

The Forward-Forward Algorithm: Some Preliminary Investigations

Geoffrey Hinton

Google Brain

Content

1. Abstract
2. What is wrong with backpropagation
3. The Forward-Forward Algorithm
4. Some experiments with FF
5. Experiments with CIFAR-10
6. Sleep
7. ~~How FF relates to other contrastive learning techniques~~
8. Learning fast and slow
9. The relevance of FF to analog hardware
10. Motral Computation
11. Future work

Abstract

- Introduce a new learning procedure for neural networks
- Demonstrate that it works well enough on a few small problems
- Forward-Forward 알고리즘 소개
 - Forward and backward passes (backpropagation) 를 두 개의 forward pass 로 대체
 - One with positive data and the other with negative data
 - 각 layer 는 각 objective function 을 가짐
 - Objective function 은 positive data 가 high goodness 를 가지도록, negative data 가 low goodness 를 가지도록 함

What is wrong with backpropagation

- As a model of how cortex learns, backpropagation (BP) remains implausible
 - 실제 뉴런이 backpropagation 방식처럼 작동하는지는 알 수 없음
 - Visual pathway 는 bottom-up connections 을 가지지 않음
- BP through time as a way of learning sequences is especially implausible
 - 인간의 뇌는 중단되는 시간이 따로 없이 sensory processing stage 를 통해서 sensory data 를 전달할 필요가 있고, 그 순간마다 learning 이 될 수 있어야 함
- BP requires perfect knowledge of the computation performed in the forward pass
 - forward 계산의 정확한 knowledge 가 필요. 즉, 미분 불가능한 black-box 에 대해 forward 한다면 BP 못함
- 이에 대한 대안으로 Reinforcement learning (RL) 이 있지만, RL 은 high variance 를 가진다는 문제가 있음

What is wrong with backpropagation

- The Forward-Forward algorithm (FF)
 - It can be used when the precise details of the forward computation are unknown
 - It can learn while pipelining sequential data through a neural network without ever storing the neural activities or stopping to propagate error derivatives
 - 즉, neural activities 를 저장하지 않거나, error derivatives 를 propagate 하지 않고도 sequential data 를 pipelining 하면서 학습 가능
- 한계점
 - 어떤 때에는 BP 보다 느림
 - Generalized 되지 않았음
- 장점
 - Cortex 에서의 모델 학습 (실제 뇌에서 학습하는 방법과 유사)
 - Very low-power analog hardware

The Forward-Forward Algorithm

- The idea is to replace the forward and backward passes of backpropagation by **two forward passes the operate in exactly the same way as each other**
 - On different data and with opposite objectives
- Positive pass
 - **Real data**
 - 매 hidden layer 마다의 **goodness** 를 증가시키도록 wieght 학습
- Negative pass
 - **Negative data**
 - 매 hidden layer 마다의 goodness 를 감소시키도록 학습
- Real data 의 goodness 는 threshold 보다 높게,
negative data 의 goodness 는 threshold 보다 낮게하는 것이 목표

The Forward-Forward Algorithm

- **Real data**

- 실제 데이터

- **Negative data**

- 임의로 생성한 데이터 (e.g. random image, 틀린 label, hybrid image ..)

- **Goodness**

- 매 layer 마다 측정
- 다양한 function 으로 계산할 수 있겠지만, 여기서는 2가지 방법 사용해서 실험
- Sum of the squared neural activities
- Negative sum of the squared neural activities

probability that an input vector is positive (*i. e.* real) is given by applying the logistic function, σ to the goodness, minus some threshold, θ

$$p(\text{positive}) = \sigma \left(\sum_j y_j^2 - \theta \right) \quad (1)$$

where y_j is the activity of hidden unit j before layer normalization. The negative data may be predicted by the neural net using top-down connections, or it may be supplied externally.



Real data

hybrid



Negative data

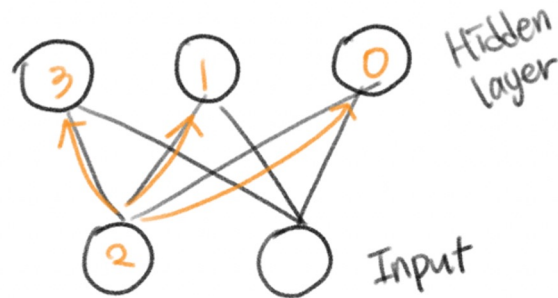
The Forward-Forward Algorithm

probability that an input vector is positive (*i. e.* real) is given by applying the logistic function, σ to the goodness, minus some threshold, θ

$$p(\text{positive}) = \sigma \left(\sum_j y_j^2 - \theta \right) \quad (1)$$

where y_j is the activity of hidden unit j before layer normalization. The negative data may be predicted by the neural net using top-down connections, or it may be supplied externally.

- The positive pass operates on real data and adjusts the weights to increase the goodness in every hidden layer
- The negative pass operates on “negative data” and adjusts the weights to decrease the goodness in every hidden layer



$$p(\text{positive}) = \sigma \left(\sum_j y_j^2 - \theta \right)$$

$$\sigma \left(3^2 + 1^2 + 0^2 - 2 \right) = \sigma(8) \approx 1$$

* negative 의 경우 ≈ 0 이 된다.

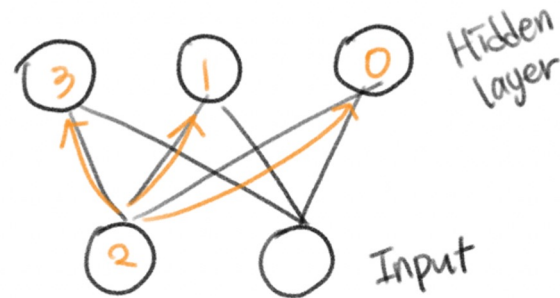
The Forward-Forward Algorithm

probability that an input vector is positive (i. e. real) is given by applying the logistic function, σ to the goodness, minus some threshold, θ

$$p(\text{positive}) = \sigma \left(\sum_j y_j^2 - \theta \right) \quad (1)$$

where y_j is the activity of hidden unit j before layer normalization. The negative data may be predicted by the neural net using top-down connections, or it may be supplied externally.

- Probability = Input data 가 positive 일 확률
- 그렇기 때문에 positive data 는 theta 보다 커야
logistic function 을 태웠을 때 1에 가까워질 수 있음



$$p(\text{positive}) = \sigma \left(\sum_j y_j^2 - \theta \right)$$

$$\sigma \left(3^2 + 1^2 + 0^2 - 2 \right) = \sigma(8) \approx 1$$

* negative 의 경우 ≈ 0 이 되겠다.

The Forward-Forward Algorithm

- 주의 - 공식 코드 아님 (https://github.com/pytorch/examples/tree/main/mnist_forward_forward)

```
31
32 class Net(torch.nn.Module):
33     def __init__(self, dims):
34
35         super().__init__()
36         self.layers = []
37         for d in range(len(dims) - 1):
38             self.layers = self.layers + [Layer(dims[d], dims[d + 1]).to(device)]
39
40     def predict(self, x):
41         goodness_per_label = []
42         for label in range(10):
43             h = overlay_y_on_x(x, label)
44             goodness = []
45             for layer in self.layers:
46                 h = layer(h)
47                 goodness = goodness + [h.pow(2).mean(1)]
48             goodness_per_label += [sum(goodness).unsqueeze(1)]
49         goodness_per_label = torch.cat(goodness_per_label, 1)
50         return goodness_per_label.argmax(1)
51
52     def train(self, x_pos, x_neg):
53         h_pos, h_neg = x_pos, x_neg
54         for i, layer in enumerate(self.layers):
55             print("training layer: ", i)
56             h_pos, h_neg = layer.train(h_pos, h_neg)
57
```

```
58
59 class Layer(nn.Linear):
60     def __init__(self, in_features, out_features, bias=True, device=None, dtype=None):
61         super().__init__(in_features, out_features, bias, device, dtype)
62         self.relu = torch.nn.ReLU()
63         self.opt = Adam(self.parameters(), lr=args.lr)
64         self.threshold = args.threshold
65         self.num_epochs = args.epochs
66
67     def forward(self, x):
68         x_direction = x / (x.norm(2, 1, keepdim=True) + 1e-4)
69         return self.relu(torch.mm(x_direction, self.weight.T) + self.bias.unsqueeze(0))
70
71     def train(self, x_pos, x_neg):
72         for i in range(self.num_epochs):
73             g_pos = self.forward(x_pos).pow(2).mean(1)
74             g_neg = self.forward(x_neg).pow(2).mean(1)
75             loss = torch.log(
76                 1
77                 + torch.exp(
78                     torch.cat([-g_pos + self.threshold, g_neg - self.threshold])
79                 )
80             ).mean()
81             self.opt.zero_grad()
82             loss.backward()
83             self.opt.step()
84             if i % args.log_interval == 0:
85                 print("Loss: ", loss.item())
86         return self.forward(x_pos).detach(), self.forward(x_neg).detach()
87
```

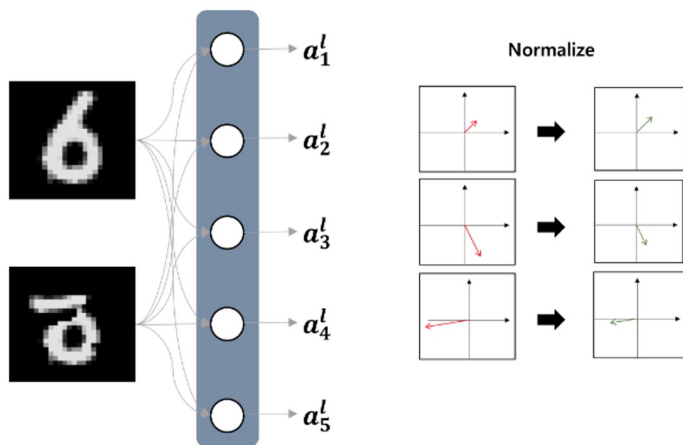
The Forward-Forward Algorithm

Learning multiple layers of representation with a simple layer-wise goodness function

- 첫 번째 hidden layer 의 activities 값이 두 번째 hidden layer 의 input 으로 사용된다면, 이후 layer 의 학습은 positive, negative data 를 구분하는 것이 쉬워짐
- 더 이상 학습할 feature 가 없어짐
- 이를 방지하기 위해서, FF normalizes the length of the hidden vector before using it as input to the next layer
- 이로써, goodness 를 결정하는데 필요한 information 이 사라지고, 두 번째 layer 에서는 첫 번째 layer 에서의 relative activities 만을 사용하도록 함
 - 첫 번째 layer 의 **activity vector** 는 **length** 와 **orientation** 을 가지고 있는데, length 는 goodness 를 정의하는데 사용됨. 따라서 orientation 만이 다음 layer 로 넘어가는 것임
 - Relative activities 는 layer-normalization 에 영향을 받지 않아서 살아있음

The Forward-Forward Algorithm

Learning multiple layers of representation with a simple layer-wise goodness function



$$\hat{a}^l = \frac{a^l}{\|a^l\|_2}$$

$$a^l = (a_1^l, a_2^l, a_3^l, a_4^l, a_5^l)$$

Some experiments with FF

The backpropagation baseline

- MNIST dataset
- CNN with a few hidden layers typically get about 0.6% test error
- Permutation-invariant, FFNN with a few fully connected hidden layers of ReLUs typically get about 1.4% test error
 - Permutation-invariant
 - 입력 벡터 요소의 순서와 상관없이 같은 출력을 생성하는 모델. 대표적인 모델로는 MLP
 - permutation-invariant task 가 아닌 모델로는 입력 이미지의 픽셀의 순서를 고려하는 CNN 이 있음

Some experiments with FF

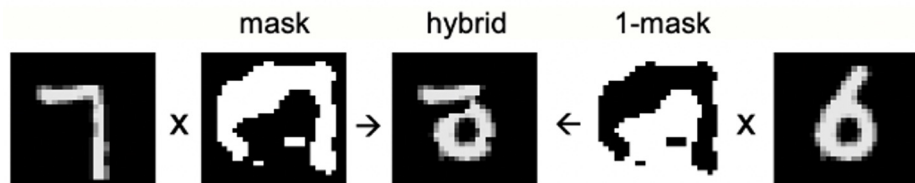


Figure 1: A hybrid image used as negative data

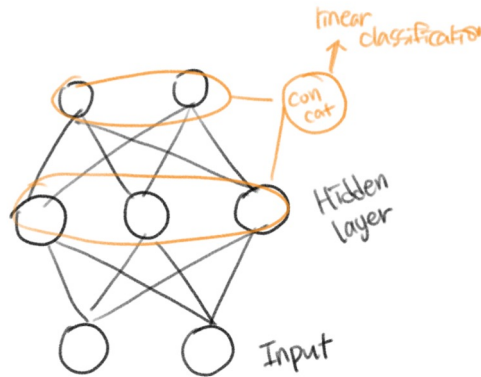
A simple **unsupervised** example of FF

- There are two main questions about FF
 1. If we have a good source of negative data, does it learn effective multi-layer representations that capture the structure in the data?
 2. Where does the negative data come from?
- Negative data 생성
 - **Very different long range correlations** but **very similar short range correlations**
 - Masks can be created by starting with a random bit image
 - Repeatedly blurring the image with a filter of the form $[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$ in both the horizontal and vertical directions

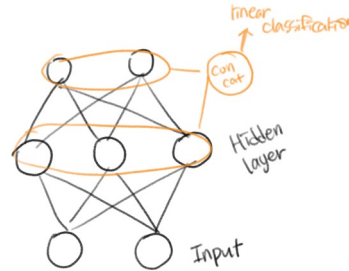
Some experiments with FF

A simple **unsupervised** example of FF

- Forward-Forward 방식으로 positive data, negative data 학습 후
- Linear classifier 학습
- 좀 더 구체적으로..
 - contrastive learning 에서 supervised learning task 를 학습하는 방식
 - Input vector 를 representation vectors 로 학습시키고 (without using any information about the labels)
 - 다음으로 learn a simple linear transformation of these representation vectors into vectors of logits which are used in a softmax to determine a probability distribution over labels
- **The learning of the linear transformation to the logits is supervised, but does not involve learning any hidden layers so it does not require backpropagation of derivatives**



Some experiments with FF



A simple **unsupervised** example of FF

- Training a network with four hidden layers of 2000 ReLUs each for 100 epochs
- 1.37% Test error rate, use the normalised activity vectors of the **last three hidden layers** as the inputs to a softmax
 - Using the first hidden layer as part of the input to the linear classifier makes the test performance worse
- Fully connected layers 대신 local receptive fields (without weight-sharing) 을 사용했을 때 성능 향상

⁸The first hidden layer used a 4x4 grid of locations with a stride of 6, a receptive field of 10x10 pixels and 128 channels at each location. The second hidden layer used a 3x3 grid with 220 channels at each grid point. The receptive field was all the channels in a square of 4 adjacent grid points in the layer below. The third hidden layer used a 2x2 grid with 512 channels and, again, the receptive field was all the channels in a square of 4 adjacent grid points in the layer below. This architecture has approximately 2000 hidden units per layer.

Some experiments with FF

A simple supervised example of FF

- One task and want to use a small model that does not have the capacity to model the full distribution of the input data, use supervised learning
- **FF is to include the label in the input**
 - **The positive data consists of an image with the correct label and the negative data consists of an image with the incorrect label.**
- Positive 데이터와 negative 데이터의 차이가 label 밖에 없기 때문에, FF should ignore all features of the image that do not correlate with the label

Some experiments with FF

A simple supervised example of FF

- 데이터 생성
 - Positive data 는 처음 10개 pixel 에 correct label (one-hot) 포함
 - Negative data 는 처음 10개 pixel 에 incorrect label (one-hot) 포함
- Test 수행 (2가지 방법)
 - One-pass softmax
 - 10개 digit 을 모두 0.1로 하여 테스트 데이터 입력
 - 첫 번째 hidden layer 를 제외한 모든 hidden activities 를 softmax input 으로 사용
 - Compute goodness for every label
 - 특정 하나의 label 로 테스트 데이터를 입력하고, goodness 합 측정
 - Accumulate the goodnesses of all but the first hidden layer
 - 모든 클래스에 대해 수행한 후, 가장 큰 goodness 를 가진 클래스로 추정

Some experiments with FF

A simple supervised example of FF

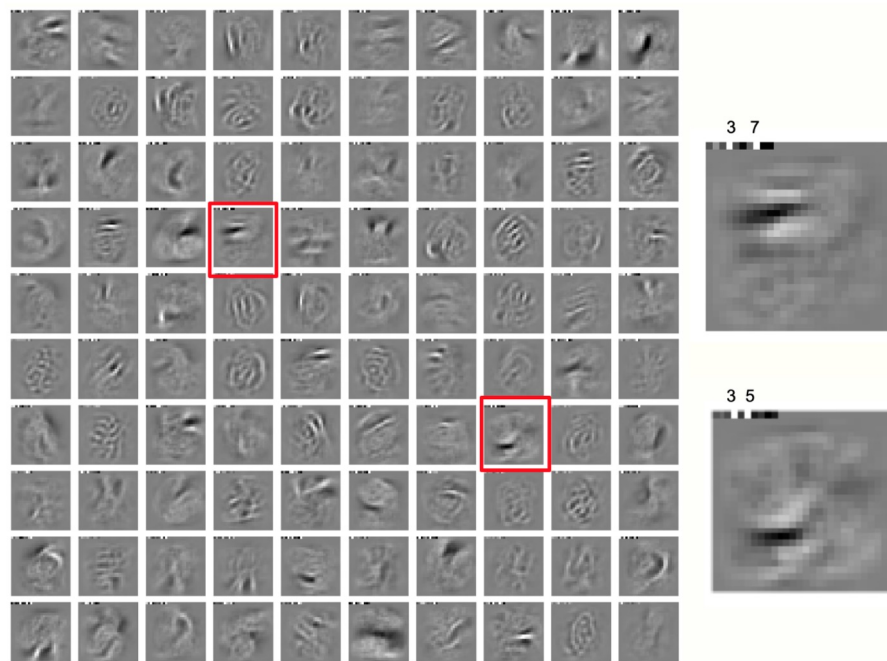


Figure 2: The receptive fields of 100 neurons in the first hidden layer of the network trained on jittered MNIST. The class labels are represented in the first 10 pixels of each image.

Some experiments with FF

Using FF to model top-down effects in perception

- All of the examples of image classification so far have used feed-forward neural networks that were learned greedily one layer at a time
- FF 는 later layers 가 earlier layers 에 영향을 줄 수 없는 구조이고, BP 와 비교했을 때 큰 약점이 됨
- 이를 위해 **Image** 를 동일한 프레임의 비디오로 간주하고, **multi-layer RNN** 구조를 사용함

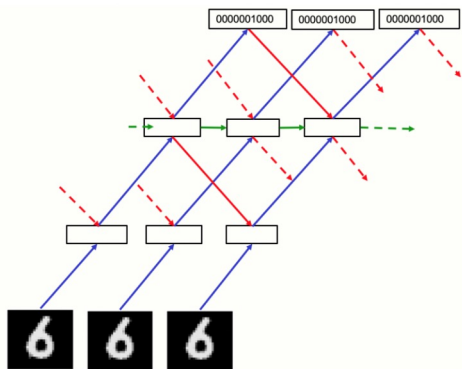


Figure 3: The recurrent network used to process video.

- 각 layer 에 있는 activity vector 는 previous time-step 에 있는 layer above, layer below 에 모두 영향을 받음
- Bottom layer 는 pixel image 이고, curve, edge 등의 정보 포함
- Top layer 는 digit class 이고, 전체 이미지 정보 포함
- 약간의 성능 향상이 있었기 때문에 논문의 실험은 synchronous updates 사용
 - 0.3 of the previous pre-normalized state + 0.7 of the computed new state

Some experiments with FF

Using FF to model top-down effects in perception

- MINST for 60 epochs
- 8 synchronous iterations with damping
- Picking the label that has the highest goodness averaged over iterations 3 to 5
- Test error 1.31%
- Negative data is generated by doing a single forward pass through the net to get probabilities for all the classes and then choosing between the incorrect classes in proportion to their probabilities
 - 이렇게하면 훨씬 효과적으로 학습할 수 있다.

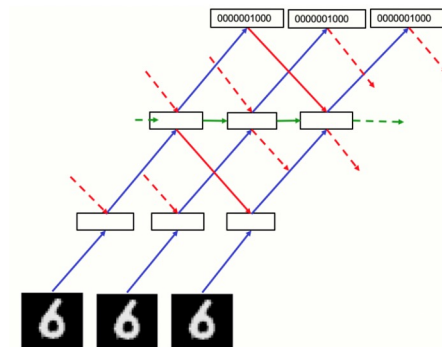


Figure 3: The recurrent network used to process video.

Some experiments with FF

Using predictions from the spatial context as a teacher

- Recurrent net 에서
 - Positive 데이터는 input from the layer above, input from the layer below 의 agreement 가 잘 일치함
 - Negative 데이터는 agreement 가 잘 일치하지 않음
- The top-down input will be determined by a larger region of the image
- Bottom-up input is based on a more local region of the image

Experiments with CIFAR-10

- FF 가 weight-sharing 이 불가능하기 때문에, local receptive field 를 사용한 BP 와 성능 비교하였음
- Network
 - Non-convolutional nets with local receptive fields of size 11×11 and 2 or 3 hidden layers
- One version is trained to maximize the sum of the squared activities on positive cases
- The other version is minimize it (This gives slightly better test performance)

learning procedure	testing procedure	number of hidden layers	training % error rate	test % error rate
BP		2	0	37
FF min ssq	compute goodness for every label	2	20	41
FF min ssq	one-pass softmax	2	31	45
FF max ssq	compute goodness for every label	2	25	44
FF max ssq	one-pass softmax	2	33	46

BP		3	2	39
FF min ssq	compute goodness for every label	3	24	41
FF min ssq	one-pass softmax	3	32	44
FF max ssq	compute goodness for every label	3	21	44
FF max ssq	one-pass softmax	3	31	46

Sleep

- FF 는 brain 에서 구현하기 훨씬 수월함
 - The positive data was processed when awake
 - The negative data was created by the network itself and processed during sleep
- 이전 draft 에서 positive data 먼저 계산하고, negative data 가 이어서 계산하는 방식을 실험했음
- 이 때, predicting the next character in a sequence from the previous ten character window 실험을 진행했는데, 어떤 이유에서인지 재현이 안됨 (버그라고 추정) -> 그래서 현재 draft 에서는 해당 내용이 없음
- Sum of squared activities 를 goodness function 으로 사용하는 것은 lr 이 매우 작고 momentum 이 매우 클 때만 작동함
- Positive, negative 과정 분리해서 사용할 수 있는 다른 goodness function 적용도 가능함
- 그리고 만약 positive, negative 과정을 분리할 수 있다면 negative 업데이트 과정을 한동안 제거하고, 실제 수면 부족일 때 발생하는 현상을 볼 수 있음

How FF relates to other contrastive learning techniques

- Relationship to Boltzmann Machines
- Relationship to Generative Adversarial Networks
- Relationship to contrastive methods that compare representations of two different image crops
- A problem with stacked contrastive learning

Learning fast and slow

- The vector of increments of the incoming weights for hidden neuron j is given by :
 - 만약 layer 가 fully connected 되어있다고 생각하면, weight update 는 input x 에 대해서 다음

$$\Delta \mathbf{w}_j = 2\epsilon \frac{\partial \log(p)}{\partial \sum_j y_j^2} y_j \mathbf{x} \quad (3)$$

where y_j is the activity of the ReLU before layer normalization, \mathbf{w}_j is the vector of incoming weights of neuron j and ϵ is the learning rate.

- $y_{\{j\}}$ 는 layer norm 이전의 ReLU 노드의 activation 값
- $w_{\{j\}}$ 는 weight 에서 뉴런 j 에 대한 벡터
- The only term that depends on j in the change of activity caused by the weight update is $y_{\{j\}}$, so all the hidden activities change by the same proportion and the **weight update does not change the orientation of the activity vector**

Learning fast and slow

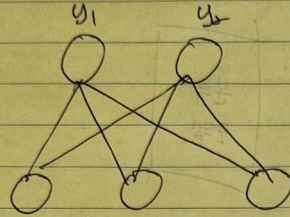
- 즉, weight 파라미터가 업데이트 되는 상황이 layer normalize 결과를 변화시키지 않기 때문에 한 번에 여러 레이어를 업데이트 할 수 있다는 장점이 있음
- Input x 를 기준으로 모든 레이어가 desired goodness S^* 을 계산하기 위해서 one step 으로 한 번에 계산이 가능하다는 의미
- Assuming that the input vector and all of the layer-normalized hidden vectors are of length 1, the learning rate that achieves this is given by :

$$\epsilon = \sqrt{\frac{S^*}{S_L}} - 1$$

- 하지만 mini-batch 단위로 학습을 진행하는 구조에서는 이를 사용할 수 없음
- Single training case 에서는 연구할만한 가치가 있다고 함

Learning fast and slow

layer norm.



y_1 y_2

W_j

X (input vector)

- ① x 가 input \rightarrow 두번째 y_1, y_2 나옴.
- ② W_j 학습
(학습된 W_j 에 의한)
- ③ 다시 x input. \rightarrow 두번째 y_1, y_2 .
- ④ 원칙대로라면, ③에서의 y_1, y_2 를 사용해야 함.

그러나,

- ⑤ W_j 가 학습 및 update 되는 과정은
linear 해 때문에, ~~정확~~ layer norm 을
다음

가치면 ① y_1, y_2 와 ③ y_1, y_2 가 동일할 것!

The relevance of FF to analog hardware

- An energy efficient way to multiply an activity vector by a weight matrix is to implement activities as voltages and weights as conductances
 - Activity vector 에 weight 를 곱하는 연산을 energy efficient 한 방법으로 구현하는 것은, **activity 를 voltage (전압) 로, weight 를 conductance (전기가 얼마나 잘 통하느냐 하는 정도를 나타내는 계수; 저항의 역수) 로 보는 것임**
- 하지만 backpropagation 과정을 구현하기엔 아날로그 회로가 부적합하기 때문에 보통은 A-to-D converters (analog-to-digital) 을 거쳐 digital computations 을 사용해왔음
- BP 대신 FF 를 사용하면 A-to-D converters 과정을 제거할 수 있음

Mortal Computation

- **The separation of software from hardware** is one of the foundations of Computer Science and it has many benefits
 - 전기 공학 (electrical engineering) 에 대한 걱정 없이 프로그램의 속성을 연구할 수 있음
 - 프로그램을 한 번 작성하고, 수백만 대의 컴퓨터에 복사하여 사용할 수 있음 (**sw immortality**)
- 만약 **sw immortality** 를 희생해서라도 연산에 필요한 에너지를 절약할 수 있다면?
 - 기존 알고리즘의 한정된 activation 과 네트워크 구조에서 벗어나 훨씬 다양한 variation 을 각 hw 에 적용 가능 -> 동일한 task 에서도 서로 다른 파라미터 셋을 사용하여 최적의 결과 도출 가능
 - 이런 파라미터는 각 하드웨어에 따라 다르기 때문에 대체 불가능하며, **mortal** 함
- 즉, **the computation they perform is mortal : it dies with the hardware**

Mortal Computation

- Hardware 에서 학습한 파라미터 value 를 다른 hardware 에 적용하는 것이 합리적이지 못할수도 있지만, **하나의 hw에서 다른 hw 로 학습한 내용을 transferring 하는 biological 방식 존재**
 - Distillation
 - 그러나, distillation 처럼 output 에 의한 knowledge transfer 은 한계가 분명히 존재함
 - Distillation 은 teacher 가 teacher's internal representations 을 포함하는 highly informative outputs 을 줄 때 성능이 가장 좋음
- 1조 개의 매개변수 신경망이 몇 와트 (watts) 만으로 작동되게 하려면, mortal computation 이 유일한 방법일수도 있음
- Mortal computation 의 실현 가능성은 learning procedure 를 잘 찾는 것에 달려 있으며,
Forward-Forward 알고리즘 역시 하나의 유력한 후보군 중 하나

Future work

- Many open questions
 - Unsupervised learning 에 필요한 negative data 를 잘 생성할 수 있는가?
 - 가장 좋은 goodness function 은 무엇인가?
 - 최근에는 단순히 sum of the unsquared activities on positive data (and maximizing on negative) 방식이 더 좋은 성능을 보였음
 - 가장 좋은 activation function 은 무엇인가?
 - 본 실험에서는 ReLU 만 사용됨
 - Spatial data 에서 FF 는 이미지의 다른 영역에 대해 local goodness function 을 가지는 것이 도움이 되었는가?
 - Sequential data 에서 simplified transformer 처럼 fast weights 를 사용하는 것이 가능한가?
 - Squared activity 값을 최대화하려는 feature detectors 와 최소화하려는 constraint violation detectors 를 사용하는 것이 이득이 되었는가?

Thank You