Distilling the Knowledge in a Neural Network

https://arxiv.org/abs/1503.02531

NIPS

Geoffrey Hinton[*†]
Google Inc.
Mountain View

Oriol Vinyals[†]
Google Inc.
Mountain View

Jeff Dean
Google Inc.
Mountain View

Freya

2020.11.15

http://www.cs.toronto.edu/~hinton/

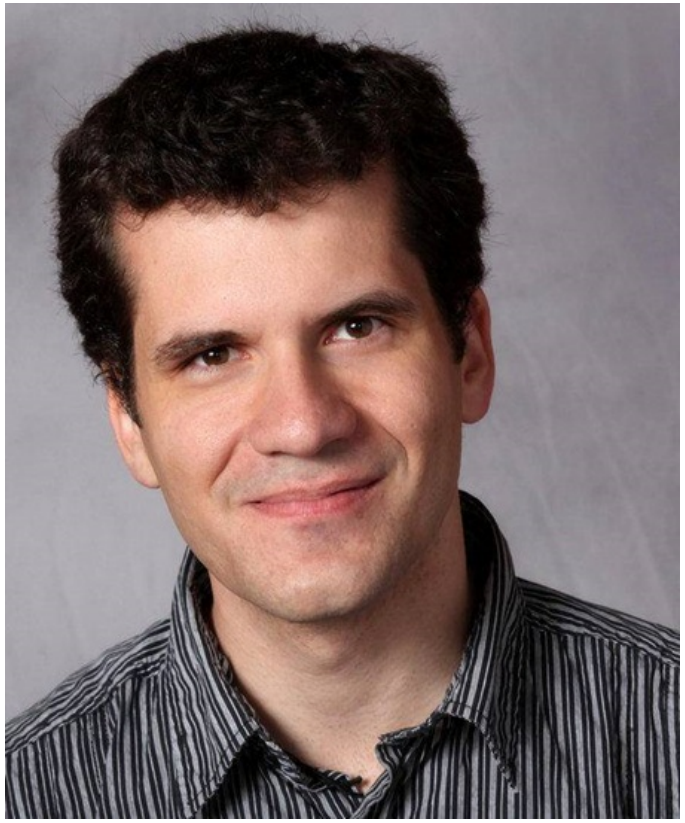
Geoffrey Hinton：

- Department of Computer Science **University of Toronto**
- Tuning award

……

- 母亲是数学老师
- 父亲是昆虫学家
- 舅舅是经济学家(Colin Clark，GDP)
- 曾曾祖父(逻辑学家George Boole，Boolean algebra)

Oriol Vinyals



Oriol Vinyals

A Research Scientist at the Google Brain

A PhD student in the Electrical Engineering & Computer Science department at the University of California, Berkeley. I currently work on interdisciplinary topics including Speech Recognition and Signal Processing under Nelson Morgan's supervision, TeleImmersive technologies with Ruzena Bajcsy, and Computer Vision and Machine Learning with Trevor Darrell. I am a recipient of the 2011 Microsoft Research PhD Fellowship.

I have recently become involved with the Berkeley Overmind, a project to build an AI that plays a popular real time strategy game.

I graduated in September 2009 with a master's degree (MSc) in Computer Science and Engineering from the University of California, San Diego, and in 2007 I completed a dual degree in Telecommunication Engineering and Mathematics from the Polytechnic University of Catalonia in Barcelona, Spain. I did my undergrad thesis at the Robotics Institute of the Carnegie Mellon University on Machine Learning and Computer Vision. In addition, I got to work with a fabulous group of people during three summer internships at Microsoft Research in Redmond, WA, and at Google Research in Mountain View, CA

Jeff Dean

- The lead of Google.ai.
- 美国工程院院士
- 华盛顿大学博士
- ACM、AAAS fellow
- Google Brain
- Google 20号员工

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.

Problem:

- Too cumbersome to Deploy to the client
- Overfitting

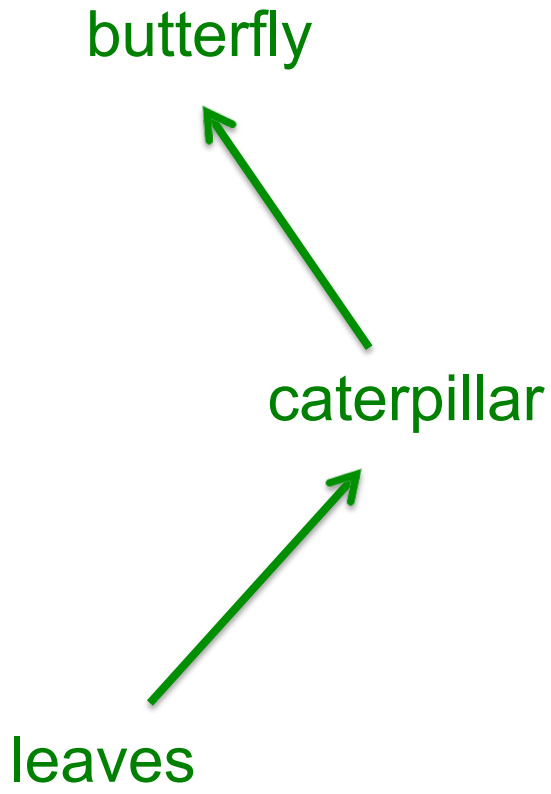
Solution :

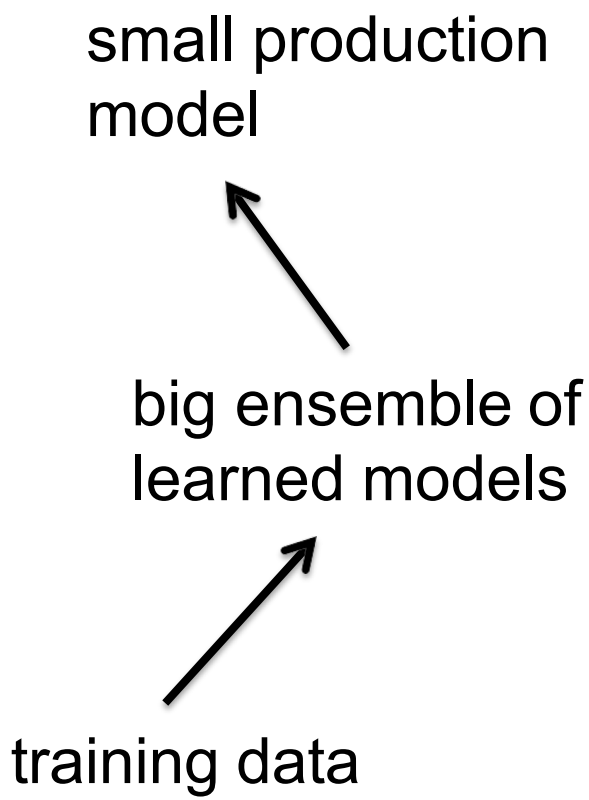
- Distilling the Knowledge in a Neural Network

1 Introduction

Many insects have a larval form that is optimized for extracting energy and nutrients from the environment and a completely different adult form that is optimized for the very different requirements of traveling and reproduction. In large-scale machine learning, we typically use very similar models for the training stage and the deployment stage despite their very different requirements: For tasks like speech and object recognition, training must extract structure from very large, highly redundant datasets but it does not need to operate in real time and it can use a huge amount of computation.

An analogy


butterfly
caterpillar
leaves

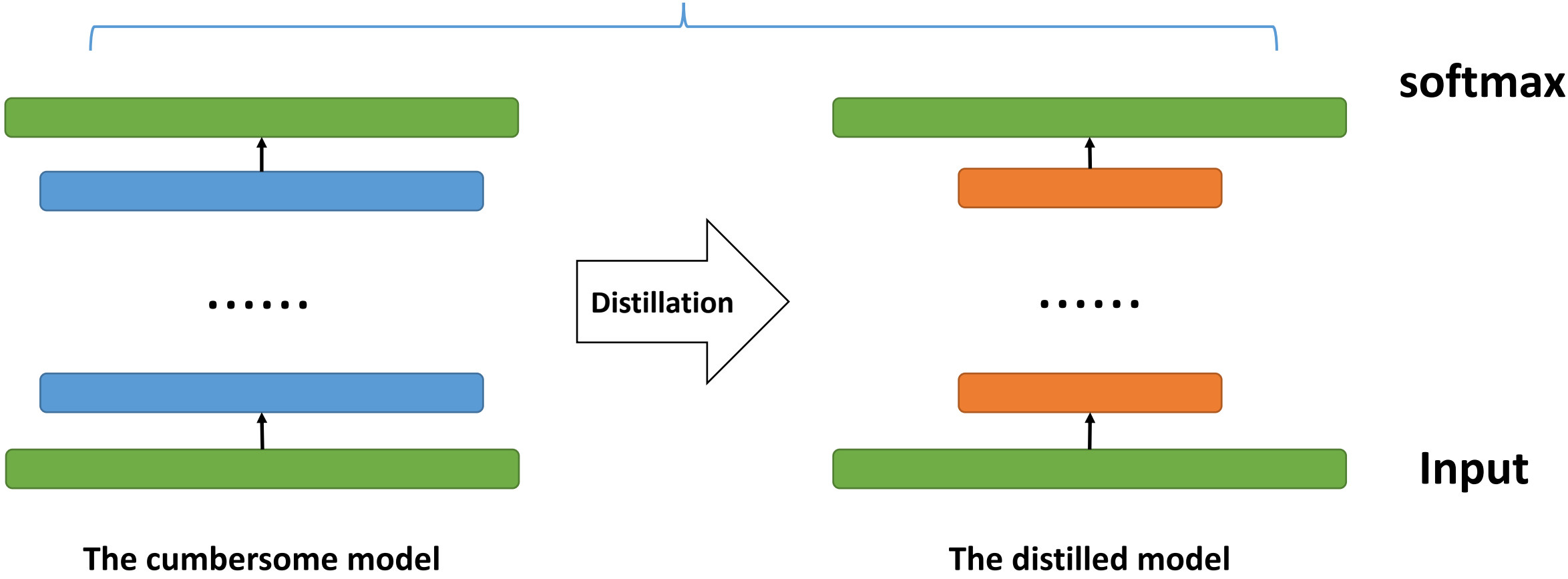

small production model
big ensemble of learned models
training data

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?

Distillation

Matching soft targets and hard target



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.
  - little information
- If we have the ensemble, we can divide the averaged logits from the ensemble by a "temperature" to get a much softer distribution.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

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

Problem： not soft enough

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

softened output of ensemble

Softened outputs reveal the dark knowledge in the ensemble.

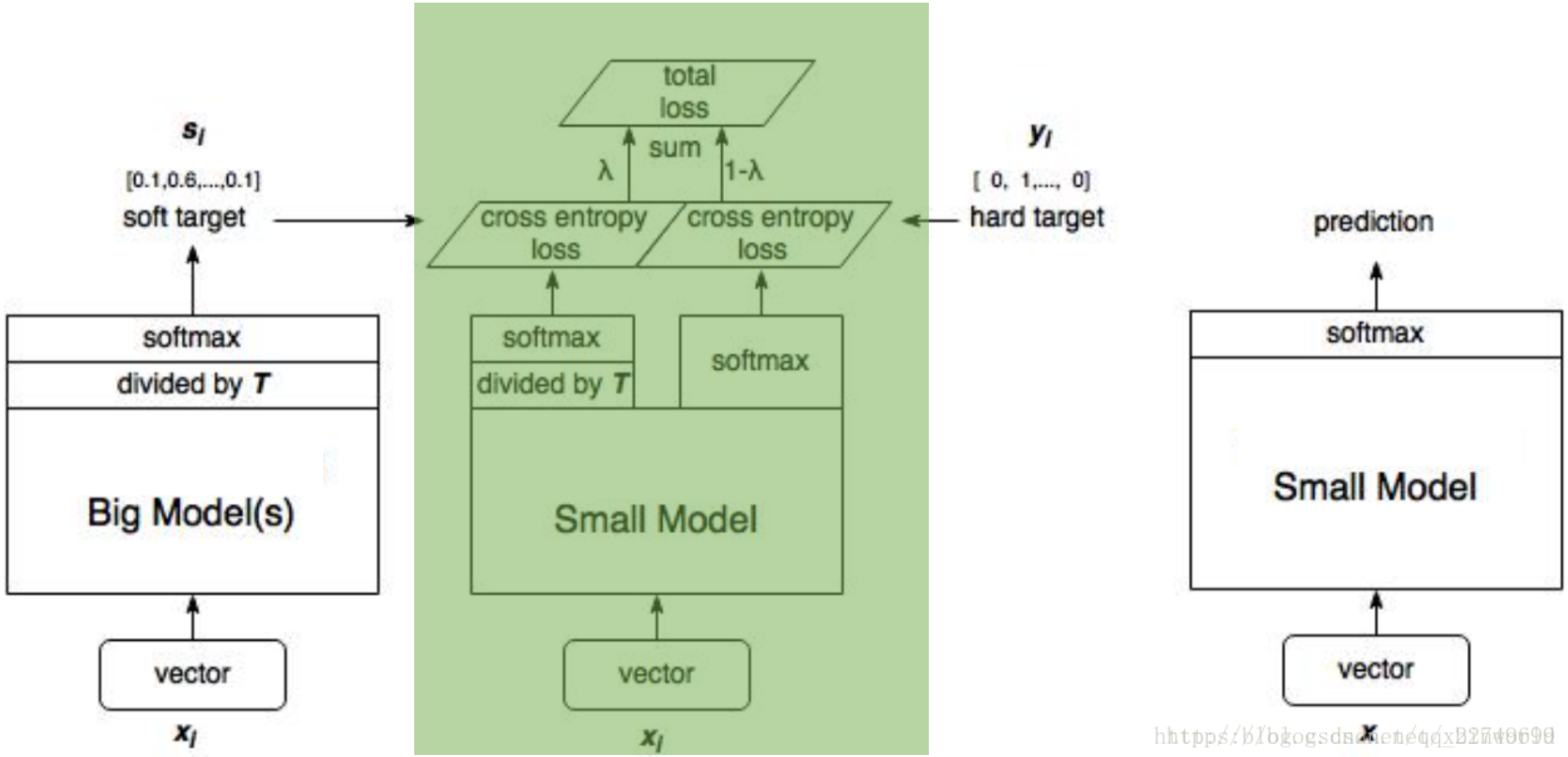
2 Distillation

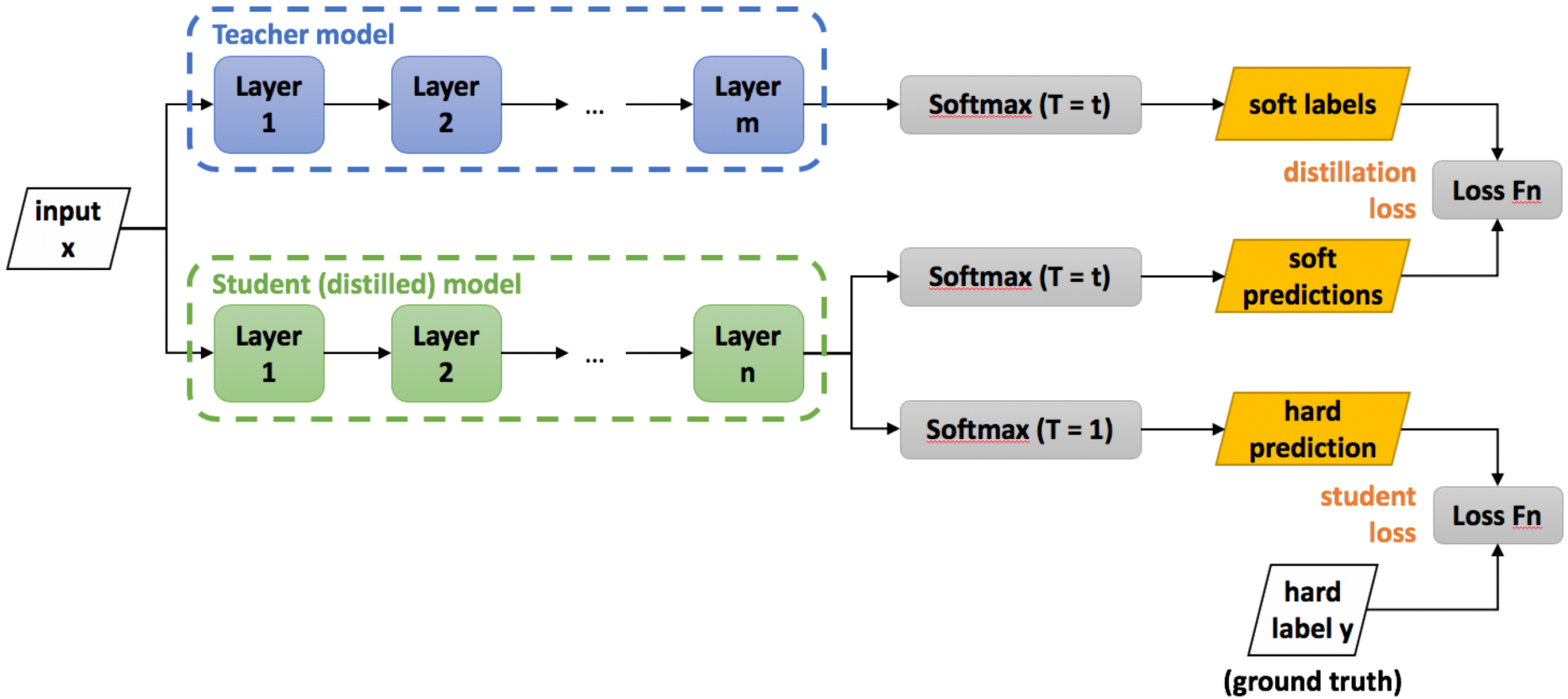
Neural networks typically produce class probabilities by using a "softmax" output layer that converts the logit, z_i, computed for each class into a probability, q_i, by comparing z_i with the other logits.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{1}$$

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.


total
loss
sum
λ
1-λ
s_i
[0.1,0.6,...,0.1]
soft target
y_i
[0, 1,..., 0]
hard target
cross entropy
loss
cross entropy
loss
softmax
divided by T
softmax
softmax
divided by T
Big Model(s)
Small Model
Small Model
vector
vector
vector
x_i
x_i
x
prediction
softmax


Teacher model
Layer 1
Layer 2
...
Layer m
input x
Softmax (T = t)
soft labels
distillation loss
Loss Fn
Student (distilled) model
Layer 1
Layer 2
...
Layer n
Softmax (T = t)
soft predictions
Softmax (T = 1)
hard prediction
student loss
Loss Fn
hard label y
(ground truth)

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.

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.
- **student学习的是teacher的泛化能力**

THE END