

Distilling the Knowledge in a Neural Network

Geoffrey Hinton, Oriol Vinyals & Jeff Dean

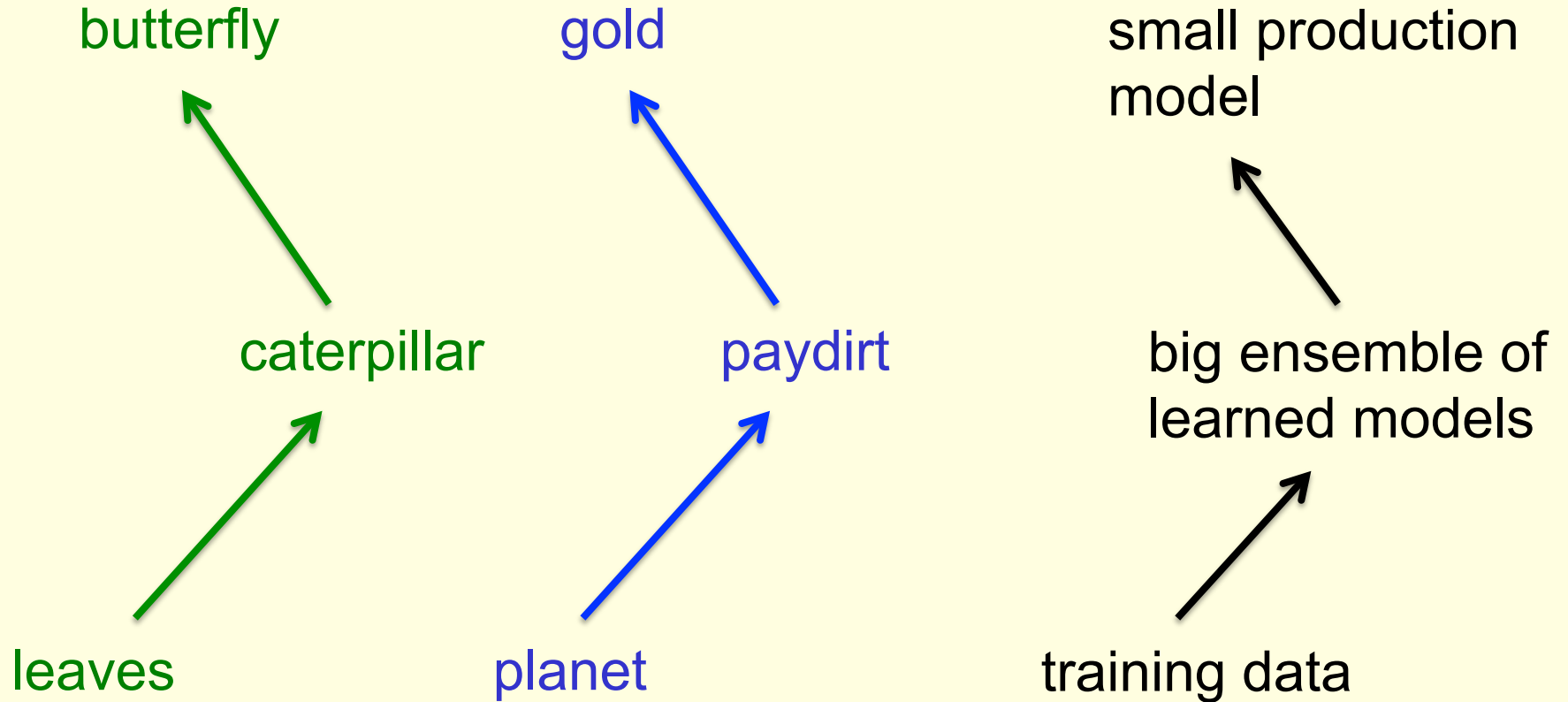
The conflicting constraints of learning and using

- 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.
- At 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



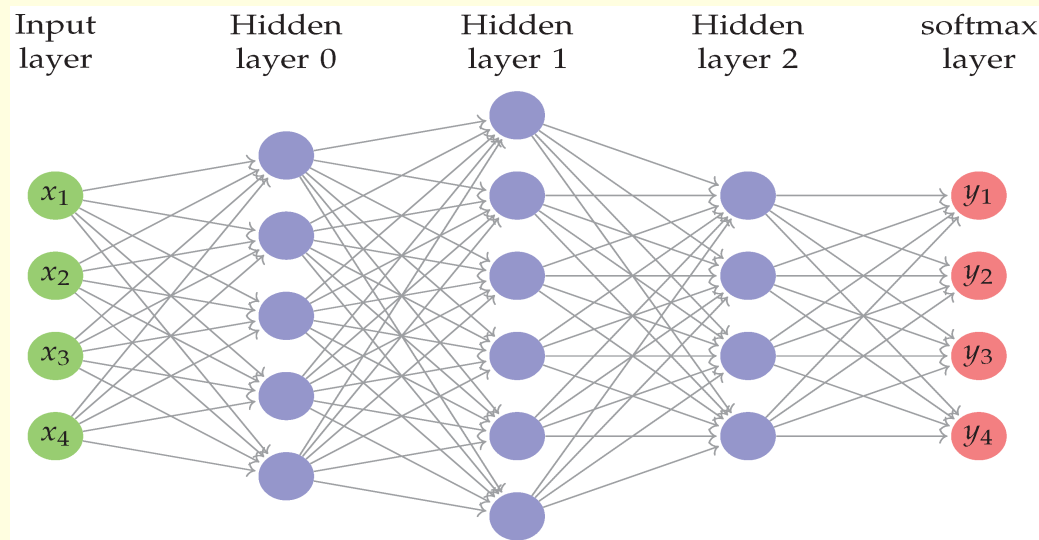
The main idea

- 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.

Softmax Activation Function

$$f_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

x_i : logit, output of unit i



$[1, 2, 3]$

$[2.718, 7.389, 20.086]$

$[0.090, 0.245, 0.665]$

cow	dog	cat	car
0	1	0	0

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

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

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.

```
[2] 1 # A simple vector that we are treating
    2 # as the output of our neurons
    3 logits = np.array([1.,2.,3.])
    4
    5 # Exponential space
    6 logits_exp = np.exp(logits)
    7
    8 logits_exp
```

```
↳ array([ 2.71828183,  7.3890561 , 20.08553692])
```

```
[3] 1 # compute logits in normalized exponential space
    2 # with temperature
    3
    4 #potential temperature values [hyper-parameter]
    5 T = [1., 5.,7.,10.]
    6
    7 for t in T:
    8     logits_exp_normalized = np.exp(logits/t) / sum(np.exp(logits/t))
    9     print(f"Temperature[{t}] - {logits_exp_normalized}")
```

```
↳ Temperature[1.0] - [0.09003057 0.24472847 0.66524096]
Temperature[5.0] - [0.2693075  0.32893292 0.40175958]
Temperature[7.0] - [0.28700357 0.33107727 0.38191915]
Temperature[10.0] - [0.30060961 0.33222499 0.3671654 ]
```

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

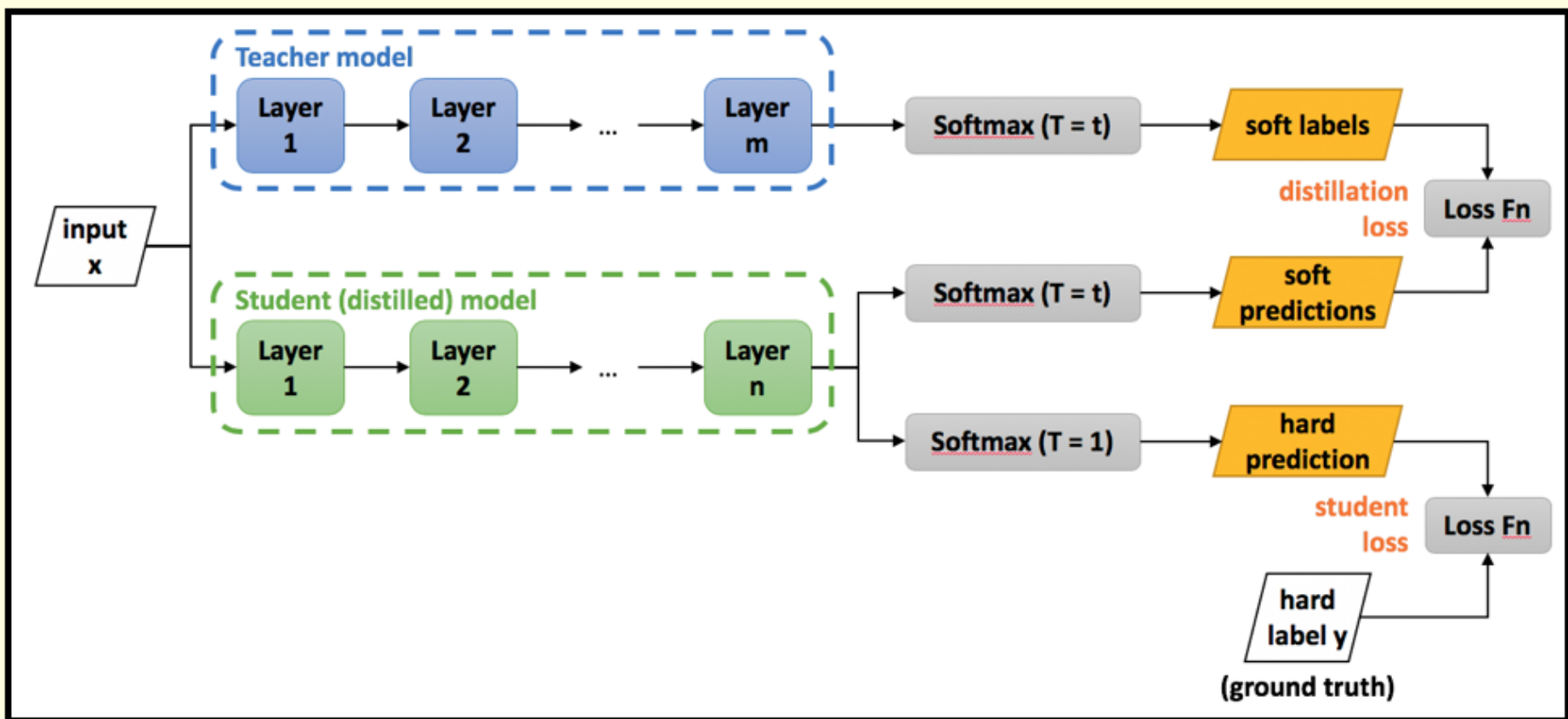
Softened outputs reveal the dark knowledge in the ensemble.

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.

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.

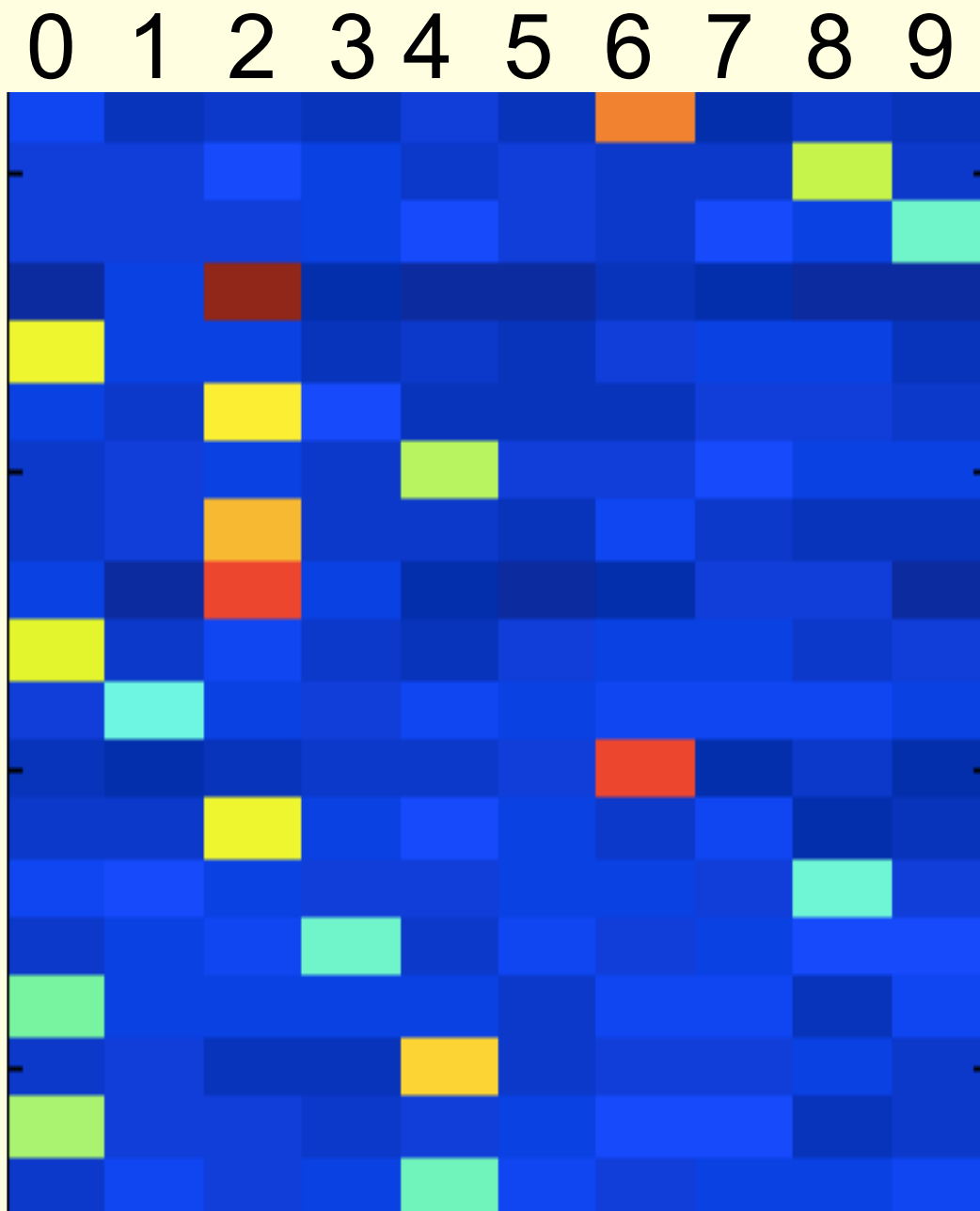
The soft outputs

(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.

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 are 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.

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%

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.

Thank You for Listening