	Dense256	Dense256	Dense256(R)	Dense256	Dense256	Dense256(R*)
Dense256	Dense256	Dense256(R)	Dense256(R)	Dense256	Dense256(R*)	Dense256(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)	Dense10(R*)	Dense10(R*)
CE	CE	CE	CE	CE		
DD0-HL	DDR1-HL	DDR2-HL	DDR4-HL	DDR1-U-HL	DDR2-U-HL	DDR4-U-HL
Dense512	Dense512	Dense512	Dense512	Dense512	Dense512	Dense512
Dense512	Dense512	Dense512	Dense512(R)	Dense512	Dense512	Dense512(R*)
Dense256	Dense256	Dense256	Dense256(R)	Dense256	Dense256	Dense256(R*)
Dense256	Dense256	Dense256(R)	Dense256(R)	Dense256	Dense256(R*)	Dense256(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)	Dense10(R*)	Dense10(R*)
Identity	Identity	Identity	Identity	Identity	Identity	Identity
HL	HL	HL	HL	HL	HL	HL

Table 4: Accuracy on test set for dense layers, 200 epoch result in the bracket

DD0	49.06%	DD0-HL	51.67%
DDR1	38.82 (41.11)%	DDR1-HL	30.42 (39.34)%
DDR2	26.94 (31.60)%	DDR2-HL	29.27 (29.63)%
DDR4	27.7 (35.08)%	DDR4-HL	28.86 (26.36)%
DDR1-U	40.27 (45.70)%	DDR1-U-HL	40.46 (45.86)%
DDR2-U	35.35 (44.80)%	DDR2-U-HL	35.83 (45.43)%
DDR4-U	45.27 (42.18)%	DDR4-U-HL	39.65 (43.93)%

4 Poggio's Net with Asymmetric Backpropagation

In this section I used poggio's net, which is a shallow convolutional neural network with maxpooling and average pooling(Q Liao, JZ Leibo, T Poggio, AAAI 2016). The random feedback occurs in the dense layers. In the table below, correctness is showed together with result at 200/300 epoch in the bracket.

From the result we can see that with random feedback in the dense layer the training is very hard. Typically more than 200 epoch is needed with Adam to achieve best result, though there is still large fluctuations even after 200 epoch. It's evident that stacking more random feedback layers prevent good result. Updating the random feedback can help, especially when we just have random feedback in the last layer. There is no significant difference between the hinge loss and crossentropy, most of time the crossentropy is better.

Using SGD works very well (68%) with (non updating) random feedback in the last layer, but given more random feedback dense layers, it failed(huge training/validation loss). Nestrov Momentum fails all the time with random feedback. Changing average pooling to max pooling in poggio's net can resolve this problem(with 2 random feedback layer), however the average pooling is very useful in improving the training result in poggio's net. Updating the random feedback in the same way as the true weight can

Table 5: Model Specification, with Softmax and Crossentropy/ Hinge Loss

PG0	PGR1	PGR2	PGR1-U	PGR2-U
Conv5*5/32	Conv5*5/32	Conv5*5/32	Conv5*5/32	Conv5*5/32
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Dense128	Dense128(R)	Dense128(R)	Dense128(R*)	Dense128(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Softmax	Softmax	Softmax	Softmax	Softmax
CE	CE	CE	CE	CE
PG0	PGR1	PGR2	PGR1-U	PGR2-U
Conv5*5/32	Conv5*5/32	Conv5*5/32	Conv5*5/32	Conv5*5/32
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64	Conv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Dense128	Dense128(R)	Dense128(R)	Dense128(R*)	Dense128(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Identity	Identity	Identity	Identity	Identity
HL	HL	HL	HL	HL

Table 6: Accuracy on test set for Poggio's net, result after 200/300 epoch in the bracket

PG0	75.87%	PG0-HL	75.12%
PGR1	29.27 (40.65, 54.13)%	PGR1-HL	28.85 (40.38, 54.10)%
PGR2	33.84 (43.79, 43.87)%	PGR2-HL	30.93 (39.02, 40.55)%
PGR1-U	28.32 (53.77, 67.48)%	PGR1-U-HL	28.27 (50.89, 62.07)%
PGR2-U	27.44 (45.61, 57.95)%	PGR2-U-HL	31.52 (34.49, 43.31)%

also help avoid this problem.

4.1 Changing RELU to Tanh

RELU has no upper bound for the hidden variables which may lead to pretty large feedback. It's might be especially unstable when using random feedback.This section explore whether the tanh can help stablize asymmetric backpropagation.

All the experiments are done with the same setting of the previous section, only changing the nonlinearity.No hinge loss is used.

From the result we can see that using Tanh for nonlinearity is inferior to using RELU, however tanh is bound so that it can help stablize the random feedback in training. Using tanh is far more stable when using the random feedback in more than one layers.

Table 7: Accuracy on test set for Poggio's net, with different number of random feedback layers

PG0	72.14%
PGR1	73.72%
PGR2	70.10 %
PGR1-U	73.35%
PGR2-U	74.01%

5 Experiments on Untied Convolution

Here untied convolution means that for each receptive kernel in a single filter, the weight is independent. The weight sharing rule efficiently reduced the number of parameters, but this time we give up this advantage because we think this is biologically impossible. In the original convolution, the receptive squares share the same weight, which is believed to be not biological plausible because cells in one position on the retina will by no means share weights(information) with cells in other locations. The detailed implementation is to flatten the data and do convolution by an extremely large matrix dot operation. The for CIFAR-10 example, one single image is fattened to be 32*32*3 = 3072 vector. The first 1024 entry for red and second for green, third for blue. The matrix is deliberately designed that most entries is 0: the number of row for the weight matrix is the same as the length of data vector. Each column stands for a single "feature pixel" in the output. For example if we do not use padding, a 5*5 filter size will yield a size 32-5+1 = 28 feature map, and we assume there is 16 output channel. The weight matrix will have 28*28*16 column. Each 28*28 column stands for one output channel. The flattened output is reshaped when feeding into the next layer. The following figure 3. shows a toy example of untied convolution on the 3*3 image with a filter kernel size 2*2, and 2 output channels. the output tensor is 2*2*2.

It has been tested that using the same weight(weight should be flipped due to the fouier transformation), the result of my matrix dot and convolution is the same (there is some numerical issue so the error is 1e-7)

5.1 Untied convolution with random feedback only in the dense layer

Now testing the untied convolution on CIFAR-10, with Adam for 200 epoch. Adam is essential because the feedback signal for a single element of the weight is extremely small. In the original convolution the weight is shared for each location, which means that the feedback is the sum of a lot of locations. In the untied convolution we have to augment the signal or the training can hardly move during training. Adam is great because the signal magnitude do not matter, so we avoid this problem implicitly. The

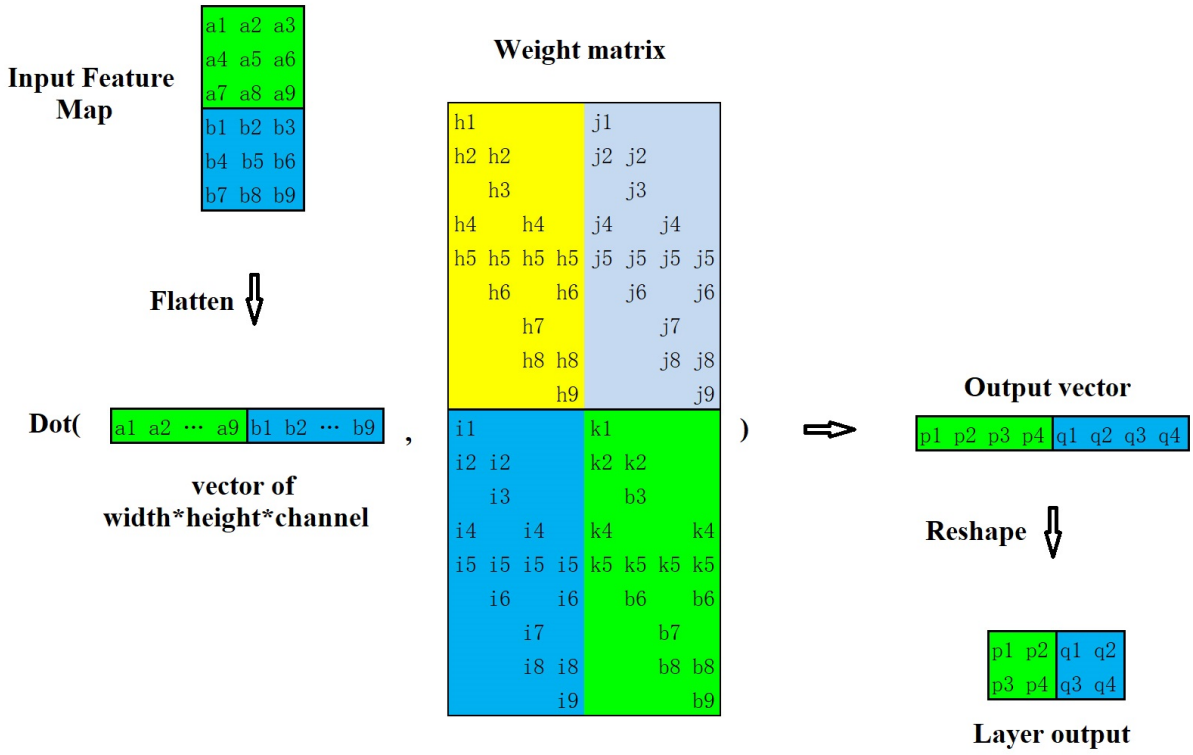

Figure 3: Toy example for untied convolution

performance for the untied convolutional network, trained from scratch with possible random feedback, is shown below.

Table 8: Model Specification, with Softmax and Crossentropy/ Hinge Loss

PG0	PGR1	PGR2	PGR1-U	PGR2-U
UntieConv5*5/32	UntieConv5*5/32	UntieConv5*5/32	UntieConv5*5/32	UntieConv5*5/32
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	Conv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Dense128	Dense128(R)	Dense128(R)	Dense128(R*)	Dense128(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Softmax	Softmax	Softmax	Softmax	Softmax
CE	CE	CE	CE	CE

Table 9: Accuracy on test set for Poggio's net

PG-Untie	64.90%
PG-Untie-R1	50.53%
PG-Untie-R2	38.70%
PG-Untie-R1-U	60.87%
PG-Untie-R2-U	34.61%

The untied convolution has a huge amout number of parameters that overfitting is very severe. It can be easily seen from the discrepancy between training loss and test loss(not shown in this summary); Updating the random feedback helps in some situation when using Relu as nonlinearity.

5.2 Untied Convolution with Tanh

Since using tanh as nonlinearity is more stable for asymmetric backpropagation, the result below changed all the nonlinearity(except softmax) into tanh. Others stay the same.

Table 10: Accuracy on test set for Poggio's net and untied convolution.

PG-Untie	68.65%
PG-Untie-R1	65.93%
PG-Untie-R2	60.44%
PG-Untie-R1-U	65.45%
PG-Untie-R2-U	64.12%

It can be seen that using untied convolution is still suffered from overfitting with tanh. However, for the random feedback using tanh is more stable.

Experiments with dropout are done selectively, with 20% dropout things do not change much, applying 50% dropout make training error very high.

5.3 Untied convolution with random feedback in both dense layer and conv layer

Based on experiments of asymmetric untied convolution, it seems that training asymmetric untied layers from scratch is not practical. The loss function become intractible during training. Changing from RELU to Tanh can resolve this problem. all the experiments below use untied convolution in all convolutional layers, with tanh as the nonlinearity.

Due to the inefficient memory usage in untied convolution, when using random feedback, we have to store one more matrix of the same size of the weight. So using more than one random feedback in the convolution caused out of memory error.

The only possible experiment is using random feedback in the last untied convolutional layer. Also using untied convolution in all the dense layer. The trsPGR3-Untie 62.59

6 Using untied conv and fixing the first layer

The first few layers of convolution is often regarded as a very good feature representation, for example edges of color patterns. however the untied convolution is more flexible in detection due to the relaxation of weight sharing rule. One possible way to improve classification is to combine lower level features from the first a few layers and using untied conv layers on the top. Experiments are done with fixing the first layer which is learned from the poggio's net.

Since previous experiments found that Tanh is more stable for asymmetric feedback, the following results are obtained from tanh nonlinearity.

6.1 Accuracy on CIFAR-10

It can be seen that when fixing the first layer to be well trained "low level feature detector", the error is lower than traininig from scratch, which gives sign that untied convolution's inferior performance is due to the failure of learning good feature representations in the low level. Random feedback do not make things worse, and updating the random feedback increased accuracy by 4%. However there is no sign of large improvement between updating the random feedback and not updating it.

6.2 Accuarcy on CIFAR-100

Now change the dataset to CIFAR-100, which has 100 classes to classify. Typical shallow network achieve accuarcy of around 40%. In this section the same network is used as previous layer and the only change is that the number of output in the softmax layer becomes 100.

Table 11: Model Specification, with Softmax and Crossentropy/ Hinge Loss

PG0	PG-Untie-R2	PG-Untie-R3	PG-Untie-R2-U	PG-Untie-R2-U
FixConv5*5/32	FixConv5*5/32	FixConv5*5/32	FixConv5*5/32	FixConv5*5/32
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64(R)	UntieConv5*5/64	Conv5*5/64(R)
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Dense128	Dense128(R)	Dense128(R)	Dense128(R*)	Dense128(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Softmax	Softmax	Softmax	Softmax	Softmax
CE	CE	CE	CE	CE

Table 12: Model Specification, with Softmax and Crossentropy/ Hinge Loss

PG0	PG-Untie-R2	PG-Untie-R3	PG-Untie-R2-U	PG-Untie-R2-U
FixConv5*5/32	FixConv5*5/32	FixConv5*5/32	FixConv5*5/32	FixConv5*5/32
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
UntieConv5*5/64	UntieConv5*5/64	UntieConv5*5/64(R)	UntieConv5*5/64	Conv5*5/64(R)
Pool3*3	Pool3*3	Pool3*3	Pool3*3	Pool3*3
Dense128	Dense128(R)	Dense128(R)	Dense128(R*)	Dense128(R*)
Dense10	Dense10(R)	Dense10(R)	Dense10(R*)	Dense10(R*)
Identity	Identity	Identity	Identity	Identity
HL	HL	HL	HL	HL

Table 13: Accuracy on test set for Poggio's net, fixing first layer as well trained convolutional layer, using tanh

PG-Untie	73.78%	PG0-Untie-HL	72.40%
PG-Untie-R2	72.81	PG-Untie-R2 -HL	71.32%
PG-Untie-R3	72.14%	PG-Untie-R3-HL	70.62%
PG-Untie-R2-U	73.58%	PG-Untie-R3-HL-U	71.75%
PG-Untie-R3-U	73.45%	PG-Untie-R3-HL-U	70.68%

From the results, the untied convolution and asymmetric backpropagation can still achieve similar result

Table 14: Accuracy on test set for Poggio's net, fixing first layer as well trained convolutional layer, using tanh

PG	40.17%	PG-HL	39.20%
PG-Untie	40.87%	PG0-Untie-HL	34.36%
PG-Untie-R2	39.52%	PG-Untie-R2 -HL	34.43%
PG-Untie-R3	38.77%	PG-Untie-R3-HL	33.06%
PG-Untie-R2-U	39.52%	PG-Untie-R2-HL-U	37.07%
PG-Untie-R3-U	38.41%	PG-Untie-R3-HL-U	37.47%

as the original network. The hinge loss become less favorable.

7 Regularized Untied Convolution With first layer fixed

Since the untied convolution contains too many parameters, which make over fitting severe, some kind of regularization is needed. Two commonly used regularization method is dropout and L1/L2 penalty. In this section some results involving applying dropout and L2 regularization on the weight of the untied convolution weight. For simplicity, only softmax with crossentropy is used.

To test the performance for applying regularization, L2 loss is applied to all the untied convolution layers on PG-Untie network in previous section, ie I fixed the first layer as well trained original convolution, and apply regularization on the second and third layer.One asymetric backpropagation is used. Under a mild regularization($\lambda = 1e-4$) the accuarcy on CIFAR-10 is 68.80%, which is inferior to the non-regularized one. Applying greater regularization ruin the training.

Applying drop out directly after the convolutional layer has similar effect on the accuarcy to the L2 regularization. When 10% drop out is applied, the accuarcy is roughly 72%, the same as non dropout network. Applying 50% of dropout ruin the traininig.

Hinge loss at CIFAR100 table 7 Regularized untie summarize lilicarp and poggio draw better model spec. weight clip