Dark knowledge

Geoffrey Hinton, Oriol Vinyals & Jeff Dean

Google Inc.

The conflicting constraints of learning and using

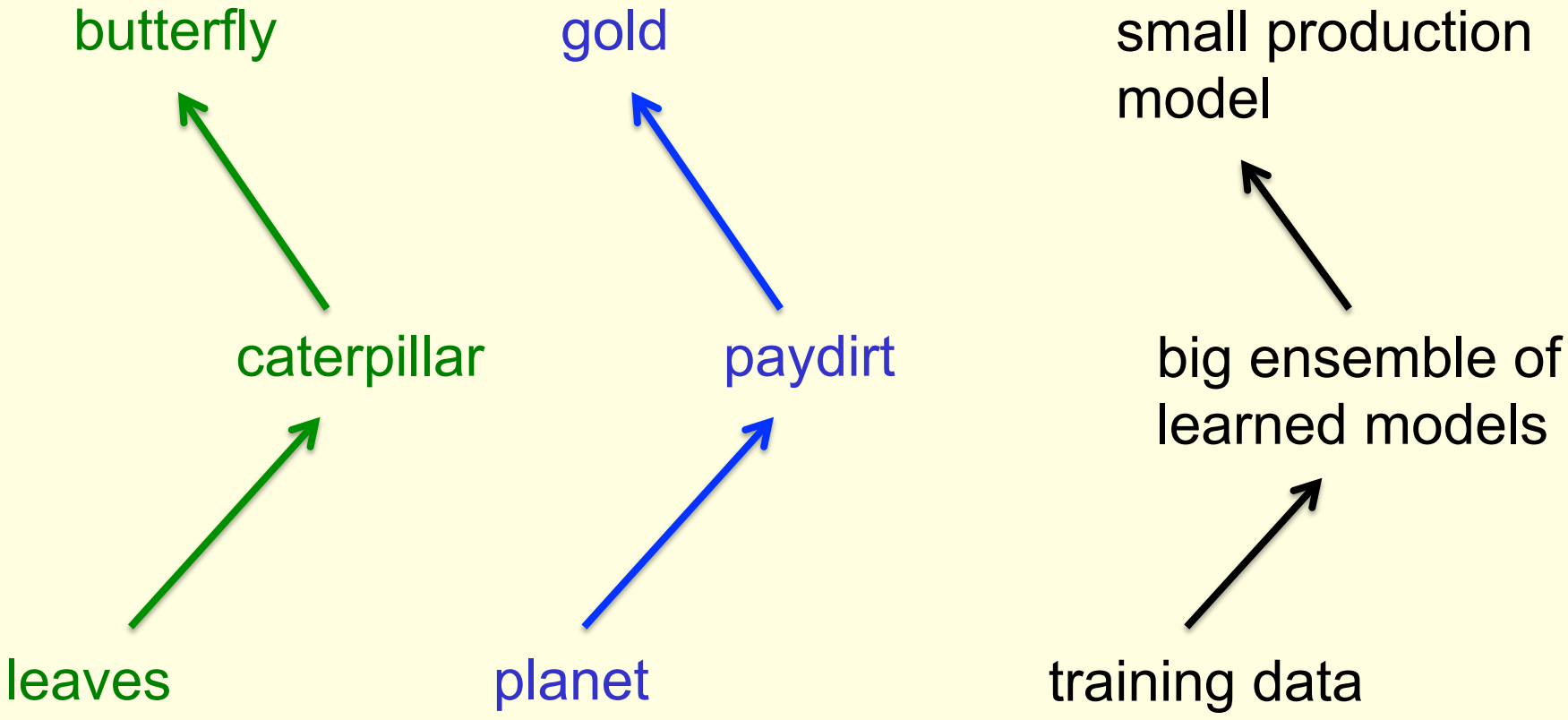
- The easiest way to extract a lot of knowledge from the training data is to learn many different models in parallel.
  - We want to make the models as different as possible to minimize the correlations between their errors.
  - We can use different initializations or different architectures or different subsets of the training data.
  - It is helpful to over-fit the individual models.
- A test time we average the predictions of all the models or of a selected subset of good models that make different errors.
  - That's how almost all ML competitions are won (e.g. Netflix)

Why ensembles are bad at test time

- A big ensemble is highly redundant. It has very very little knowledge per parameter.

- At test time we want to minimize the amount of computation and the memory footprint.
  - These constraints are generally much more severe at test time than during training.

An analogy


butterfly
gold
small production model
caterpillar
paydirt
big ensemble of learned models
leaves
planet
training data

The main idea

- The ensemble implements a function from input to output. Forget the models in the ensemble and the way they are parameterized and focus on the function.
  - After learning the ensemble, we have our hands on the function.
  - Can we transfer the knowledge in the function into a single smaller model?
- Caruana et. al. 2006 had the same idea but used a different way of transferring the knowledge.

Soft targets: A way to transfer the function

- If the output is a big N-way softmax, the targets are usually a single 1 and a whole lot of 0's.
  - On average each target puts at most log N bits of constraint on the function.
- If we have the ensemble, we can divide the averaged logits from the ensemble by a "temperature" to get a much softer distribution.

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

This reveals much more information about the function on each training case.

An aside: Two ways to average models

- We can combine models by averaging their output probabilities:

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

- 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

An example of hard and soft targets

cow	dog	cat	car
0	1	0	0

original hard targets

cow	dog	cat	car
10^{-6}	.9	.1	10^{-9}

output of geometric ensemble

cow	dog	cat	car
.05	.3	.2	.005

softened output of ensemble

Softened outputs reveal the dark knowledge in the ensemble.

Adding in the true targets

- If we just train the final model on the soft targets from the ensemble, we do quite well.
- We learn fast because each training case imposes much more constraint on the parameters than a single hard target.
- But it works better to fit both the hard targets and the soft targets from the ensemble.

How to add hard targets during distillation

- We try to learn logits in the distilled model that minimize the sum of two different cross entropies.
- Using a high temperature in the softmax, we minimize the cross entropy with the soft targets derived from the ensemble at high temperature.
- Using the very same logits at a temperature of 1, we minimize the cross entropy with the hard targets.

Relative weighting of the hard and soft cross entropies

- The derivatives for the soft targets tend to be much smaller.
  - They also have much less variance from case to case.

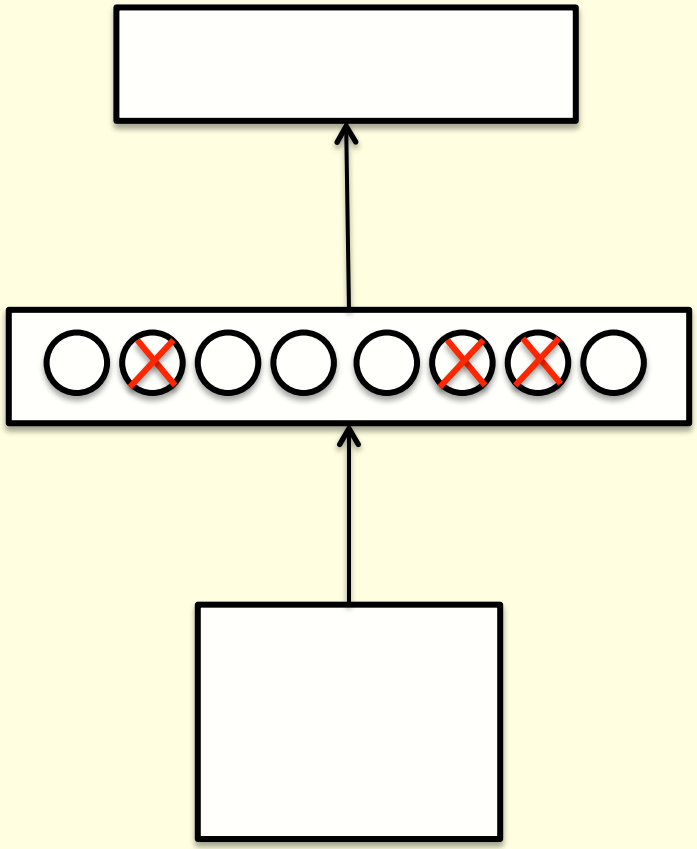
- So we down-weight the cross entropy with the hard targets.
  - Even though its down-weighted, this extra term is important for getting the best results.

Training an ensemble of models

- This is a beautiful problem to parallelize.
- But if we do not want to use a lot of cores, is there a way to train an ensemble that involves much less total computation?

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.
- 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.
  - It pulls the weights towards what other models want.

But what do we do at test time?

- We could sample many different architectures and take the geometric mean of their output distributions.
- Its faster 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.

Experiment on MNIST

- Vanilla backprop in a 784 -> 800 -> 800 -> 10 net with rectified linear hidden units gives 146 test errors.

  RELU: $y = \max(0, x)$

- If we train a 784 -> 1200 -> 1200 -> 10 net using dropout and weight constraints and jittering the input, we eventually get 67 errors.

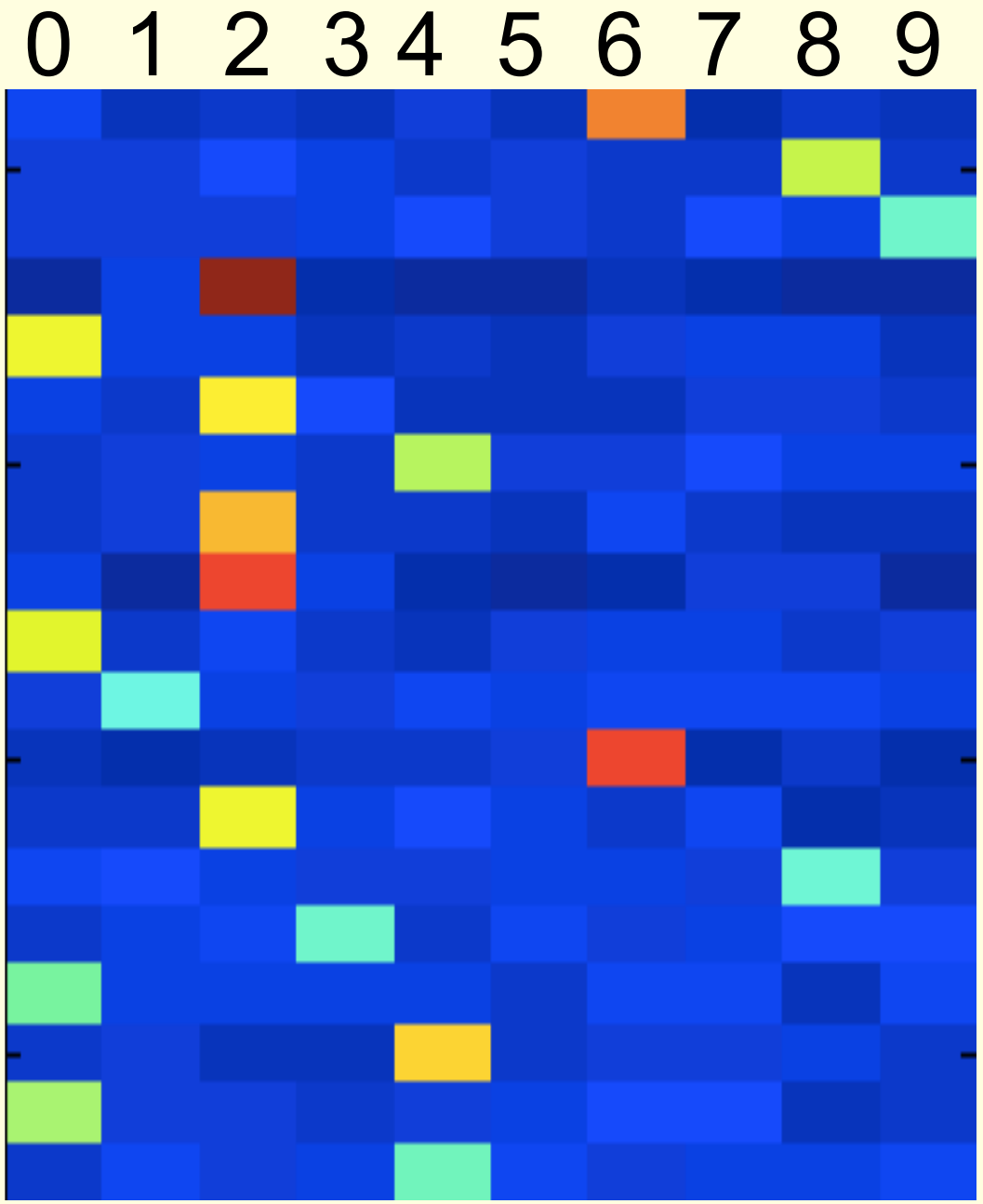
- How much of this improvement can be transferred to the 784 -> 800 -> 800 -> 10 net?

Transfer to the small net

- Using both the soft targets obtained from the big net and the hard targets, we get 74 errors in the 784 -> 800 -> 800 -> 10 net.
  - The transfer training uses the same training set but with no dropout and no jitter.
  - Its just vanilla backprop (with added soft targets).
- The soft targets contain almost all the knowledge.
  - The big net learns a similarity metric for the training digits even though this isn't the objective function for learning.

The soft outputs

(one row per training case)



this 2 resembles a 1
and nothing much else

this 2 resembles
0, 3, 7, 8

this 2 resembles
4 and 7

A very surprising result on MNIST

- Train the 784 -> 800 -> 800 -> 10 net on a transfer set that does not contain any examples of a 3. After this training, raise the bias of the 3 by the right amount.
  - The distilled net then gets 98.6% of the test threes correct even though it never saw any threes during the transfer training.

An even more surprising result on MNIST

- Train the 784 -> 800 -> 800 -> 10 net on a transfer set that only contains images of 7 and 8.
- After training, lower the biases of 7 and 8 by the optimal amount.
- The net then gets 87% correct over all classes.

Conclusion so far

- It is well known that object recognition is greatly improved by transforming the input images in ways that do not change the label.
  - But this brute-force method means we need to train on a lot more images.
- Transforming the targets has similarly big effects on generalization.
  - This does not change the size of the training set.
  - But you have to get the soft targets from somewhere.

A popular way to transform the targets

- Organize the labels into a tree and instead of just predicting a label, predict all of the labels on the path to the root.
  - Many groups have tried this and it helps, but not nearly as much as using soft targets produced by a good model.
- Visual similarity cannot be modeled well by any tree.

An intriguing result on speech

- Start with a trained model that classifies 58.9% of the test frames correctly.
  - The model is a slightly outdated version of the acoustic model used in Android phones.
- Use that model to provide soft targets for a new model (that also sees hard targets).
  - The new model converges to 57.0% correct even when it is only trained on 3% of the data.
  - Without the soft targets it peaks at 44.5% on 3% of the data and then gets much worse.

Conclusion

- Soft targets area a VERY good regularizer.
  - They prevent the model from being too sure.
  - They allow each training case to impose much more constraint on the weights.

Improving a production speech model

- Train 10 models separately. The individual models average 58.9% correct.
  - The models only differ in their initial weights.
  - The ensemble gets 61.1% correct.

- Now distill the ensemble into a single model of the same size using both hard and soft targets.
  - The distilled model gets 60.8% correct.
  - This is 6/7 of the ensemble win.

Relation to prior work by Caruana and his collaborators

- They perform transfer from the ensemble by trying to match the logits produced by the ensemble average.
  - This is just a regression problem during the transfer learning.

- How does matching logits relate to minimizing the cross entropy with soft targets derived by using a high temperature in the softmax?

The high temperature limit

$$e^{\varepsilon} \approx 1+\varepsilon \quad \text{if } \varepsilon \text{ is small}$$

logits of distilled model

logits for soft targets

$$T\frac{\partial C}{\partial z_i} = p_i - t_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)} - \frac{\exp\left(\frac{v_i}{T}\right)}{\sum_j \exp\left(\frac{v_j}{T}\right)}$$

$$T\frac{\partial C}{\partial z_i} \approx \frac{1+\frac{z_i}{T}}{N+\sum_j \frac{z_j}{T}} - \frac{1+\frac{v_i}{T}}{N+\sum_j \frac{v_j}{T}} = \frac{1}{NT}\left(z_i - v_i\right)$$

assume we have zero-meaned both sets of logits for every case

Training a community of neural nets

- If we train ten 784 -> 500 -> 300 -> 10 nets independently on MNIST, they average about 158 test errors, but the geometric ensemble gets 143 errors.

- What if we let each net try to match soft targets derived by averaging the opinions of the whole community as it is training (in addition to matching the hard targets)?
  - The nets now average 126 errors!
  - The ensemble gets 120 errors.

How to make an ensemble mine knowledge more efficiently

- We can encourage different members of the ensemble to focus on resolving different confusions.
  - In ImageNet, one “specialist” net could see examples that are enriched in mushrooms.
  - Another specialist net could see examples enriched in sports cars.
- We can choose the confusable classes in several ways.
  - K-means clustering on the soft target vectors produced by a generalist model works nicely.

The main problem with specialists

- Specialists tend to over-fit.
- To prevent this we need a very strong regularizer.
  - Making them small doesn't work well. They need all the lower levels of a general vision system to make the right fine distinctions.
  - Freezing the lower levels does not work well. The early filters need to be slightly adapted.

- So how can we regularize specialists effectively without making them too weak?

One way to prevent specialists over-fitting

- Each specialist uses a reduced softmax that has one dustbin class for all the classes it does not specialize in.
- The specialist estimates two things:
  - 1. Is this image in my special subset?
  - 2. What are the relative probabilities of the classes in my special subset?
- After training we can adjust the logit of the dustbin class to allow for the data enrichment.
- The specialist is initialized with the weights of a previously trained generalist model and uses early stopping to prevent over-fitting.

The JFT dataset

- This is a Google internal dataset with about 100 million images with 15,000 different class labels.
- A large convolutional neural net trained for about six months on many machines gets 25% correct on the test set (using top-1 criterion).
- Can we improve this significantly with only a few weeks of training?

Early stopping specialists on JFT

- Start from JFT model that gets 25% top-1 correct.

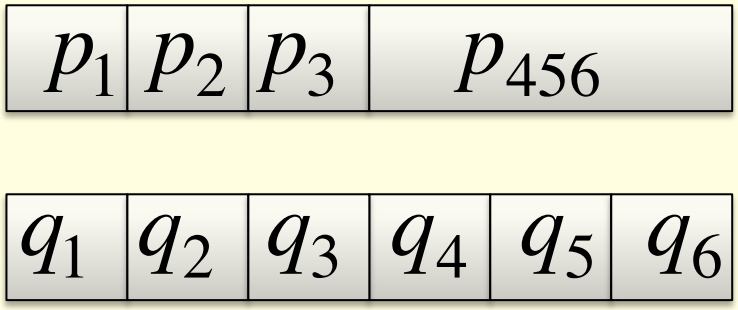
#spec	#cases	#win	relative accuracy
0	350037	0	0.0%
1	141993	+1421	+3.4%
2	67161	+1572	+7.4%
3	38801	+1124	+8.8%
4	26298	+835	+10.5%
5	16474	+561	+11.1%
6	10682	+362	+11.3%
7	7376	+232	+12.8%
8	4703	+182	+13.6%
9	4706	+208	+16.6%
10+	9082	+324	+14.1%

Combining models that have dustbin classes

- Its not trivial. A specialist is NOT claiming that everything in its dustbin class is equally probable. Its making a claim about the sum of those probabilities.
- Basic idea: For each test case, we iteratively revise the logits for the detailed classes to try to agree with all of the specialists.
  - i.e. We try to make the sum of the relevant detailed probabilities match the dustbin probability.

A picture of how to combine models that each have a dustbin class

- For each test or transfer case we run a fast iterative loop to find the set of logits that fit best with the partial distributions produced by the trained specialists.



A better way to prevent specialists over-fitting?

- Each specialist gets data that is very enriched in its particular subset of classes but its softmax covers all of the classes.
- On data from its special subset (50% of its training cases) it just tries to fit the hard targets with T=1.
- On the remaining data it just tries to match the soft targets produced by a previously trained generalist model at high temperature.
  - The soft targets will prevent overfitting.
  - Remember the 3% effect in the speech experiment.

- Conclusion: When extracting knowledge from data we do not need to worry about using very big models or very big ensembles of models that are much too cumbersome to deploy.
  - If we can extract the knowledge from the data it is quite easy to distill most of it into a much smaller model for deployment.

- Speculation: On really big datasets, ensembles of specialists should be more efficient at extracting the knowledge.
  - Soft targets for their non-special classes can be used to prevent them from over-fitting.

THE END