

10707

Deep Learning: Spring 2021

Andrej Risteski

Machine Learning Department

Lecture 7:

Generalization in deep
learning

Recap: definition of generalization

Generalization: difference in performance on train set vs all data.

The i.i.d. assumption: the samples in training set are **identically, independently distributed**.

Mathematically, there is an underlying **data distribution** \mathcal{D} . A new training sample is generated by drawing a sample from \mathcal{D} independently from the previous ones.

“Ideal” (all data) loss is the expected loss over \mathcal{D} . We minimize expected loss over the training set. Hope: for (not too large training sets), training loss \approx ideal loss.

Theoretical idealization, broken in many ways in practice:

Data is not “identically” distributed, e.g.:

Images (ala Imagenet) are scraped in slightly different ways.

Data has systemic bias (e.g. patients are tested based on symptoms they exhibit)

Data is result of interaction (e.g. agent interacting with environment in RL)

Recap: definition of generalization

Generalization: difference in performance on training set vs all data.

The **i.i.d.** assumption
identically, independent

Mathematically, the data is
generated by drawing samples

“Ideal” (all data) loss is
training set. Hope: generalization

Theoretical idealization

Data is not “identical”

Images (ala ImageNet)

Data has systemic bias (e.g. patients are tested based on symptoms they exhibit)

Data is result of interaction (e.g. agent interacting with environment in RL)

But!!
We **really** don't
understand even this
simple idealization.

training sample is different from previous ones.

expected loss over the whole data vs ideal loss.

Recap: definition of generalization

Mathematical formalization of generalization

For a classifier f , and loss l , the **generalization gap** is:

$$|\mathbb{E}_{(x,y): \text{random training sample}} l(f(x), y) - \mathbb{E}_{(x,y) \sim \mathcal{D}} l(f(x), y)|$$


As shorthand, we denote this as

$$\hat{\mathbb{E}} l(f(x), y) - \mathbb{E} l(f(x), y)$$

Good generalization: $\hat{\mathbb{E}} l(f(x), y) \approx \mathbb{E} l(f(x), y)$

Bad generalization\overfitting: $\hat{\mathbb{E}} l(f(x), y) \ll \mathbb{E} l(f(x), y)$

The classifier f “looks” much better on the training set



Classical view of generalization

Decoupled view of generalization and optimization: the generalization depends on the complexity of the predictor class considered.

Ignore the algorithm*: to have a good generalization, make sure that the predictor class we are using is not too “complex”.

Some popular strategies to do this:

Capacity control: bound the size of the network/amount of connections, clip the weights during training, etc.

Regularization: add to the objective a penalization term for “complex” predictors, e.g. l_2 norm of weights of network, dropout, etc.

Classical view of generalization

Meta-“theorem” of generalization: with probability $1 - \delta$ over the choice of a training set of size m , we have

$$\sup_{f \in \mathcal{F}} | \hat{\mathbb{E}} l(f(x), y) - \mathbb{E} l(f(x), y) | \leq O \left(\sqrt{\frac{\text{complexity}(\mathcal{F}) + \ln 1/\delta}{m}} \right)$$

Some measures of “complexity”:

(Effective) number of elements in \mathcal{F}

VC (Vapnik-Chervonenkis)

Rademacher complexity

PAC-Bayes

Basic version: Chernoff and union bound

Basic version (discrete classes): with probability $1 - \delta$ over the choice of a training set of size m , for a loss l bdd. by 1, we have

$$\sup_{f \in \mathcal{F}} |\hat{\mathbb{E}} l(f(x), y) - \mathbb{E} l(f(x), y)| \leq O\left(\sqrt{\frac{\ln |\mathcal{F}| + \ln 1/\delta}{m}}\right)$$

Proof: (1): Write empirical loss as $\hat{\mathbb{E}} l(f(x), y) = \frac{1}{m} \sum_i l(f(x_i), y_i)$

Consider a **fixed** classifier f . As this is an average of i.i.d random variables, each bounded by 1, the Chernoff bound applies. Recall:

Chernoff bound: Let $X_i, i \in [n]$ be i.i.d. random variables bounded by 1 with expectation μ . Then,

$$\Pr \left[\left| \frac{1}{m} \sum_i X_i - \mu \right| \geq \alpha \right] \leq 2e^{-2\alpha^2 m}$$

Hence, apply **Chernoff** with $l(f(x_i), y_i)$ corresponding to X_i and **union bounding**:

$$\Pr_{\exists f} \left[\left| \hat{\mathbb{E}} l(f(x), y) - \mathbb{E} l(f(x), y) \right| \geq \sqrt{\frac{\ln |\mathcal{F}| + \ln 1/\delta}{m}} \right] \leq \delta$$

Bounds in this family: VC dimension

Do we really need to consider ***every*** classifier in \mathcal{F} ?

Intuitively, **pattern of classifications** on the training set should suffice.

(Two predictors that predict identically on the train set should generalize “identically”)

Consider binary prediction (say, labels are $\{\pm 1\}$), with l being 0-1 loss.

Theorem (Vapnik-Chervonenkis): with probability $1 - \delta$ over the choice of a training set of size m , we have

$$\sup_{f \in \mathcal{F}} | \widehat{\mathbb{E}} l(f(x), y) - \mathbb{E} l(f(x), y) | \leq O \left(\sqrt{\frac{\mathbf{VCdim}(\mathcal{F}) \ln m + \ln 1/\delta}{m}} \right)$$

Bounds in this family: VC dimension

VC dimension complexity: “**how many patterns can you produce**”?

Let $\mathcal{F} = \{f: \mathcal{X} \rightarrow \{\pm 1\}\}$ be a class of predictors.

Max # of possible label sequences

The **growth function** $\Pi_{\mathcal{F}}: \mathbb{N} \rightarrow \mathbb{N}$ of \mathcal{F} is defined as

$$\Pi_{\mathcal{F}}(m) = \max_{(x_1, x_2, \dots, x_m)} |\{(f(x_1), f(x_2), \dots, f(x_m)) \mid f \in \mathcal{F}\}|$$

The **VC (Vapnik-Chervonenkis) dimension** of \mathcal{F} is defined as

$$\text{VCdim}(\mathcal{F}) = \max\{m: \Pi_{\mathcal{F}}(m) = 2^m\}$$

Equivalently: the largest m , s.t. \mathcal{F} can **shatter** some set of size m :

that is, for **some** (x_1, x_2, \dots, x_m) and **any** labeling (b_1, b_2, \dots, b_m) , $b_i \in \{\pm 1\}$,

some $f \in \mathcal{F}$ can produce that labeling, that is

$$(f(x_1), f(x_2), \dots, f(x_m)) = (b_1, b_2, \dots, b_m)$$

Bounds in this family: Rademacher complexity

Rademacher complexity: how well can a classifier “**fit random labels**”

For a training set $S = \{x_1, x_2, \dots, x_n\}$, and class \mathcal{F} , denote

$$\hat{R}_n(S) = \mathbb{E}_\sigma \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_i \sigma_i f_i(x_i)$$

Let $R_n = \mathbb{E}_S \hat{R}_n(S)$. Then:

$$R_m = O\left(\sqrt{\frac{VCdim(\mathcal{F}) \ln m}{m}}\right)$$

Let l be a loss bounded in $[0,1]$.

Theorem (Rademacher generalization): with probability $1 - \delta$ over the choice of a training set of size m , we have

$$\sup_{f \in \mathcal{F}} | \hat{\mathbb{E}} l(f(x), y) - \mathbb{E} l(f(x), y) | \leq 2R_m + O\left(\sqrt{\frac{\ln 1/\delta}{m}}\right)$$

Bounds in this family: PAC-Bayesian

PAC-Bayesian complexity: “**how much does algorithm memorize**”

Setup: Let P be a prior over predictors in class \mathcal{F} , let Q be a posterior (after algorithm’s training)

Theorem (PAC-Bayes): with probability $1 - \delta$ over the choice of a training set of size m , we have

$$| \widehat{\mathbb{E}}_Q l(f(x), y) - \mathbb{E}_Q l(f(x), y) | \leq \sqrt{\frac{\text{KL}(Q || P) + \ln 1/\delta}{m}}$$

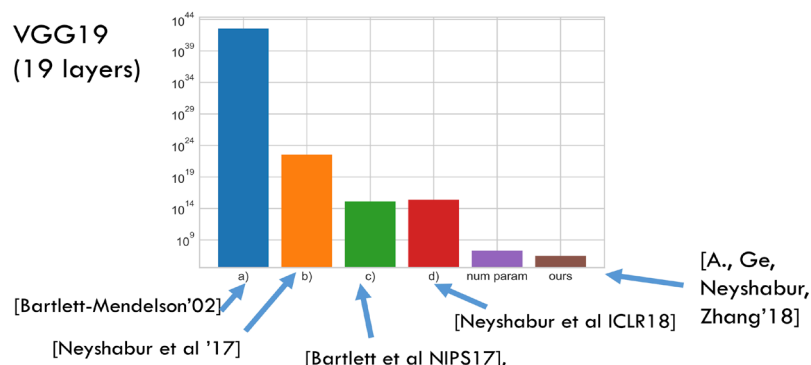
What do these bounds look like?

VC-dimension of **fully connected net** with **N** edges in total is $O(\mathbf{N \log N})$.

Note: doesn't depends on magnitude of weights at all.

Rademacher bounds: a lot of recent activity [*Neyshabur et al '15, 17; Bartlett-Foster-Telgarsky '18, Golowich-Rakhlin-Shamir '19*]. Roughly, best bounds look like:

$$R_m = O\left(\frac{\sqrt{\prod_{i=1}^d ||W_i||_2 \text{poly}(d)}}{\sqrt{m}}\right), \text{ where } d \text{ is the depth}$$



When plugged in for standard values used in practice, the values these quantities give are **rarely “non-trivial”** (i.e. give a quantity < 1)

(Jiang et al'19): a large-scale investigation of the correlation of extant generalization measures with true generalization.

They show some newer Rademacher bounds have *worse* correlation.

Problems with the standard view I

Modern models are **very** complex, by any measure of complexity.
(Not uncommon: # of params in neural net \gg # training samples)

Observations in seminal paper of *Zhang-Bengio-Hardt-Recht-Vinyals '17*:

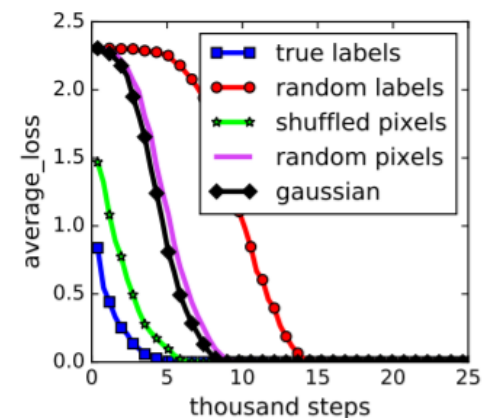
Modern architectures can fit **random noise!** (reminiscent of *high Rademacher complexity*).

Can achieve 0 training error *even if we*:

Randomly permute pixels by a **fixed** permutation.

Randomly permute pixels using a **different** permutation for each image.

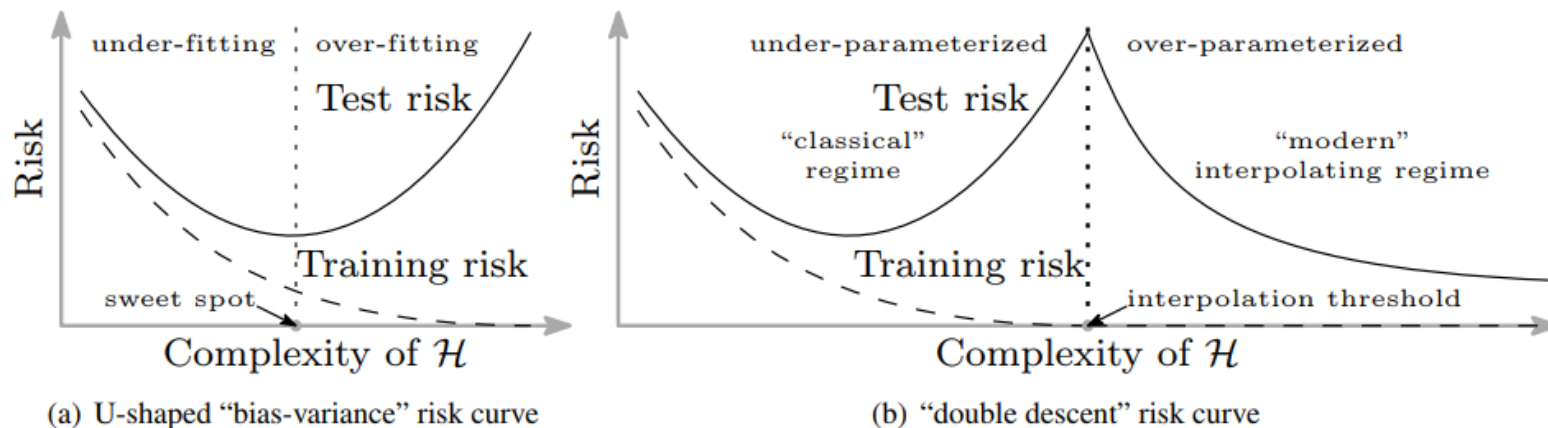
Replace the true labels by random labels;



(a) learning curves

Problems with the standard view II

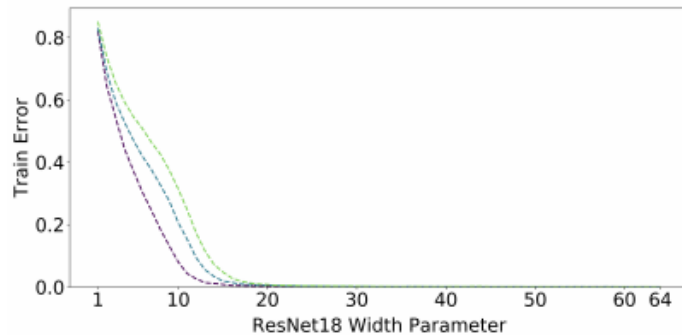
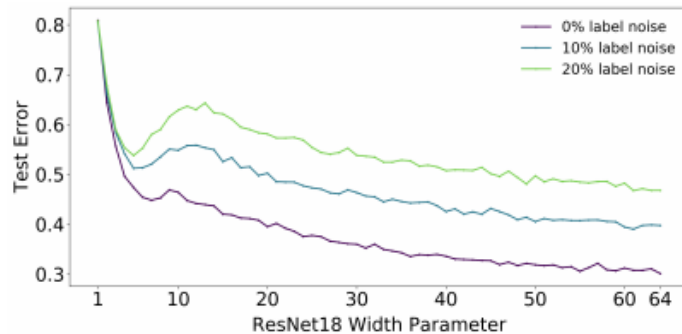
Classical view of the “model complexity” / “over-fitting” curve is broken, as observed in a seminal paper by *Belkin-Hsu-Ma-Mandal '18*



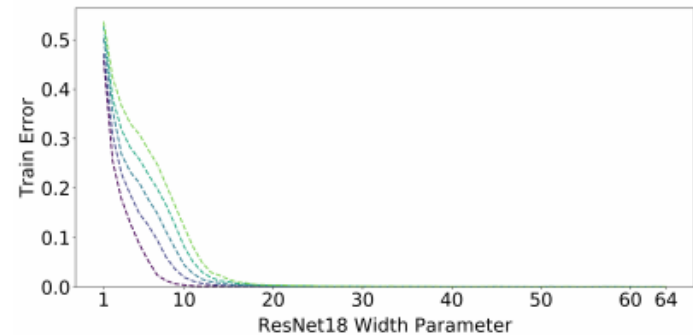
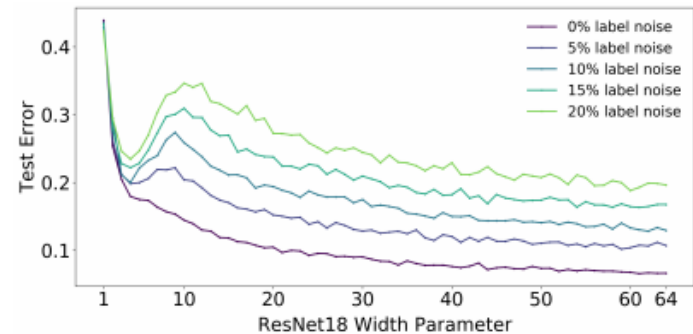
There are cases where the model gets bigger, yet the (test!) loss goes down, sometimes even lower than in the classical “underparametrized” regime.

Problems with the standard view II

Widespread phenomenon, across architectures, and **even other** quantities like train time, dataset size (Nakkiran et al '19):



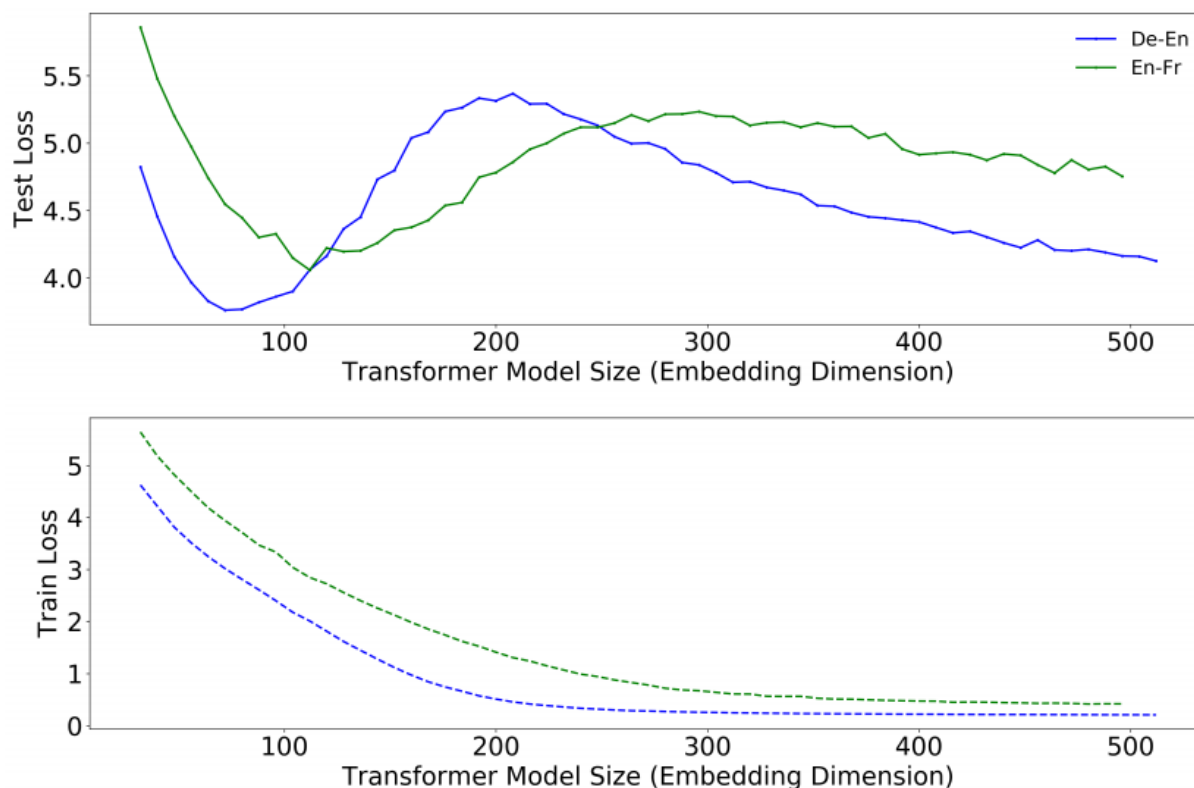
(a) **CIFAR-100**. There is a peak in test error even with no label noise.



(b) **CIFAR-10**. There is a “plateau” in test error around the interpolation point with no label noise, which develops into a peak for added label noise.

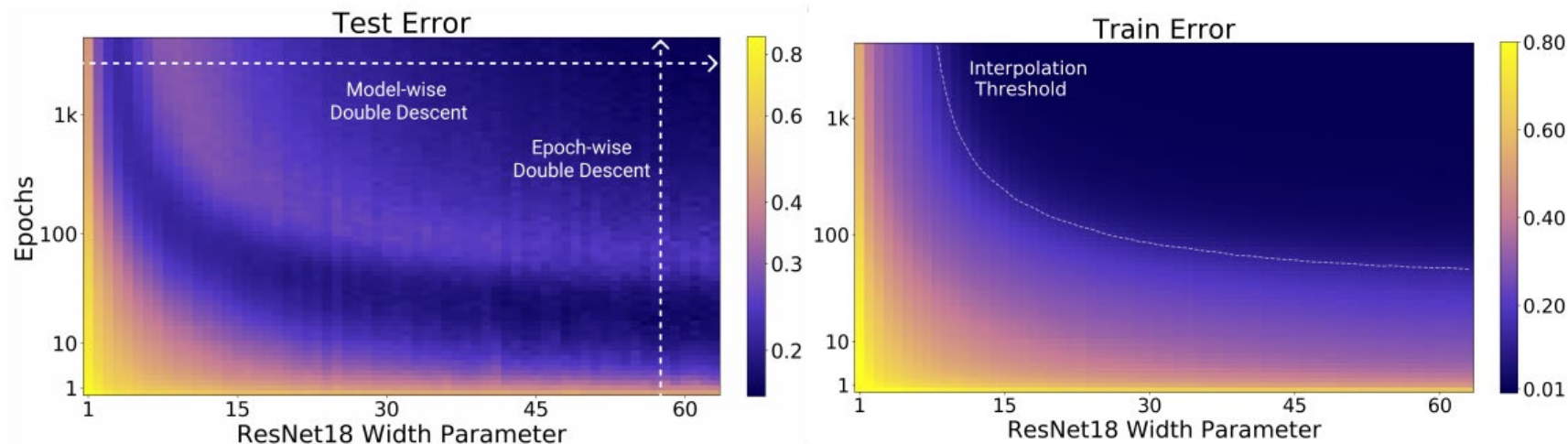
Problems with the standard view II

Widespread phenomenon, across architectures, and **even other** quantities like train time, dataset size (Nakkiran et al '19):



Problems with the standard view II

Widespread phenomenon, across architectures, and **even other** quantities like train time, dataset size (Nakkiran et al '19):



Train time manifests, but only for large enough models!

Best way to use computational budget

[Li et al. '20]: “*Train large, then compress*”: for NLP settings, the best usage of computational budget is to *train a very big model, stop early, and prune the model*.

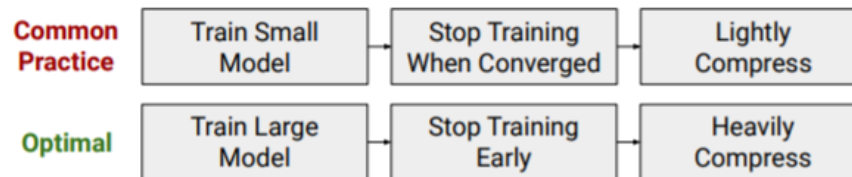


Figure 1. Under the usual presumption that models are trained to convergence, only small models that are fast-to-execute are feasible in resource-constrained settings. Our work shows that the most compute-efficient training scheme is instead to train very large models, stop them well short of convergence, and then heavily compress them to meet test-time constraints.

Best way to use computational budget

[Li et al. '20]: “*Train large, then compress*”: for NLP settings, the best usage of computational budget is to *train a very big model, stop early, and prune the model*.

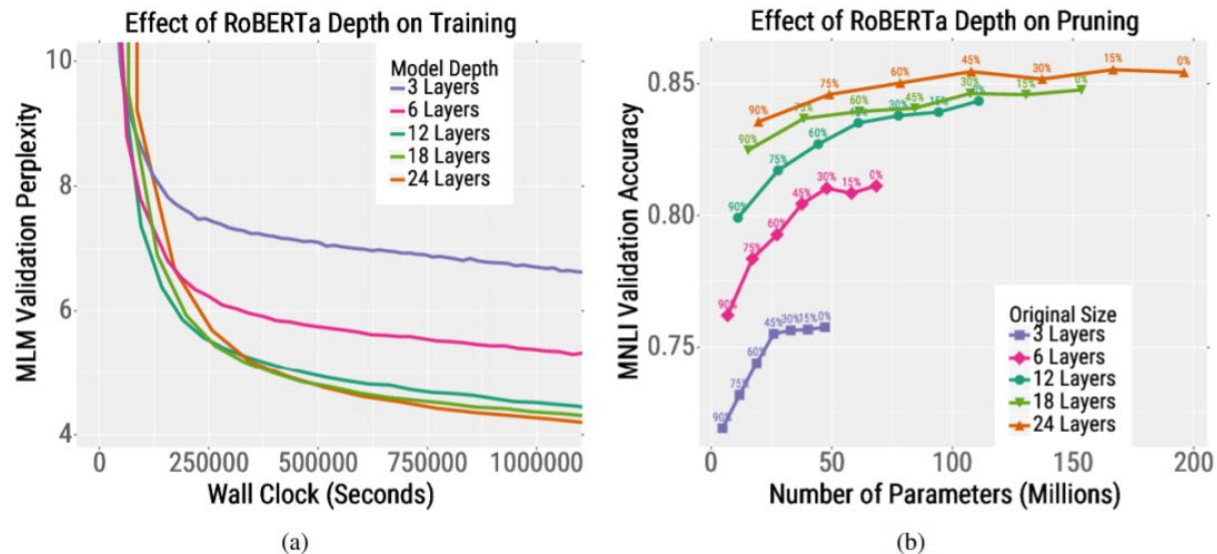
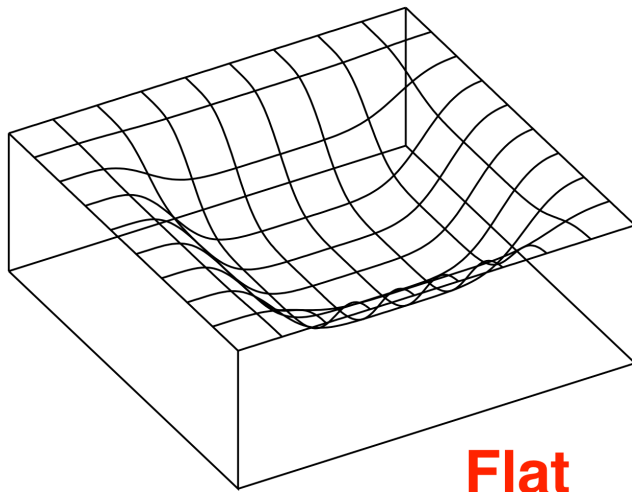


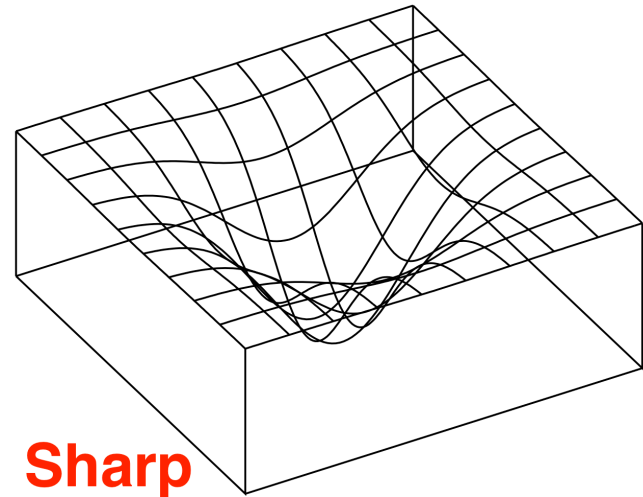
Figure 2. Increasing Transformer model size results in lower validation error as a function of *wall-clock time* and better test-time accuracy for a given *inference budget*. (a) demonstrates the training speedup for RoBERTa models of different sizes on the masked language modeling pretraining task. In (b), we take RoBERTa checkpoints that have been pretrained for the *same* amount of wall-clock time and finetune them on a downstream dataset (MNLI). We then iteratively prune model weights to zero and find that the best models for a given test-time memory budget are ones which are trained large and then heavily compressed.

Flat vs sharp minima

The practical observation: “flat” minima seem to generalize better.
([Hochreiter and Schmidhuber 1995](#), [Keskar et al 2016](#), [Hinton and Camp 1993](#))



Flat



Sharp

Original intuition ([Hinton and Camp 1993](#)): if the minimum occupies “volume” V , the number of distinct “minima regions” is at most $\text{total_volume}/V$. Hence, the “cardinality” of the neural nets corresponding to flat minima is on the order of $\text{total_volume}/V$.

Flat vs sharp minima

The practical observation: “flat” minima seem to generalize better.
([Hochreiter and Schmidhuber 1995](#), [Keskar et al 2016](#), [Hinton and Camp 1993](#))

Possible algorithmic bias as well: SGD might be biased towards flat minima: a sharp minimum is hard to land into because of the noise.

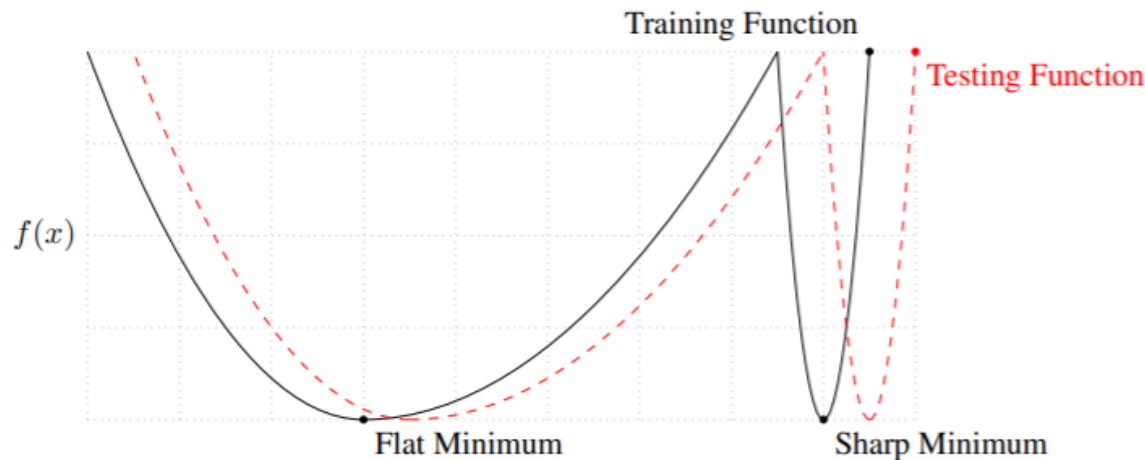


Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

Figure from Keskar et al '16

Flat vs sharp minima

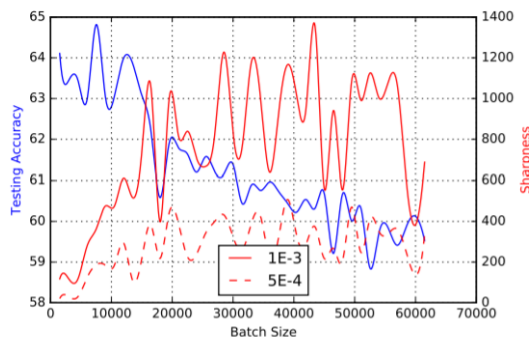
How to measure flatness?

- **Volume** of nearby points where loss is approx. the same? (How to check??)
- **Curvature** of loss (i.e. magnitude of eigenvalues of Hessian)? (Also expensive to evaluate)
- **Stability** of loss wrt to random perturbations?

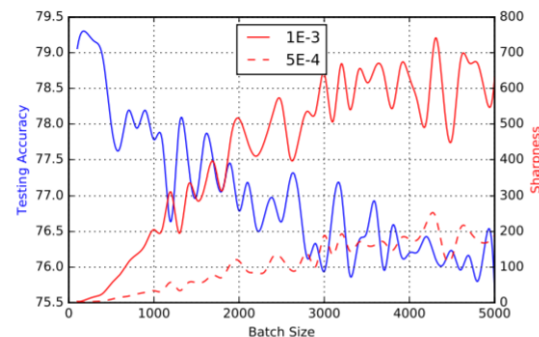
Some positive results: expts indicate that flatness may correlate with generalization. (Precisely, they argue SGD generalizes better than GD because it gets to flatter minima)

Metric 2.1. Given $x \in \mathbb{R}^n$, $\epsilon > 0$ and $A \in \mathbb{R}^{n \times p}$, we define the (C_ϵ, A) -sharpness of f at x as:

$$\phi_{x,f}(\epsilon, A) := \frac{(\max_{y \in C_\epsilon} f(x + Ay)) - f(x)}{1 + f(x)} \times 100. \quad (4)$$



(a) F_2



(b) C_1

Flat vs sharp minima

How to measure flatness?

- **Volume** of nearby points where loss is approx. the same? (How to check??)
- **Curvature** of loss (i.e. magnitude of eigenvalues of Hessian)? (Also expensive to evaluate)
- **Stability** of loss wrt to random perturbations?

Some negative results: Goyal et al '20 manage to train to SOTA accuracy w/ batches of $\sim 8k$ on Imagenet.

Moreover, they claim the issue for large batches is optimization difficulties, rather than generalization difficulties:

With an inappropriate learning schedule, even large-batch training error doesn't match small-batch training error.

With an appropriate learning schedule, both train and test error are comparable for large and small-batch methods.

Flat vs sharp minima

Main idea in Dinh et al '17: ReLU network f with 0 bias satisfies $f(\alpha x) = \alpha f(x)$, e.g.

$$f(\alpha x) = w^T \sigma(W \alpha x) = \alpha w^T \sigma(W x) = \alpha f(x)$$

Hence, $f_{\alpha w, \frac{1}{\alpha} W} = f_{w, W}$. (i.e. we can rescale the two layers to get the same function).

Taking derivatives twice, you can show (L denotes the loss):

$$(\nabla^2 L)(\alpha w, \alpha^{-1} W) = \begin{bmatrix} \alpha^{-1} I_{n_1} & 0 \\ 0 & \alpha I_{n_2} \end{bmatrix} (\nabla^2 L)(w, W) \begin{bmatrix} \alpha^{-1} I_{n_1} & 0 \\ 0 & \alpha I_{n_2} \end{bmatrix}.$$

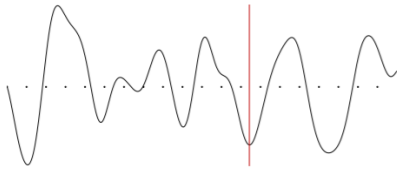
The RHS matrix is similar to $\begin{bmatrix} \alpha^{-2} I_{n_1} & 0 \\ 0 & \alpha^2 I_{n_2} \end{bmatrix} (\nabla^2 L)(w, W)$.

Since $(\nabla^2 L)(w, W) \neq 0$ taking either $\alpha \rightarrow 0$ or $\alpha \rightarrow \infty$, we can make the Frobenius (and hence spectral norm) as large as we want.

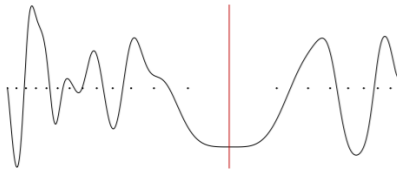
Flat vs sharp minima

How to measure flatness?

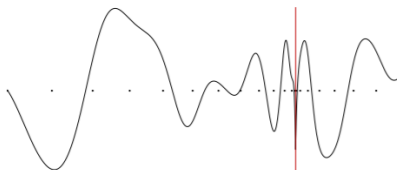
- **Volume** of nearby points where loss is approx. the same? (How to check??)
- **Curvature** of loss (i.e. magnitude of eigenvalues of Hessian)?
- **Stability** of loss wrt to random perturbations?



(a) Loss function with default parametrization



(b) Loss function with reparametrization



(c) Loss function with another reparametrization

Some negative results: multiple parameter settings might represent the same function. (Recall, neural nets have lots of invariances.)

For all these metrics, easy to choose different parameters, s.t. function is the same, but the “flatness” changes immensely. (Dinh et al ‘17)

There’ve been some attempts to come up with “reparametrization-invariant” measures, but gradient descent *is* sensitive to normal Euclidean metric. (Liang et al ‘17)