Learning Curves for Analysis of Deep Networks



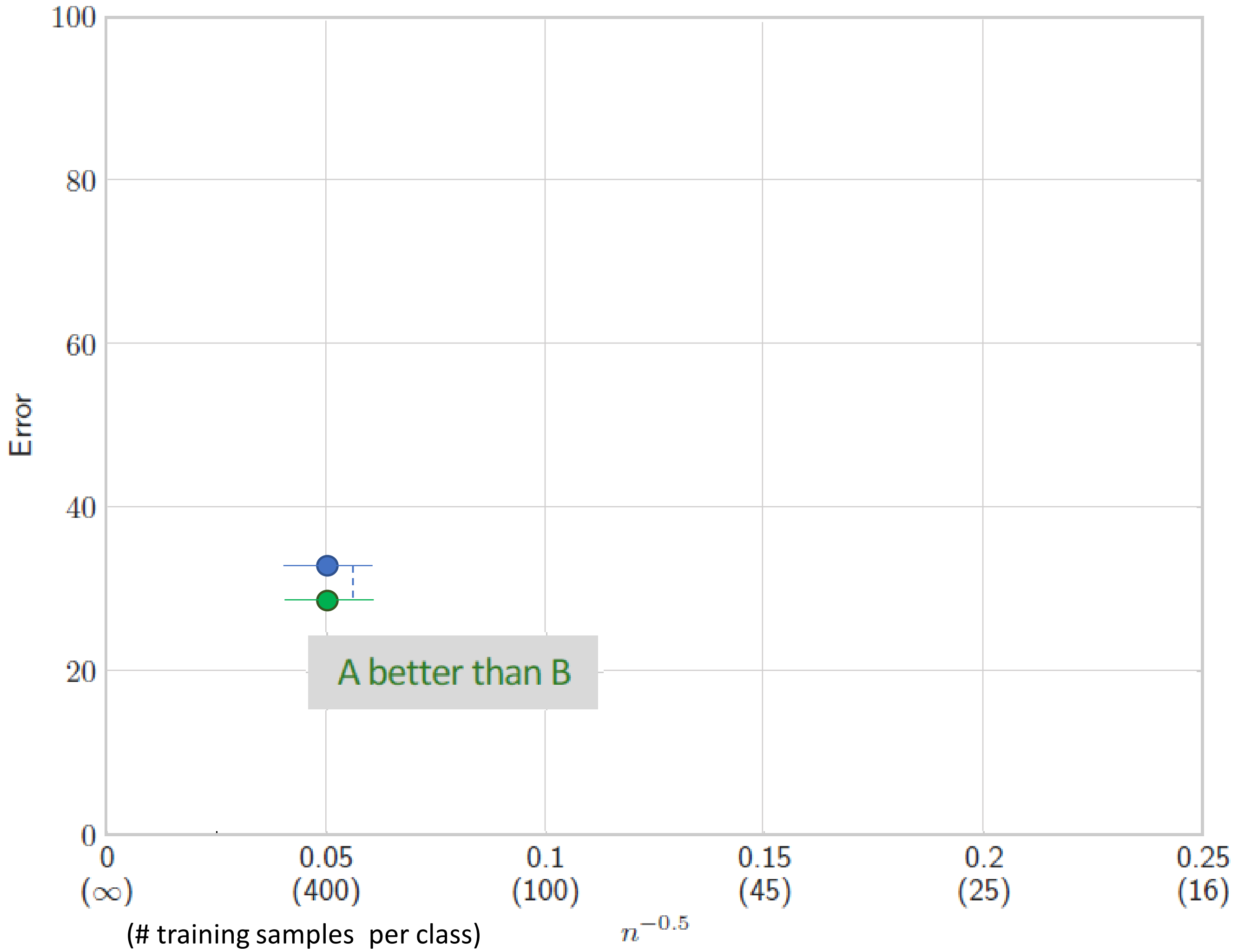

Derek Hoiem[1] Tanmay Gupta[2] Zhizhong Li[1] Michal M. Shlapentokh-Rothman[1]

[1] ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

[2] AI2
Allen Institute for AI

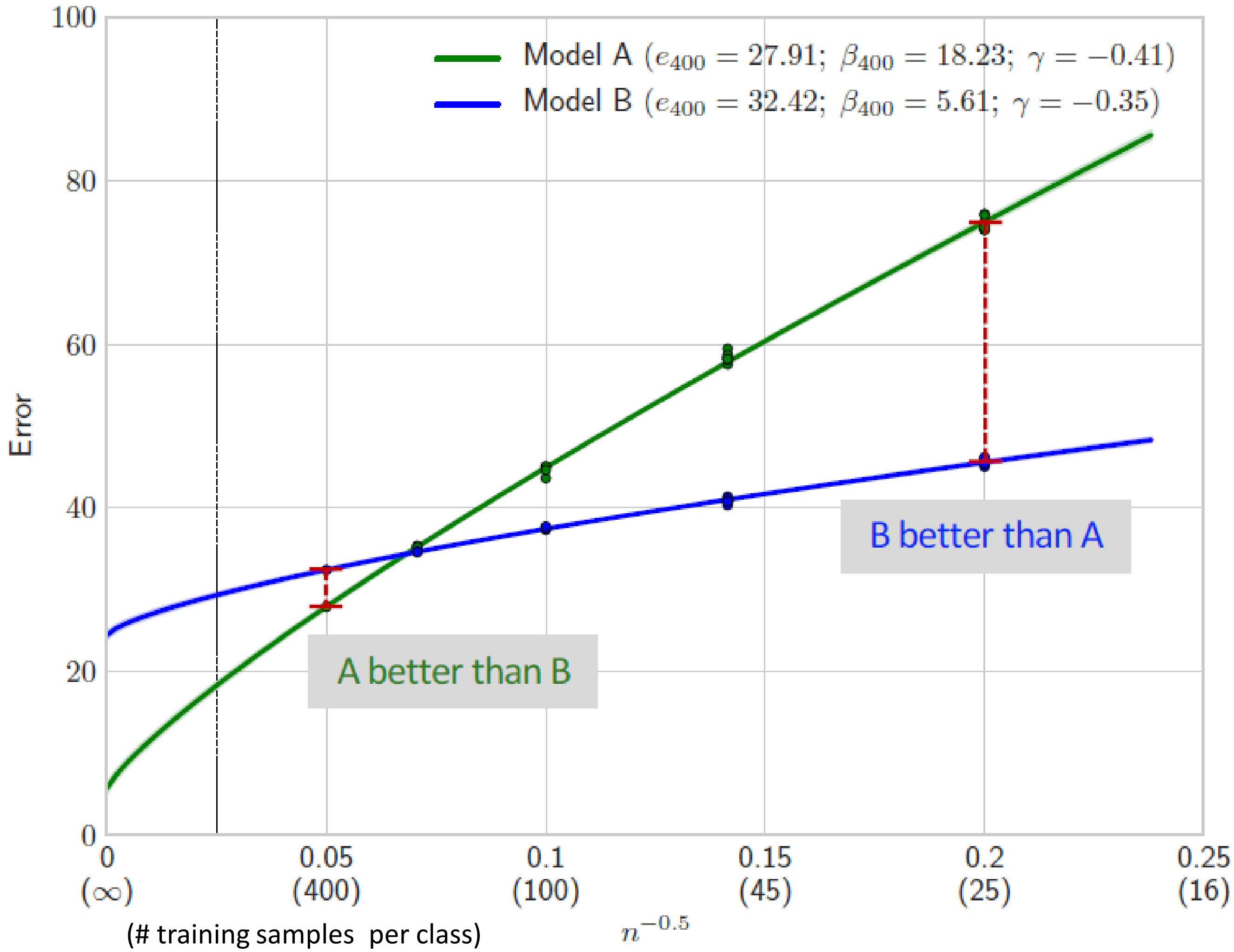
Which classifier is better?

Model	Error after full training (n=400)
A	27.9%
B	32.4%



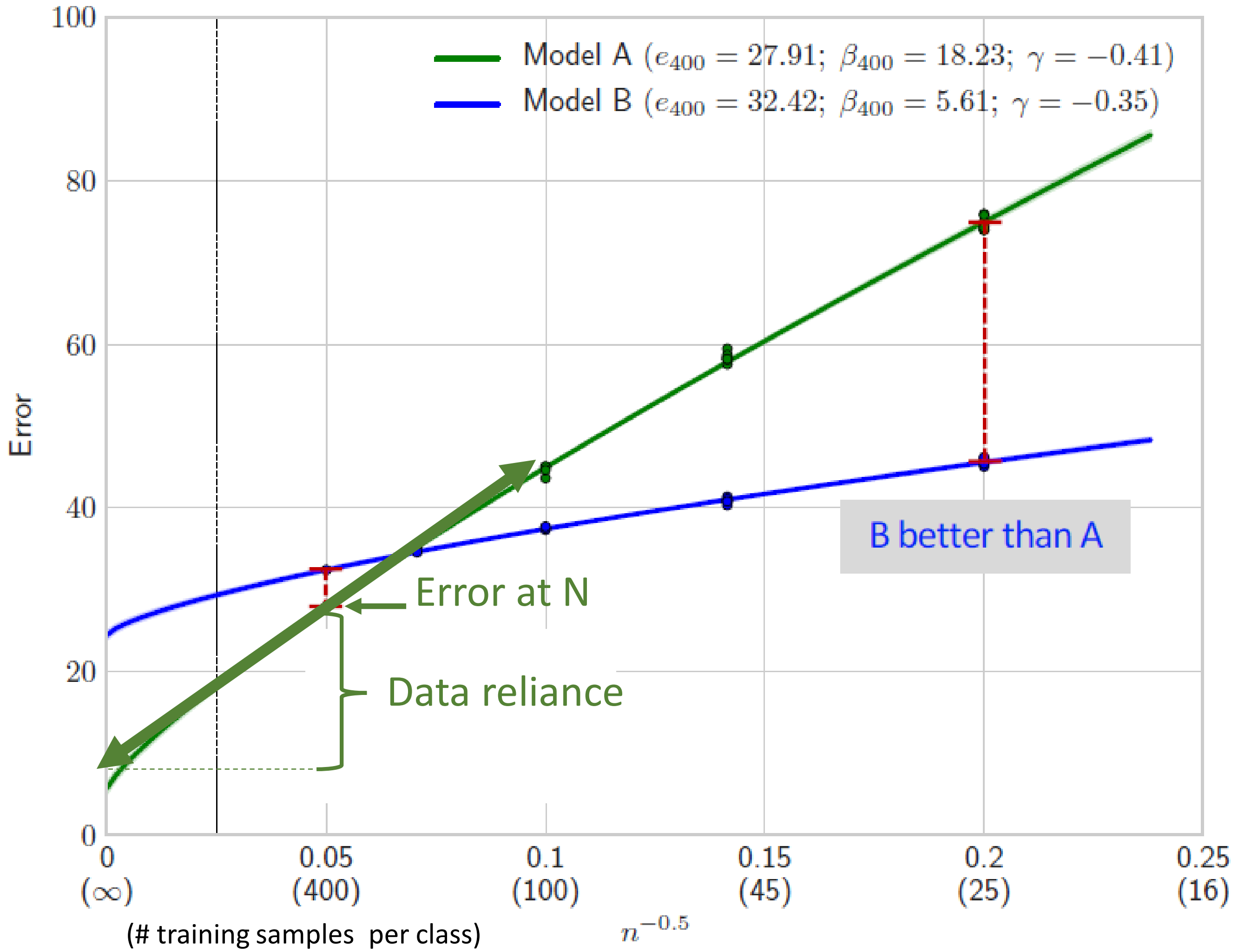
Which classifier is better?

Model	Error after full training (n=400)
A	27.9%
B	32.4%



Better characterize classifier performance with learning curve and measure of data reliance

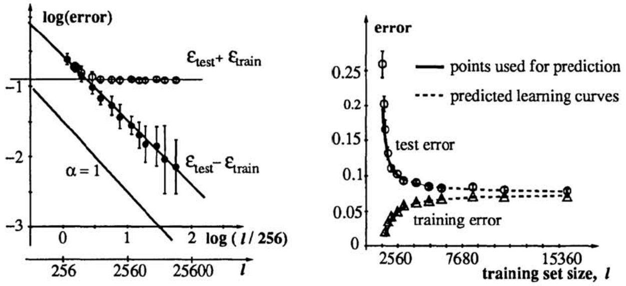
Model	Error after full training (n=400)	Data Reliance
A	27.9%	18.2
B	32.4%	5.6



Learning curves have been shown useful, but there is no established methodology for how to use them in evaluation

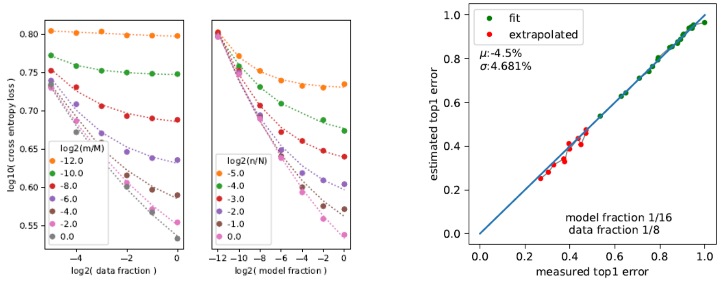
model selection

Cortes et al.
NIPS 1993



relationship analysis
of model size, training size, computation

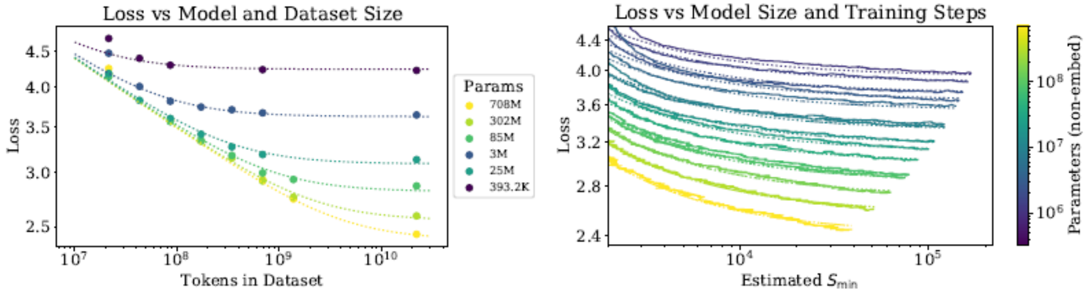
Rosenfeld et al.
ICLR 2020



(b) Extrapolation on ImageNet

extrapolation

Kaplan et al.
arxiv 2020



Our goal: make it easy to improve classifier evaluations with learning curves

- Show how to model, fit, and display without using a lot of computation or paper space
- Show that learning curves provide useful insights

Model learning curves with extended power law

$$e(n) = \alpha + \eta n^{\gamma}$$

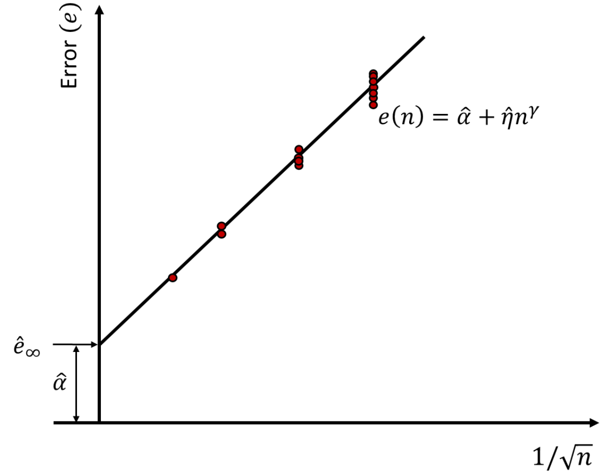
n: Number of training samples (per class)

e: Test error

Well supported by

- Theory: bias-variance trade-off, many generalization bounds
- Practice: Hestness et al. 2017, Johnson & Nguyen 2017, Kaplan et al. 2020, Rosenfeld et al. 2020
- Our experiments

Fit learning curves with weighted least squares



1. Given γ, solve for α, η

$$\mathcal{G}(\gamma) = \min_{\alpha,\eta} \sum_{i=1}^{S} \sum_{j=1}^{F_i} w_{ij} \left(e_{ij} - \alpha - \eta n^{\gamma} \right)^2$$

e_{ij}: observation test error on split i with size F_i
w_{ij}: accounts for variance of e_{ij} and number of splits

2. Step over γ

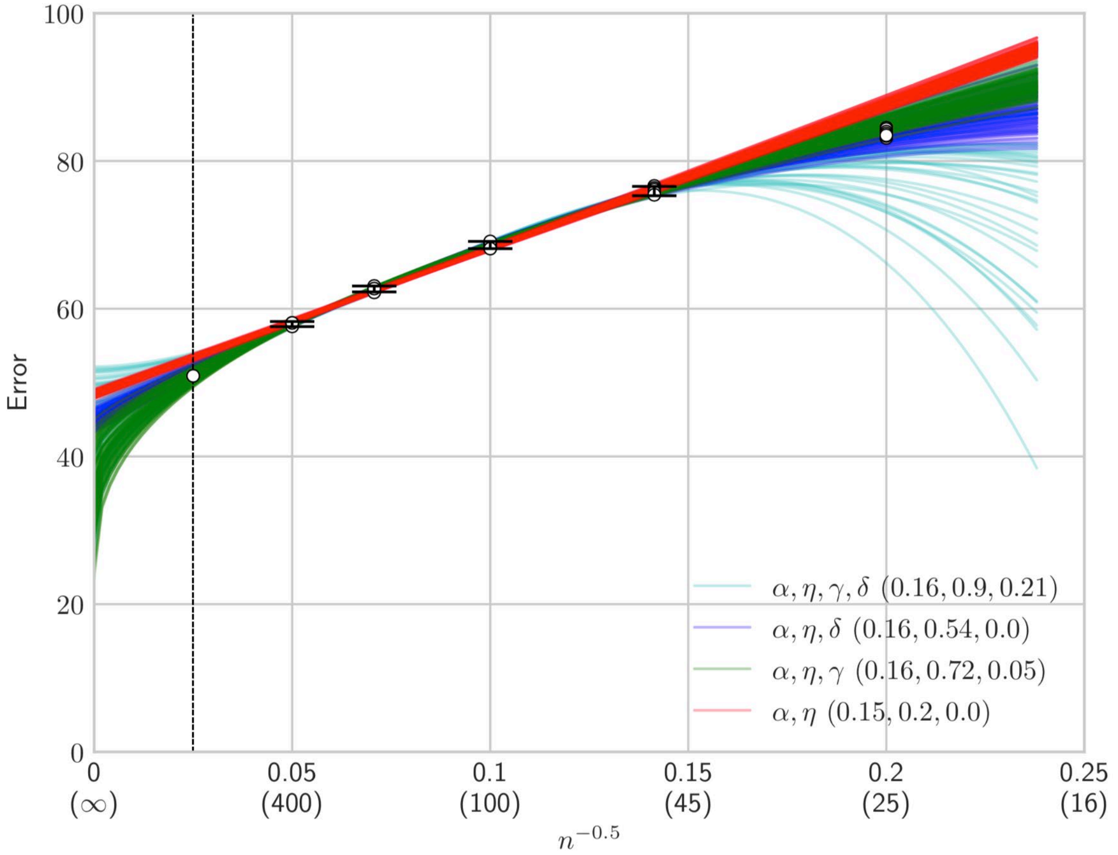
$$\min_{\gamma \in (-1,0)} \mathcal{G}(\gamma) + \lambda |\gamma + 0.5|$$

Extended power law model and weighted fitting lead to better prediction of error and more stable parameters

Functional Form	Parameters
$e(n) = \alpha + \eta n^{-0.5}$	α, η
$e(n) = \alpha + \eta n^{\gamma}$	α, η, γ
$e(n) = \alpha + \eta n^{-0.5} + \delta n^{-1}$	α, η, δ
$e(n) = \alpha + \eta n^{\gamma} + \delta n^{2\gamma}$	$\alpha, \eta, \gamma, \delta$

			RMSE						
Params	**Weights**	R^2	**25**	**50**	**100**	**200**	**400**	**avg**	**p-value**
α, η, γ	$\frac{1}{\sigma_i^2 F_i}$	0.998	2.40	0.86	**<u>0.54</u>**	0.57	**<u>0.85</u>**	**<u>1.04</u>**	-
	$\frac{1}{\sigma_i^2}$	0.999	**<u>2.38</u>**	0.83	0.69	0.54	1.08	1.10	0.06
	1	0.998	2.66	0.86	0.79	**<u>0.50</u>**	1.26	1.21	0.008
α, η	$\frac{1}{\sigma_i^2 F_i}$	0.988	3.41	1.09	0.69	0.72	1.21	1.42	$<$0.001
α, η, δ	$\frac{1}{\sigma_i^2 F_i}$	0.999	2.89	**<u>0.74</u>**	0.68	0.56	0.94	1.16	0.05
$\alpha, \eta, \delta, \gamma$	$\frac{1}{\sigma_i^2 F_i}$	0.999	3.46	**<u>0.74</u>**	0.70	0.59	1.00	1.30	0.02

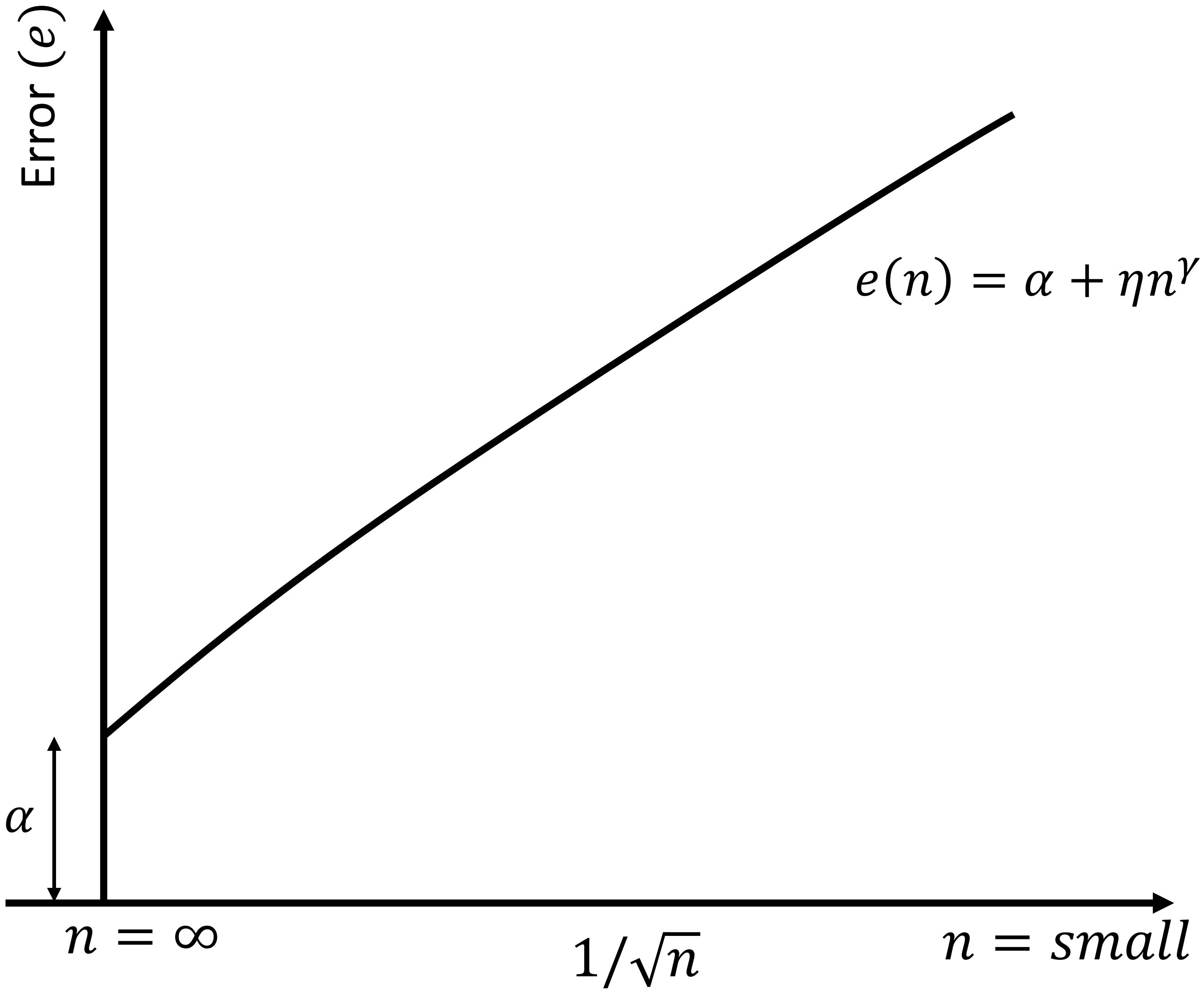
Leave-one-train-size-out error analysis:
Model accounts for 99.8% of *e(n)* variance and typically predicts on held out training size within 1% error



Stability analysis:
Given only 4 error observations (evaluations for 4 training sizes), our model better extrapolates and leads to more stable parameters with resampling

Display learning curves as error vs $n^{-0.5}$

- $\gamma \approx -0.5$ typically
- Curves are roughly linear in $n^{-0.5}$
- Can see full range of n

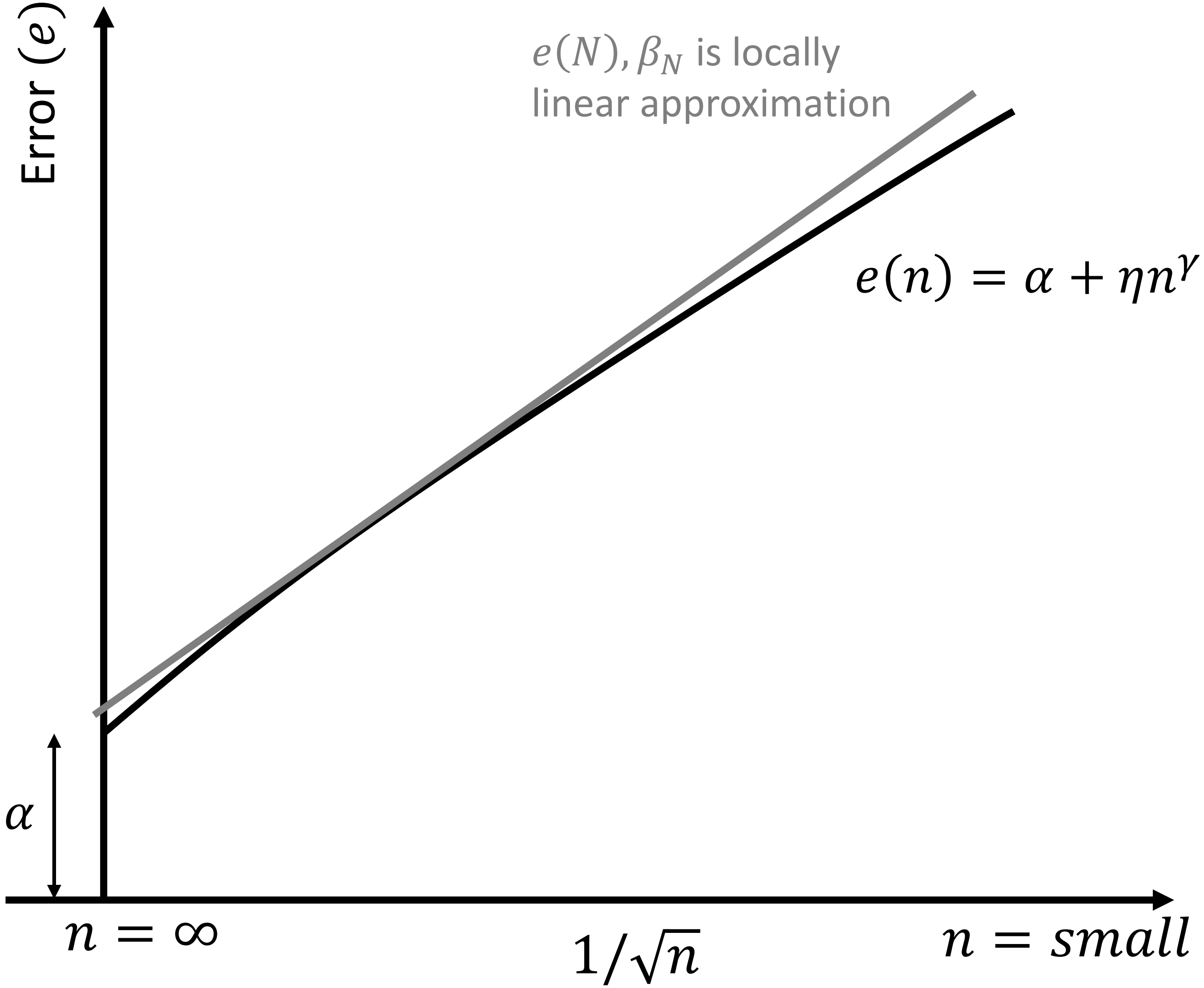


How to characterize/summarize learning curves

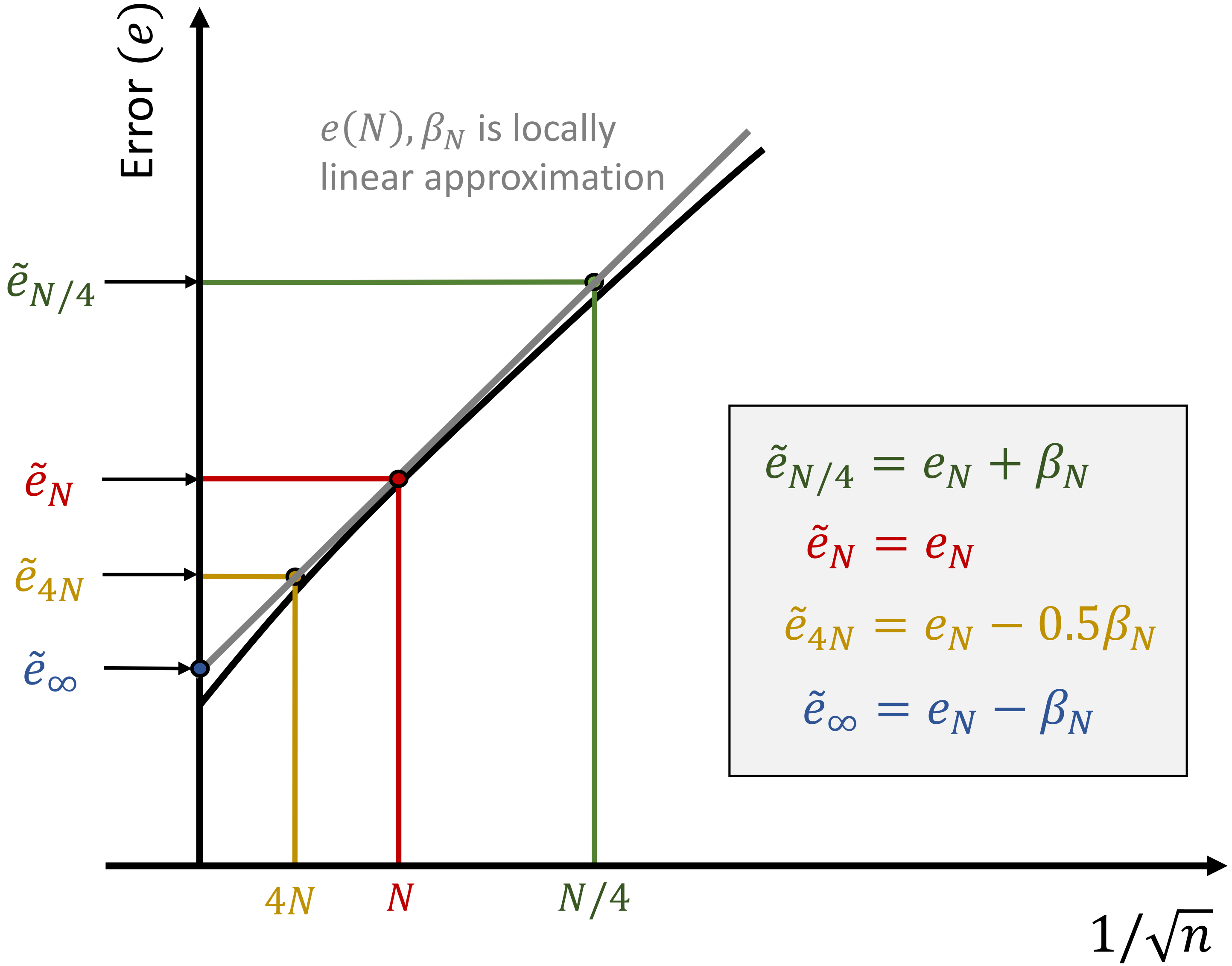
Problem: γ, η, α highly covariant with observation perturbations and not individually comparable across curves

Solution: re-parameterize

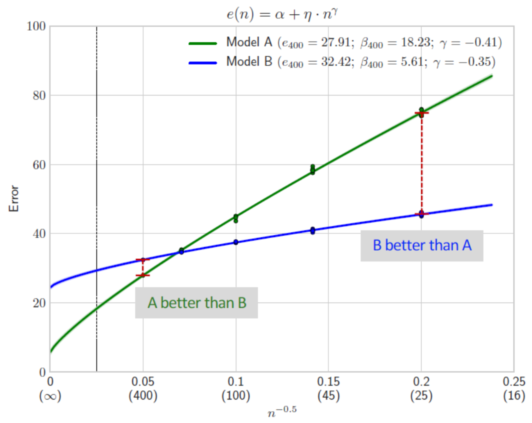
- $e_N = e(N)$ is error at full training size N
- $\beta_N = e'(N)/\sqrt{N}$ is data-reliance
- Can recover α, γ, η from e_N, β_N, γ



β_N characterizes how error depends on data size, is stably estimated under perturbation, and easy to derive for other models



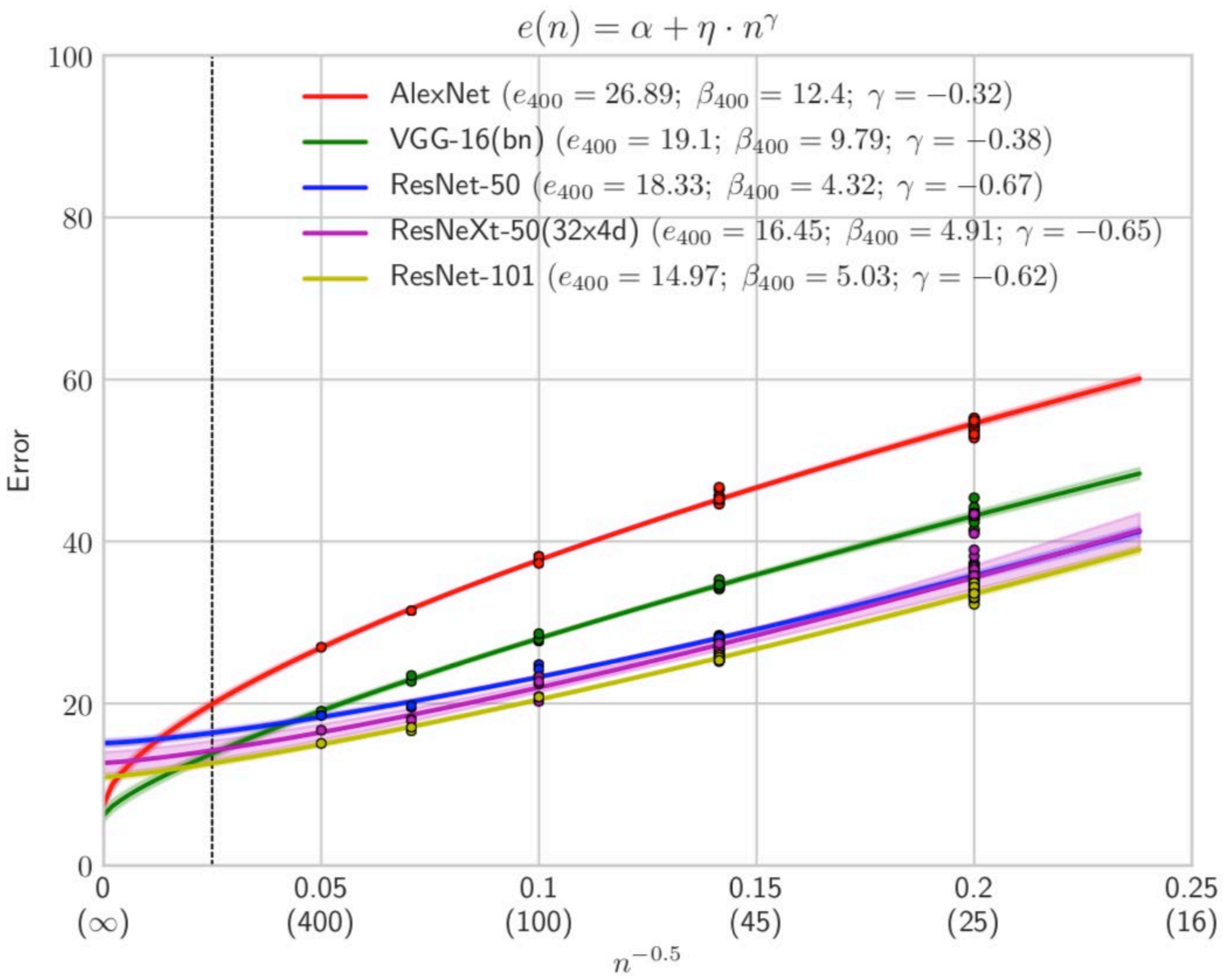
Display learning curve analysis with only one extra column for β_N



Model	e_{400}	β_{400}
A	27.91	18.23
B	32.42	5.61

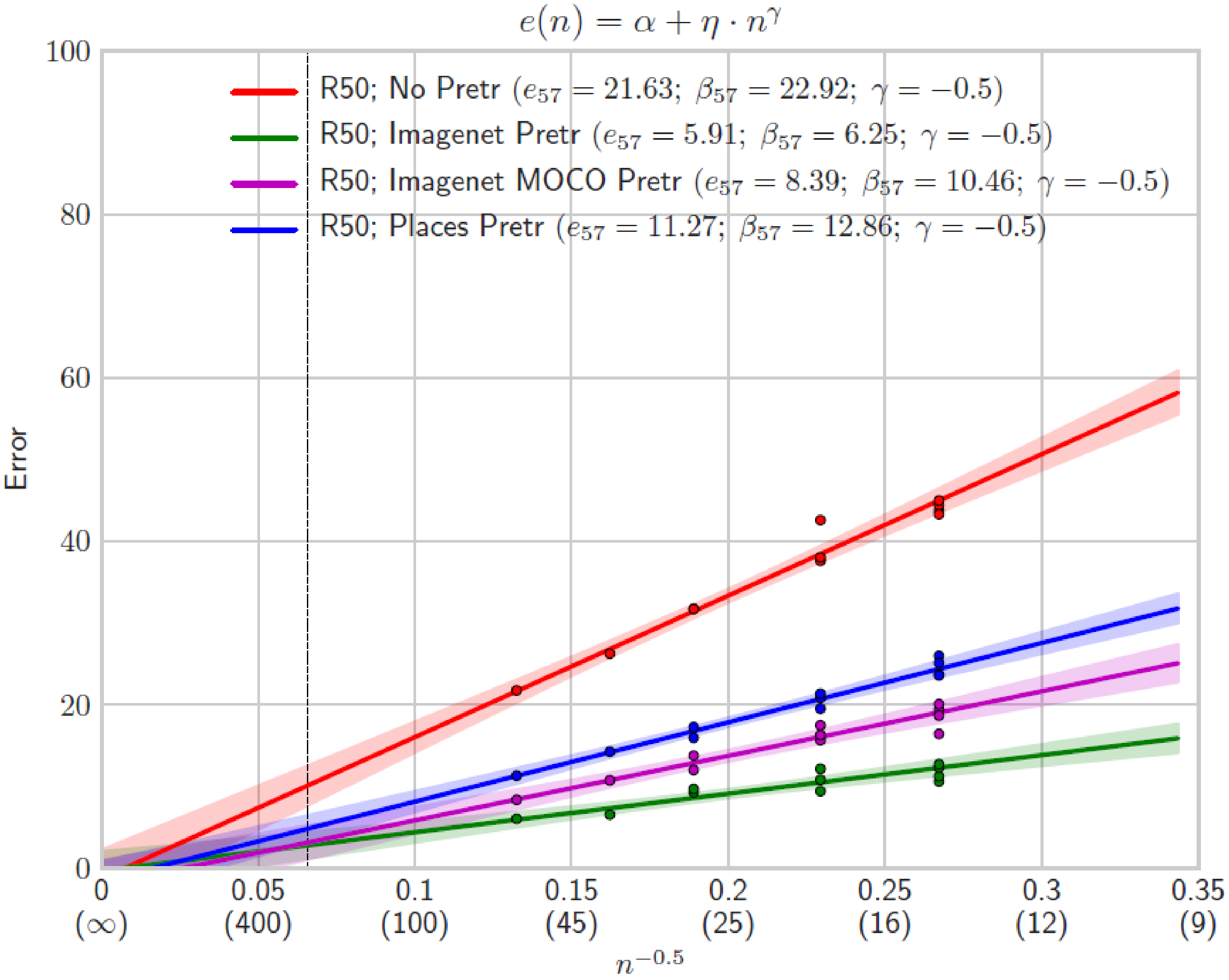
Comparing Architectures
(Cifar-100, no pretraining)

More recent architectures achieve lower error with less data-reliance



Effect of Pretraining Source
(Caltech-101)

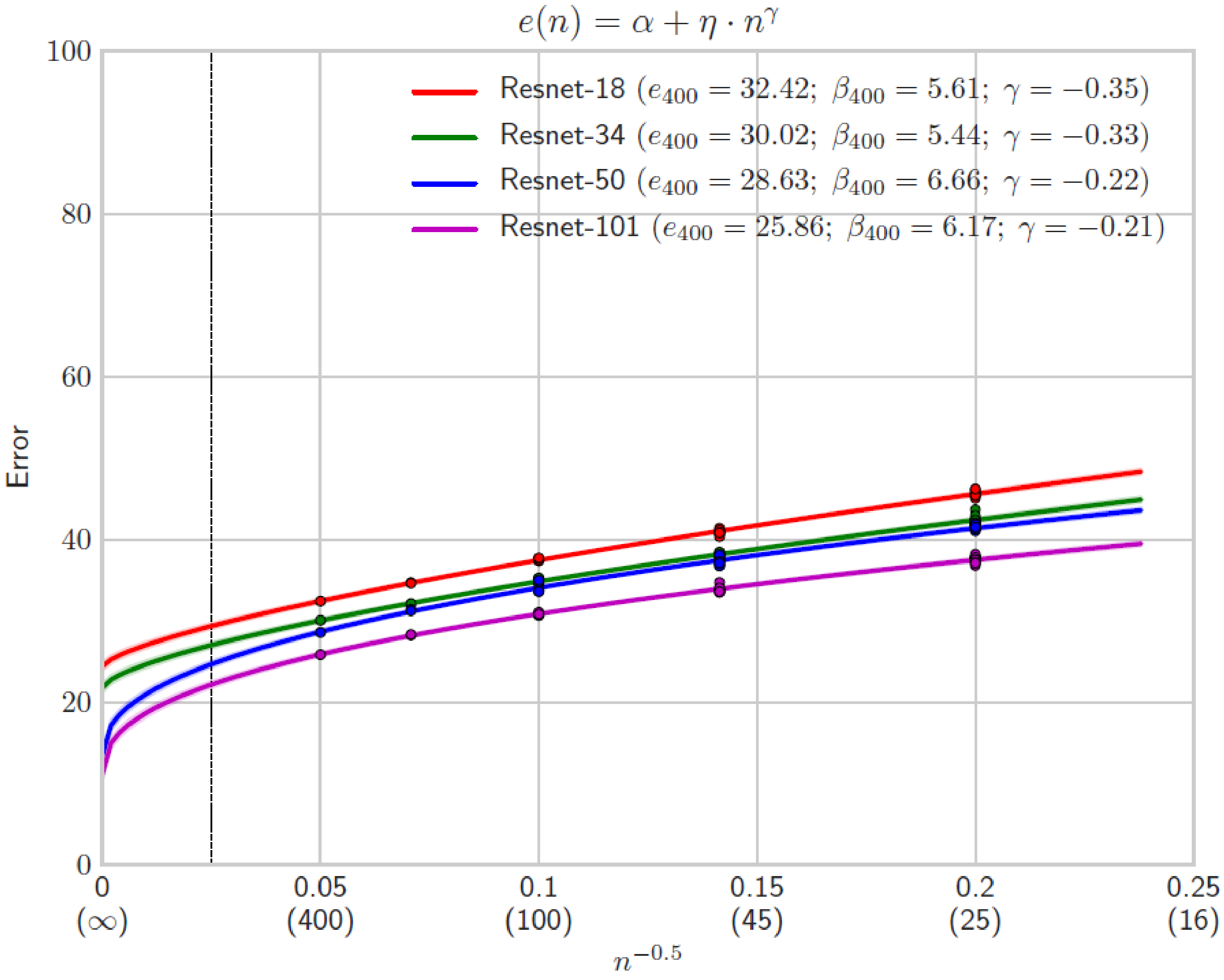
Supervised ImageNet pretraining provides much lower data-reliance than others (for classification)



Effect of Depth

(Cifar-100, ImageNet pretrained, not fine-tuned)

Features from deeper networks are better across range of training sizes (deeper is almost always better)



Check out our deep learning quiz and additional experiments in the paper to test your beliefs

Popular beliefs	Your guess	Supp-orted?	Exp. figures
Pre-training on similar domains nearly always helps compared to training from scratch.	☐	Y	5a, 5b, 6
Pre-training, even on similar domains, introduces bias that would harm performance with a large enough training set.	☐	U	6
Self-/un-supervised training performs better than supervised pre-training for small datasets.	☐	N	6
Fine-tuning the entire network (vs. just the classification layer) is only helpful if the training set is large.	☐	N	5a, 5b
Increasing network depth, when fine-tuning, harms performance for small training sets, due to an overly complex model.	☐	N	7a
Increasing network depth, when fine-tuning, is more helpful for larger training sets than smaller ones.	☐	N	7a
Increasing network depth, if the backbone is frozen, is more helpful for smaller training sets than larger ones.	☐	N	7d
Increasing depth or width improves more than ensembles of smaller networks with the same number of parameters.	☐	Y	7f
Data augmentation is roughly equivalent to using a m-times larger training set for some m.	☐	Y	8

Use learning curves to better evaluate your research contributions

```
from lc.measurements import CurveMeasurements
from lc.curve import LearningCurveEstimator
from omegaconf import OmegaConf
import matplotlib
import matplotlib.pyplot as plt

# Load error measurements
curvems = CurveMeasurements()
curvems.load_from_json('data/no_pretr_ft.json')

# Load config
cfg = OmegaConf.load('lc/config.yaml')

# Estimate curve
curve_estimator = LearningCurveEstimator(cfg)
curve, objective = curve_estimator.estimate(curvems)

# Plot
curve_estimator.plot(curve,curvems,label='No Pretr; Ft')
plt.show()
```

Thank You

https://github.com/allenai/learning-curve

https://prior.allenai.org/projects/lcurve