IPAM Summer School 2012

Tutorial on:

Deep Learning

Geoffrey Hinton
Canadian Institute for Advanced Research
&
Department of Computer Science
University of Toronto

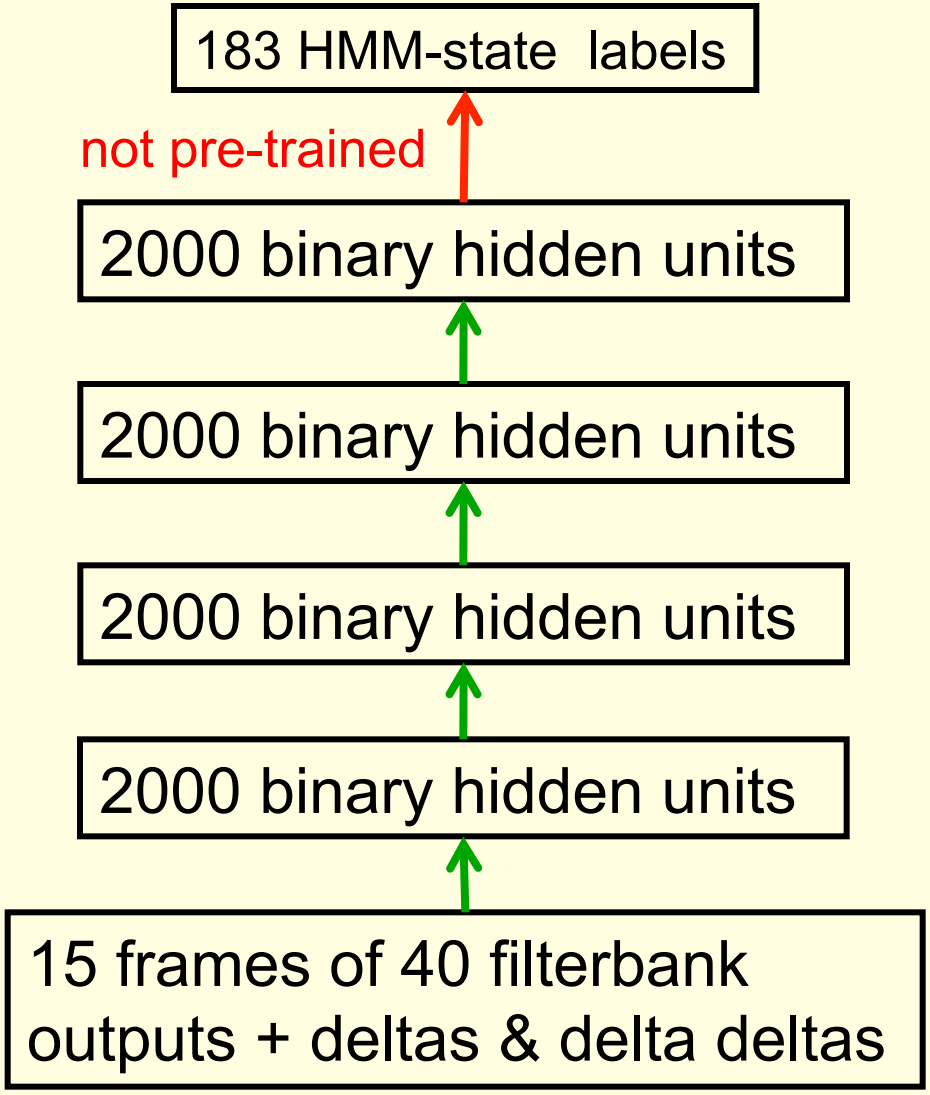
PART 3:
Some applications of deep learning

- Speech recognition
  - Deep learning is now being deployed in the latest speech recognition systems.
- Object recognition
  - Deep learning is breaking records on really tough object recognition tasks.
- Image retrieval
  - Deep learning finds efficient codes for images.
- Predicting the next character in a string
  - Deep learning reads Wikipedia and discovers the meaning of life.

Using deep neural nets for speech recognition

- Each phoneme is modeled by one or more three-state Hidden Markov Model.
- The sound-wave is pre-processed to yield frames of acoustic coefficients every 10ms.
- A deep neural net looks at a window of frames and outputs the probabilities of the various possible HMM states for the central frame.
- A decoder then uses these probabilities to find the best string of words.

Phone recognition on the TIMIT benchmark
(Mohamed, Dahl, & Hinton, 2011)



- After standard post-processing using a bi-phone model, a deep net with 8 layers gets 20.7% error rate.
- The best previous speaker-independent result on TIMIT was 24.4% and this required averaging several models.
- A deep net that uses 3-way factors to model the covariance structure of the filterbank outputs gets 20.5% without using deltas (Dahl et.al. 2010)

What happened next

- This method of using deep nets for speech recognition was ported to MSR by George Dahl and Abdel-rahman Mohamed and to Google by Navdeep Jaitly.
- It forms the basis of a speech recognizer recently deployed by Microsoft.
- Google has also been developing this technology (see next slide).

Word error rates from MSR, IBM, and the Google speech group

task	hours of training data	DNN-HMM	GMM-HMM with same data
Switchboard (test set 1)	309	18.5	27.4
Switchboard (test set 2)	309	16.1	23.6
English Broadcast News	50	17.5	18.8
Bing Voice Search (Sentence error rates)	24	30.4	36.2
Google Voice Input	5,870	12.3	
Youtube	1,400	47.6	52.3

Word error rates from MSR, IBM, and the Google speech group

	hours of training data	DNN-HMM	GMM-HMM with same data	GMM-HMM with more data
	309	18.5	27.4	18.6 (2000 hrs)
	309	16.1	23.6	17.1 (2000 hrs)
	50	17.5	18.8	
	24	30.4	36.2	
	5,870	12.3		16.0 (>>5,870hrs)
	1,400	47.6	52.3	

Experiment on ImageNet 1000 classes
(1.3 million high-resolution training images)

- The 2010 competition winner got 47% error for its first choice and 25% error for top 5 choices.
- The current record is 45% error for first choice.
  - This uses methods developed by the winners of the 2011 competition.
- Our chief critic, Jitendra Malik, has said that this competition is a good test of whether deep neural networks really do work well for object recognition.

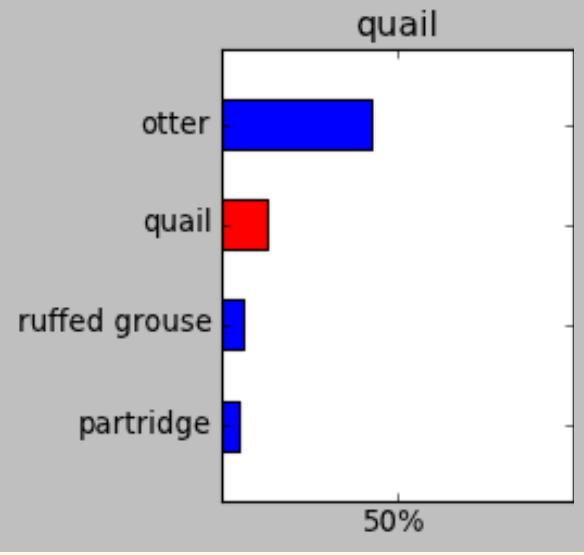
A convolutional net for ImageNet

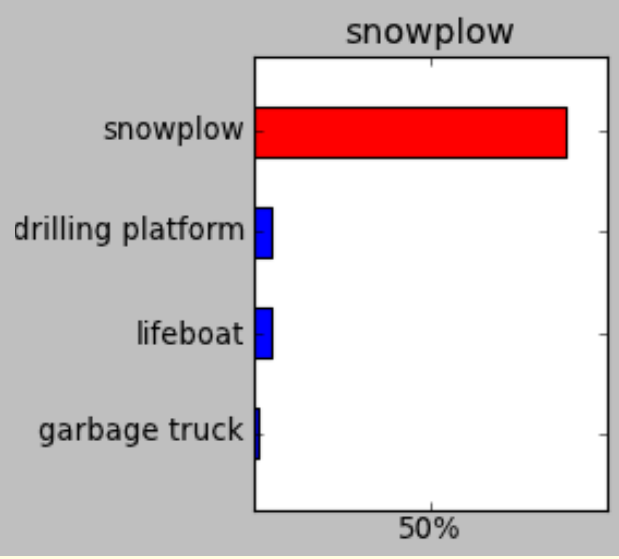
- Alex Krizhevsky developed a very deep convolutional neural net.
  - 7 hidden layers not counting max pooling.
  - Early layers are convolutional, last two layers are globally connected.
  - Alex trains on random 224x224 patches from 256x256 images to get more data.
  - Uses rectified linear units in every layer.
  - Uses competitive normalization to suppress hidden activities.

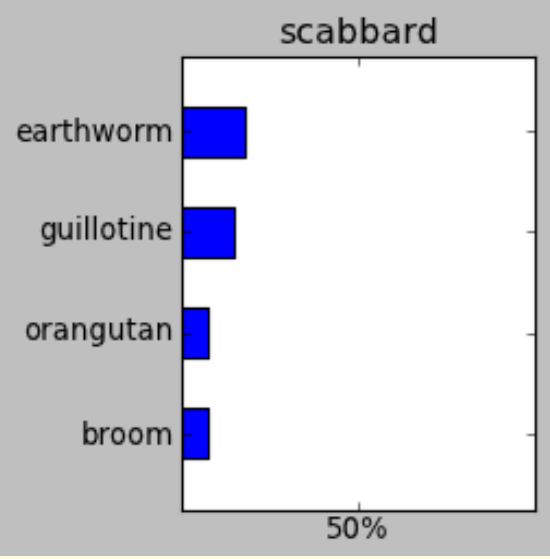
The performance of Alex's net

- At test time, use the central patch and the 4 corner patches and their reflections.
  - By combining the 10 predicted distributions Alex gets an error rate which is about the same as the state-of-the-art.
- If he uses "dropout" to regularize the weights in the globally connected layers (which contain most of the parameters) he gets 39% error for first choice and 19% for top 5 choices.
  - This is a big improvement over the current state of the art (45% for top choice & 25% for top 5).

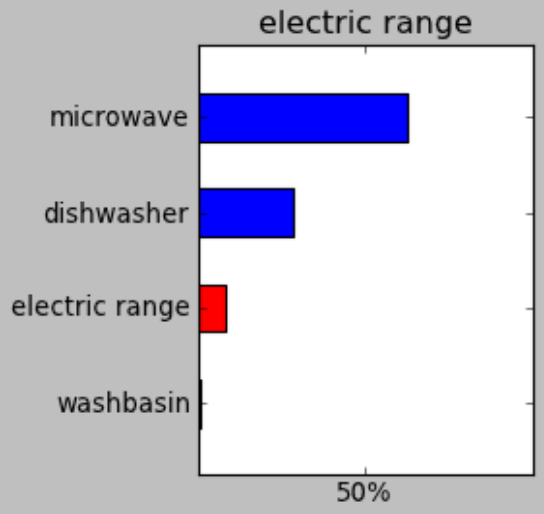
Some examples from an earlier version of the net

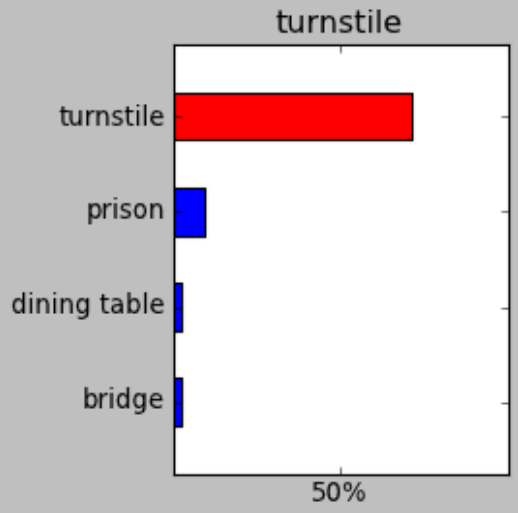


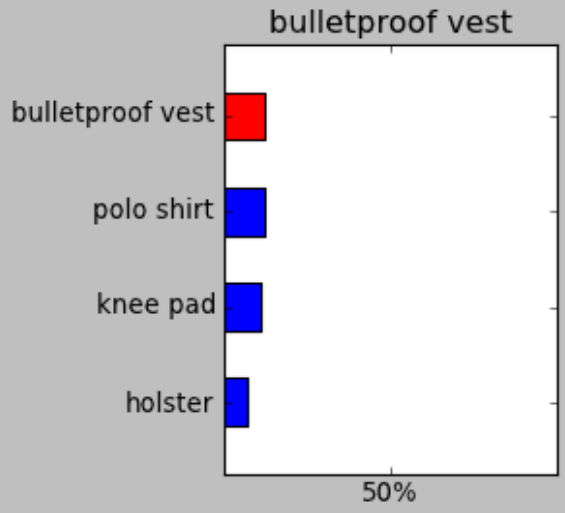




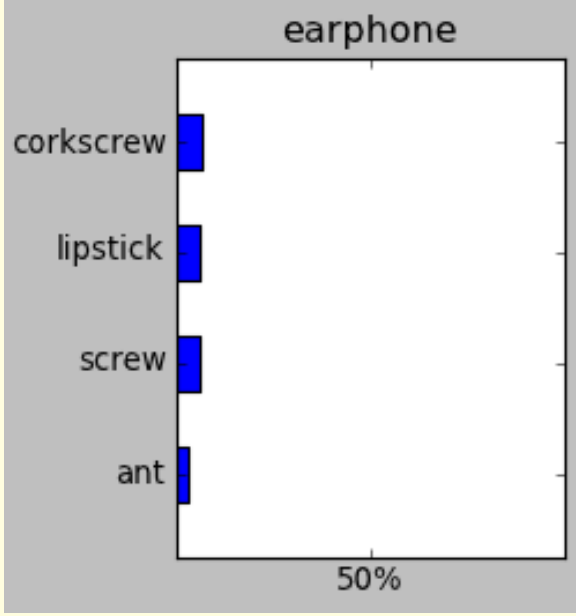
It can deal with a wide range of objects

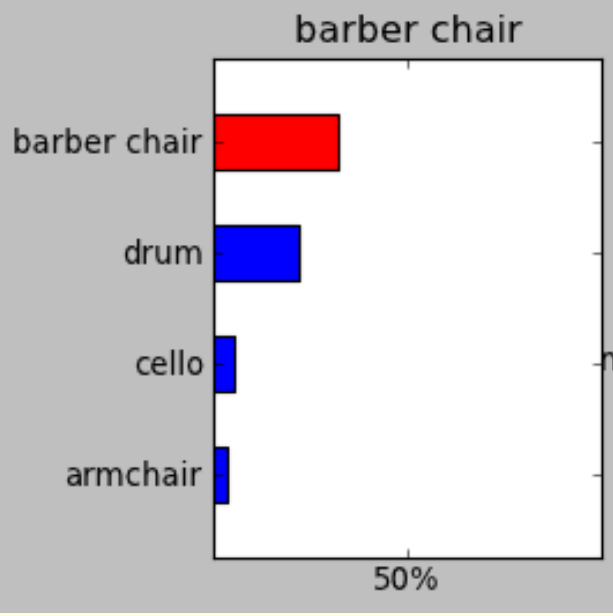


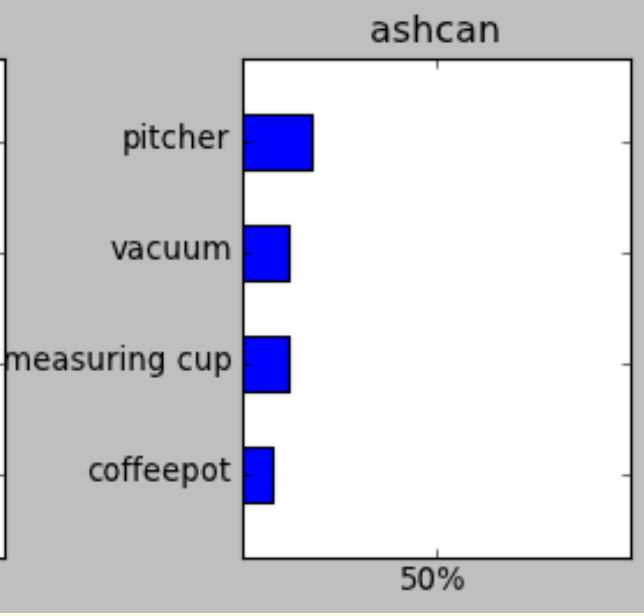




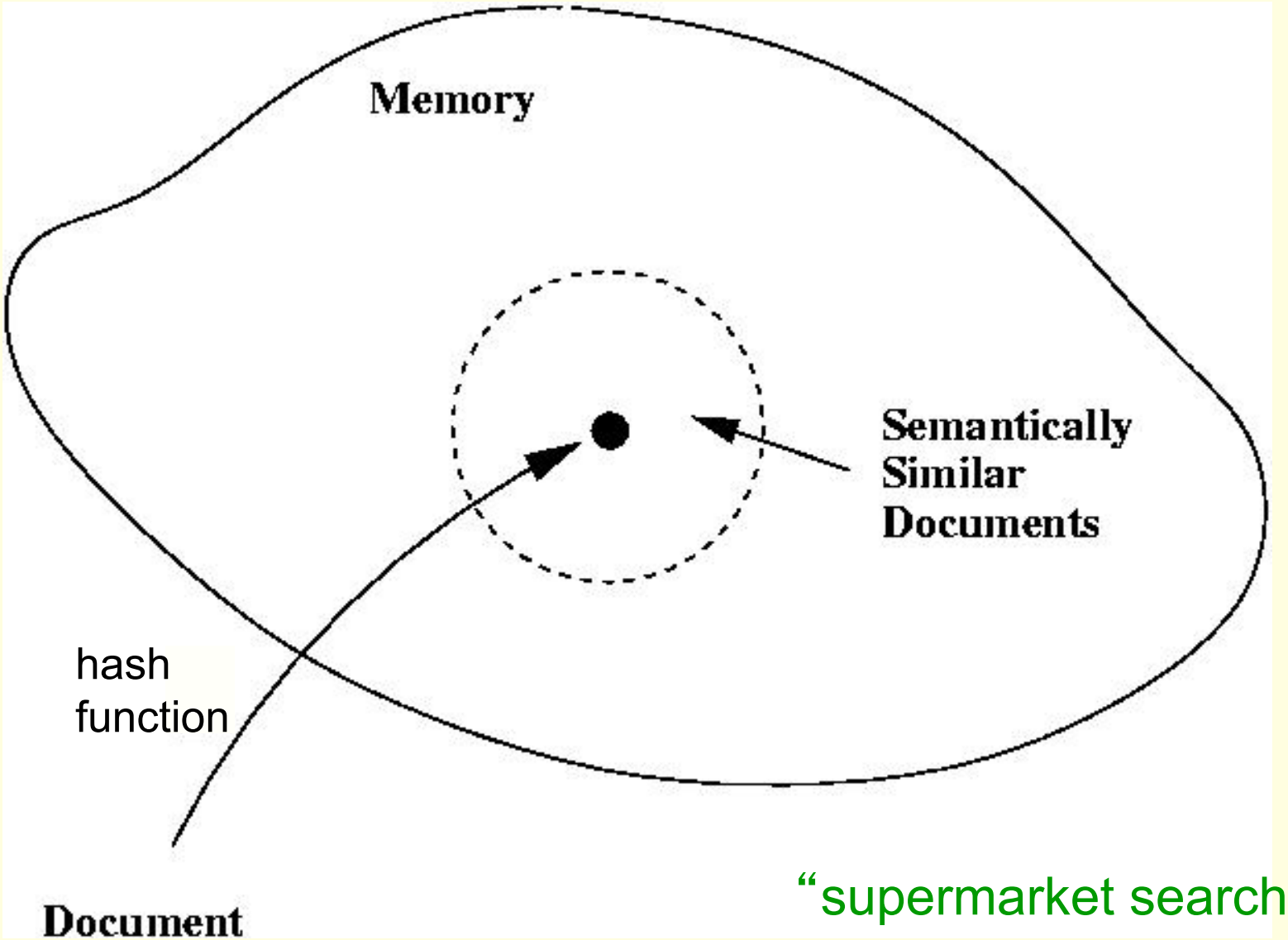
It makes some really cool errors







Using a deep autoencoder as a hash-function for finding approximate matches



"supermarket search"

Another view of semantic hashing

- Fast retrieval methods typically work by intersecting stored lists that are associated with cues extracted from the query.
- Computers have special hardware that can intersect 32 very long lists in one instruction.
  - Each bit in a 32-bit binary code specifies a list of half the addresses in the memory.
- Semantic hashing uses machine learning to map the retrieval problem onto the type of list intersection the computer is good at.

Semantic hashing for image retrieval

- Currently, image retrieval is typically done by using the captions. Why not use the images too?
  - Pixels are not like words: individual pixels do not tell us much about the content.
  - Extracting object classes from images is hard.
- Maybe we should extract a real-valued vector that has information about the content?
  - Matching real-valued vectors in a big database is slow and requires a lot of storage
- Short binary codes are easy to store and match

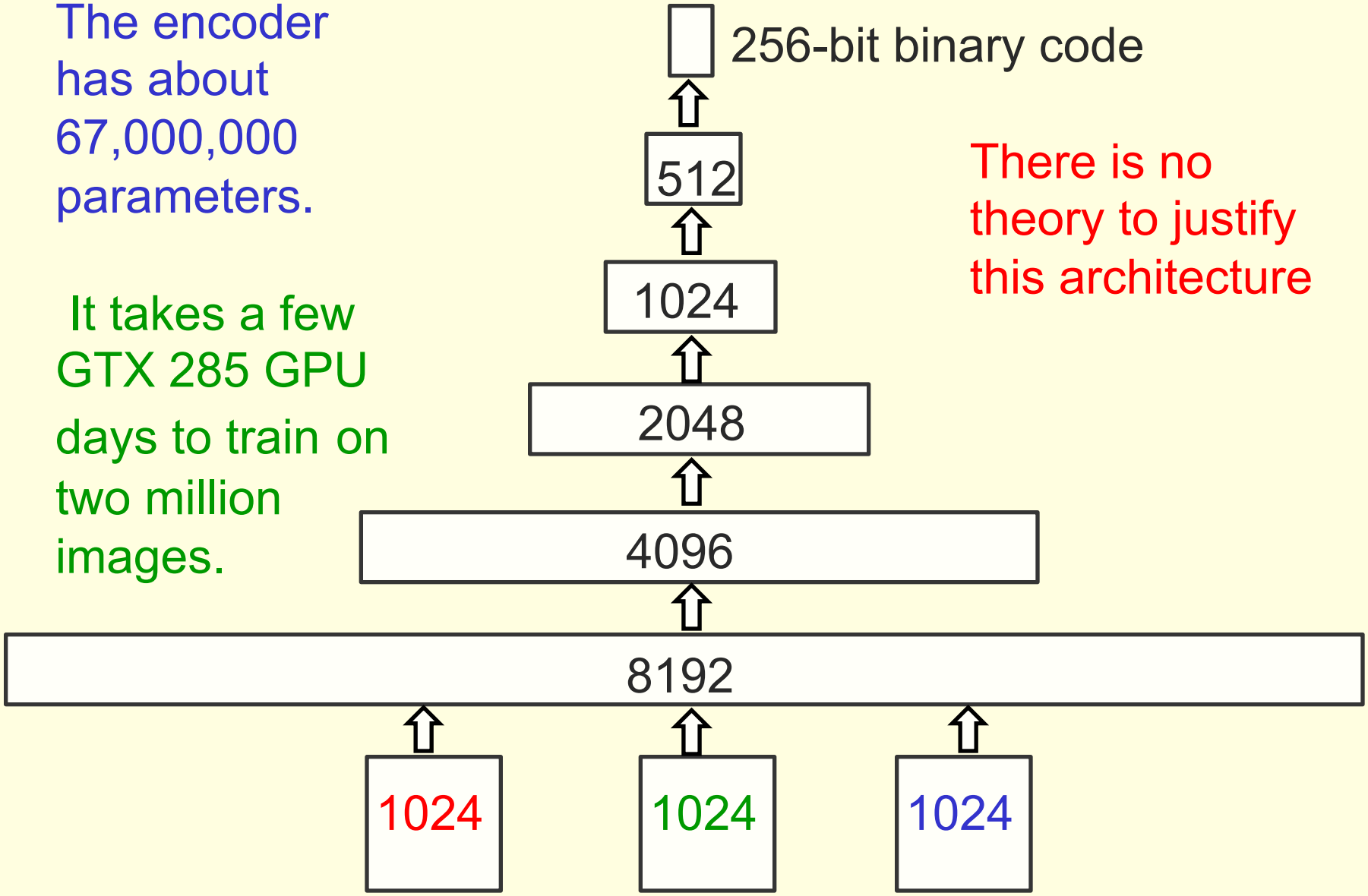
A two-stage method

- First, use semantic hashing with 30-bit binary codes to get a long “shortlist” of promising images.
- Then use 256-bit binary codes to do a serial search for good matches.
  - This only requires a few words of storage per image and the serial search can be done using fast bit-operations.
- But how good are the 256-bit binary codes?
  - Do they find images that we think are similar?

Krizhevsky's deep autoencoder

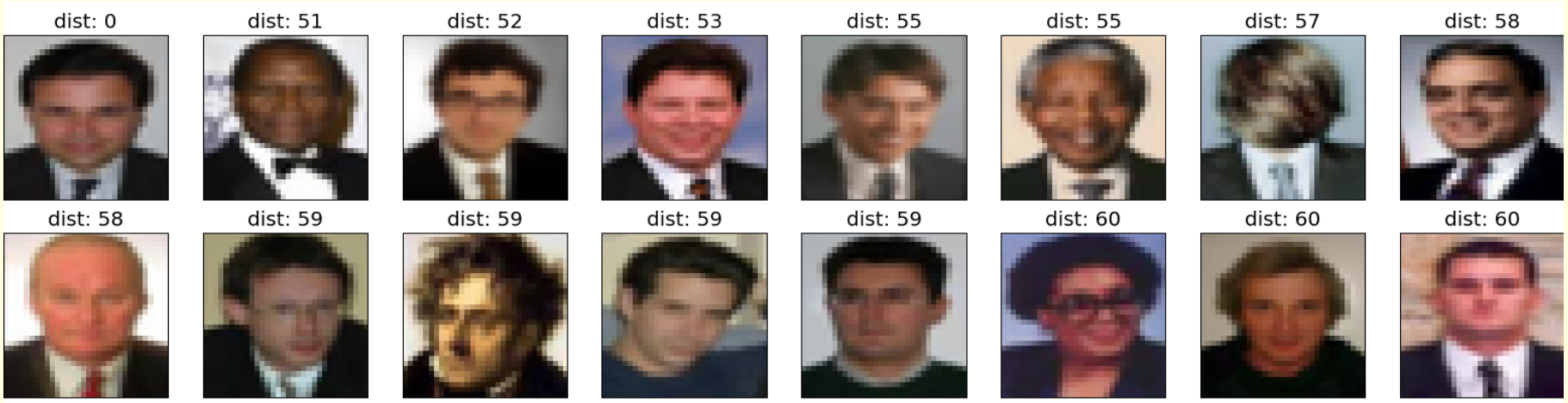
The encoder has about 67,000,000 parameters.

It takes a few GTX 285 GPU days to train on two million images.

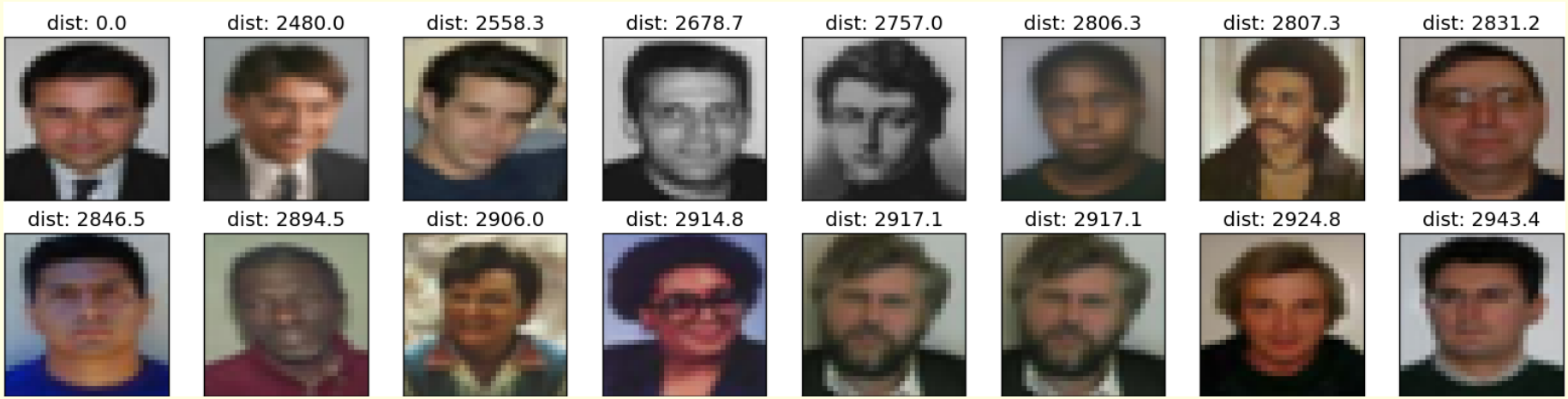


There is no theory to justify this architecture

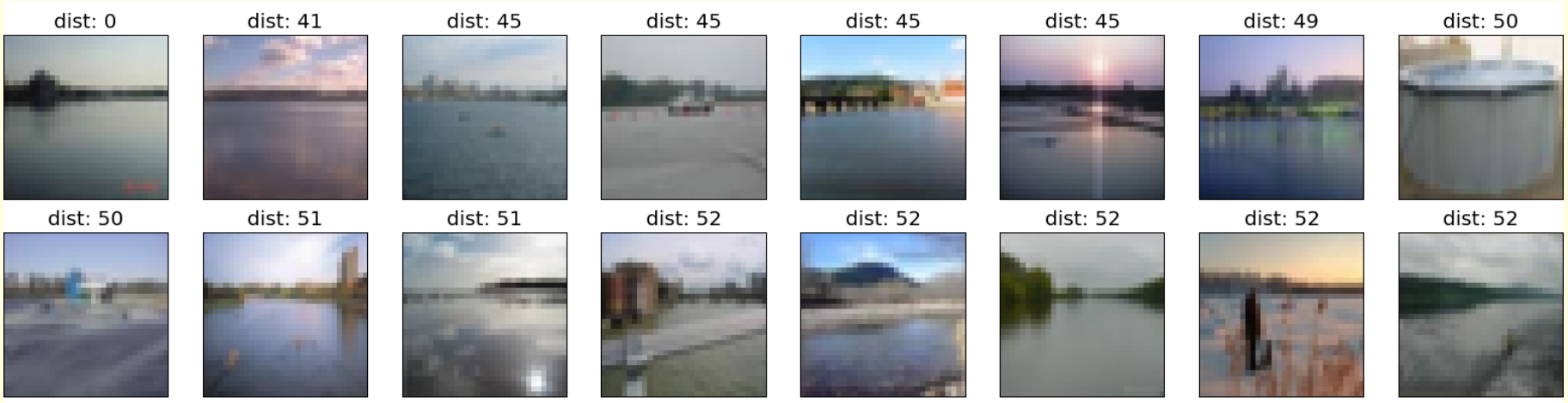
Autoencoder



Euclidean distance in pixel intensity space

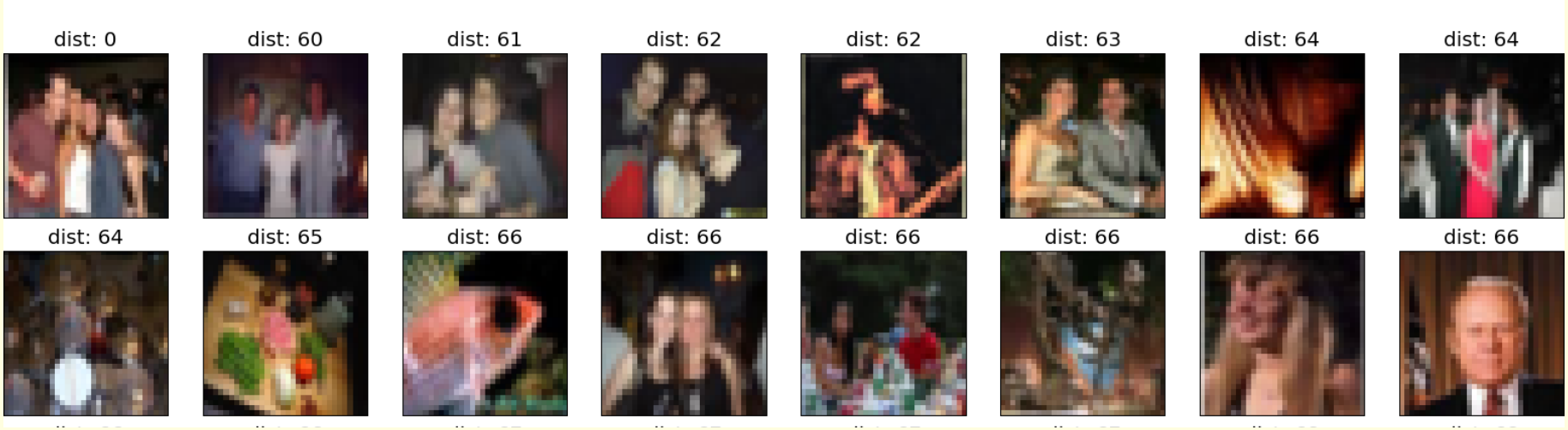


Autoencoder



Euclidean distance in pixel intensity space

Autoencoder



Euclidean distance in pixel intensity space

An obvious extension

- Use a multimedia auto-encoder that represents captions and images in a single code.
  - The captions should help it extract more meaningful image features such as "contains an animal" or "indoor image"
- RBM's already work much better than standard LDA topic models for modeling bags of words.
  - So the multimedia auto-encoder should be
    + a win (for images)
    + a win (for captions)
    + a win (for the interaction during training)

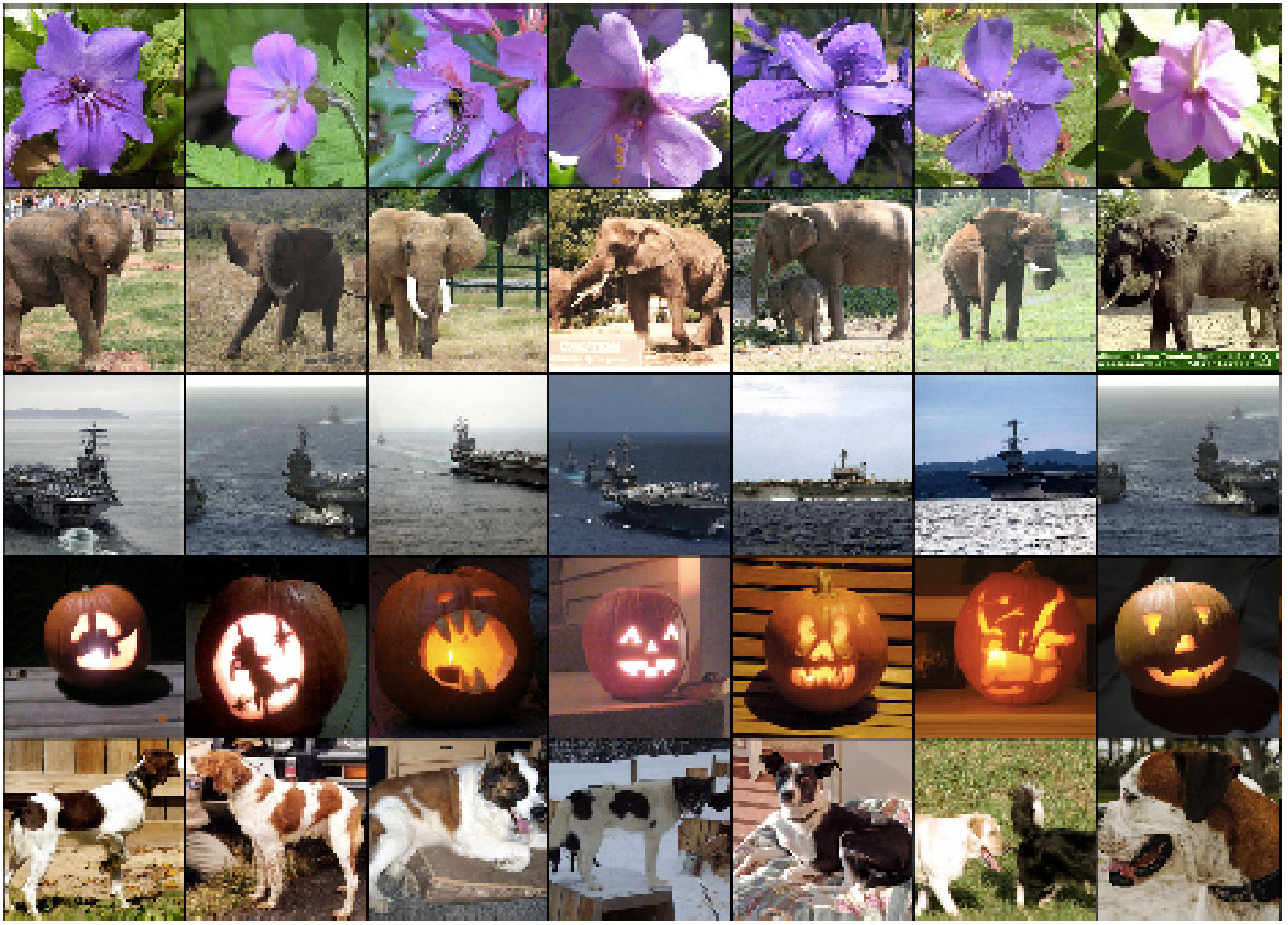
A less obvious extension

- Semantic hashing gives incredibly fast retrieval but its hard to go much beyond 32 bits.
- We can afford to use semantic hashing several times with variations of the query and merge the shortlists
  - Its easy to enumerate the hamming ball around a query image address in ascending address order, so merging is linear time.
- Apply many transformations to the query image to get transformation independent retrieval.
  - Image translations are an obvious candidate.

A better starting point for semantic hashing

- The last hidden layer of a deep net for object recognition has a lot of information about the prominent objects in the image.
  - So use the last hidden layer as the input to an autoencoder.

Images that give similar activity vectors in the last hidden layer

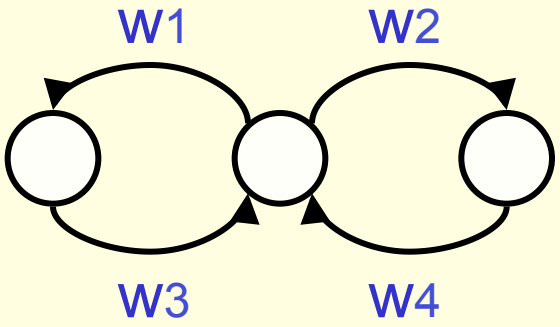


Another failure of back-propagation

The most exciting version is back-propagation through time. This should be able to learn distributed sequential "programs" to predict the next input vector.

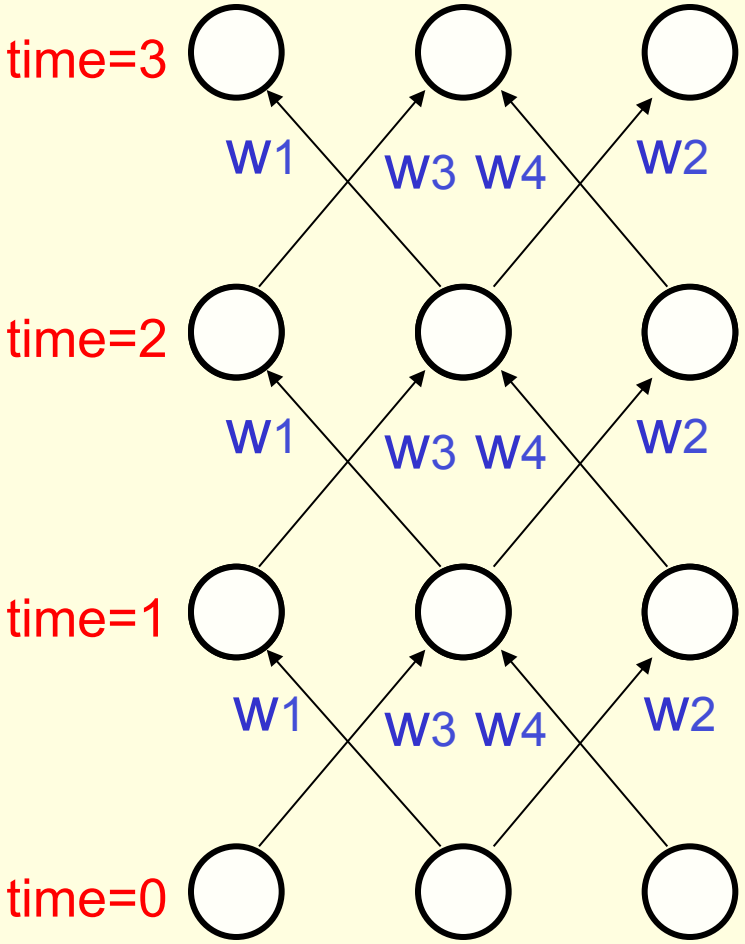
- It doesn't work properly.
- The gradient either explodes or dies.

The equivalence between layered, feedforward nets and recurrent nets



Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.



Two ways to use curvature information to improve optimization

- Quasi-Newton: Exact minimization on a very crude quadratic approximation to the curvature.

- Hessian-Free: partial minimization on a much better quadratic approximation to the curvature
  - Put a huge amount of work into coming up with a good Gauss-Newton approximation to the curvature.
  - (See ICML 2010 paper by James Martens)

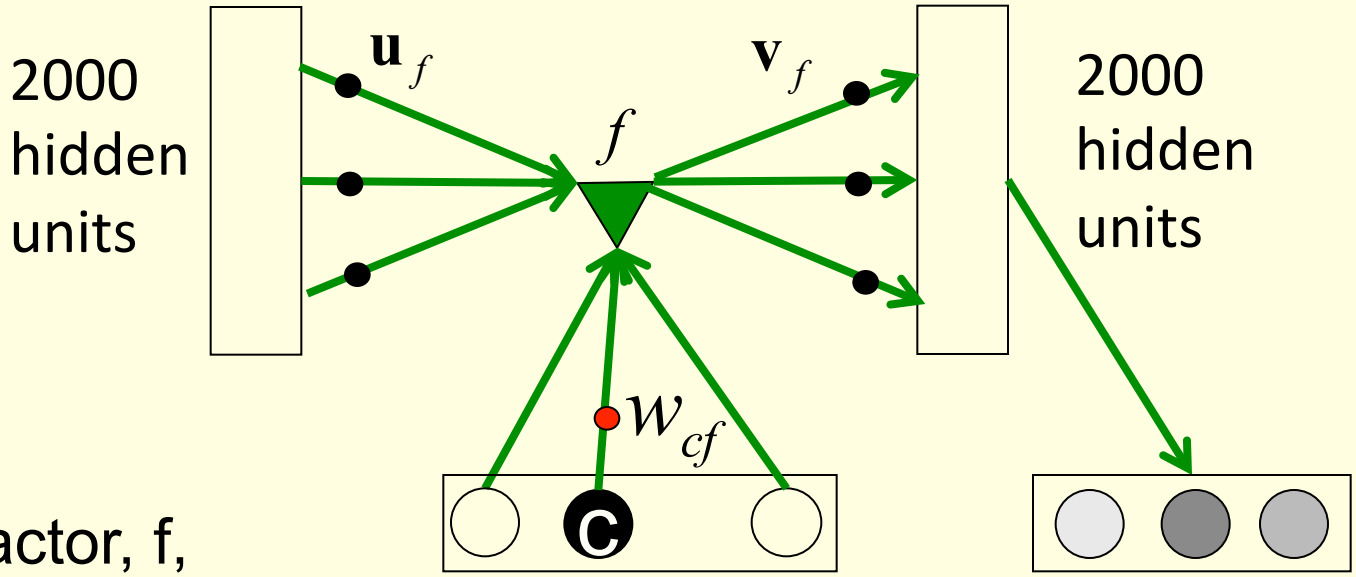
What is a word

- A word operates on a mental state to produce a new mental state.
- If we want to model the mental state as a big vector, a word needs to define a transition matrix.
  - How can each of 100,000 words define a really big transition matrix without requiring a huge number of parameters?
- Maybe we should think of a word as a sequence of characters and allow each character to define a transition matrix?

Advantages of working with characters

- The web is composed of character strings.
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true).
- Pre-processing text to get words is a big hassle
  - What about morphemes (prefixes, suffixes etc)
  - What about subtle effects like “sn” words?
  - What about New York?
  - What about Finnish
    - ymmärtämättömyydellänsäkään

Using 3-way factors to allow a character to create a whole transition matrix



Each factor, f, defines a rank one matrix , $\mathbf{u}_f \mathbf{v}_f^T$

character: 1-of-86

predicted distribution for next character.

Each character, c, determines a gain w_{cf} for each of these rank one matrices

It is a lot easier to predict 86 characters than 100,000 words.

Training the character model

- Ilya Sutskever used 5 million strings of 100 characters taken from wikipedia. For each string he starts predicting at the 11th character.
- It takes a month on a GPU board to get a really good model. It needs very big mini-batches.
- Ilya's best model is about equal to the state of the art for character prediction, but works in a very different way from the best other models.
  - It can balance quotes and brackets over long distances. Markov models cannot do this.

How to use the model to continue a character string

- Given an initial string, we could generate whatever character the net thinks is most probable.
  - This degenerates into “the United States of the United States of the United States of …”
- Its better to get the net to produce a probability distribution for the next character and then sample from this distribution.
  - We then tell the net that the character we sampled was the real next character and ask it to predict the one after that, and so on.

Some text generated by the model

In 1974 Northern Denver had been overshadowed by CNL, and several Irish intelligence agencies in the Mediterranean region. However, on the Victoria, Kings Hebrew stated that Charles decided to escape during an alliance. The mansion house was completed in 1882, the second in its bridge are omitted, while closing is the proton reticulum composed below it aims, such that it is the blurring of appearing on any well-paid type of box printer.

He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters‘ sisters in lower coil trains were always operated on the line of the ephemerable street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade."

Some completions produced by the model

- Sheila thrunges (most frequent)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6th try)

What does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers.
- It is good at balancing quotes and brackets.
  - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what form this knowledge has.
  - Its syntactic knowledge is not modular.
- It knows a lot of semantic associations
  - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable.

Completing a sentence using the neural network

(after a lot more training)

The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger.

PART 4:

A computational principle that explains sex, the brain, and sparse coding

Theme of this lecture

- An interesting discovery in machine learning provides an explanation for two puzzling phenomena in biology.

- The two phenomena appear to have nothing to do with one another, but actually there is one principal that explains both of them.

- Also, we can make neural nets work much better and explain why sparse coding improves classification.

A problem with sexual reproduction

- Fitness depends on genes working well together. But sexual reproduction breaks up sets of co-adapted genes.
  - This is a puzzle in the theory of evolution.
- A recent paper by Livnat, Papadimitriou and Feldman (PNAS 2008) claims that breaking up complex co-adaptations is actually a good thing even though it may be bad in the short term.
  - It may help optimization in the long run.
  - It may make organisms more robust to changes in the environment. We show this is a big effect.

A problem with neural communication

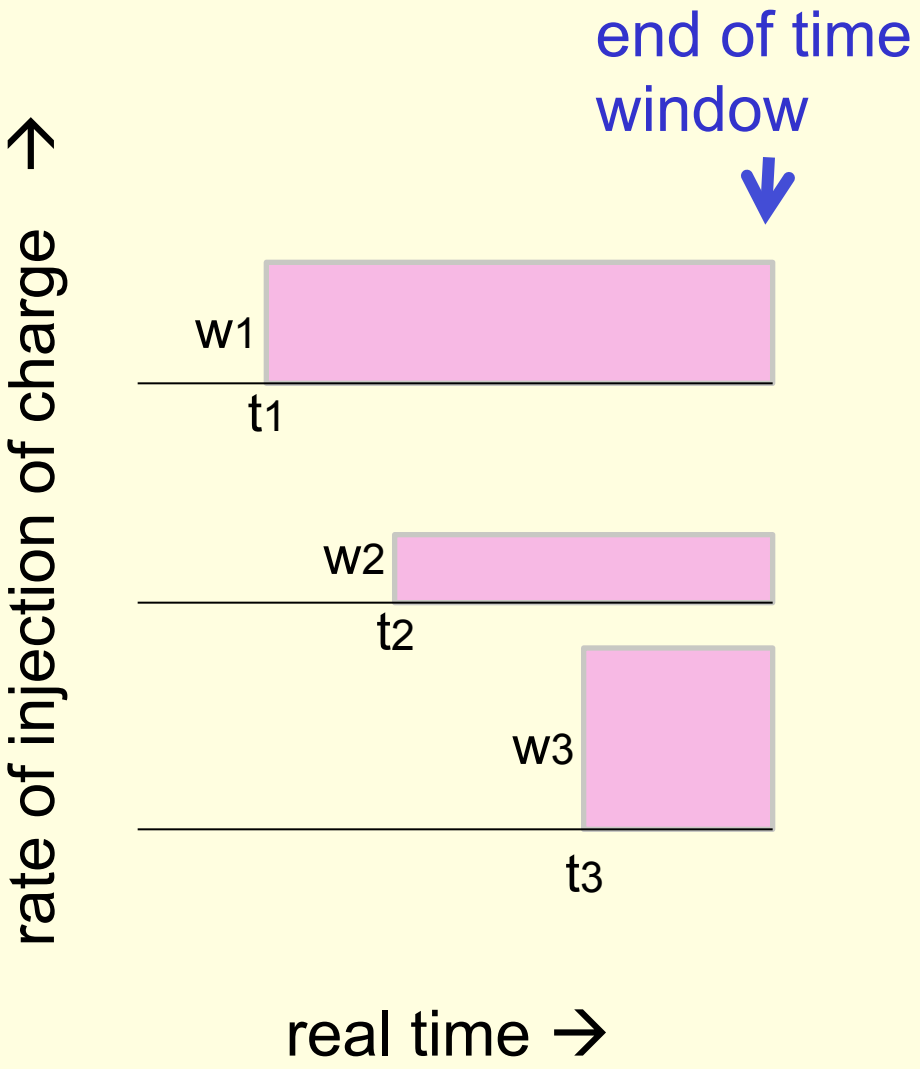
- Cortical neurons do signal processing by sending discrete spikes of activity with apparently random timing.
  - Why don't they communicate precise analog values by using the precise times of spikes.
  - Surely analog values are more useful for signal processing?

How spike timing could be used to compute scalar products

- We want to take the scalar product of a vector of activations (coded as spike times) with a vector of parameters (coded as synaptic weights).

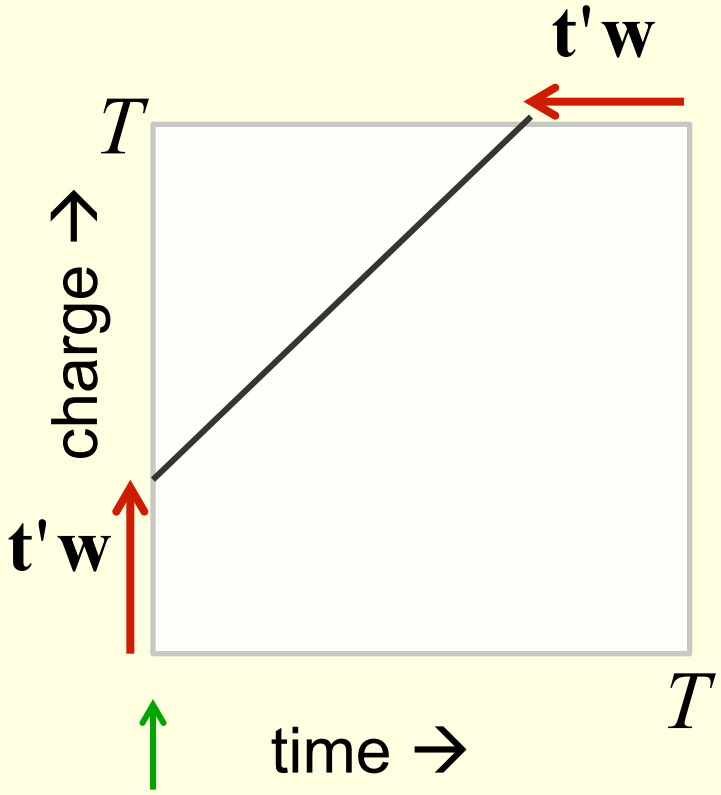
- The answer must then be converted back to a spike time.

A picture of how an integrate-and-fire neuron could compute a scalar product



- At the end of the time window, the total injected charge is the scalar product of the time advances and the weights.
- But how do we convert this back into a spike time?

Converting accumulated charge into a spike time



end of first time window

- At the end of the time window, add an additional input that injects charge at the rate:

$$1 - \sum_i w_i$$

- The total rate of injection of charge is then 1 and so the additional time taken to reach a threshold of T is: $T - \mathbf{t'w}$
- So in the next window, the advance of the outgoing spike represents the scalar product $\mathbf{t'w}$

Does the brain really make use of spike times to communicate real numbers?

- In the hippocampus of a rat, the location of the rat is represented by place cells.
  - The precise time at which a place cell fires probably indicates where the rat is within that place field.
- But there is not much evidence that spike times are used this way in sensory cortex. This leaves two main possibilities:
  - Evolution failed to discover an obvious trick.
  - Our ideas about signal processing are hopelessly wrong.

How could 1 bit possibly be better?

- Maybe the kind of signal processing problem that the brain solves is very different from the kind of signal processing problem engineers solve.
  - Engineers want to fit one model to data that they understand (e.g. a linear dynamical system).
  - The brain is confronted by a buzzing, blooming confusion. It needs to fit many different models and use the wisdom of crowds.

Why engineers need to understand the principles underlying neural communication

- If we spread a really big neural net over many cores, we have to communicate the states of the neurons.
- It would be really helpful if communicating 1 bit was actually better than sending a real number.

An apparent change of topic:

Is there anything we cannot do with very big, deep neural networks?

- It appears to be hard to do massive model averaging:
  - Each net takes a long time to learn.
  - At test time we don't want to run lots of different large neural nets.

Averaging many models

- To win a machine learning competition (e.g. Netflix) you need to use many different types of model and then combine them to make predictions at test time.

- Decision trees are not very powerful models, but they are easy to fit to data and very fast at test time.
  - Averaging many decision trees works really well. Its called random forests. Kinect uses this.
  - We make the individual trees different by giving them different training sets. That's called bagging

Two ways to average models

- Mixture: We can combine models by taking the arithmetic means of their output probabilities:

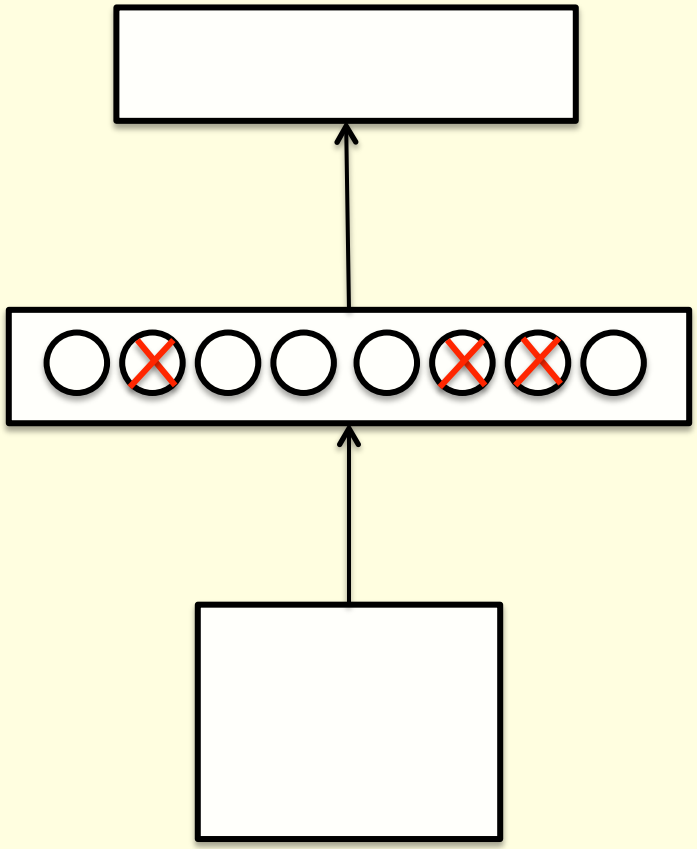
Model A:	.3	.2	.5
Model B:	.1	.8	.1
Combined	.2	.5	.3

- Product: We can combine models by taking the geometric means of their output probabilities:

Model A:	.3	.2	.5	
Model B:	.1	.8	.1	
Combined	$\sqrt{.03}$	$\sqrt{.16}$	$\sqrt{.05}$	/sum

Dropout: An efficient way to average many large neural nets.

- Consider a neural net with one hidden layer.
- Each time we present a training example, we randomly omit each hidden unit with probability 0.5.
- So we are randomly sampling from 2^H different architectures.
  - All architectures share weights.



Dropout as a form of model averaging

- We sample from 2^H models. So only a few of the models ever get trained, and they only get one training example.
  - This is as extreme as bagging can get.
- The sharing of the weights means that every model is very strongly regularized.
  - It's a much better regularizer than L2 or L1 penalties that pull the weights towards zero.

But what do we do at test time?

- We could sample many different architectures and take the geometric mean of their output distributions.
- It better to use all of the hidden units, but to halve their outgoing weights.
  - This exactly computes the geometric mean of the predictions of all 2^H models.

What if we have more hidden layers?

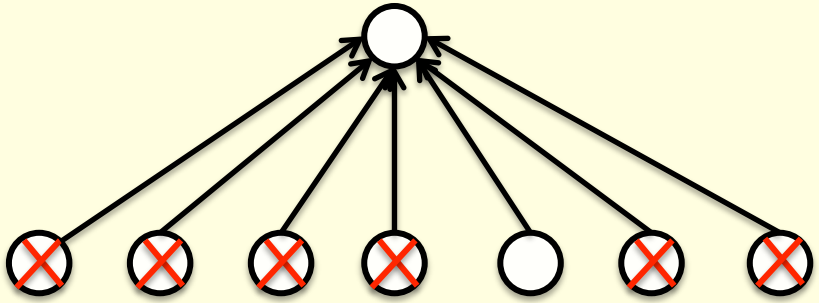
- Use dropout of 0.5 in every layer.
- At test time, use the “mean net” that has all the outgoing weights halved.
- This is not exactly the same as averaging all the separate dropped out models, but it’s a pretty good approximation, and its fast.

What about the input layer?

- It helps to use dropout there too, but with a higher probability of keeping an input unit.
  - This trick is already used by the “denoising autoencoders” developed in Yoshua Bengio’s group.

A familiar example of dropout

- Do logistic regression, but for each training case, dropout all but one of the inputs.



- At test time, use all of the inputs.
  - Its better to divide the learned weights by the number of features, but if we just want the best class its unnecessary.
- This is called “Naïve Bayes”.
  - Why keep just one input?

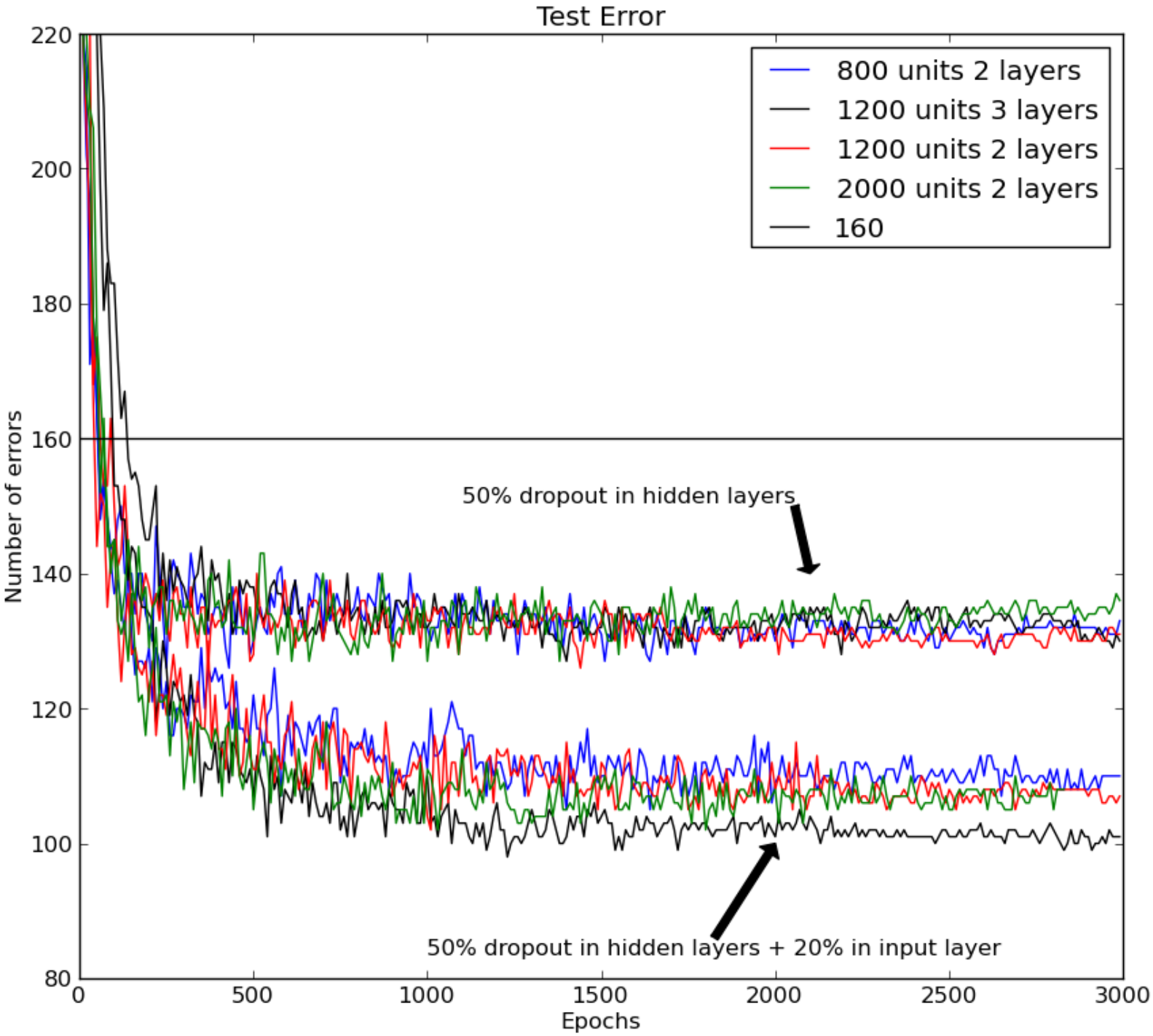
How well does dropout work?

- If your deep neural net is significantly overfitting, it will reduce the number of errors by a lot.
  - Any net that uses “early stopping” can do better by using dropout (at the cost of taking quite a lot longer to train).

- If your deep neural net is not overfitting you should be using a bigger one.

Initial experiments on permutation invariant MNIST (Nitish Srivastava)

- MNIST is a standard machine learning benchmark.
- It has 60,000 training images of hand-written digits and 10,000 test images.
- There are many ways of improving performance:
  - Put in prior knowledge of geometry
  - Add extra training data by transforming the images.
- Without using these tricks, the record for neural nets was 160 errors on the test set.

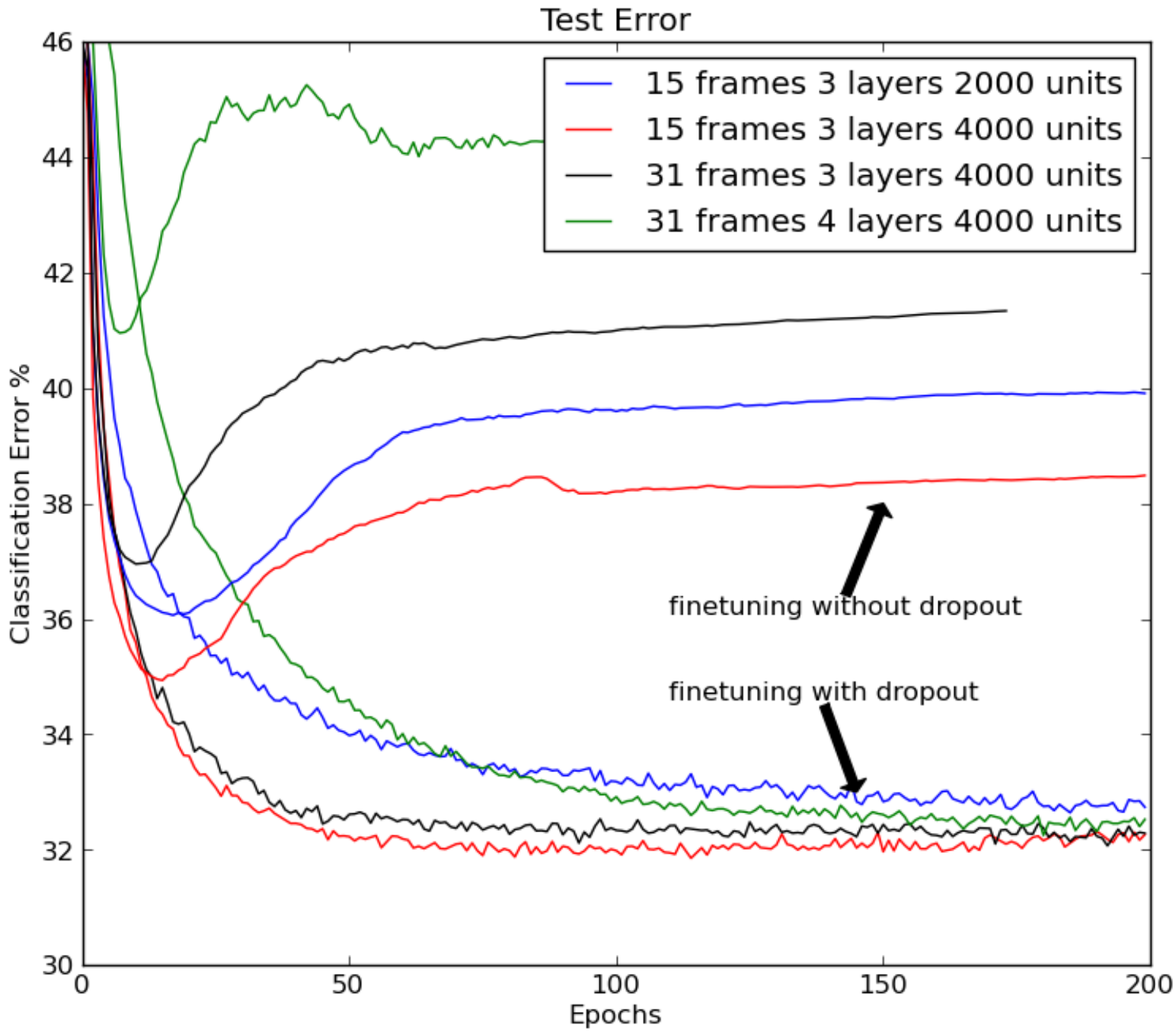

Test Error
800 units 2 layers
1200 units 3 layers
1200 units 2 layers
2000 units 2 layers
160
50% dropout in hidden layers
50% dropout in hidden layers + 20% in input layer
Number of errors
Epochs
220
200
180
160
140
120
100
80
0
500
1000
1500
2000
2500
3000

Using weight constraints

- In neural nets, it is standard to use an L2 penalty on the weights (called weight-decay).
  - This improves generalization by keeping the weights small.
- It is generally better to constrain the length of the incoming weight vector of each hidden unit.
  - If the weight vector becomes longer than allowed, the weights are renormalized by division.
- Weight constraints make it possible to use a very big initial learning rate that then decays.

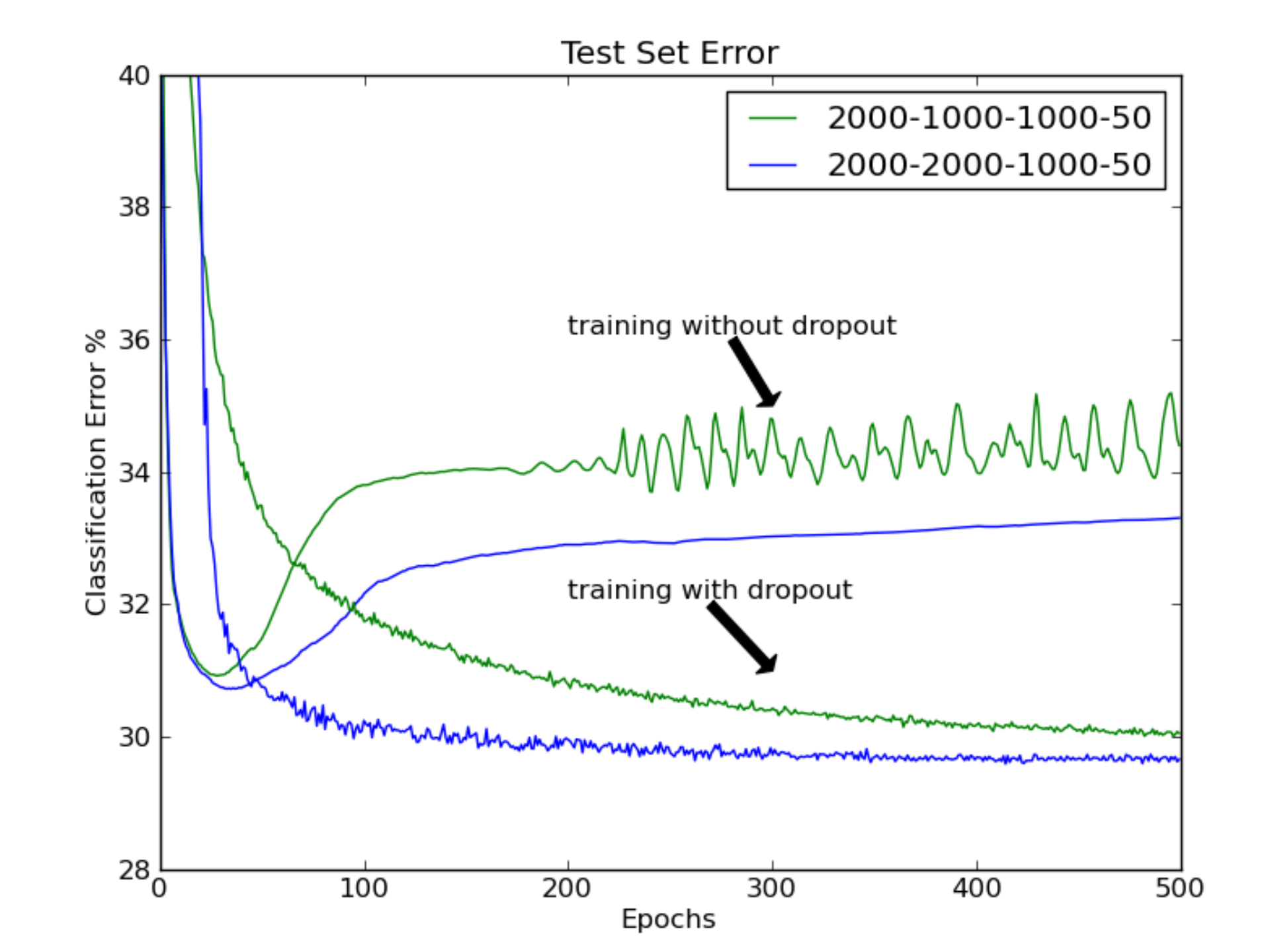
Experiments on TIMIT (Nitish Srivastava)

- First pre-train a deep neural network one layer at a time on unlabeled windows of acoustic coefficients.
- Then fine-tune to discriminate between the classes using a small learning rate.
- Standard fine-tuning: 22.7% error on test set
- Dropout fine-tuning: 19.7% error on test set
  - This is a record for speaker-independent methods.


Test Error
15 frames 3 layers 2000 units
15 frames 3 layers 4000 units
31 frames 3 layers 4000 units
31 frames 4 layers 4000 units
finetuning without dropout
finetuning with dropout
Classification Error %
Epochs
46
44
42
40
38
36
34
32
30
0
50
100
150
200

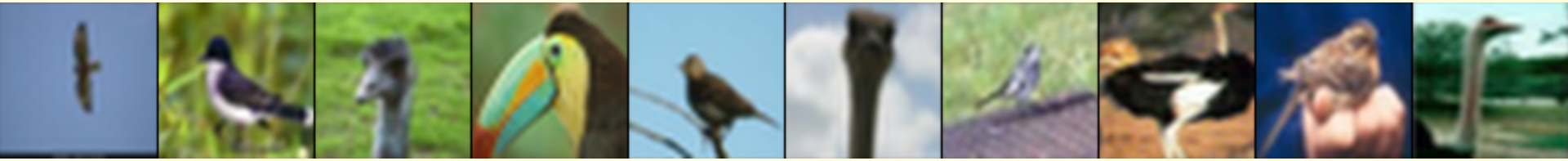
Experiment on document classification (Nitish Srivastava & Ruslan Salakhutdinov)

- Represent a document by a vector of counts of the 2000 most frequent non-stop words.
- Use a subset of the documents from 50 non-overlapping classes.
- Predict the class from the word count vector using a net with two hidden layers.


Test Set Error
2000-1000-1000-50
2000-2000-1000-50
training without dropout
training with dropout
Classification Error %
Epochs
40
38
36
34
32
30
28
0
100
200
300
400
500

Experiment on CIFAR-10

- Benchmark task for object recognition:
  - 10 classes, 32x32 color images downsampled from web and carefully hand-labeled.
  - 50,000 training cases, 10,000 test cases.



  - A big convolutional net gets 18% error.
  - With dropout in the last layer it gets 16% error

Another way to think about dropout

- If a hidden unit knows which other hidden units are present, it can co-adapt to them on the training data.
  - But complex co-adaptations are likely to go wrong on new test data.
  - Big, complex conspiracies are not robust.
- If a hidden unit has to work well with combinatorially many sets of co-workers, it is more likely to do something that is individually useful, but also marginally useful given what its co-workers typically achieve.

A simple example for a linear model

- Training data:

  1, 1, 0 0 → 6

  1, 1, 1 1 → 4

  -5 +11 +4 -6 co-adapted weights

  +3, +3, -1, -1 less co-adapted weights

Noise versus model averaging

- Here are two apparently different ways to improve generalization:
  - Method 1: Regularize by adding noise to the weights or neural activities (equivalent to weight penalties).
  - Method 2: Average the predictions of many different models.

- Dropout is an example of both. So these methods are not as different as they appear.

Sparse coding

- If we use a large “dictionary” of hidden units and code an image by only activating a few hidden units, we get an efficient code.
  - This is often done by using an L1 penalty on the hidden activities and then iterating to drive many of the activities to zero.

- One big advantage of sparse coding is that the sparse codes are usually good for predicting class labels.
  - Why is this? Is it because each active hidden unit is more informative?
  - We now know that randomly setting most of the activities to zero allows us to learn very good codes.
  - So maybe sparse coding produces good codes for classification precisely because its unstable: A hidden unit does not know which other units will be present.

Another advantage of using dropout

- Dropout forces neurons to be robust to the loss of co-workers.
- This should make “genetic algorithms” work better.
- Two nets produce an offspring by randomly picking each hidden unit to come from one or other parent (we need to know the correspondence of hidden units).
  - The offspring already works quite well and after some training it will work about as well as the parents.

A simple way to run on a cluster
(suggested by Inman Harvey)

- Every so often, a “mother” network advertises for a mate. After considering various possible mates, the mother network produces an offspring.
- The mother network is then suspended and that core is used to run the child network.
- After a while we compare the child with the mother and decide which one to kill.
- This algorithm requires very little interaction.
  - It could happily use a million cores.
  - Nets need to change gender frequently.

Now for something not completely different

An alternative to dropout

- In dropout, each neuron computes an activity, p, using the logistic function. Then it sends p to the next layer with a probability of 0.5.
- This has exactly the same expected value as sending 0.5 with probability p.
  - That is exactly what a stochastic binary neuron does (if we call 0.5 one spike)
  - So what happens if we use stochastic binary neurons in the forward pass but do the backward pass as if we had done a “normal” forward pass?

The effect of only sending one bit

- The deep neural network learns slower and gets more errors on the training data.
  - But it generalizes much better.
  - Its about the same win as dropout, but we have not properly compared them yet.

- Dropout variance = $p^2/4$
- Stochastic bit variance = $p(1-p)/4$
  - Stochastic bits have more variance for small p.
  - This is the Poisson limit and resembles neurons

An amusing piece of history

- In 2005 we discovered that deep nets can be pre-trained effectively on unlabeled data by learning a stack of "Restricted Boltzmann Machines" (see my 2007 Youtube Techtalk for details).
- The pre-training uses stochastic binary units. After pre-training we cheat and use backpropagation by pretending that they are deterministic units that send the real-valued outputs of logistics.
  - We would get less overfitting if we stayed with stochastic binary neurons in the forward pass.

Some explanations for why cortical neurons don't send analog values

- There is no efficient way for them to do it.
  - But some neurons use the precise times of spikes very effectively.
- Evolution just didn't figure it out.
  - Evolution had hundreds of millions of years. If neurons wanted to send analog values, evolution would have found a way.
- Its better to send stochastic spikes because they act as a great regularizer.
  - This helps the brain to use a lot of neurons without overfitting (10^14 parameters,10^9 seconds)

Another look at Restricted Boltzmann Machines

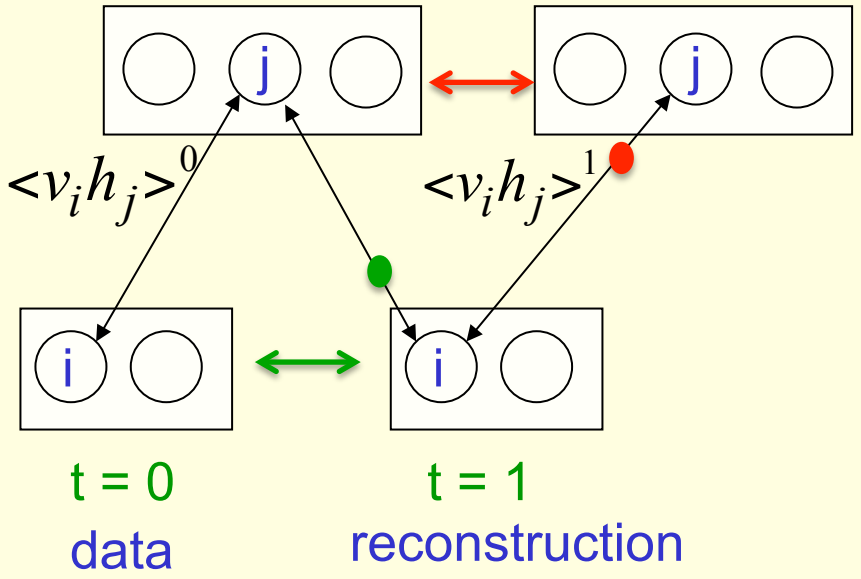
- When pre-training a deep net, we can use one-step Contrastive Divergence as a shortcut to make maximum likelihood training faster.
  - But, for pre-training, CD actually works better than proper maximum likelihood training of the RBM.
- So maybe there are better ways to understand what CD training is achieving.
  - Maybe its fitting a type of auto-encoder!

Autoencoders vs RBMs trained with ML

- An autoencoder tries to make the reconstruction match the data.
- An RBM trained with ML tries to make the distribution of the reconstructions match the distribution of the data.
  - So the RBM is happy to sometimes reconstruct A as B provided it also sometimes reconstructs B as A.
- Consider a pixel that is pure noise.
  - The autoencoder will use its hidden state to code the value of the noise so that it can reconstruct it.
  - An RBM trained with ML will totally ignore that pixel because the bias of the pixel can get the distribution correct.

A picture of CD training

One term changes the generative weight. The other term changes the recognition weight.



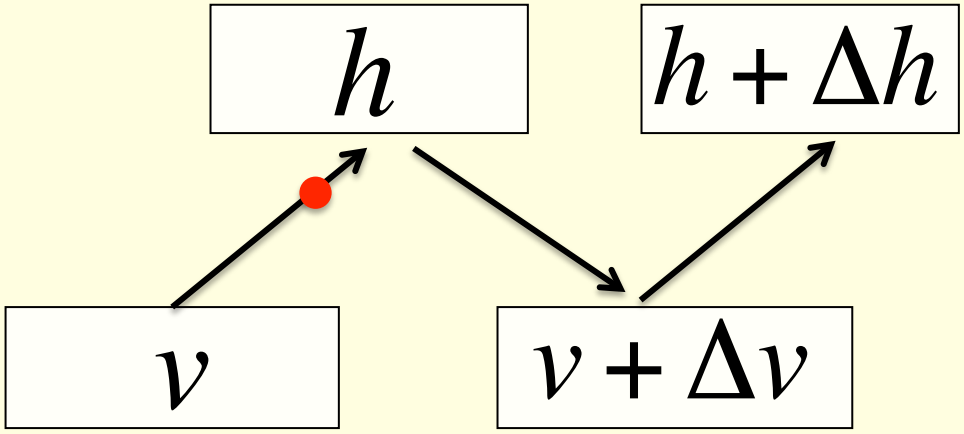
$$s_j^0(s_i^0 - s_i^1) +$$

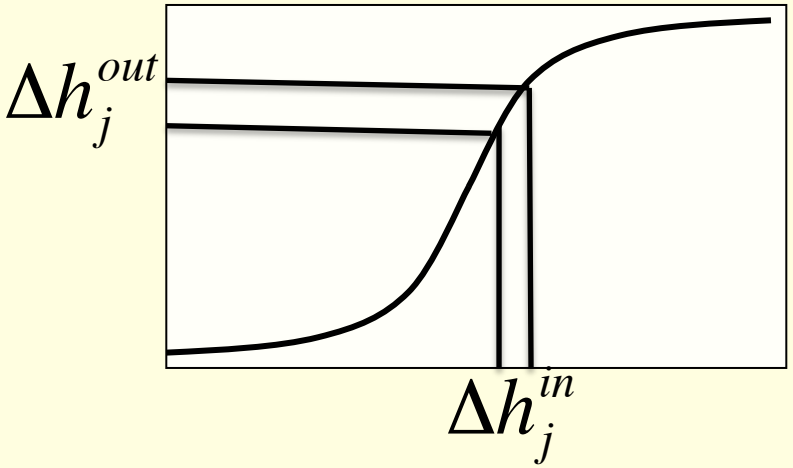
$$s_i^1(s_j^0 - s_j^1) +$$

$$s_j^1(s_i^1 - s_i^2) + \ldots$$

$$s_j^\infty s_i^\infty$$

How CD learning back-propagates through stochastic neurons





Assume that Δv is small.

To first order, Δh is the derivative of the reconstruction error w.r.t. the inputs to the hidden units.

An improved version of Contrastive Divergence learning for density modeling

- The main worry with CD is that there will be deep minima of the energy function far away from the data.
  - To find these we need to run the Markov chain for a long time (maybe thousands of steps).
  - But we cannot afford to run the chain for too long for each update of the weights.
- Maybe we can run the same Markov chain over many weight updates? (Neal, 1992)
  - If the learning rate is very small, this should be equivalent to running the chain for many steps and then doing a bigger weight update.

Persistent CD
(Tijmen Teileman, ICML 2008 & 2009)

- Use minibatches of 100 cases to estimate the first term in the gradient. Use a single batch of 100 fantasies to estimate the second term in the gradient.

- After each weight update, generate the new fantasies from the previous fantasies by using one alternating Gibbs update.
  - So the fantasies can get far from the data.

A puzzle

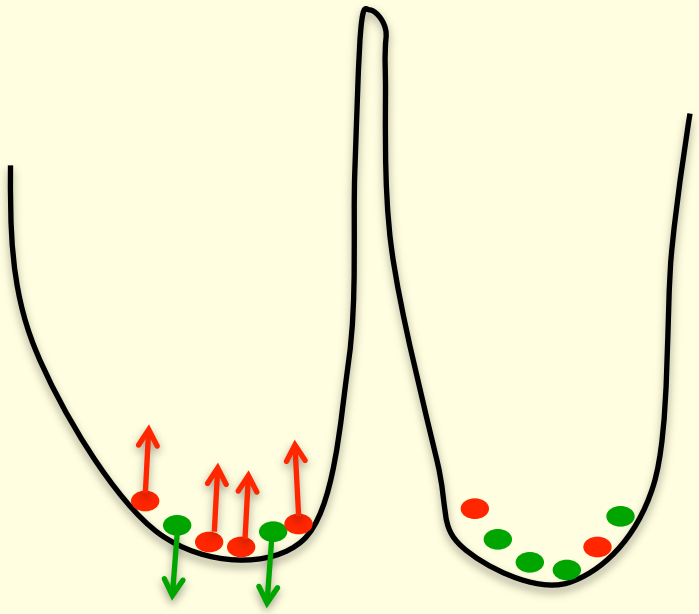
- Why does persistent CD work so well with only 100 negative examples to characterize the whole partition function?
  - For all interesting problems the partition function is highly multi-modal.
  - How does it manage to find all the modes without starting at the data?

The learning causes very fast mixing

- The learning interacts with the Markov chain.
- Persisitent Contrastive Divergence cannot be analysed by viewing the learning as an outer loop.
  - Wherever the fantasies outnumber the positive data, the free-energy surface is raised. This makes the fantasies rush around hyperactively.

How persistent CD moves between the modes of the model's distribution

- If a mode has more fantasy particles than data, the free-energy surface is raised until the fantasy particles escape.
  - This can overcome free-energy barriers that would be too high for the Markov Chain to jump.
- The free-energy surface is being changed to help mixing in addition to defining the model.



Fast PCD (Tieleman & Hinton 2009)

- To settle on a good set of weights, it helps to turn down the learning rate towards the end of learning.
- But with a small learning rate, we dont get the fast mixing of the fantasy particles.
- In addition to the “real” weights that define the model, we could have temporary weights that learn fast and decay fast.
- The fast weights provide an additive overlay that achieves fast mixing even when the real weights are hardly changing.

Training a multilayer Boltzmann machine

- For a full Boltzmann machine (i.e. with connections between hidden units), we cannot use variational learning because one of the terms has the wrong sign.

$$\frac{\partial \log p(data)}{\partial w_{ij}} = < s_i s_j >_{data} - < s_i s_j >_{model}$$

- Variational learning maximizes the sum over all training cases of:

$$\log p(data) - KL(Q||P)$$

approximate posterior

true posterior

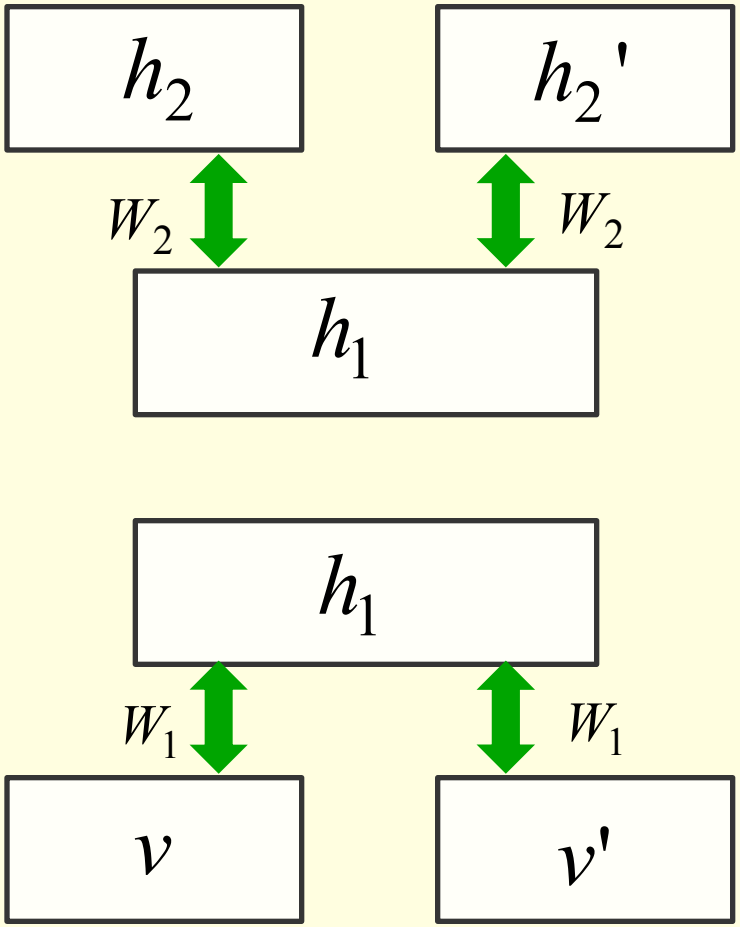
How to train a Boltzmann machine

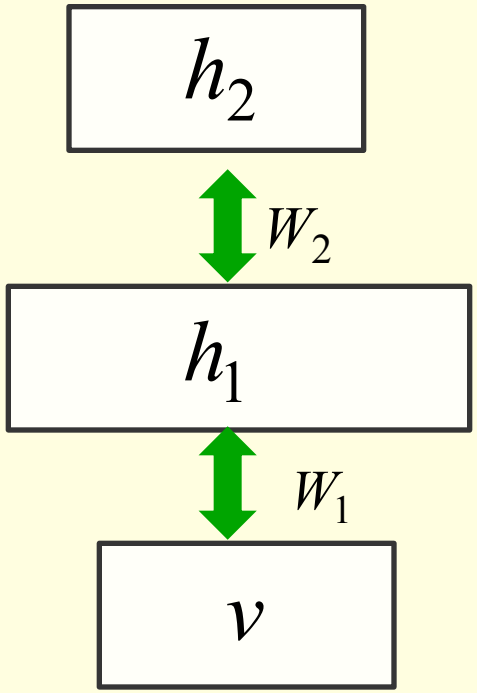
- For the data-dependent expectations, assume the energy landscape is unimodal and use a mean-field approximation to the posterior.
- For the model's expectations uses PCD.
- Ruslan Salakhutdinov showed that this worked, but it works much better if the weights of the hidden units are initialized sensibly.

How to pre-train a deep Boltzmann machine
(Salakhutdinov & Hinton, Neural Computation, 2012)

- In a DBN, each RBM replaces the prior over the previous hidden layer (that is implicitly defined by the lower RBM) by a better prior.
- Suppose we just replace half of the prior defined by the lower RBM by half of a better prior defined by the higher RBM.
  - The new prior is then the geometric mean of the priors defined by the two RBMs
  - The geometric mean is a better prior than the old one due to the convexity of KL divergence.

Combining two RBMs to make a DBM





Each of these two RBMs is a product of two identical experts

Readings on deep belief nets

A reading list (that is still being updated) can be found at

www.cs.toronto.edu/~hinton/deeprefs.html