The Forward-Forward Algorithm: Some Preliminary Investigations

Geoffrey Hinton
Google Brain
geoffhinton@google.com

Abstract

The aim of this paper is to introduce a new learning procedure for neural networks and to demonstrate that it works well enough on a few small problems to be worth serious investigation. The Forward-Forward algorithm replaces the forward and backward passes of backpropagation by two forward passes, one with positive (*i.e.* real) data and the other with negative data which could be generated by the network itself. Each layer has its own objective function which is simply to have high goodness for positive data and low goodness for negative data. The sum of the squared activities in a layer can be used as the goodness but there are many other possibilities, including minus the sum of the squared activities. If the positive and negative passes can be separated in time, the negative passes can be done offline, which makes the learning much simpler in the positive pass and allows video to be pipelined through the network without ever storing activities or stopping to propagate derivatives.

1 What is wrong with backpropagation

The astonishing success of deep learning over the last decade has established the effectiveness of performing stochastic gradient descent with a large number of parameters and a lot of data. The gradients are usually computed using backpropagation (Rumelhart et al., 1986), and this has led to a lot of interest in whether the brain implements backpropagation or whether it has some other way of getting the gradients needed to adjust the weights on connections.

As a model of how cortex learns, backpropagation remains implausible despite considerable effort to invent ways in which it could be implemented by real neurons (Lillicrap et al., 2020; Richards and Lillicrap, 2019; Guerguiev et al., 2017; Scellier and Bengio, 2017). There is no convincing evidence that cortex explicitly propagates error derivatives or stores neural activities for use in a subsequent backward pass. The top-down connections from one cortical area to an area that is earlier in the visual pathway do not mirror the bottom-up connections as would be expected if backpropagation was being used in the visual system. Instead, they form loops in which neural activity goes through about half a dozen cortical layers in the two areas before arriving back where it started.

Backpropagation through time as a way of learning sequences is especially implausible. To deal with the stream of sensory input without taking frequent time-outs, the brain needs to pipeline sensory data through different stages of sensory processing and it needs a learning procedure that can learn on the fly. The representations in later stages of the pipeline may provide top-down information that influences the representations in earlier stages of the pipeline *at a later time step*, but the perceptual system needs to perform inference and learning in real time without stopping to perform backpropagation.

Preprint. Under review.

A further serious limitation of backpropagation is that it requires perfect knowledge of the computation performed in the forward pass[1] in order to compute the correct derivatives[2]. If we insert a black box into the forward pass, it is no longer possible to perform backpropagation unless we learn a differentiable model of the black box. As we shall see, the black box does not change the learning procedure at all for the Forward-Forward Algorithm because there is no need to backpropagate through it.

In the absence of a perfect model of the forward pass, it is always possible to resort to one of the many forms of reinforcement learning. The idea is to make random perturbations of the weights or the neural activities and to correlate these perturbations with the resulting changes in a payoff function. But reinforcement learning procedures suffer from high variance: it is hard to see the effect of perturbing one variable when many other variables are being perturbed at the same time. To average away the noise caused by all the other perturbations, the learning rate needs to be inversely proportional to the number of variables that are being perturbed and this means that reinforcement learning scales badly and cannot compete with backpropagation for large networks containing many millions or billions of parameters[3].

The main point of this paper is to show that neural networks containing unknown non-linearities do not need to resort to reinforcement learning. The Forward-Forward algorithm (FF) is comparable in speed to backpropagation but has the advantage that it can be used when the precise details of the forward computation are unknown. It also has the advantage that it can learn while pipelining sequential data through a neural network without ever storing the neural activities or stopping to propagate error derivatives.

The forward-forward algorithm is somewhat slower than backpropagation and does does not generalize quite as well on several of the toy problems investigated in this paper so it is unlikely to replace backpropagation for applications where power is not an issue. The exciting exploration of the abilities of very large models trained on very large datasets will continue to use backpropagation. The two areas in which the forward-forward algorithm may be superior to backpropagation are as a model of learning in cortex and as a way of making use of very low-power analog hardware without resorting to reinforcement learning(Jabri and Flower, 1992).

2 The Forward-Forward Algorithm

The Forward-Forward algorithm is a greedy multi-layer learning procedure inspired by Boltzmann machines (Hinton and Sejnowski, 1986) and Noise Contrastive Estimation (Gutmann and Hyvärinen, 2010). The idea is to replace the forward and backward passes of backpropagation by two forward passes that operate in exactly the same way as each other, but on different data and with opposite objectives. 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. This paper explores two different measures of goodness – the sum of the squared neural activities and the negative sum of the squared activities, but many other measures are possible.

Let us suppose that the goodness function for a layer is simply the sum of the squares of the activities of the rectified linear neurons in that layer[4] 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. More specifically, the aim is to correctly classify input vectors as positive data or negative data when the

[1]I use "forward pass" to refer to either a forward pass through a multi-layer network or a forward pass in time in a recurrent network.

[2]It is possible to use random weights in the backward pass instead of the transpose of the forward weights, and this yields fake derivatives that work surprisingly well on small-scale tasks like MNIST (Lillicrap et al., 2016), but it does not scale well to large networks and it is particularly bad at squeezing information through narrow bottlenecks.

[3]One possible way to make reinforcement learning scale better is to have many different objective functions and to train subsets of the neurons or weights using their own local objective function (Dayan and Hinton, 1992; Ren et al., 2022). The network can then be scaled up by keeping the subsets the same size but having more subsets, each with its own objective function.

[4]There are two main reasons for using the squared length of the activity vector as the goodness function. First, it has very simple derivatives. Second, layer normalization removes all trace of the goodness.

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

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

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.

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

It is easy to see that a single hidden layer can be learned by making the sum squared activities of the hidden units be high for positive data and low for negative data. But if the activities of the first hidden layer are then used as input to the second hidden layer, it is trivial to distinguish positive from negative data by simply using the length of activity vector in the first hidden layer. There is no need to learn any new features. To prevent this, FF normalizes the length of the hidden vector before using it as input to the next layer (Ba et al., 2016b; Carandini and Heeger, 2013) This removes all of the information that was used to determine the goodness in the first hidden layer and forces the next hidden layer to use information in the *relative* activities of the neurons in the first hidden layer. These relative activities are unaffected by the layer-normalization[5]. To put it another way, the activity vector in the first hidden layer has a length and an orientation. The length is used to define the goodness for that layer and only the orientation is passed to the next layer.

3 Some experiments with FF

The aim of this paper is to introduce the FF algorithm and to show that it works in relatively small neural networks containing a few million connections. A subsequent paper will investigate how well it scales to large neural networks containing orders of magnitude more connections.

3.1 The backpropagation baseline

Most of the experiments described in the paper use the MNIST dataset of handwritten digits. 50,000 of the official training images are used for training and 10,000 for validation during the search for good hyper-parameters. The official test set of 10,000 images is then used to compute the test error rate. MNIST has been well studied and the performance of simple neural networks trained with backpropagation is well known. This makes MNIST very convenient for testing out new learning algorithms to see if they actually work.

Sensibly-engineered convolutional neural nets with a few hidden layers typically get about 0.6% test error. In the "permutation-invariant" version of the task, the neural net is not given any information about the spatial layout of the pixels so it would perform equally well if all of the training and test images were subjected to the same random permutation of the pixels before training started. For the permutation-invariant version of the task, feed-forward neural networks with a few fully connected hidden layers of Rectified Linear Units (ReLUs) typically get about 1.4% test error[6] and they take about 20 epochs to train. This can be reduced to around 1.1% test error using a variety of regularizers such as dropout (Srivastava et al., 2014) (which makes training slower) or label smoothing (Pereyra et al., 2017) (which makes training faster). It can be further reduced by combining supervised learning of the labels with unsupervised learning that models the distribution of images.

To summarize, 1.4% test error on the permutation-invariant version of the task without using complicated regularizers, shows that a learning procedure works about as well as backpropagation[7].

[5]FF uses the simplest version of layer normalization which does not subtract the mean before dividing by the length of the activity vector.

[6]I have trained thousands of different neural networks on MNIST and, in the spirit of Ramon y Cajal, I believe that it is more informative to report the performance of the non-existent typical net than to report the performance of any particular net.

[7]Some papers on biologically plausible alternatives to backpropagation report test error rates greater than 2% on permutation invariant MNIST. This indicates that the learning procedure does not work nearly as well as backpropagation or that the authors failed to tune it properly.

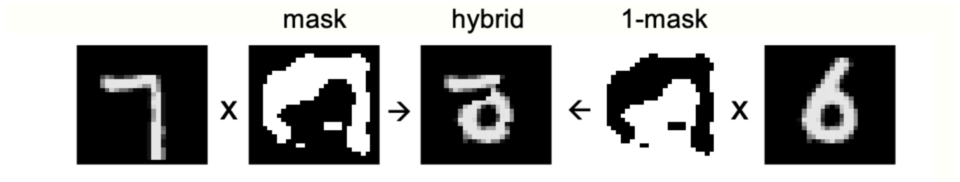

Figure 1: A hybrid image used as negative data

3.2 A simple unsupervised example of FF

There are two main questions about FF that need to be answered. First, if we have a good source of negative data, does it learn effective multi-layer representations that capture the structure in the data? Second, where does the negative data come from? We start by trying to answer the first question using a hand-crafted source of negative data as a temporary crutch so that we can investigate the first question separately.

A common way to use contrastive learning for a supervised learning task is to first learn to transform the input vectors into representation vectors without using any information about the labels and then to 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. This is called a linear classifier despite its glaring non-linearity. 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. FF can be used to perform this kind of representation learning by using real data vectors as the positive examples and corrupted data vectors as the negative examples. There are many very different ways to corrupt the data.

To force FF to focus on the longer range correlations in images that characterize shapes, we need to create negative data that has very different long range correlations but very similar short range correlations. This can be done by creating a mask containing fairly large regions of ones and zeros. We then create hybrid images for the negative data by adding together one digit image times the mask and a different digit image times the reverse of the mask as shown in figure 1. Masks like this can be created by starting with a random bit image and then repeatedly blurring the image with a filter of the form $[1/4, 1/2, 1/4]$ in both the horizontal and vertical directions. After repeated blurring, the image is then thresholded at 0.5.

After training a network with four hidden layers of 2000 ReLUs each for 100 epochs, we get a test error rate of 1.37% if we use the normalised activity vectors of the last three hidden layers as the inputs to a softmax that is trained to predict the label. Using the first hidden layer as part of the input to the linear classifier makes the test performance worse.

Instead of using fully connected layers we can use local receptive fields (without weight-sharing) and this improves the performance. Only one architecture was tried[8]. After training for 60 epochs it gave 1.16% test error. It used "peer normalization" of the hidden activities to prevent any of the hidden units from being extremely active or permanently off[9].

[8]The 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.

[9]It is helpful to measure the average activity of a unit and to introduce an additional derivative that causes this average activity to regress towards some sensible target value. To avoid having to pick a target value that might not be compatible with the demands of the task, peer normalization uses the running mean of the average activities of all the hidden units in the layer as the target. When the activity of a ReLU is zero, there is no derivative with respect to the input to the ReLU but we pretend that there is so that dead units have their incoming weights increased. The coefficient determining the strength of this regularizer is set so that the mean activities of the units in a layer are similar, but not too similar.

3.3 A simple supervised example of FF

Learning hidden representations without using any label information is quite sensible for large models that eventually need to be able to perform a wide variety of tasks. The unsupervised learning extracts a smorgasbord of features and the individual tasks can use whichever features are helpful. But if we are only interested in one task and we want to use a small model that does not have the capacity to model the full distribution of the input data, it makes more sense to use supervised learning. One way to achieve this with 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. Since the only difference between positive and negative data is the label, FF will ignore all features of the image that do not correlate with the label.

MNIST images contain a black border to make life easy for convolutional neural nets. If we replace the first 10 pixels by a one of N representation of the label, it is very easy to display what the first hidden layer learns. A network with 4 hidden layers each containing 2000 ReLUs and full connectivity between layers gets 1.36% test errors on MNIST after 60 epochs. Backpropagation takes about 20 epochs to get similar test performance. Doubling the learning rate of FF and training for 40 epochs instead of 60 gives a slightly worse test error of 1.46% instead of 1.36%.

After training with FF, it is possible to classify a test digit by doing a single forward pass through the net starting from an input that consists of the test digit and a neutral label composed of ten entries of 0.1. The hidden activities in all but the first hidden layer are then used as the inputs to a softmax that has been learned during training. This is a quick but sub-optimal way to classify an image. It is better to run the net with a particular label as part of the input and accumulate the goodnesses of all but the first hidden layer. After doing this for each label separately, the label with the highest accumulated goodness is chosen. During training, a forward pass from a neutral label was used to pick hard negative labels and this made the training require about a third as many epochs.

We can augment the training data by jittering the images by up to two pixels in each direction to get 25 different shifts for each image. This uses knowledge of the spatial layout of the pixels so it is no longer permutation invariant. If we train the same net for 500 epochs with this augmented data, we get 0.64% test error which is similar to a convolutional neural net trained with backpropagation. We also get interesting receptive fields in the first hidden layer as shown in figure 2.

3.4 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. This means that what is learned in later layers cannot affect what is learned in earlier layers. This seems like a major weakness compared with backpropagation.

The key to overcoming this apparent limitation of FF is to treat a static image as a rather boring video that is processed by a multi-layer recurrent neural network (Hinton, 2021). FF runs forwards in time for both the positive and negative data, but, as figure 3 shows, the activity vector at each layer is determined by the normalized activity vectors at both the layer above and the layer below *at the previous time-step*[10].

As an initial check that this approach actually works, we can use "video" input that consists of a static MNIST image which is simply repeated for each time-frame. The bottom layer is the pixel image and the top layer is a one-of-N representation of the digit class. There are two or three intermediate layers each of which has 2000 neurons. In a preliminary experiment, the recurrent net was run for 10 time-steps and at each time-step the even layers were updated based on the normalized activities in the odd layers and then the odd layers were updated based on the new normalized activities in the even layers. This alternating update was designed to avoid biphasic oscillations, but it appears to be unnecessary: With a little damping, synchronous updates of all hidden layers based on the normalized states at the previous time step actually learn slightly better which is good news for less regular architectures. So the experiments reported here use synchronous updates with the new pre-normalized states being set to 0.3 of the previous pre-normalized state plus 0.7 of the computed new state.

[10]There is clearly no problem adding skip-layer connections, but the simplest architecture is the easiest to understand and the aim of this paper is understanding, not performance on benchmarks.

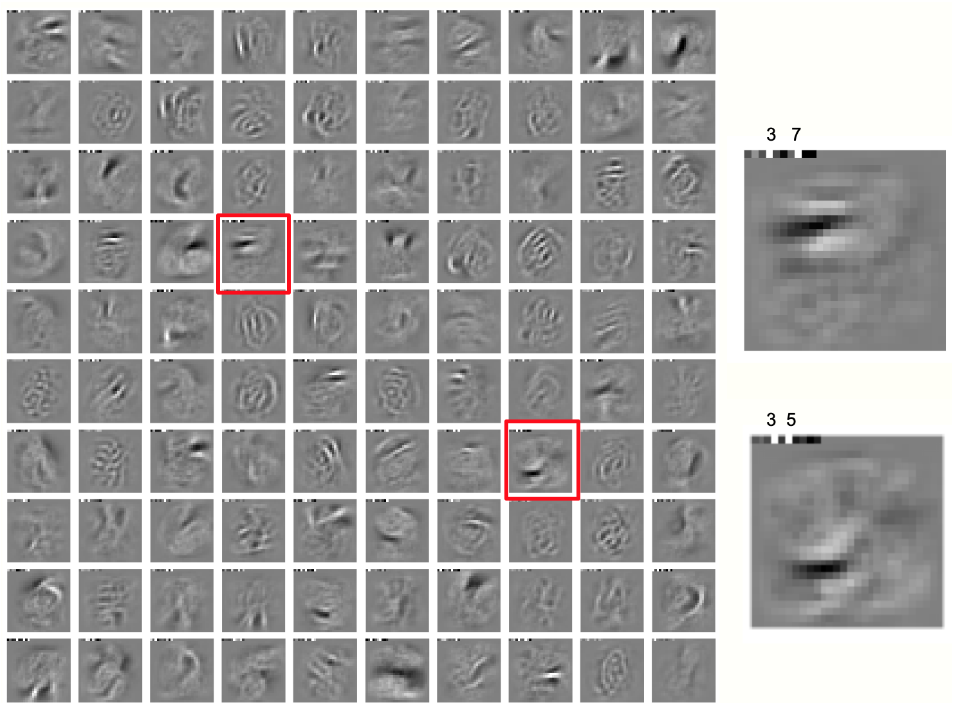

Figure 2: The receptive fields of 100 neurons in the first hidden layer of the network trained on jittered MNIST. The class labels are represented in the first 10 pixels of each image.

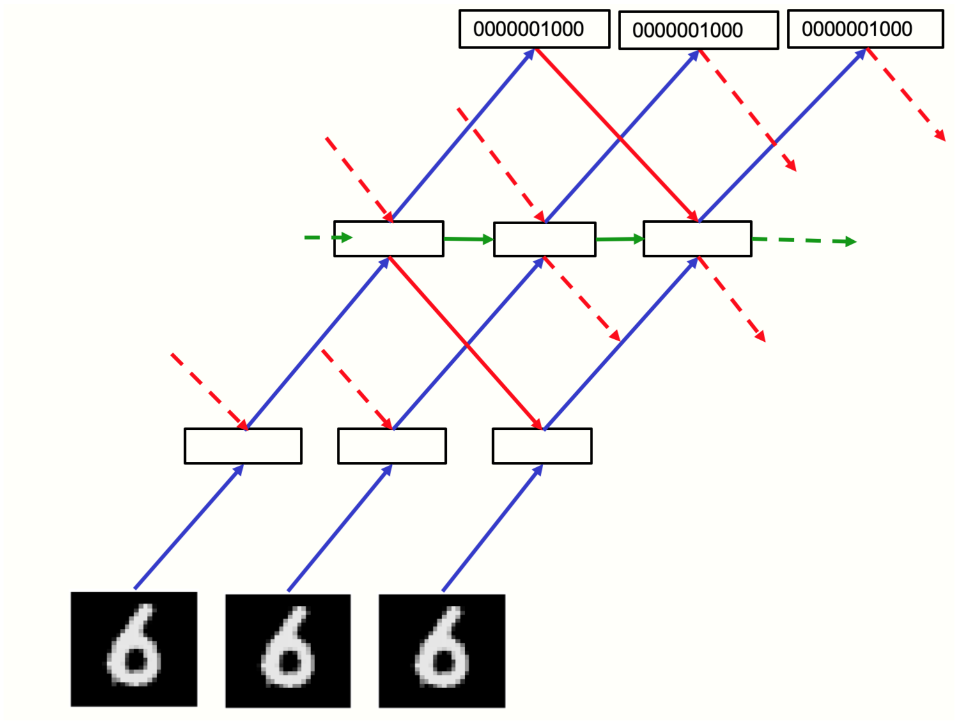

Figure 3: The recurrent network used to process video.

The net shown in figure 3 was trained on MNIST for 60 epochs. For each image, the hidden layers are initialized by a single bottom-up pass. After this the network is run for 8 synchronous iterations with damping. The performance of the network on the test data is evaluated by running it for 8 iterations with each of the 10 labels and picking the label that has the highest goodness averaged over iterations 3 to 5. It gets 1.31% test error. 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.

3.5 Using predictions from the spatial context as a teacher

In the recurrent net, the objective is to have good agreement between the input from the layer above and the input from the layer below for positive data and bad agreement for negative data[11]. In a network with spatially local connectivity, this has a very desirable property: The top-down input will be determined by a larger region of the image and will be the result of more stages of processing so it can be viewed as a contextual prediction for what should be produced by the bottom-up input which is based on a more local region of the image. If the input is changing over time, the top-down input will be based on older input data so it will have to learn to predict the representations of the bottom-up input. If we reverse the sign of the objective function and aim for low squared activities for positive data, the top-down input should learn to cancel out the bottom-up input on positive data which looks very like predictive coding (Rao and Ballard, 1999; van den Oord et al., 2018). The layer normalization means that plenty of information gets sent to the next layer even when the cancellation works pretty well. If all the prediction errors are small, they get exaggerated by the normalization. Dolby would approve.

The idea of learning by using a contextual prediction as a teaching signal for local feature extraction has been around for a long time but it has been difficult to make it work in neural networks using the spatial context as opposed to the one-sided temporal context. The obvious method of using the consensus of the top-down and bottom-up inputs as the teaching signal for both the top-down and bottom-up weights leads to collapse and this problem is not fully solved by using the contextual predictions from other images to create the negative pairs. It seems to be crucial to use negative *data* rather than any negative internal representation.

4 Experiments with CIFAR-10

CIFAR-10 (Krizhevsky and Hinton, 2009) has 50,000 training images that are 32 x 32 with three color channels for each pixel. Each image, therefore, has 3072 dimensions. The images have complicated backgrounds that are highly variable and cannot be modeled well given such limited training data. A fully connected net with two or three hidden layers overfits badly when trained with backpropagation unless the hidden layers are very small, so nearly all of the reported results are for convolutional nets.

Since FF is intended for networks where weight-sharing is not feasible, it was compared with a backpropagation net that used local receptive fields to limit the number of weights without seriously restricting the number of hidden units. The aim was simply to show that with plenty of hidden units, FF is comparable in performance to backpropagation for images that contain highly variable backgrounds.

The networks contained two or three hidden layers of 3072 ReLUs each. Each hidden layer is a 32 x 32 topographic map with 3 hidden units at each location. Each hidden unit has an 11 x 11 receptive field in the layer below so it has 363 bottom-up inputs[12]. For FF, hidden units in the last hidden layer have 10 top-down inputs and in other layers they have up to 363 top-down inputs from an 11 x 11 receptive field in the layer above[13].

Table 1 shows the test performance of networks trained with backpropagation and FF, with both methods using weight-decay to reduce overfitting. To test a net trained with FF, we can either use a

[11]There is also pressure to make the different parts of the bottom-up input agree on which neurons to activate, and the same goes for the top-down input.

[12]For hidden neurons near the edge of the map the receptive field is truncated at the edge of the image.

[13]An inefficient but very simple way to implement this connectivity on a GPU is to learn with full connectivity but to use a precomputed mask to reset all of the non-existent weights to zero after each weight update. This approach makes it very easy to explore any kind of connectivity between layers

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

Table 1: A comparison of backpropagation and FF on CIFAR-10 using non-convolutional nets with local receptive fields of size 11x11 and 2 or 3 hidden layers. One version of FF is trained to maximize the sum of the squared activities on positive cases. The other version is trained to minimize it and this gives slightly better test performance. After training with FF, a single forward pass through the net is a quick but sub-optimal way to classify an image. It is better to run the net with a particular label as the top-level input and record the average goodness over the middle iterations. After doing this for each label separately, the label with the highest goodness is chosen. For a large number of labels, a single forward pass could be used to get a candidate list of which labels to evaluate more thoroughly.

single forward pass or we can let the network run for 10 iterations with the image and each of the 10 labels and accumulate the energy for a label over iterations 4 to 6 (that is when the goodness-based error is the lowest). Although the test performance of FF is worse than backpropagation it is only slightly worse, even when there are complicated confounding backgrounds. Also, the gap between the two procedures does not increase with more hidden layers. However, backpropagation reduces the training error much more quickly.

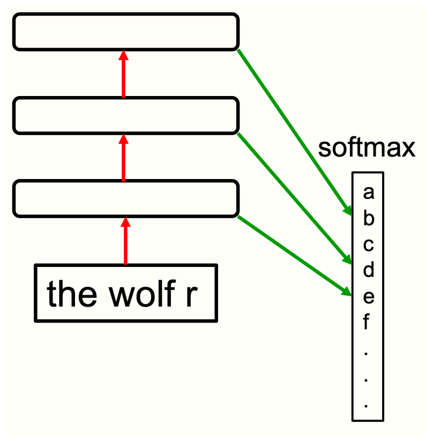

Figure 4: The net used to predict the next character.

4.1 A simple sequence learning example of FF

Generating video frames is difficult so for this initial paper on FF, the recurrent net shown in figure 3 assumed that the positive and negative video input was given and the focus was on discriminative learning. Generating discrete symbols is much easier, so this section uses the task of predicting the next character in a sequence to demonstrate that a network trained with FF can generate its own negative data.

Consider the task of learning to predict the next character in a string from the previous 10 characters. One simple way to do this is to use several hidden layers to extract higher-order features of the preceding 10 character string and then to use the activities of the hidden units in these layers as inputs to a softmax that predicts the probability distribution over all possible next characters. The obvious way to train this network is by using backpropagation, but FF provides a more biologically realistic alternative.

For a toy demonstration that this can be made to work, 248 strings of 100 characters each were taken from Aesop's fables. Upper case letters were converted to lower case and characters were deleted if they were not in the 30 symbol alphabet consisting of the lower case letters, space, comma, semi-colon, and full stop. The first 10 characters of each string were provided as context and the neural network shown in figure 4 was trained to predict the last 90 characters. Each of the three hidden layers had 2000 ReLUs.

The hidden layers can be trained by using strings of 10 characters from the real data as the positive data and strings in which the last character has been replaced by a prediction from the previous 10 characters as the negative data. This is a form of teacher forcing since the first 9 characters of a positive and negative pair are identical.

Alternatively, the negative data could be entirely generated by the predictive model starting from the initial 10 characters of a real data string. This works very nearly as well as teacher forcing and it would allow the negative data to be generated offline in a sleep phase[14]. The big advantage of this is that the FF learning in the wake phase becomes simpler: For the connections to the hidden layers, it just consists of doing a forward pass and raising the goodness of hidden layers that are not good enough. For the connections back to the input, it just consists of learning these connections to maximize the log probability of correctly predicting the next character.

Figure 5 shows that FF can learn hidden representations that are good for predicting the next character.

[14]Assuming the net could somehow remember the initial 10 characters.

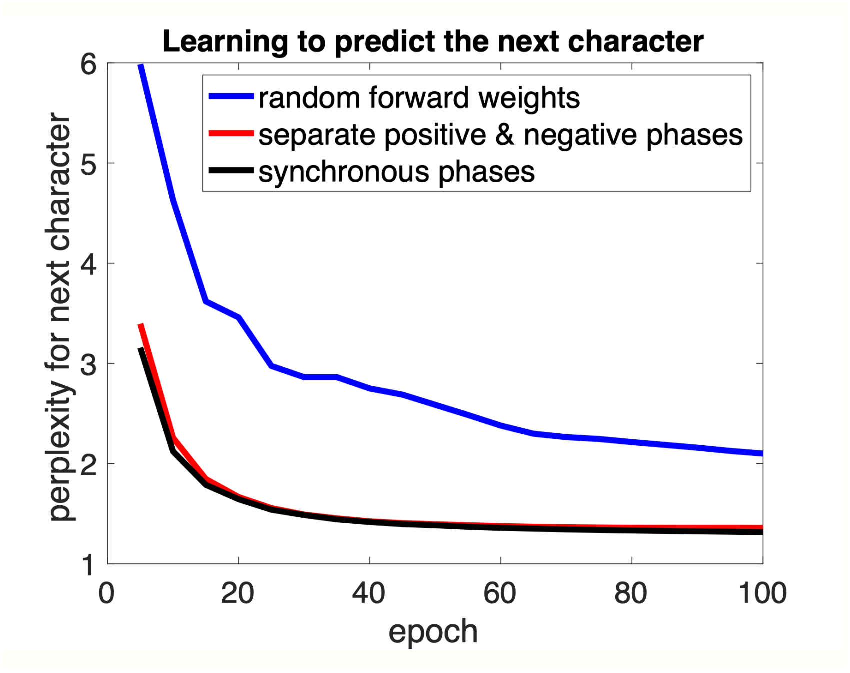

Figure 5: Learning curves showing the perplexity of predicting the next character for three different learning regimes. To demonstrate that it is helpful to learn the hidden representations, the top learning curve uses fixed random weights for computing the hidden representations and simply learns the linear classifier defined by the top-down connections. The two lower curves are almost identical. For one curve, the weights are updated by combining the positive gradients computed for the data sequences with the negative gradients computed for sequences generated by the model. For the other curve, the weights are first updated using only the positive gradients from data sequences. Negative sequences are then generated from from the new model and the negative gradients are used to update the weights again. The close similarity of the two curves is some evidence that the negative passes can be done offline. Each weight update uses gradients from 22,320 character predictions. If the weight updates were performed online, there would be 22,320 positive updates followed by 22,320 negative updates.

5 How FF relates to other contrastive learning techniques

5.1 Relationship to Boltzmann Machines

In the early 1980s there were two promising learning procedures for deep neural networks. One was backpropagation and the other was Boltzmann Machines (Hinton and Sejnowski, 1986) which performed unsupervised contrastive learning. A Boltzmann Machine is a network of stochastic binary neurons with pairwise connections that have the same weight in both directions. When it is running freely with no external input, a Boltzmann machine repeatedly updates each binary neuron by setting it to the on state with a probability equal to the logistic of the total input it receives from other active neurons. This simple update procedure eventually samples from the equilibrium distribution in which each global configuration (an assignment of binary states to all of the neurons) has a log probability proportional to its negative energy. The negative energy is simply the sum of the weights between all pairs of neurons that are on in that configuration.

A subset of the neurons in a Boltzmann Machine is “visible” and binary data vectors are presented to the network by clamping them on the visible neurons and then letting it repeatedly update the states of the remaining, hidden neurons. The aim of Boltzmann machine learning is to make the distribution of binary vectors on the visible neurons when the network is running freely match the data distribution. Very surprisingly, the Kullback-Liebler divergence between the data distribution and the model distribution exhibited on the visible neurons by the freely running Boltzmann machine at thermal equilibrium has a very simple derivative w.r.t. any weight, w_{ij} in the network:

$$\frac{\partial KL(P_{data}||P_{model})}{\partial w_{ij}} = < s_i s_j >_{data} \; - \; < s_i s_j >_{model} \tag{2}$$

where the angle brackets denote an expectation over the stochastic fluctuations at thermal equilibrium and also the data for the first term.

The exciting thing about this result is that it gives derivatives for weights deep within the network without ever propagating error derivatives explicitly. Instead, it propagates neural activities in two different phases which were intended to correspond to wake and sleep. Unfortunately, the mathematical simplicity of the learning rule comes at a very high price. It requires a deep Boltzmann Machine to approach its equilibrium distribution and this makes it impractical as a machine learning technique and implausible as a model of cortical learning: There is not time for a large network to approach its equilibrium distribution during perception. There is also no evidence for detailed symmetry of connections in cortex and there is no obvious way to learn sequences. Moreover, the Boltzmann machine learning procedure fails if many positive updates of the weights are followed by many negative updates as would be required if the negative phase corresponded to REM sleep (Crick and Mitchison, 1983).[15]. Despite these insuperable objections, Boltzmann Machine learning remains intellectually interesting because it replaces the forward and backward passes of backpropagation with two iterative settlings that work in exactly the same way, but with different boundary conditions on the visible neurons – clamped to data versus free.

The Boltzmann machine can be seen as a combination of two ideas:

1. Learn by minimizing the free energy on real data and maximizing the free energy on negative data generated by the network itself.
2. Use the Hopfield energy as the energy function and use repeated stochastic updates to sample global configurations from the Boltzmann distribution defined by the energy function.

Retrospectively, it is obvious that the first idea, which is about contrastive learning, could be used with many other energy functions. For example, (Hinton et al., 2006a) uses the output of a feedforward neural net to define the energy and then uses backpropagation through this net to compute the derivatives of the energy w.r.t. both the weights and the visible states. Negative data is then generated by following the derivatives of energy w.r.t. the visible states. This approach is used very effectively in (Grathwohl et al., 2019). It is also obvious that the negative data does not have to be produced by sampling data vectors from the Boltzmann distribution defined by the energy function. Methods like contrastive divergence (Hinton, 2002) , for example, greatly improve the efficiency of learning in Boltzmann machines with a single hidden layer by not sampling from the equilibrium distribution. But the mathematical simplicity of equation 2 and the fact that the stochastic update procedure performs correct Bayesian integration over all possible hidden configurations were just too beautiful to abandon, so the idea of replacing the forward and backward passes of backpropagation with two settlings that only need to propagate neural activities remained entangled with the complexities of Markov Chain Monte Carlo.

FF combines the contrastive learning from Boltzmann machines with a simple, local goodness function that is far more tractable than the free energy of a network of binary stochastic neurons.

5.2 Relationship to Generative Adversarial Networks

A GAN (Goodfellow et al., 2014) uses a multi-layer neural network to generate data and it trains its generative model by using a multi-layer discriminative network to give it derivatives w.r.t. the output of the generative model of the probability that the output is real data as opposed to generated data. GANs are tricky to train because the discriminative and generative models are competing. In practice they generate very nice images but suffer from mode collapse: There can be large regions of image

[15]The need to interleave positive and negative updates on a fine timescale can be somewhat reduced by storing the average of $< s_i s_j >_{model}$ during sleep and subtracting this baseline from the positive updates during wake, but this trick fails as soon as the weight updates make the stored baseline significantly incorrect. The trick works much better for FF because, unlike the Boltzmann machine, the updates in the positive phase stop changing the weights once the layers are correctly classifying an example as positive.

space in which they never generate examples. Also they use backpropagation to adapt each network so it is hard to see how to implement them in cortex.

FF can be viewed as a special case of a GAN in which every hidden layer of the discriminative network makes its own greedy decision about whether the input is positive or negative so there is no need to backpropagate to learn the discriminative model. There is also no need to backpropagate to learn the generative model because instead of learning its own hidden representations, it just reuses the representations learned by the discriminative model. The only thing the generative model has to learn is how to convert those hidden representations into generated data and if this is done using a linear transformation to compute the logits of a softmax no backpropagation is required. One advantage of using the same hidden representations for both models is that it eliminates the problems that arise when one model learns too fast relative to the other model. It also eliminates mode collapse.

5.3 Relationship to contrastive methods that compare representations of two different image crops

There is a family of self-supervised contrastive methods like SimCLR (Chen et al., 2020a; Bachman et al., 2019; He et al., 2020; Grill et al., 2020; Chen et al., 2020b) that learn by optimizing an objective function that favors agreement between the representations of two different crops from the same image and disagreement between the representations of crops from two different images. These methods generally use many layers to extract the representations of the crops and they train these layers by backpropagating the derivatives of the objective function. They do not work if the two crops always overlap in exactly the same way because then they can simply report the intensities of the shared pixels and get perfect agreement.

In a real neural network, it would be non-trivial to measure the agreement between two different representations[16] and there is no obvious way to extract the representations of two crops at the same time using the same weights.

FF uses a different way to measure agreement, which seems easier for a real neural network. Many different sources of information provide input to the same set of neurons. If the sources agree on which neurons to activate there will be positive interference resulting in high squared activities and if they disagree the squared activities will be lower[17]. Measuring agreement by using positive interference is a lot more flexible than comparing two different representation vectors because there is no need to arbitrarily divide the input into two separate sources.

A major weakness of SimCLR-like methods is that a lot of computation is used to derive the representations of two image crops, but the objective function only provides a modest amount of constraint on the representations and this limits the rate at which information about the domain can be injected into the weights. To make the representation of a crop be closer to its correct mate than to a million possible alternatives only takes 20 bits of information. This problem seems even worse for FF since it only takes one bit to distinguish a positive case from a negative one.

The solution to this poverty of constraint is to divide each layer into many small blocks and to force each block separately to use the length of its pre-normalized activity vector to decide between positive and negative cases. The information required to satisfy the constraints then scales linearly with the number of blocks which is much better than the logarithmic scaling achieved by using a larger contrast set in SimCLR-like methods.

5.4 A problem with stacked contrastive learning

An obvious unsupervised way to learn multiple layers of representation is to first learn one hidden layer that captures some of the structure in the data and to then treat the activity vectors in this layer as data and apply the same unsupervised learning algorithm again. This is how multiple layers of representation are learned using Restricted Boltzmann machines (RBMs)(Hinton et al., 2006b) or stacked autoencoders (Bengio et al., 2007). But it has a fatal flaw.

[16]An obvious exception is when the two representations occur at adjacent times on the same set of neurons. The temporal derivatives of the neural activities then represent the amount of disagreement.

[17]It is possible to measure agreement much more sharply if the inputs are spikes which arrive at particular times and the post-synaptic neurons only fire if they get many spikes within a small time window.

Suppose we map some random noise images through a random weight matrix. The resulting activity vectors will have correlational structure that is created by the weight matrix and has nothing to do with the data. When unsupervised learning is applied to these activity vectors it will discover some of this structure, but that tells the system nothing about the external world[18].

The original Boltzmann Machine learning algorithm was designed to avoid this flaw (Hinton and Sejnowski, 1986) (page 297) by contrasting statistics caused by two different *external* boundary conditions. This cancels out all the structure that is just the result of other parts of the network. When contrasting positive and negative *data*, there is no need to restrict the wiring or have random spatial relationships between crops to prevent the network from cheating. This makes it easy to have a large number of interconnected groups of neurons, each with its own objective of distinguishing positive from negative data.

6 Learning fast and slow

If there is full connectivity between layers, it can be shown that the weight updates used to change the goodness function of a layer for a particular input vector $\mathbf{x}$ have no effect on the layer-normalized output of that layer *when the input vector is* $\mathbf{x}$. The vector of increments of the incoming weights for hidden neuron j is given by:

$$\Delta \mathbf{w_j} = 2\epsilon \frac{\partial log(p)}{\partial \sum_j y_j^2} \, y_j \, \mathbf{x} \tag{3}$$

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.

After the weight update has occurred, the change in the activity of neuron j is simply the scalar product $\Delta \mathbf{w_j} \mathbf{x}$. 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.

The fact that the weight update does not change the layer normalized output for that input vector means that it is possible to perform simultaneous online weight updates in many different layers because the weight updates in earlier layers do not change the activity vectors in later layers *for that input vector*. It is possible to change all the weights in one step so that every layer exactly achieves a desired goodness of S^* for input vector $\mathbf{x}$. 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:

$$\epsilon = \sqrt{\frac{S^*}{S_L}} - 1 \tag{4}$$

where S_L is the current goodness of layer L.

Currently, we do not exploit this interesting property of FF because we still use mini-batches, but the ability of a deep neural net to absorb a lot of information from a single training case by jumping to a set of weights that handles that case perfectly could be of interest to psychologists who are tired of creeping down gradients[19]

Unlike backpropagation, FF has no problem learning if we insert a black box between layers that are being trained by FF. The black box applies an unknown and possibly stochastic transformation to the

[18]Stacked RBMs can deal with this issue, though not very well, by initializing each RBM to have the transpose of the weights of the previous RBM. This means that the initial weights have already captured the structure induced by the previous weight matrix and changes to those initial weights may capture structure due to the external world, but a lot of the capacity is used up just modeling the previous weight matrix.

[19]The perceptron convergence procedure (Rosenblatt, 1958) also has the property that it is possible to make big updates to the weights without using a learning rate. For perceptrons, any set of weights in the "feasible" region of weight-space will give the correct binary classification for all of the training examples. Sadly, there is no feasible region if the positive and negative examples are not linearly separable. Given a training example that is currently misclassified, the weights can be updated in a single step to make the classification correct for that example. The weight update is perpendicular to the hyper-plane that separates good weights from bad weights for that particular example and the step size can be chosen to move the weights to the correct side of that hyperplane. The update may cause many other examples to be misclassified, but the perceptron convergence procedure does not converge by reducing the number of errors or the sum squared error. It works by always getting closer to the feasible reqion.

output of one layer and presents this transformed activity vector as the input to the next layer. One interesting possibility is to make these black boxes be neural nets with a few hidden layers. If these hidden layers learn very slowly, the “outer loop” FF learning can quickly adapt to new data under the assumption that the black boxes are stationary. Slow learning in the black boxes can then improve the system over a much longer timescale. For example, a slow reinforcement learning procedure could add small random noise vectors to the inputs to neurons inside the black box and then multiply these activity perturbation vectors by the change in the cost function used by the positive phase of FF to get a noisy but unbiased estimate of the derivative of the FF cost function w.r.t. the activities of neurons inside the black box (Ren et al., 2022).

7 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. Their products, per unit time, are charges which add themselves. This seems a lot more sensible than driving transistors at high power to model the individual bits in the digital representation of a number and then performing $O(n^2)$ single bit operations to multiply two n-bit numbers together. Unfortunately, it is difficult to implement the backpropagation procedure in an equally efficient way, so people have resorted to using A-to-D converters and digital computations for computing gradients (Kendall et al., 2020). The use of two forward passes instead of a forward and a backward pass should make these A-to-D converters unnecessary.

8 Mortal Computation

General purpose digital computers were designed to faithfully follow instructions because it was assumed that the only way to get a general purpose computer to perform a specific task was to write a program that specified exactly what to do in excruciating detail. This is no longer true, but the research community has been slow to comprehend the long-term implications of deep learning for the way computers are built. More specifically the community has clung to the idea that the software should be separable from the hardware so that the same program or the same set of weights can be run on a different physical copy of the hardware. This makes the knowledge contained in the program or the weights immortal: The knowledge does not die when the hardware dies[20].

The separation of software from hardware is one of the foundations of Computer Science and it has many benefits. It makes it possible to study the properties of programs without worrying about electrical engineering. It makes it possible to write a program once and copy it to millions of computers. If, however, we are willing to abandon immortality it should be possible to achieve huge savings in the energy required to perform a computation and in the cost of fabricating the hardware that executes the computation. We can allow large and unknown variations in the connectivity and non-linearities of different instances of hardware that are intended to perform the same task and rely on a learning procedure to discover parameter values that make effective use of the unknown properties of each particular instance of the hardware. These parameter values are only useful for that specific hardware instance, so the computation they perform is mortal: it dies with the hardware.

Even though it makes no sense to copy the parameter values to a different piece of hardware that works differently, there is a more biological way of transferring what has been learned by one piece of hardware to a different piece of hardware. For a task like classification of objects in images, what we are really interested in is the function relating pixel intensities to class labels, not the parameter values that implement that function in a particular piece of hardware. The function itself can be transferred (approximately) to a different piece of hardware by using distillation (Hinton et al., 2014): The new hardware is trained not only to give the same answers as the old hardware but also to output the same probabilities for incorrect answers. These probabilities are a much richer indication of how the old model generalizes than just the label it thinks is most likely. So by training the new model to match the probabilities of incorrect answers we are training it to generalize in the same way as the old model. This is a rare example of neural net training actually optimizing what we care about: generalization.

[20]Assuming the program or weights are faithfully stored in some medium.

If you want your trillion parameter neural net to only consume a few watts, mortal computation may be the only option. Its feasibility rests on finding a learning procedure that can run efficiently in hardware whose precise details are unknown, and the Forward-Forward algorithm is a promising candidate, though it remains to be seen how well it scales to large neural networks.

9 Future work

The investigation of the Forward-Forward algorithm has only just begun and there are many open questions:

- Can FF produce a generative model of images or video that is good enough to create the negative data needed for unsupervised learning?
- Can the positive and negative passes be very widely separated in time as they would be if the negative passes were done during sleep?
- If the negative phase is eliminated for a while, do the effects mimic the devastating effects of severe sleep deprivation?
- What is the best goodness function to use? This paper uses the sum of the squared activities in most of the experiments but minimizing the sum squared activities for positive data and maximizing it for negative data seems to work slightly better. More recently, just minimizing the sum of the activities on positive data (and maximizing on negative) has worked well[21].
- What is the best activation function to use? So far, only ReLUs have been explored. There are many other possibilities whose behaviour is unexplored in the context of FF. Making the activation be the negative log of the density under a t-distribution is an interesting possibility(Osindero et al., 2006).
- For spatial data, can FF benefit from having lots of local goodness functions for different regions of the image?(Löwe et al., 2019) If this can be made to work, it should allow learning to be much faster.
- For sequential data, is it possible to use fast weights to mimic a simplified transformer(Ba et al., 2016a)?
- Can FF benefit from having a set of feature detectors that try to maximize their squared activity and a set of constraint violation detectors that try to minimize their squared activity?(Welling et al., 2003).

10 Acknowledgements

First I would like to thank Jeff Dean and David Fleet for creating the environment at Google that made this research possible. Many people made helpful contributions including, in no particular order, Simon Kornblith, Mengye Ren, Renjie Liao, David Fleet, Ilya Sutskever, Terry Sejnowski, Peter Dayan, Chris Williams, Yair Weiss, and Max Welling.

References

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. (2016a). Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016b). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. In Wallach, H., Larochelle, H., Beygelzimer, A., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 15535–15545. Curran Associates, Inc.

[21]When using the unsquared activities, the layer normalization must normalize the sum of the activities, not the sum of their squares.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *NIPS'2006*.

Carandini, M. and Heeger, D. J. (2013). Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51–62.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. (2020b). Big self-supervised models are strong semi-supervised learners. *arXiv:2006.10029*.

Crick, F. and Mitchison, G. (1983). The function of dream sleep. *Nature*, 304:111–114.

Dayan, P. and Hinton, G. E. (1992). Feudal reinforcement learning. *Advances in neural information processing systems*, 5.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., Norouzi, M., and Swersky, K. (2019). Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning. *arXiv:2006.07733*.

Guerguiev, J., Lillicrap, T. P., and Richards, B. A. (2017). Towards deep learning with segregated dendrites. *ELife*, 6:e22901.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hinton, G. (2021). How to represent part-whole hierarchies in a neural network. *arXiv preprint arXiv:2102.12627*.

Hinton, G., Osindero, S., Welling, M., and Teh, Y.-W. (2006a). Unsupervised discovery of nonlinear structure using contrastive backpropagation. *Cognitive science*, 30(4):725–731.

Hinton, G. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 282–317. MIT Press, Cambridge, MA.

Hinton, G., Vinyals, O., and Dean, J. (2014). Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning Workshop*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006b). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

Jabri, M. and Flower, B. (1992). Weight perturbation: An optimal architecture and learning technique for analog vlsi feedforward and recurrent multilayer networks. *IEEE Transactions on Neural Networks*, 3(1):154–157.

Kendall, J., Pantone, R., Manickavasagam, K., Bengio, Y., and Scellier, B. (2020). Training end-to-end analog neural networks with equilibrium propagation. *arXiv preprint arXiv:2006.01981*.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Lillicrap, T., Cownden, D.and Tweed, D., and Akerman, C. (2016). synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7.

Lillicrap, T., Santoro, A., Marris, L., Akerman, C., and Hinton, G. E. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21:335–346.

Löwe, S., O'Connor, P., and Veeling, B. (2019). Putting an end to end-to-end: Gradient-isolated learning of representations. *Advances in neural information processing systems*, 32.

Osindero, S., Welling, M., and Hinton, G. E. (2006). Topographic product models applied to natural scene statistics. *Neural Computation*, 18(2):381–414.

Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., and Hinton, G. (2017). Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*.

Rao, R. and Ballard, D. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2:79–87.

Ren, M., Kornblith, S., Liao, R., and Hinton, G. (2022). Scaling forward gradient with local losses. *arXiv preprint arXiv:2210.03310*.

Richards, B. A. and Lillicrap, T. P. (2019). Dendritic solutions to the credit assignment problem. *Current opinion in neurobiology*, 54:28–36.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Scellier, B. and Bengio, Y. (2017). Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv:1807.03748*.

Welling, M., Williams, C., and Agakov, F. (2003). Extreme components analysis. *Advances in Neural Information Processing Systems*, 16.