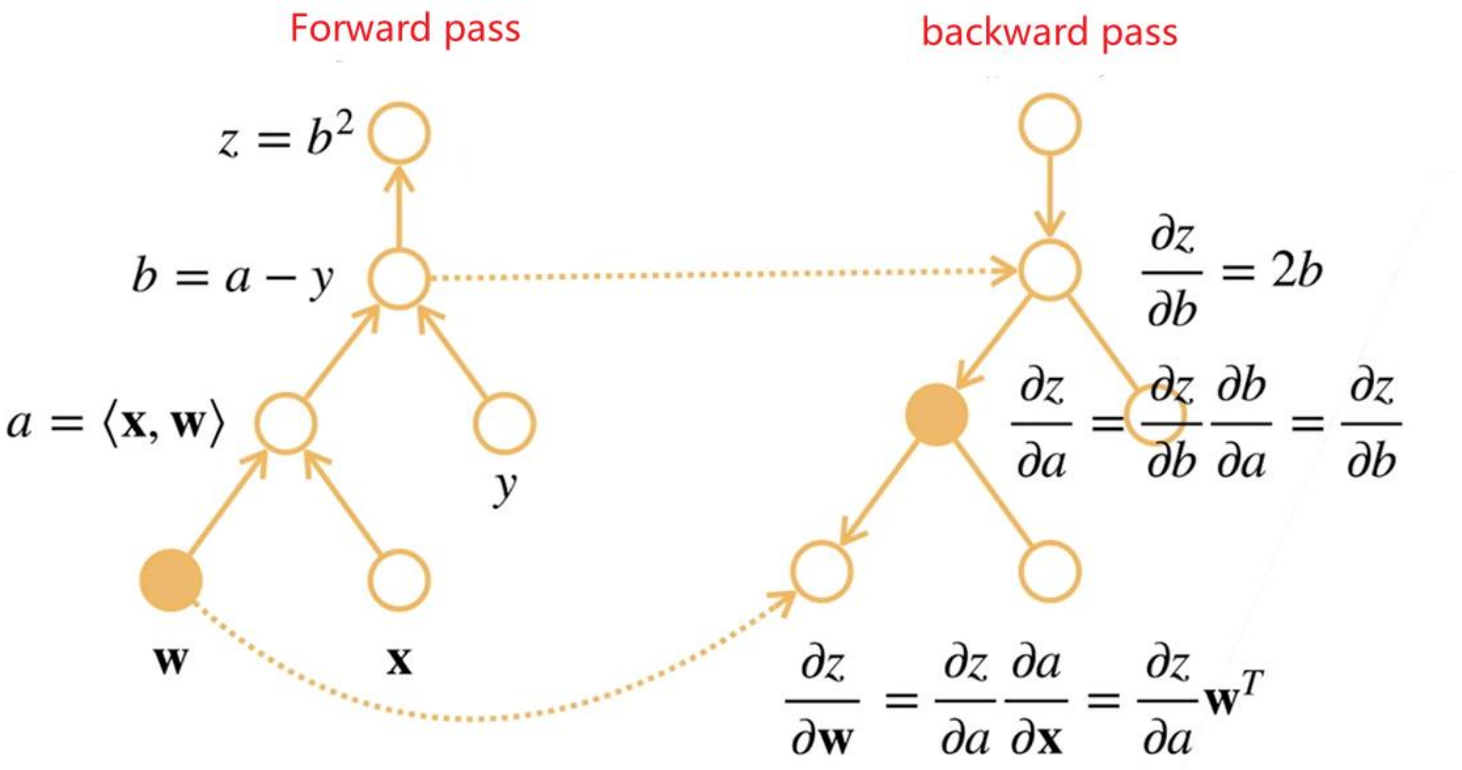
The Forward-Forward Algorithm: Some Preliminary Investigations

Geoffrey Hinton
Google Brain
geoffhinton@google.com

What is wrong with backpropagation

$$z = \left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)^2$$



1. the perceptual system needs to perform inference and learning in real time without stopping to perform backpropagation

2. it requires perfect knowledge of the computation performed in the forward pass in order to compute the correct derivatives. (It is possible to resort to reinforcement learning)

The Forward-Forward Algorithm

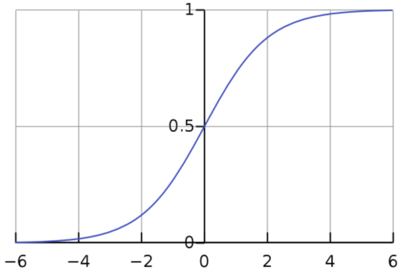
- **Replace the forward and backward passes of backpropagation by two forward passes.**
- **Positive pass** on **real data** adjusts the weight to **increase the goodness** in every hidden layer
- **Negative pass** on **negative data** adjusts the weight to **decrease the goodness** in every hidden layer
- **The aim of the learning** is to make the goodness be well **above** θ for real data and well **below** θ for negative data.

Goodness= $\sum_j y_j^2$

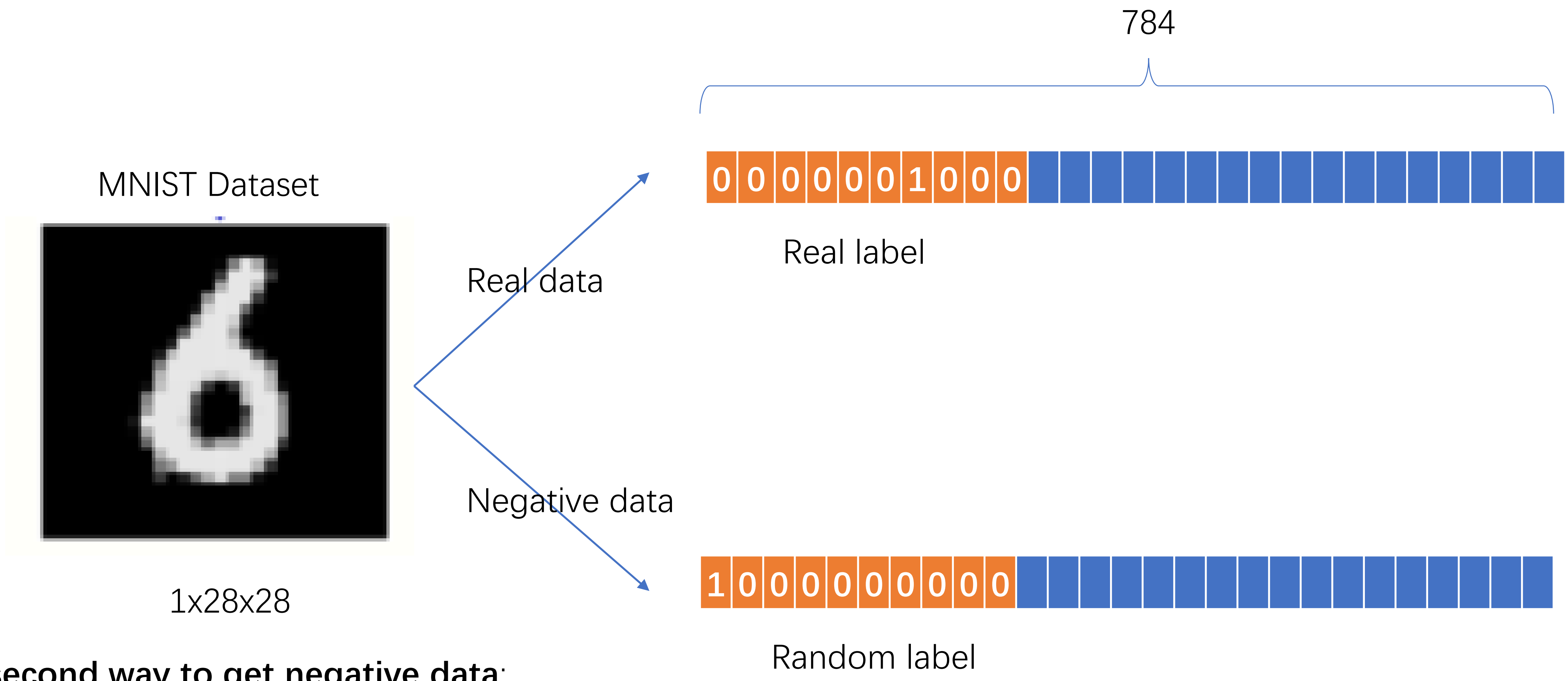
y_j is the activity of hidden unit j

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

θ: Hyperparameter
σ: logistic function



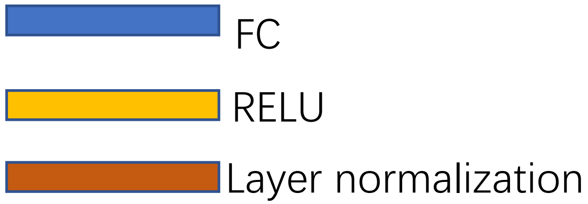
Where does the real & negative data from?



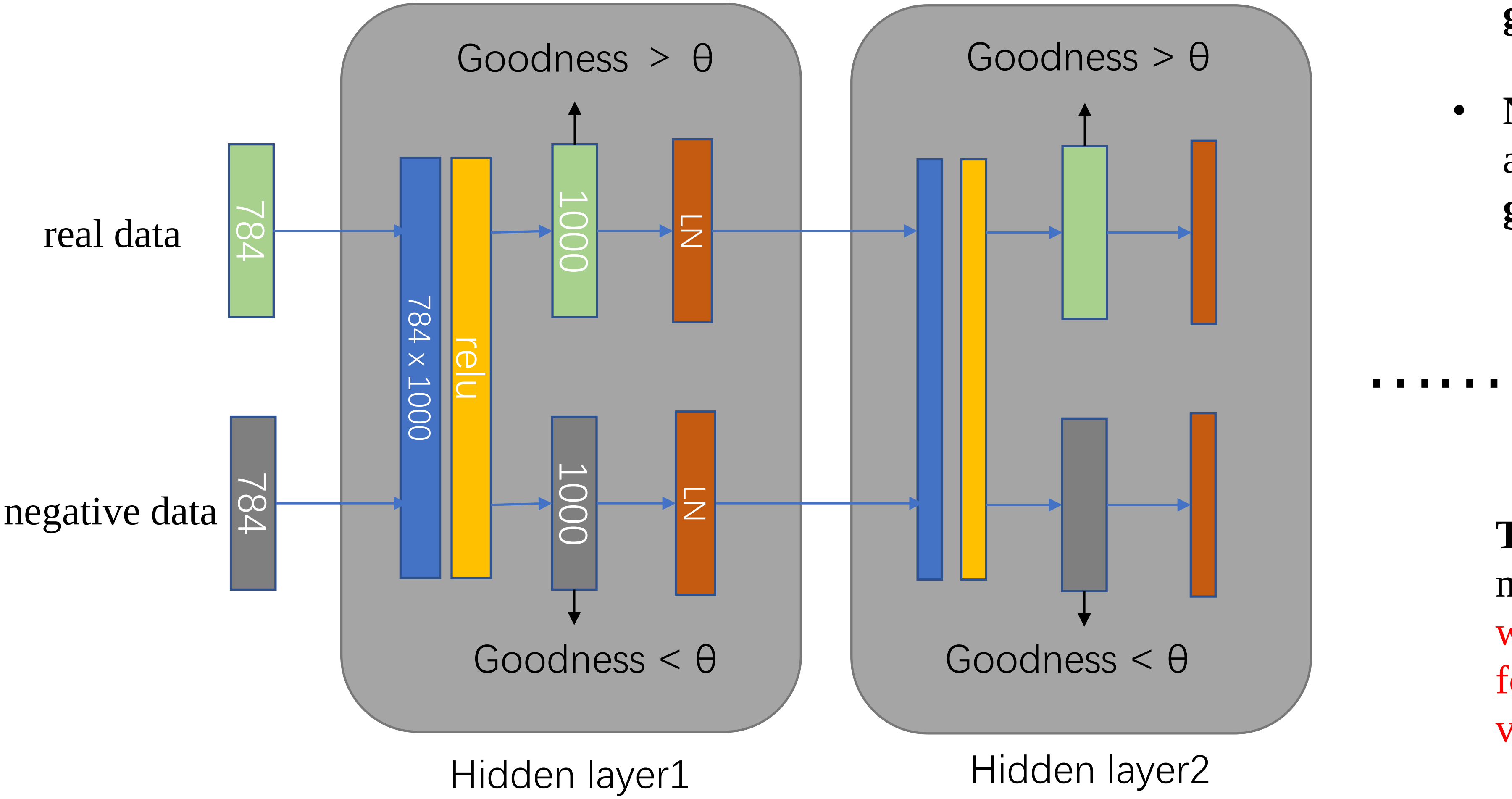
The second way to get negative data:
Negative data is generated by doing a single forward pass through the net to get probabilities for all the classes and then choosing between the incorrect classes in proportion to their probabilities. This makes the training much more efficient.

0.3	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.6**	**0.01**	**0.01**	**0.03**

The Forward-Forward Algorithm



$$\text{Goodness} = \sum_j y_j^2$$

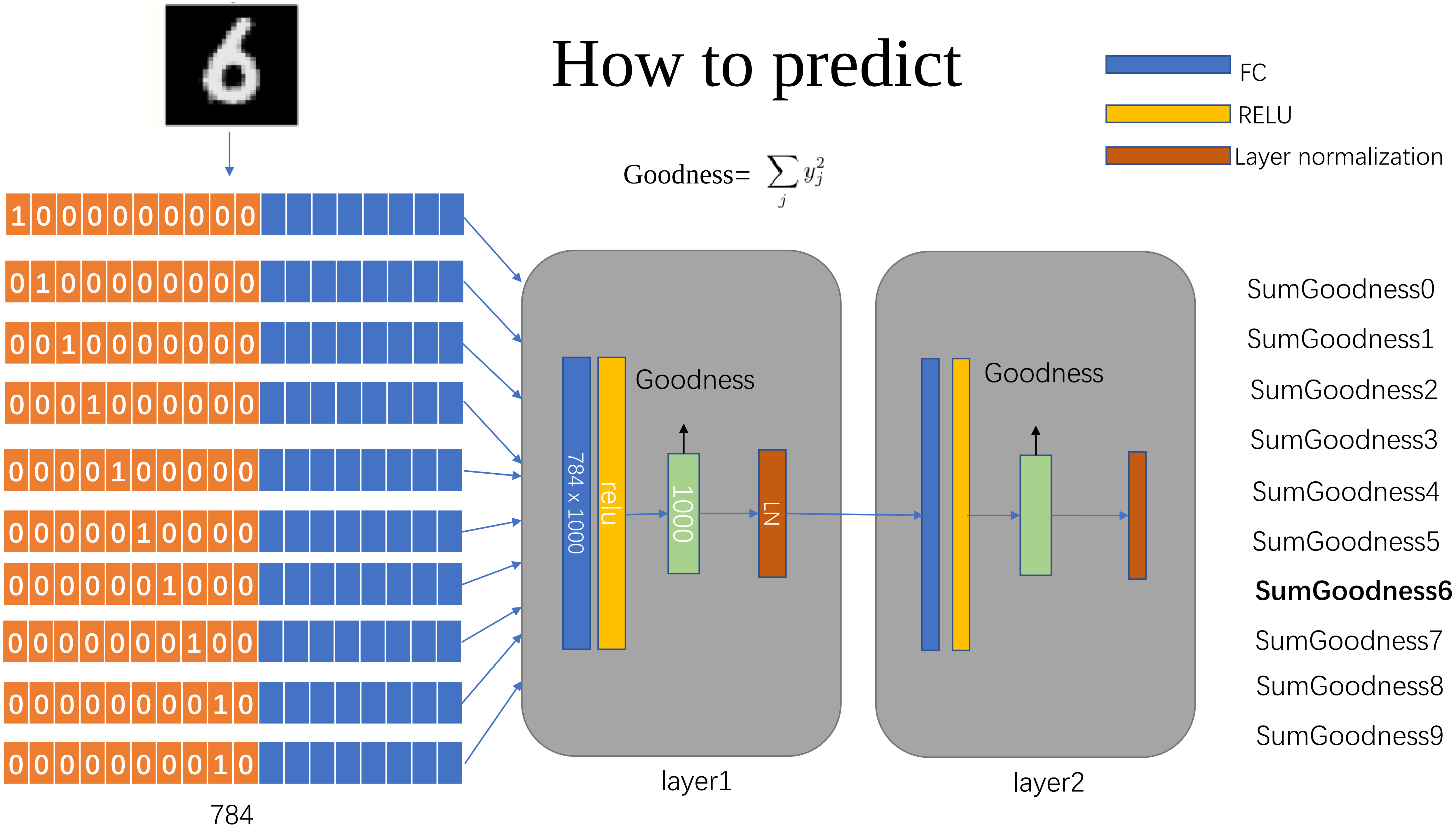


- **Positive pass** on **real data** adjusts the weight to **increase the goodness** in every hidden layer
- **Negative pass** on **negative data** adjusts the weight to **decrease the goodness** in every hidden layer

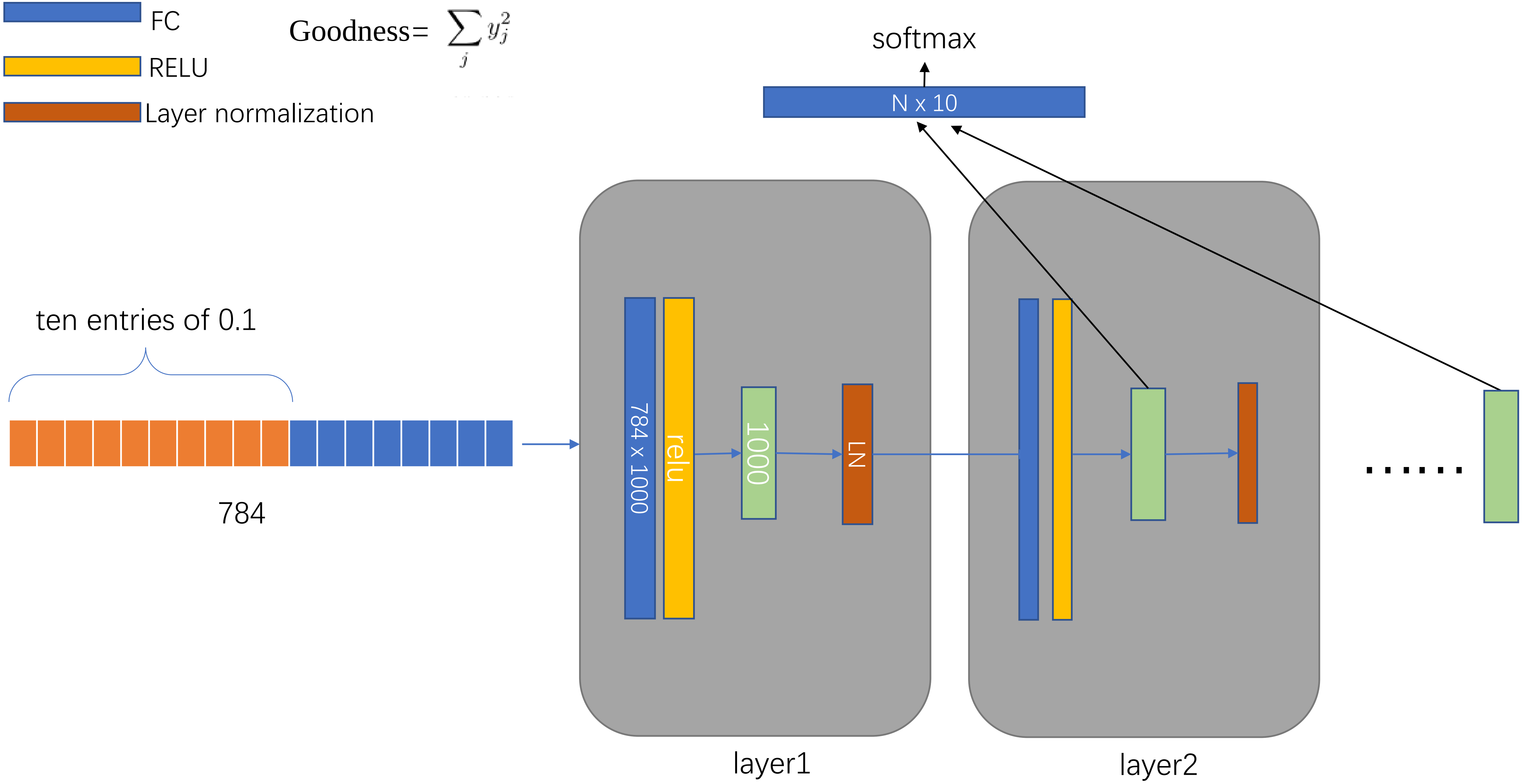
The aim of the learning is to make the goodness be well above some threshold value for real data and well below that value for negative data.

How to predict

Goodness= $\sum_j y_j^2$



Another way to predict(quick but sub-optimal)



The advantage of FF

- It can be used when the precise details of the forward computation are unknow.

- It can learn while pipelining sequential data through a neural network ever without storing the neural activities or stopping to propagate error derivatives.

What is wrong with FF

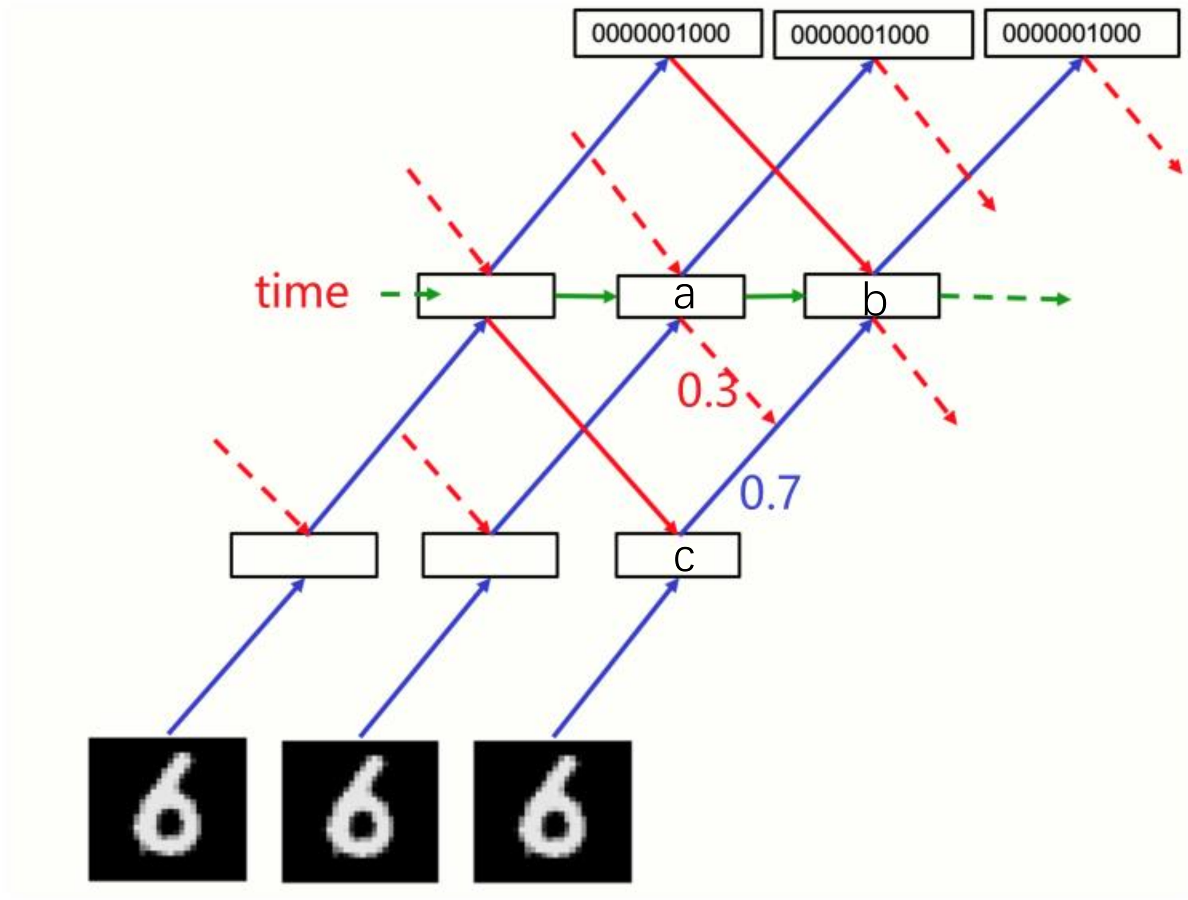
- FF is somewhat slower than backpropagation

- FF does not generalize quite as well as backpropagation

- What is learned in later layers cannot affect what is learned in earlier layers.

How to resolve the third disadvantage?

What is learned in later layers cannot affect what is learned in earlier layers.



Activity vector b =
0.3 * Activity vector a
+ 0.7 * LN(Activity vector c)

Experiments with CIFAR-10

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

Conclusion: Although the test performance of FF is worse than backpropagation it is only slightly worse, even when there are complicated confounding backgrounds.

Max(min) ssq: maximize(minimize) the sum of the squared activities

Other interesting points

- An energy efficient way to multiply an activity vector by a weight matrix is to implement activities as voltages and weights as conductances.
- There are many other possible activation function to explore in the context of FF.
- Mortal Computation makes trillion parameter neural net to consume a few watts.

Code

```python
class Layer(nn.Linear):
    def __init__(self, in_features, out_features,
                 bias=True, device=None, dtype=None):
        super().__init__(in_features, out_features, bias, device, dtype)
        self.relu = torch.nn.ReLU()
        self.opt = Adam(self.parameters(), lr=0.03)
        self.threshold = 2.0
        self.num_epochs = 500

    def forward(self, x):
        x_direction = x / (x.norm(2, 1, keepdim=True) + 1e-4)
        return self.relu(
            torch.mm(x_direction, self.weight.T) +
            self.bias.unsqueeze(0))

    def train(self, x_pos, x_neg):
        for i in tqdm(range(self.num_epochs)):
            g_pos = self.forward(x_pos).pow(2).mean(1)
            g_neg = self.forward(x_neg).pow(2).mean(1)
            # The following loss pushes pos (neg) samples to
            # values larger (smaller) than the self.threshold.
            loss = torch.log(1 + torch.exp(torch.cat([
                -g_pos + self.threshold,
                g_neg - self.threshold]))).mean()
            self.opt.zero_grad()
            # this backward just compute the derivative and hence
            # is not considered backpropagation.
            loss.backward()
            self.opt.step()
        return self.forward(x_pos).detach(), self.forward(x_neg).detach()
```

```
(detectron2) root@lyhlab2:/opt/data/private/lyh/FFA# python main.py
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MN
100%|                                                       | 9912422/9
T/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MN
100%|                                                            | 288
T/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNI
100%|                                                         | 1648877/
/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNI
100%|
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

training layer 0 ...
100%|
training layer 1 ...
100%|
train error: 0.06754004955291748
test error: 0.06850004196166992
```