How to do backpropagation in a brain

Geoffrey Hinton

Canadian Institute for Advanced Research
&
University of Toronto
&
Google Inc.

Prelude

- I will start with three slides explaining a popular type of deep learning.

- It is this kind of deep learning that makes back propagation easy to implement.

Pre-training a deep network

- First train a layer of features that receive input directly from the pixels.
  - The features are trained to be good at reconstructing the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features of features in a second hidden layer.
  - They are good at reconstructing the activities in the first hidden layer.
- Each time we add another layer of features we capture more complex, longer range, regularities in the ensemble of training images.

Discriminative fine-tuning

- First train multiple hidden layers greedily to be good autoencoders. This is unsupervised learning.
- Then connect some classification units to the top layer of features and do back-propagation through all of the layers to fine-tune all of the feature detectors.
- On a dataset of handwritten digits called MNIST this worked much better than standard back-propagation and better than Support Vector Machines. (2006)
- On a dataset of spoken sentences called TIMIT it beat the state of the art and led to a major shift in the way speech recognition is done. (2009).

Why does pre-training followed by fine-tuning work so well?

- Greedily learning one layer at a time scales well to really big networks, especially if we have locality in each layer.
- We do not start backpropagation until we already have sensible features in each layer.
  - So the initial gradients are sensible and back-propagation only needs to perform a local search.
- Most of the information in the final weights comes from modeling the distribution of input vectors.
  - The precious information in the labels is only used for the final fine-tuning. It slightly modifies the features. It does not need to discover features.
  - So we can do very well when most of the training data is unlabelled.

But how can the brain back-propagate through a multilayer neural network?

- Some very good researchers have postulated inefficient algorithms that use random perturbations.
  - Do you really believe that evolution could not find an efficient way to adapt a feature so that it is more useful to higher-level features in the same sensory pathway? (have faith!)

Three obvious reasons why the brain cannot be doing backpropagation

- Cortical neurons do not communicate real-valued activities.
  - They send spikes.

- The neurons need to send two different types of signal
  - Forward pass: signal = activity = y
  - Backward pass: signal = dE/dx

- Neurons do not have point-wise reciprocal connections with the same weight in both directions.

Small data: A good reason for spikes

- Synapses are much cheaper than training cases.
  - We have 10^14 synapses and live for 10^9 seconds.
- A good way to throw a lot of parameters at a task is to use big neural nets with strong, zero-mean noise in the activities.
  - Noise in the activities has the same regularization advantages as averaging big ensembles of models but makes much more efficient use of hardware.
- In the small data regime, noise is good so sending random spikes from a Poisson process is better than sending real values.
  - Poisson noise is special because it is exactly neutral about the sparsity of the codes.
  - Multiplicative noise penalizes sparse codes .

A way to simplify the explanations

- Lets ignore the Poisson noise for now.
  - We are going to pretend that neurons communicate real analog values.
- Once we have understood how to do backprop in a brain, we can treat these real analog values as the underlying rates of a Poisson.
  - We will get the same expected value for the derivatives from the Poisson spikes, but with added noise.
  - Stochastic gradient descent is very robust to added noise so long as it is not biased.

A way to encode error derivatives

- Consider a logistic output unit , j, with a cross-entropy error function.

$$-\partial E / \partial x_j \quad = \quad d_j \; - \; p_j$$

⇧ derivative of the error w.r.t. The total input to j

⇧ target value

⇧ output probability when driven bottom-up

Suppose we start with pure bottom-up output, p_j , and then we take a weighted average of the target value and the bottom-up output. We make the weight on the target value grow linearly with time.

$$y_j(t) \quad = \quad p_j + t\,d_j - t\,p_j$$

A fundamental representational decision: temporal derivatives represent error derivatives

- This allows the rate of change of the blended output to represent the error derivative w.r.t. the neuron's input

$$-\partial E / \partial x_j \quad = \quad \dot{y}_j$$

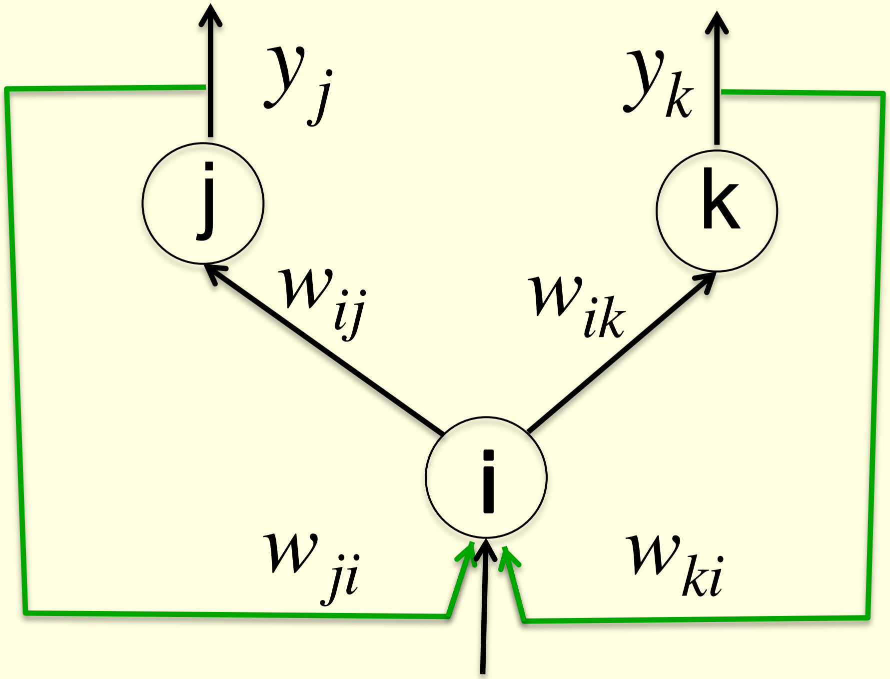
$$\Uparrow \qquad\qquad\qquad \Uparrow$$

error derivative = temporal derivative

This allows the same neuron to code both the normal activity and the error derivative (for a limited time).

The payoff

- In a pre-trained stack of auto-encoders, this way of representing error derivatives makes back-propagation through multiple layers of neurons happen automatically.



If the auto-encoder is perfect, replacing the bottom-up input to i by the top down input will have no effect on the output of i.

If we then start moving y_j and y_k towards their target values, we get:

$$\dot{x}_i = w_{ji}\,\dot{y}_j + w_{ki}\,\dot{y}_k = \frac{dE}{dy_i}$$

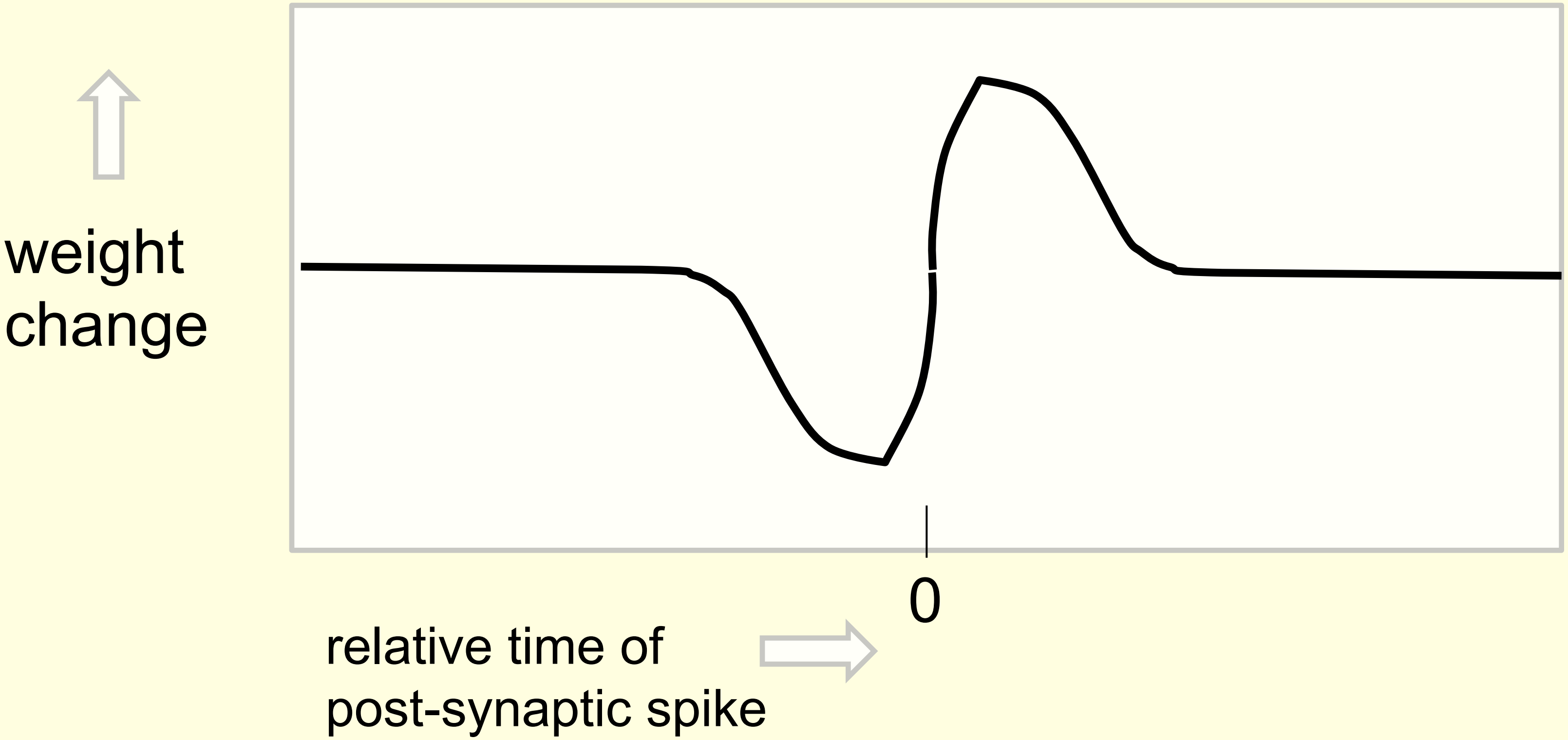
$$\dot{y}_i = \left(w_{ji}\,\dot{y}_j + w_{ki}\,\dot{y}_k\right)\frac{dy_i}{dx_i} = \frac{dE}{dx_i}$$

The synaptic update rule

- First do an upward (forward) pass as usual.
- Then do top-down reconstructions at each level.
- Then perturb the top-level activities by blending them with the target values so that the rate of change of activity of a top-level unit represents the derivative of the error w.r.t. the total input to that unit.
  - This will make the activity changes at every level represent error derivatives.
- Then update each synapse in proportion to:

pre-synaptic activity X rate-of-change of post-synaptic activity

If this is what is happening, what should neuroscientists see?



- Spike-time-dependent plasticity is just a derivative filter. You need a computational theory to recognize what you discovered!

An obvious prediction

- For the top-down weights to stay symmetric with the bottom-up weights, their learning rule should be:

  rate-of-change of

  pre-synaptic activity X post-synaptic activity

A problem (this is where the woffle starts)

- This way of performing backpropagation requires symmetric weights
  - But auto-encoders can still be trained if we first split each symmetric connection into two oppositely directed connections and then we randomly remove many of these directed connections.

Functional symmetry

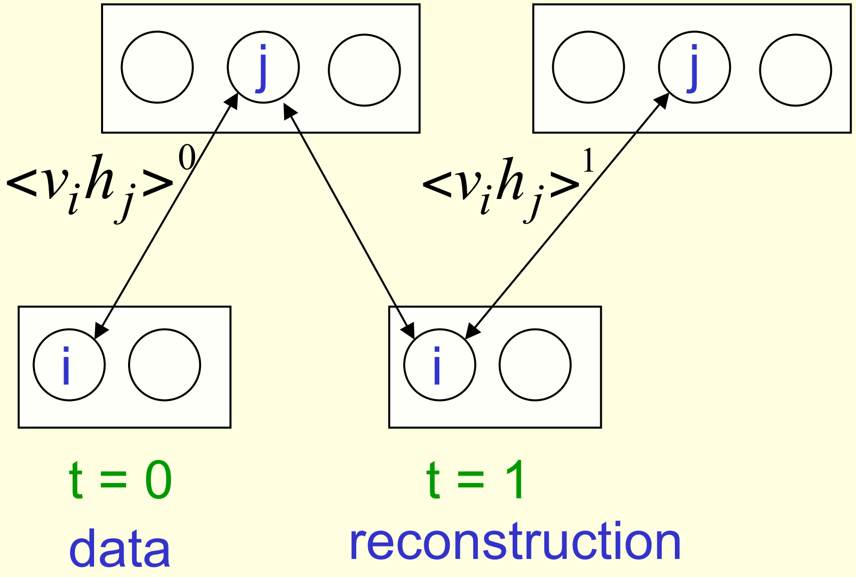
- If the representations are highly redundant, the state of a hidden unit can be estimated very well from the states of the other hidden units in the same layer that have similar receptive fields.
  - So top-down connections from these other correlated units could learn to mimic the effect of the missing top-down part of a symmetric connection.
  - All we require is functional symmetry on and near the data manifold.

But what about the backpropagation required to learn the stack of autoencoders?

- One step Contrastive Divergence learning was initially viewed as a poor approximation to running a Markov chain to equilibrium.
  - The equilibrium statistics are needed for maximum likelihood learning.

- But a better view is that its a neat way of doing gradient descent to learn an auto-encoder when the hidden units are stochastic with discrete states.
  - It uses temporal derivatives as error derivatives.

Contrastive divergence learning: A quick way to learn an RBM



Start with a training vector on the visible units.

Update all the hidden units in parallel

Update all the visible units in parallel to get a "reconstruction".

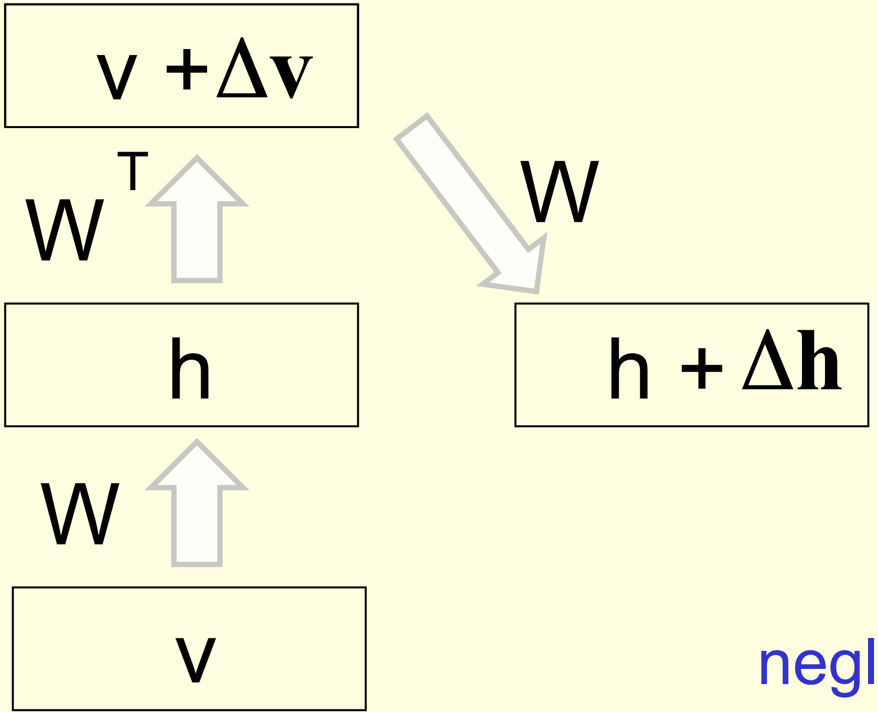
Update all the hidden units again.

$$\Delta w_{ij} = \varepsilon \, (<v_i h_j>^0 - <v_i h_j>^1)$$

This is not following the gradient of the log likelihood. But it works well.

It is approximately following the gradient of another objective function.

One-step CD is just backpropagation learning of an auto-encoder using temporal derivatives



$v + \Delta \mathbf{v}$

W^T

W

h

$h + \Delta \mathbf{h}$

W

v

negligible to first order

true gradient $-\mathbf{v}\Delta\mathbf{h} - \mathbf{h}\Delta\mathbf{v}$ (to first order)

CD: $\mathbf{vh} - (\mathbf{v} + \Delta\mathbf{v})(\mathbf{h} + \Delta\mathbf{h}) = -\mathbf{v}\Delta\mathbf{h} - \mathbf{h}\Delta\mathbf{v} - \underbrace{\Delta\mathbf{v}\Delta\mathbf{h}}$

THE END