Bayesian Model Selection & Generalization in Deep Learning

Clare Lyle

GenU Workshop

Oct 12, 2021

Outline

Philosophy of Model Selection

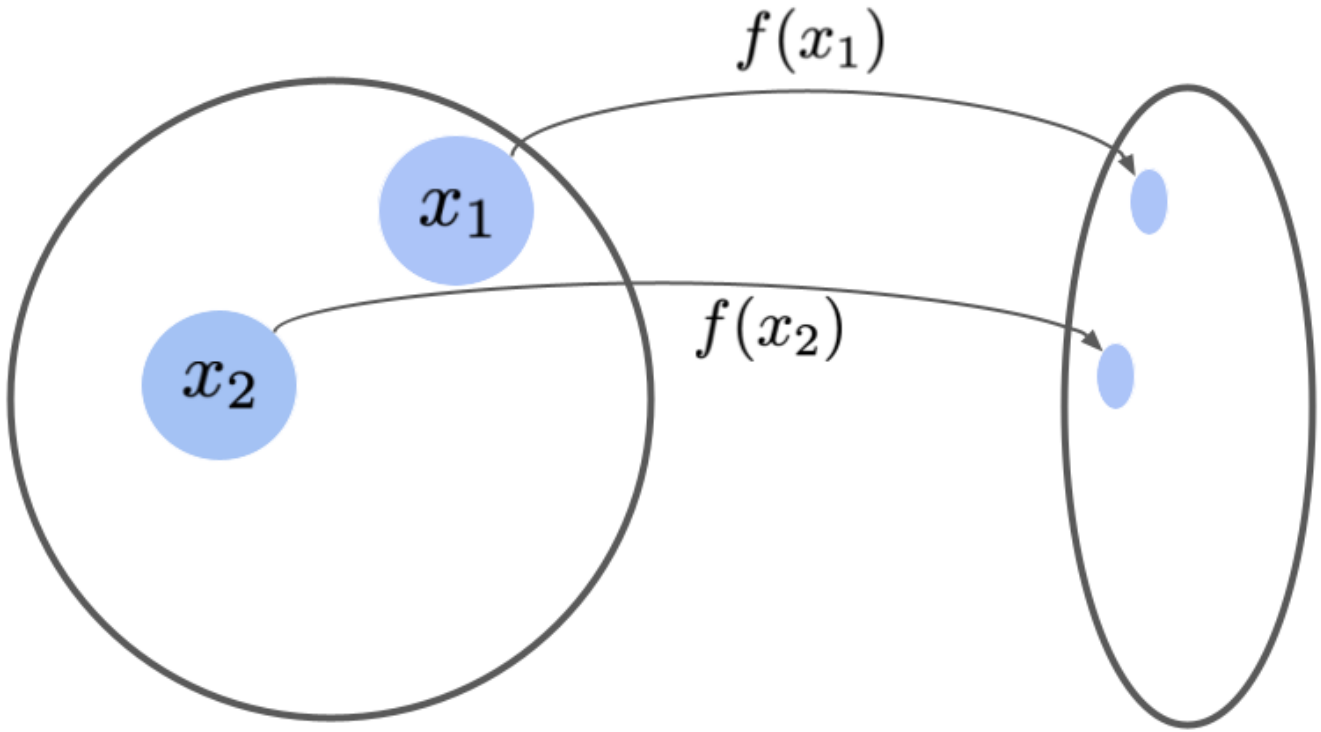
TL;DR

Bayesian model selection: 'Which model (i.e. prior distribution over parameters and likelihood function) is most likely to have generated my data?'

Non-Bayesian model selection: 'Which model (i.e. function from inputs to outputs produced by a learning algorithm) is going to be the most accurate on future data?'

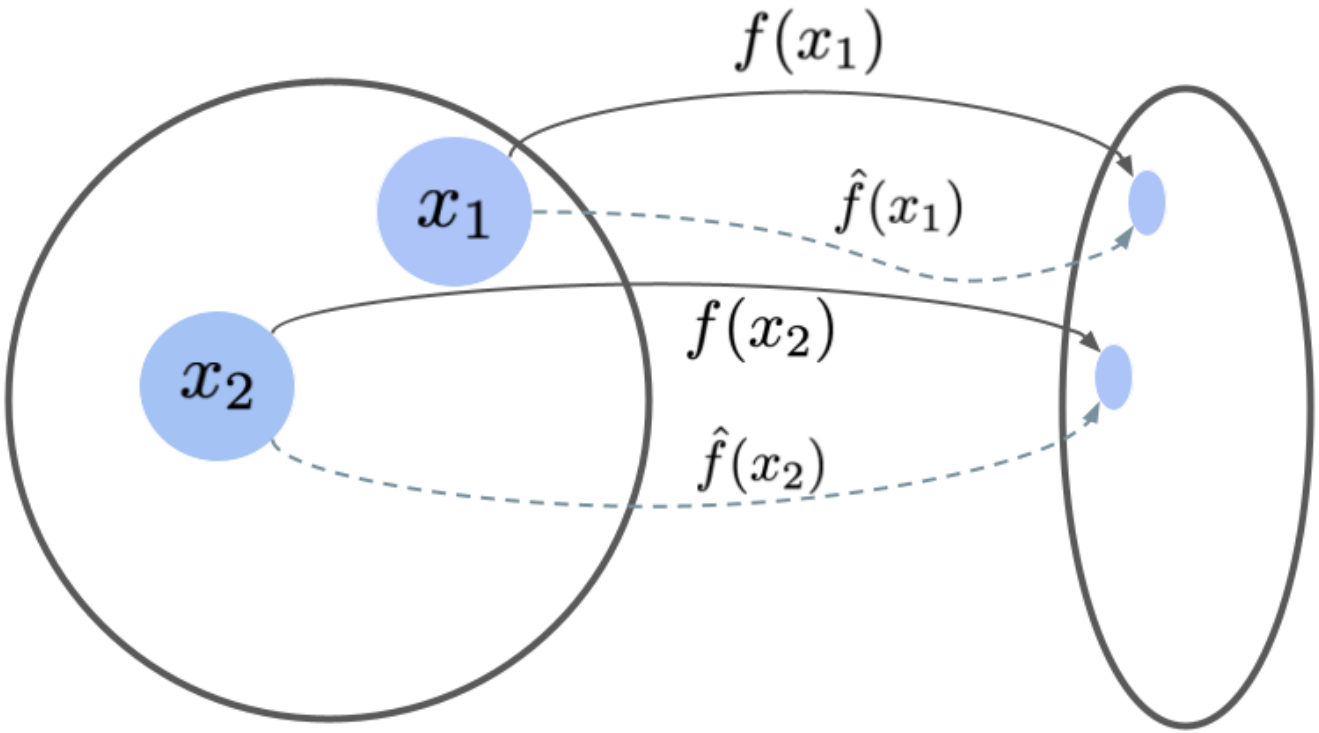
Non-bayesian model selection hasn't done a great job at explaining generalization in deep neural networks. Maybe Bayesian model selection is a better path forward?

- Given: some function f that maps inputs to outputs $\mathcal{X} \to \mathcal{Y}$.



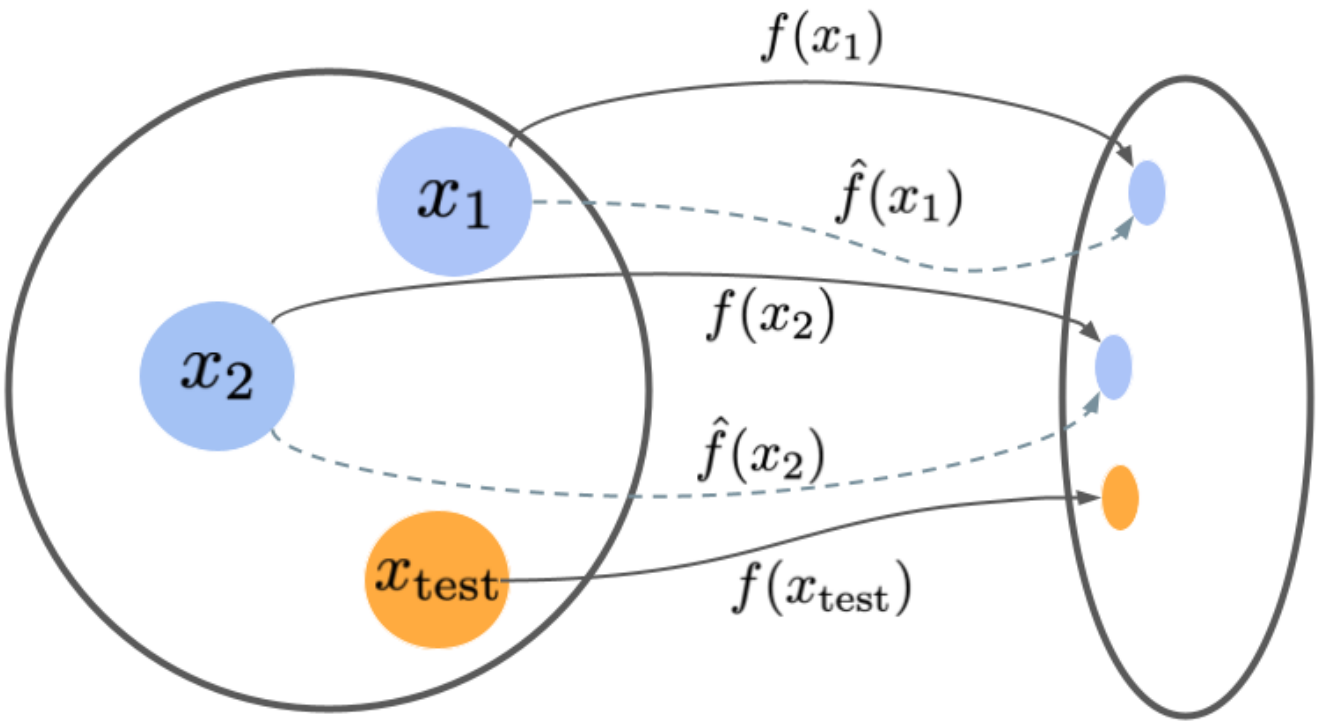
Preliminaries: generalization

- Want to know: if I find a function $\hat{f}$ that's close to f on a finite sample $\mathbf{X}_n, \mathbf{y}_n$, how close to f will it be on new inputs?



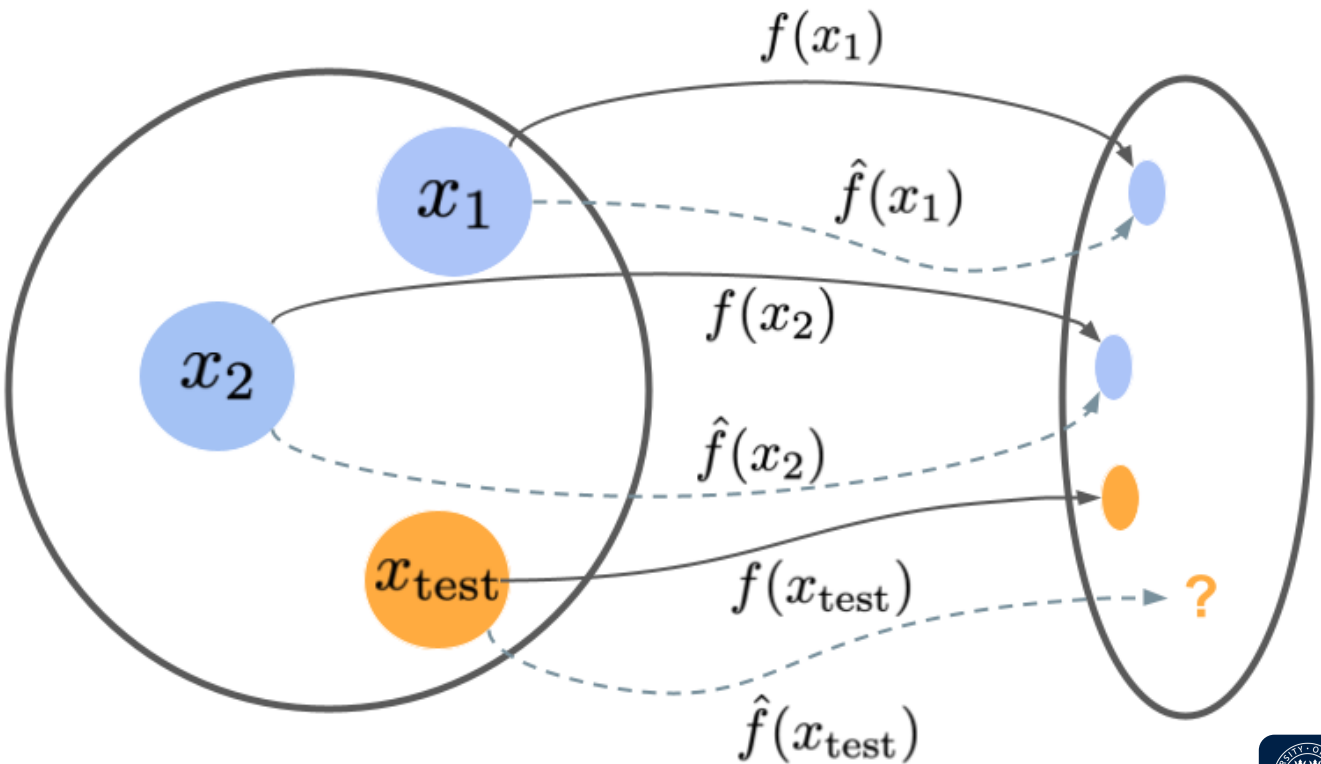
Preliminaries: generalization

- Want to know: if I find a function $\hat{f}$ that's close to f on a finite sample $\mathbf{X}_n, \mathbf{y}_n$, how close to f will it be on new inputs?



Preliminaries: generalization

- Want to know: if I find a function $\hat{f}$ that's close to f on a finite sample $\mathbf{X}_n, \mathbf{y}_n$, how close to f will it be on new inputs?



- Want to know: if I find a function $\hat{f}$ that's close to f on a finite sample $\mathbf{X}_n, \mathbf{y}_n$, how close to f will it be on new inputs?
- Want to minimize the expected error ℓ obtained by $\hat{f}$ on the distribution $P(X)$.

$$R_\ell(f) = \mathbb{E}_{x \sim P(X)}[\ell(f(x), \hat{f}(x))] \tag{1}$$

Problem: you have a set of functions $f_1, \ldots, f_n$. You want to pick the one with the best $R_\ell(f_i)$, but you don't have access to P except through samples.

Easy solution: pick the function ($\sim$ model) with the best validation loss!

Easy solution: pick the function ($\sim$ model) with the best validation loss! This has limitations:

- **Data-expensive** – requires leaving some data out.
- **Computationally expensive** – requires training the set of models to completion.
- Doesn't help us to **understand** what makes deep neural networks generalize.

Easy solution: pick the function ($\sim$ model) with the best validation loss! This has limitations:

- **Data-expensive** – requires leaving some data out.
- **Computationally expensive** – requires training the set of models to completion.
- Doesn't help us to **understand** what makes deep neural networks generalize.
- ... and therefore doesn't give great insights into **how to train even better models** later.

- **Scientific theory** $\approx$ an explanation of a phenomenon that makes testable predictions.
- Not just a set of predictions (e.g. validation loss).
- Not just an intuitive explanation of known phenomena

Generalization Bounds

Can we use high-probability bounds to explain generalization?

$$R_\ell(f) \overset{\text{w.p.}1-\delta}{\leq} \overbrace{\widehat{R}(f)}^{\text{training loss}} + \overbrace{\sqrt{\frac{C(f)}{n}}}^{\text{'Model complexity' term}} \tag{2}$$

Ideally, model complexity term should provide a theory (as defined in previous slide) for why some models generalize.

Generalization Bounds

Problem: most of these bounds are vacuous, and often *increasing* the 'model complexity' term *improves* generalization performance [Jiang et al., 2019].

$$R_\ell(f) \overset{\text{w.p.}1-\delta}{\leq} \overbrace{\widehat{R}(f)}^{\text{training loss}} + \sqrt{\frac{\overbrace{\Pi_{i=1}^{d}\|W_i\|_2^2 \sum_{j=1}^{d} \frac{\|w_j - w_j^0\|_F^2}{\|w_j\|_2^2}}^{\text{Spectral norm of trained weights}}}{n}}$$

Problem: most of these bounds are vacuous, and often *increasing* the 'model complexity' term *improves* generalization performance [Jiang et al., 2019].

$$R_\ell(f) \overset{\text{w.p.}1-\delta}{\leq} \overbrace{\widehat{R}(f)}^{\text{training loss}} + \overbrace{\sqrt{\frac{\mathrm{KL}(Q||P) + \log(n/\delta)}{2n}}}^{\text{KL between prior and posterior}}$$

Theory $\iff$ practice

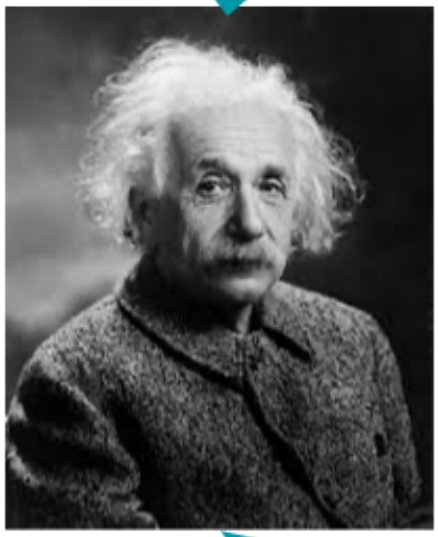
generalization in deep neural networks

Old theories of generalization break in NNs

[Zhang et al., 2016.]

Theory

"If your model can memorize random labels, we can't guarantee it won't overfit"

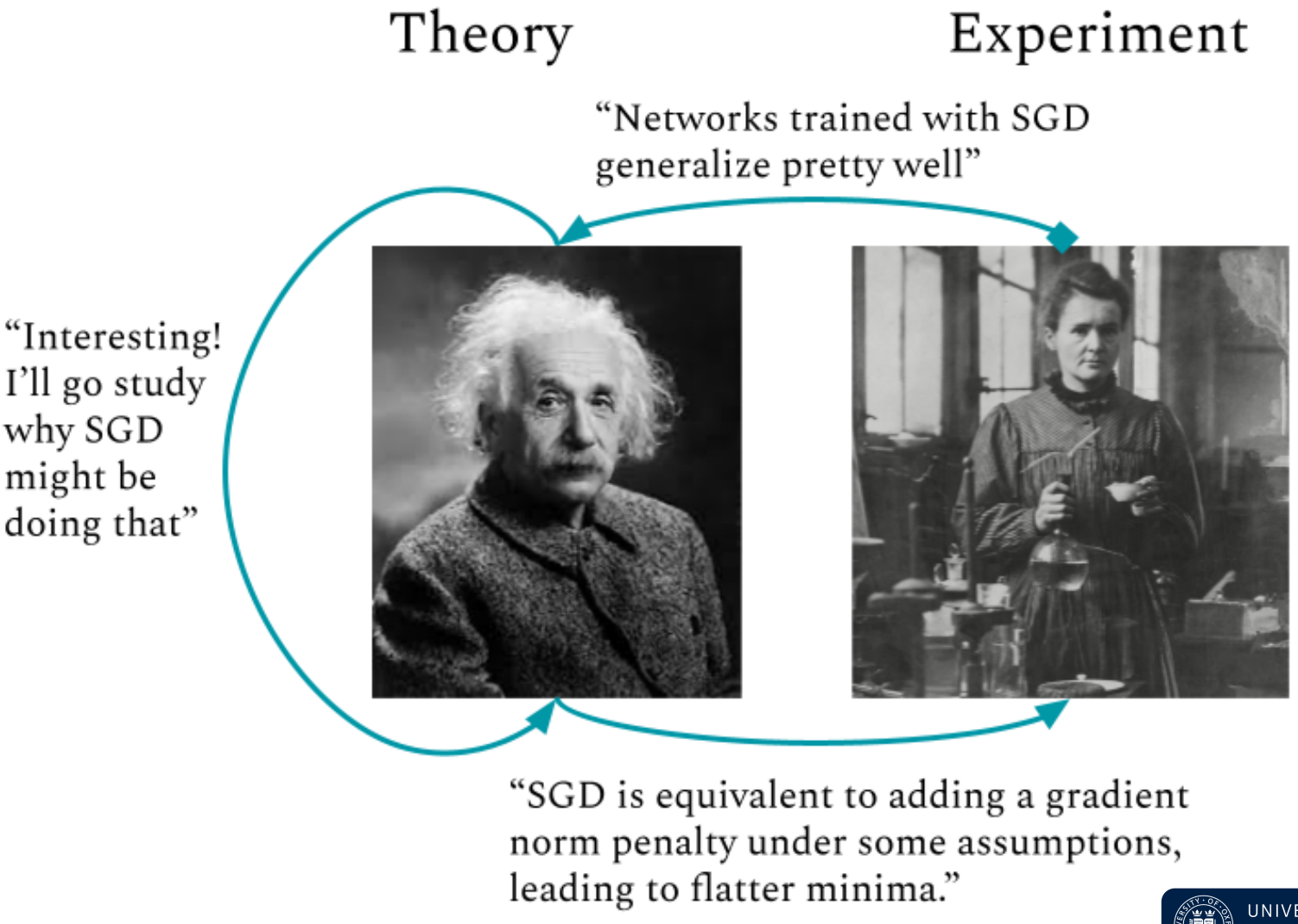


Experiment

"Interesting! I'll go check if neural networks can memorize random labels"

"So... it turns out neural networks can memorize random labels and they don't overfit on normal data."

Theorists can explain some empirical phenomena



The state of the world

We know lots of properties that (empirically) make a model generalize better.

The state of the world

We know lots of properties that (empirically) make a model generalize better.

- Flat minima generalize better than sharp minima
- Models which share known geometric properties with the data-generating distribution generalize well.

The state of the world

We know lots of properties that (empirically) make a model generalize better.

- Flat minima generalize better than sharp minima
- Models which share known geometric properties with the data-generating distribution generalize well.
- Smooth functions tend to generalize well.
- Models trained with stochastic optimizers tend to generalize better.

But we don't have a unified explanation of generalization and model selection in deep learning.

An alternative model selection approach.

Would ideally like a framework with

- a built-in Occam's razor

An alternative model selection approach.

Would ideally like a framework with

- a built-in Occam's razor
- a means of encoding preferences for properties like smoothness and symmetry

An alternative model selection approach.

Would ideally like a framework with

- a built-in Occam's razor
- a means of encoding preferences for properties like smoothness and symmetry
- a probabilistic framework that includes stochastic predictors by default.

An alternative model selection approach.

Would ideally like a framework with

- a built-in Occam's razor [MacKay, 2003]
- a means of encoding preferences for properties like smoothness and symmetry [van der Wilk et al., 2018]
- a probabilistic framework that includes stochastic predictors by default. [Germain et al., 2016]

Bayesian Inference

Type I and Type II Bayesian Inference

Bayesian inference can occur on two levels.

- **Model fitting (Type I)** assumes the data $\mathcal{D}$ was generated by a specific model $\mathcal{H}_i$ and computes a posterior belief distribution over parameters w given this data.
- **Model selection (Type II)** takes a set of models $\mathcal{H}_1, \ldots, \mathcal{H}_n$, and computes a posterior belief about how likely each model is to have generated the data $\mathcal{D}$.

Marginal Likelihood

The marginal likelihood is the quantity $P(\mathcal{D}|\mathcal{H}_i)$ that we need to compute the posterior over models in Type II inference.

$$P(\mathcal{D}|\mathcal{H}_i) = \int_w P(\mathcal{D}|w)p(w)dw \tag{2}$$

Prequential Coding Perspective

We can write $\log P(\mathcal{D}|\mathcal{H})$ as follows:

$$\log P(\mathcal{D}|\mathcal{H}) = \sum_{i=1}^{n} \log P(\mathcal{D}_i|\mathcal{D}_{<i}, \mathcal{H}) \tag{3}$$

Intuitively, the marginal likelihood measures how much information each successive data point gives about the model.

The Marginal Likelihood and Cross-Validation

This looks superficially like leave-one-out cross-validation.

$$CV(\mathcal{D}) = \overbrace{\mathbb{E}_{\sigma \in \Sigma_n}[\log P(\mathcal{D}_{\sigma(n)}|\mathcal{D}_{\sigma(<n)})]}^{\text{loglik. of } \mathcal{D}_{\sigma(n)} \text{ over permutations } \sigma} \tag{4}$$

Cross-validation looks only at final data point; the marginal likelihood looks at *all* data points.

$$P(\mathcal{D}|\mathcal{H}) = P(\mathcal{D}_1|\mathcal{H})P(\mathcal{D}_2|\mathcal{D}_1,\mathcal{H})P(\mathcal{D}_3|\mathcal{D}_{1,2},\mathcal{H})\ldots \overbrace{P(\mathcal{D}_n|\mathcal{D}_{<n},\mathcal{H})}^{\text{LOO-CV}}$$

Model Selection vs Model Averaging

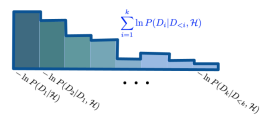
Option 1: average over models according to posterior weight.

$$P(\mathcal{H}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{H})P(\mathcal{H})}{P(\mathcal{D})} \tag{5}$$

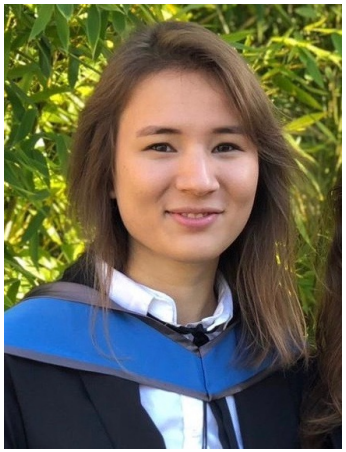
Option 2: pick the model with the highest likelihood.

$$\mathcal{H}^* = \max_{\mathcal{H}} \frac{P(\mathcal{D}|\mathcal{H})P(\mathcal{H})}{P(\mathcal{D})} \approx \max_{\mathcal{H}} P(\mathcal{D}|\mathcal{H}) \tag{6}$$

Bayesian Model Selection and Training Speed

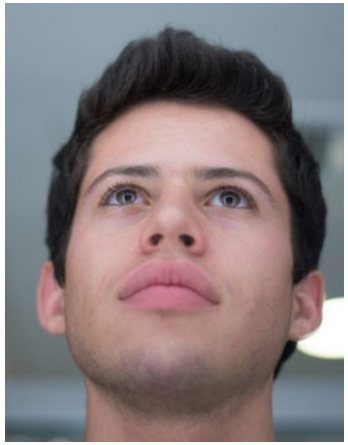


Credits



Lisa Schut

Robin Ru

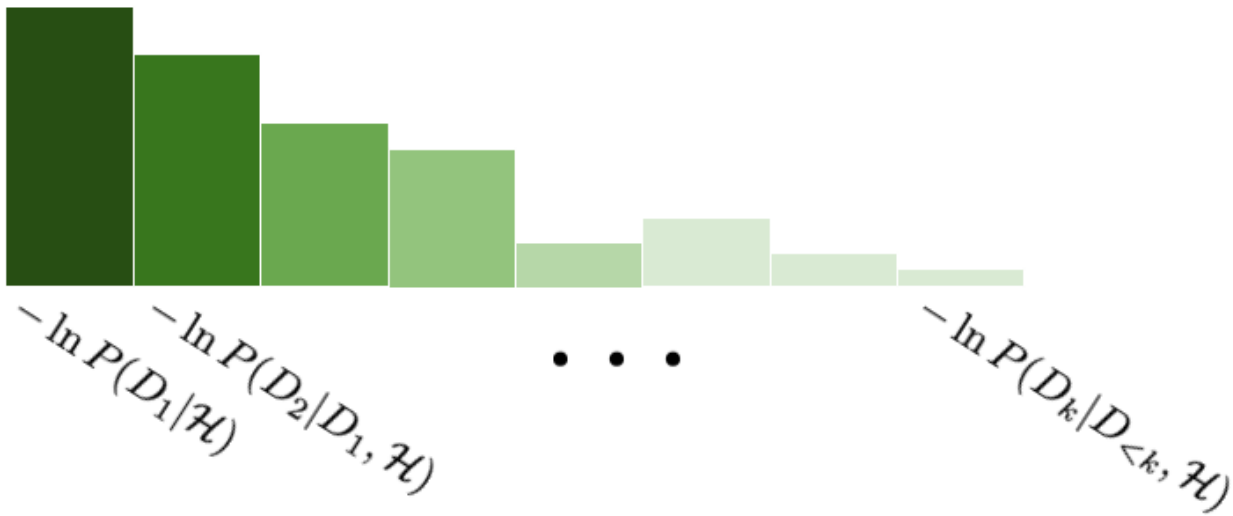


Yarin Gal

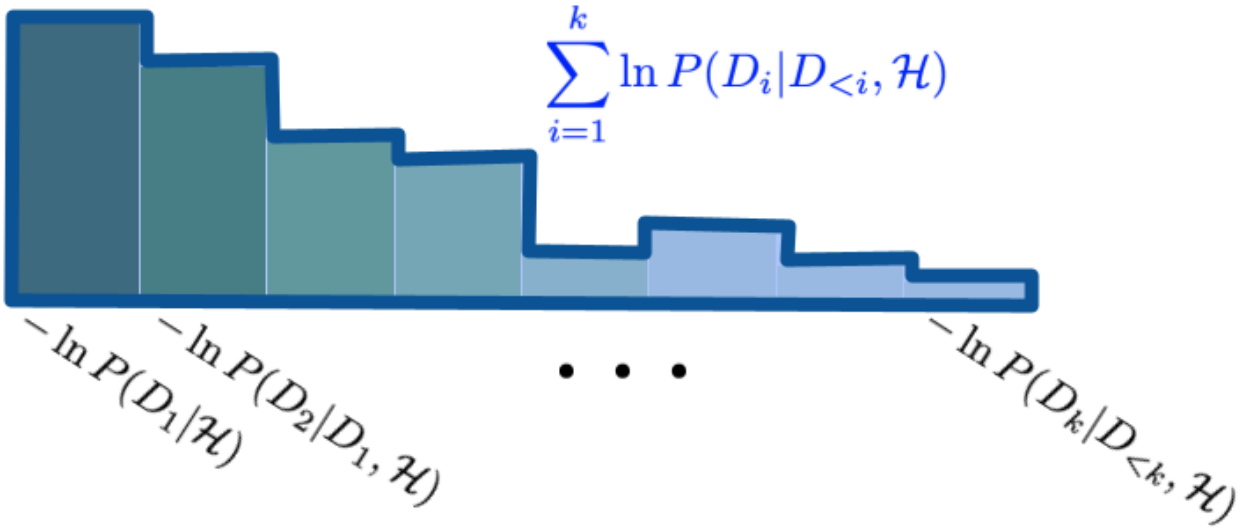
Mark van der Wilk

[Lyle et al., 2020; Ru et al., 2021]

Marginal Likelihood and Training Speed



Marginal Likelihood and Training Speed



Lower bounding the ML

Observation: Can write (a lower bound on) the log marginal likelihood as a sum of "training losses".

$$\log P(\mathcal{D}) = \sum_{i=1}^{n} \log[\mathbb{E}_{P(\theta|\mathcal{D}_{<i})} P(\mathcal{D}_i|\theta)]$$

$$\text{(by Jensen's inequality)} \geq \sum_{i=1}^{n} \mathbb{E}_{P(\theta|\mathcal{D}_{<i})}[\log P(\mathcal{D}_i|\theta)]$$

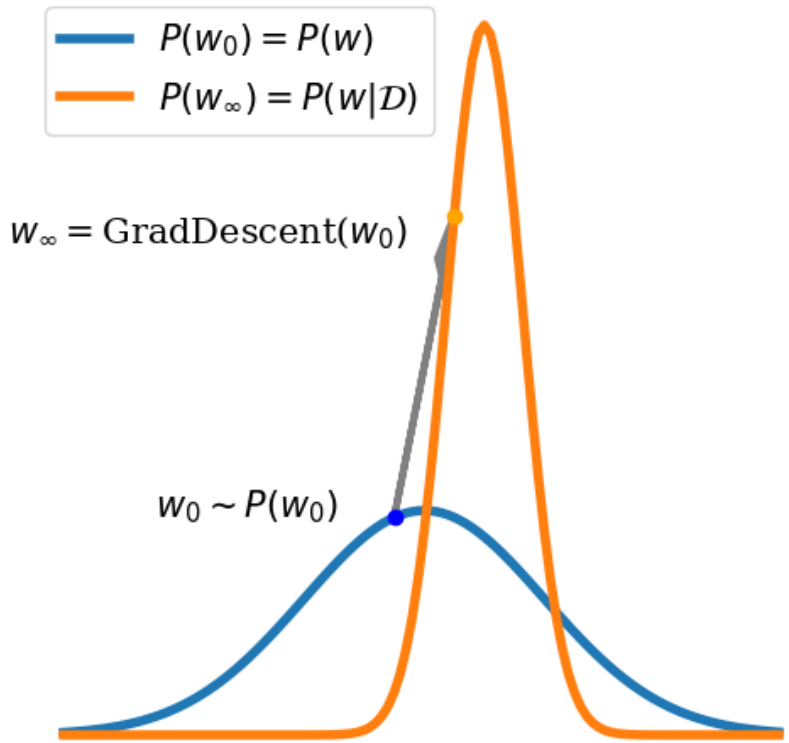
$$= \mathcal{L}(\mathcal{D}) \iff -\sum \mathbb{E}_{\theta}[\ell(f_\theta(x_i), y_i)]$$

We can estimate a **lower bound** on the marginal likelihood using samples from the posterior

$$\hat{\mathcal{L}}(\mathcal{D}) = \sum_{i=1}^{n} \frac{1}{k} \sum_{j=1}^{k} \log P(\mathcal{D}_i|\theta_j) \qquad \hat{\mathcal{L}}_k(\mathcal{D}) = \sum_{i=1}^{n} \log \frac{1}{k} \sum_{j=1}^{k} P(\mathcal{D}_i|\theta_j). \tag{7}$$

Posterior Sampling via Gradient Descent

For linear models and the infinite-width limit of neural networks, we can get **posterior samples** using **gradient descent**.



Algorithm 1: Marginal Likelihood Estimation for Linear Models

Input: A dataset $\mathcal{D} = (x_i, y_i)_{i=1}^n$, parameters $\mu_0, \sigma_0^2, \sigma_N^2$
Result: An estimate of $\mathcal{L}(\mathcal{D})$

$\theta_t \leftarrow \theta_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$; $\tilde{Y} \leftarrow Y + \epsilon \sim \mathcal{N}(0, \sigma_N^2)$; sumLoss $\leftarrow 0$;

$\ell(\mathcal{D}_{\leq i}, w) \leftarrow \|\tilde{Y}_{\leq i} - \theta^\top X_{\leq i}\|_2^2 + \frac{\sigma_N^2}{\theta_0^2}\|\theta - \theta_0\|_2^2$;

for $\mathcal{D}_i \in \mathcal{D}$ **do**

  sumLoss = sumLoss + $\frac{(\theta_t^\top x_i - y_i)^2}{2\sigma_N^2}$;

  $\theta_t \leftarrow$ GradientDescent$(\ell, \theta_t, \mathcal{D}_{\leq i})$;

end

return sumLoss

What about neural networks?

Lots of great papers have used the *Laplace approximation* to estimate marginal likelihood in neural networks [Smith &. Le, 2017; Immer et al., 2021].

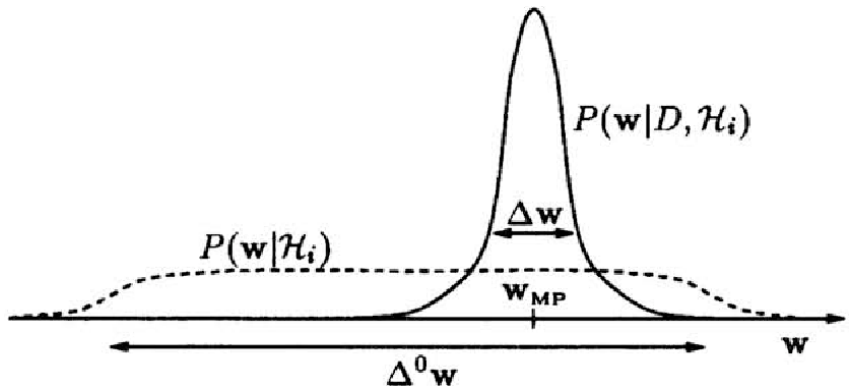


Figure: MacKay, 2003

Flat loss landscape $\implies$ higher marginal likelihood.

A lazier option

Laplace approximations require performing additional computation or modifying your training procedure.

A lazier option

Laplace approximations require performing additional computation or modifying your training procedure.

Question: Can we do something that looks like marginal likelihood estimation without doing any additional computation?

A lazier option

Laplace approximations require performing additional computation or modifying your training procedure.

Question: Can we do something that looks like marginal likelihood estimation without doing any additional computation?

Answer: Yes!

A lazier option

Laplace approximations require performing additional computation or modifying your training procedure.

Question: Can we do something that looks like marginal likelihood estimation without doing any additional computation?

Answer: Yes! (Sort of, if you don't care about theoretical guarantees and you squint really hard.)

TSE: Training Speed Estimator

Recall:

$$\log P(\mathcal{D}|\mathcal{H}) = \sum_{i=1}^{n} \overbrace{\log P(\mathcal{D}_i|\mathcal{D}_{<i}, \mathcal{H})}^{\text{sum of log likelihoods}} \tag{8}$$

TSE: Training Speed Estimator

Recall:

$$\log P(\mathcal{D}|\mathcal{H}) = \sum_{i=1}^{n} \overbrace{\log P(\mathcal{D}_i|\mathcal{D}_{<i}, \mathcal{H})}^{\text{sum of log likelihoods}} \tag{8}$$

Naive application to SGD trajectories:

$$\text{TSE} = \sum_{t=1}^{T} \left[\frac{1}{B} \sum_{i=1}^{B} \overbrace{\ell\left(f_{\theta_{t,i}}(\mathbf{X}_i), \mathbf{y}_i\right)}^{\text{sum of training losses}} \right] \tag{9}$$

Is the TSE metric useful for non-Bayesian model selection?

- Want a cheap estimator to tell us whether a network architecture is good
- ideally, this estimator should work well with early stopping
- and give us a reliable ranking of which architecture will obtain the best performance after training

Training Speed as a Performance Estimator

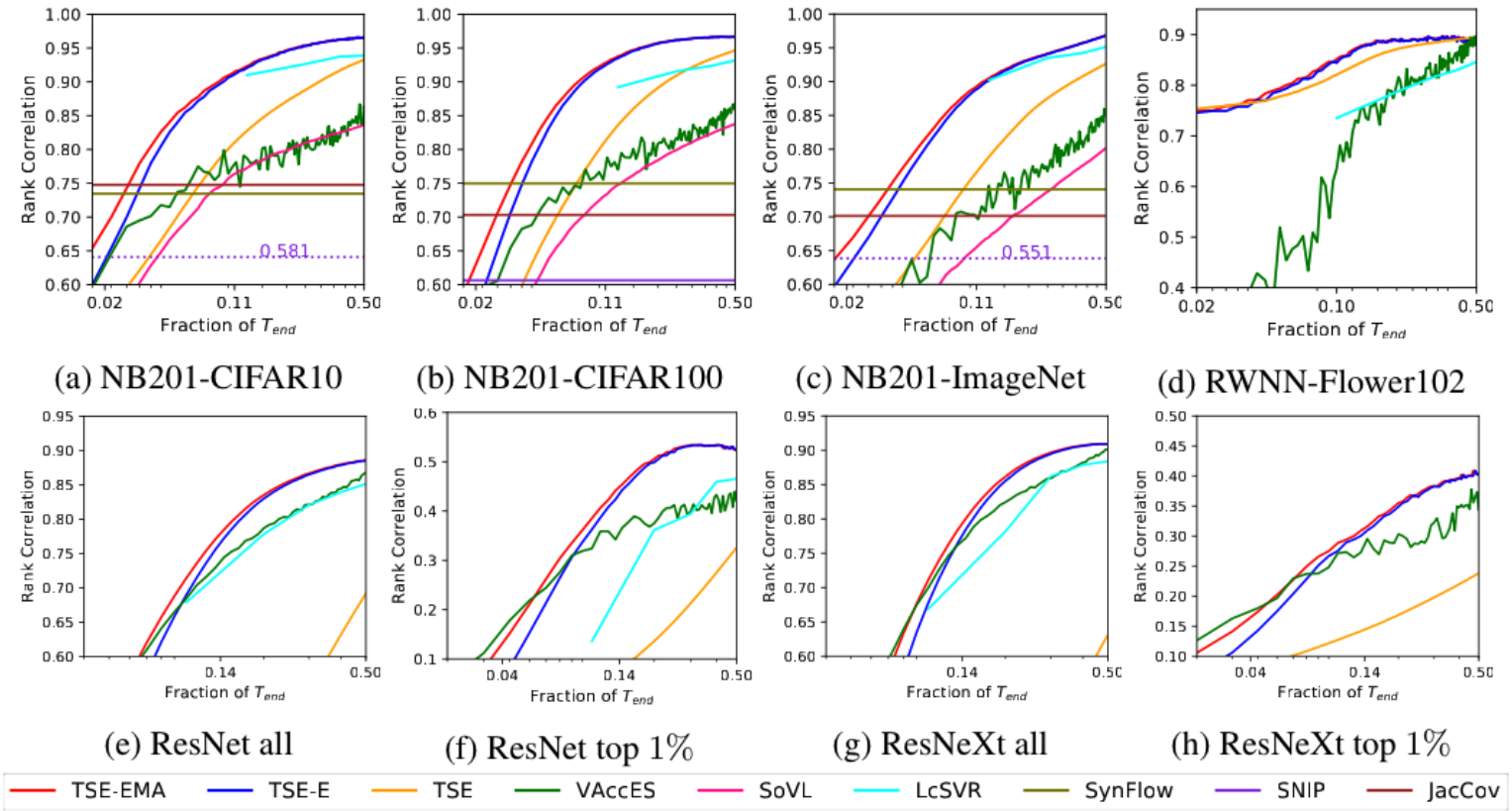


Figure: Rank correlation between final test accuracy and several performance estimators computed at different points in training.

Training speed vs validation loss

The validation loss is an estimator of how much information the training set has given about the test set so far

$$P(\mathcal{D}|\mathcal{H}) = P(\mathcal{D}_1|\mathcal{H})\overset{\mathcal{D}_1\to\mathcal{D}_2}{P}(\mathcal{D}_2|\mathcal{D}_1,\mathcal{H})\overset{\mathcal{D}_{1,2}\to\mathcal{D}_3}{P}(\mathcal{D}_3|\mathcal{D}_{1,2},\mathcal{H})\ldots\overbrace{P(\mathcal{D}_n|\mathcal{D}_{<n},\mathcal{H})}^{\text{LOO-CV}}$$

The marginal likelihood (and the estimator described previously) looks iteratively at how updating on one data point affects the rest of the training set.

Is the TSE metric useful for neural architecture search?

- High rank correlation doesn't guarantee good performance in NAS
- E.g. if the method can't distinguish between 'good' and 'excellent' architectures, it might not produce great results.
- This is what we evaluate next.

Results in Neural Architecture Search

B	Estimator	Rank Correlation				Average Accuracy of Top 10 Architectures			
		NB201-CIFAR10			DARTS	NB201-CIFAR10			DARTS
		RandNAS	FairNAS	MultiPaths	RandNAS	RandNAS	FairNAS	MultiPaths	RandNAS
100	TSE	**0.70 (0.02)**	**0.84 (0.01)**	**0.83 (0.01)**	**0.30(0.04)**	**92.67 (0.12**	**92.7 (0.1)**	**92.63 (0.12)**	**93.64(0.04)**
	Val Acc	0.44 (0.15)	0.56 (0.17)	0.67 (0.05)	0.11(0.04)	91.47 (0.31)	91.73 (0.21)	91.77 (0.78)	93.20(0.04)
200	TSE	**0.70 (0.03)**	**0.850 (0.01)**	**0.83 (0.01)**	**0.32(0.04)**	**92.70 (0.00)**	**92.77 (0.06)**	**92.73 (0.06)**	**93.55(0.04)**
	Val Acc	0.41 (0.10)	0.56 (0.17)	0.53 (0.11)	0.09(0.02)	91.53 (0.55)	92.40 (0.10)	92.23 (0.23)	93.34(0.02)
300	TSE	**0.71 (0.03)**	**0.851 (0.00)**	**0.82 (0.01)**	**0.34(0.04)**	**92.70 (0.00)**	**92.77 (0.06)**	**92.70 (0.00)**	**93.65(0.04)**
	Val Acc	0.44 (0.04)	0.62 (0.08)	0.59 (0.71)	0.06(0.02)	91.20 (0.35)	92.10 (0.50)	91.43 (0.72)	93.31(0.02)

Substituting training speed for validation accuracy improves performance in several NAS methods.

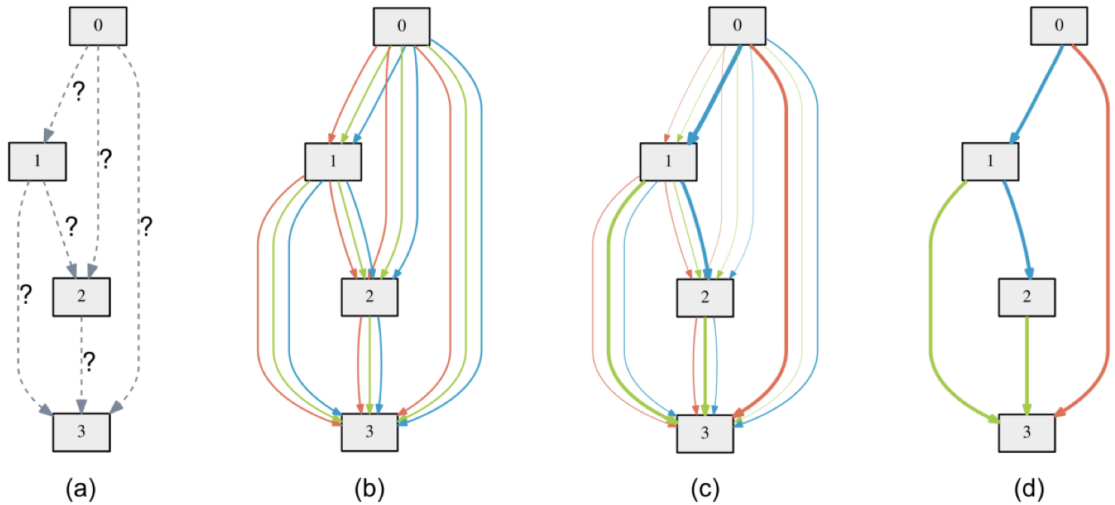
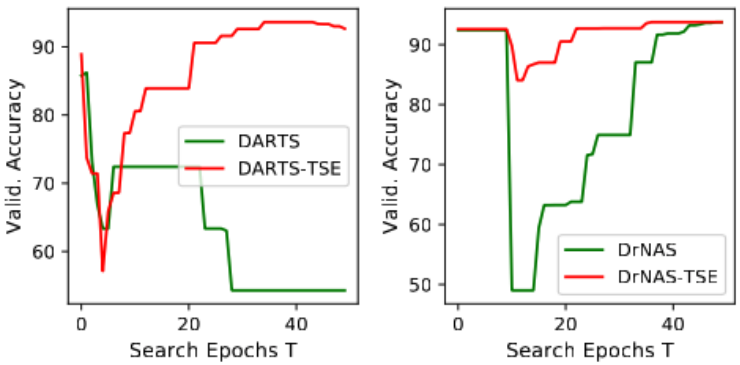
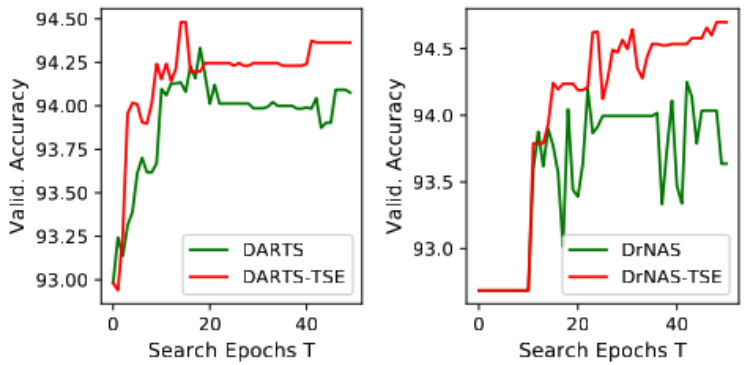


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Can you differentiate through training speed?



(a) NB201-CIFAR10



(b) DARTS-NB301

Using TSE instead of validation loss improves performance in differentiable NAS methods.

Understanding Training Speed

Train faster, generalize better?

Prior work (Hardt et al., 2016) has studied the training speed-generalization connection:

$$\overbrace{\mathbb{E}_z[f(w_T; z) - f(w'_T; z)]}^{\epsilon \text{ term to bound generalization error}} \leq \overbrace{\frac{t_0}{n} \sup f(w; z)}^{\text{linear in number of steps}} + \overbrace{L\mathbb{E}[\delta_T | \delta_{t_0} = 0]}^{\text{Non-negative}}$$

High-probability guarantees, but vacuous for long training runs!

SOTL is measuring something different.

What is SOTL measuring?

Hypotheses:

- **Gradient correlation.** If the gradient step for one minibatch lowers the loss on the rest of the training set, then the training loss will decrease more quickly than if updates only affect the minibatch they're computed on.

What is SOTL measuring?

Hypotheses:

- **Gradient correlation.** If the gradient step for one minibatch lowers the loss on the rest of the training set, then the training loss will decrease more quickly than if updates only affect the minibatch they're computed on.

- **Flatness.** In a flat loss landscape, it may be easier to quickly find a good minimum. In later stages of training, loss is equal to loss of minimum + parameter noise, smoother loss surfaces produce less noise.

Training speed doesn't make predictions w.r.t. absolute value of test set error, but does make predictions on relative performance of comparable models.

- **Gradient correlation view** predicts that models which generalize well to test data will also exhibit good generalization between disjoint minibatches in the training data.
- **Flatness view** predicts that flatter loss landscapes contain functions that generalize well.

- Data augmentation
- Dropout
- Comparisons across different network families

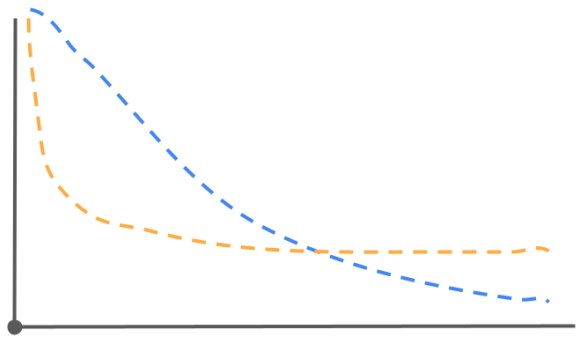


Figure: A slow-and-steady tortoise vs a hare that plateaus early.

Discussion

Understanding training speed

- What properties is training speed measuring?
- How can we directly incentivize these properties during training?
- How does this relate to existing approaches in meta- and continual learning?
- What are other ways that the prequential coding framing of the marginal likelihood might be useful?

- Type II maximum likelihood answers the question: 'which model is most likely to have generated this data?'
- It does **not** answer: 'which model will assign the highest likelihood to the data I'll see in the future?'

- Type II maximum likelihood answers the question: 'which model is most likely to have generated this data?'
- It does **not** answer: 'which model will assign the highest likelihood to the data I'll see in the future?'
- It therefore doesn't directly give worst-case bounds on performance on future data points.
- How can we bridge the gap between heuristics for model selection and provable guarantees?


Discussion period