

# Deep Learning Generalization

---



# Measure of Generalization

---

**Generalization:** difference in performance on train vs. test.

$$\frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim \mathcal{D}}[\ell(f(x), y)]$$

Assumption  $(x_i, y_i) \text{ i.i.d. } \sim \mathcal{D}$

# Problems with the theoretical idealization

---

Data is not identically distributed:

- Images (Imagenet) are scraped in slightly different ways
- Data has systematic bias (e.g., patients are tested based on symptoms they exhibit)
- Data is result of interaction (reinforcement learning)
- Domain / distribution shift

# Meta Theorem of Generalization

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

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim D} [\ell(f(x), y)] \right| = O \left( \sqrt{\frac{\text{Complexity}(\mathcal{F}) + \log(1/\delta)}{n}} \right)$$

**Some measures of complexity:**

- (Log) number of elements
- VC (Vapnik-Chervonenkis) dimension
- Rademacher complexity
- PAC-Bayes
- ...

# Classical view of generalization

**Decoupled** view of generalization and optimization:

- Optimization: find a global minimum:  $\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^m \ell(f(x_i), y_i)$
- Generalization: how well does the global optimizer generalize

**Practical implications:** to have a good generalization, make sure  $\mathcal{F}$  is not too “complex”.

Strategies:

- **Direct capacity control:** bound the size of the network / amount of connections, clip the weights, etc.
- **Regularization:** add a penalty term for “complex” predictors: weight decay ( $\ell_2$  norm), dropout, etc.

# Techniques for Improving Generalization

---

W

# Weight Decay

---

**L2 regularization:**  $\frac{\lambda}{2} \|\theta\|_2^2$

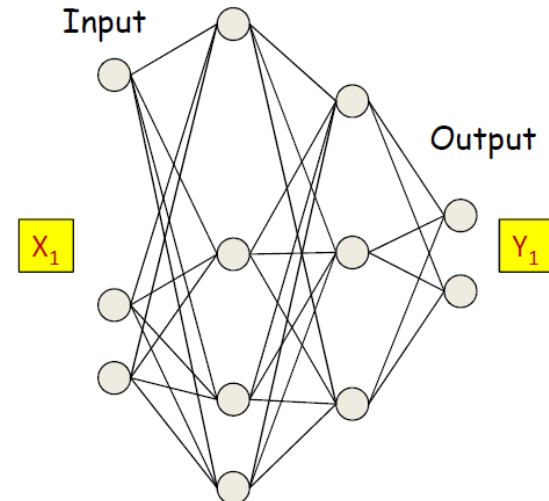
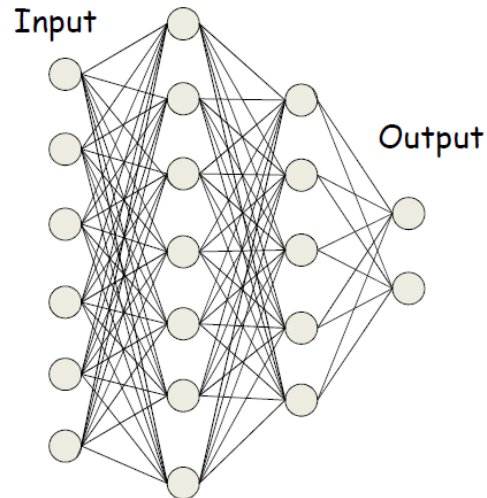
**Implementation:**  $\theta \leftarrow (1 - \eta\lambda)\theta - \eta \nabla f(\theta)$

# Dropout

**Intuition:** randomly cut off some connections and neurons.

**Training:** for each input, at each iteration, randomly “turn off” each neuron with a probability  $1 - \alpha$

- Change a neuron to 0 by sampling a Bernoulli variable.
- Gradient only propagated from non-zero neurons.



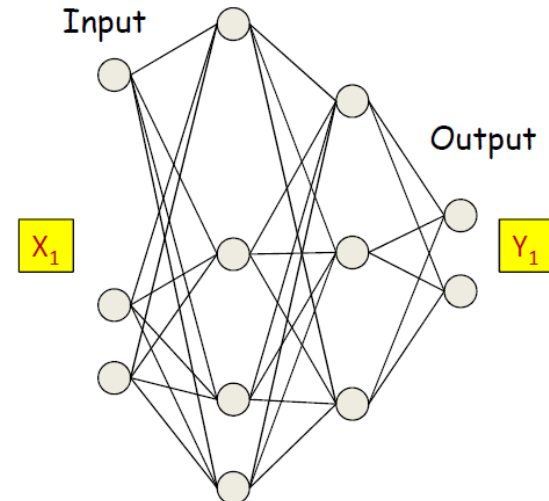
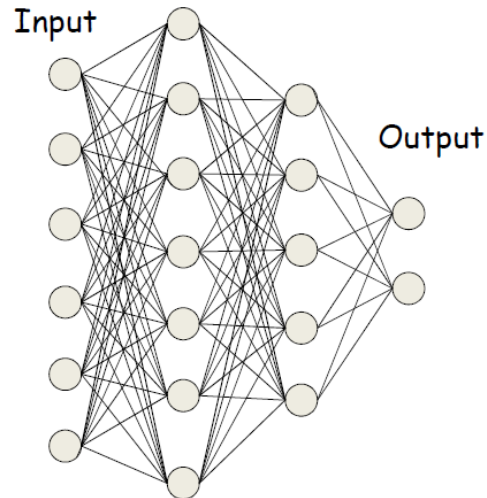


# Dropout

Dropout changes the scale of the output neuron:

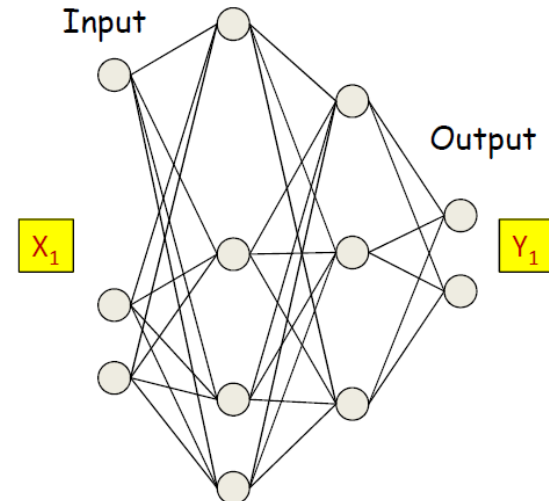
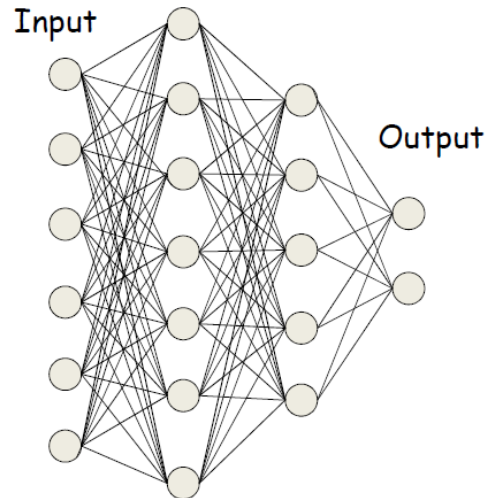
- $y = \text{Dropout}(\sigma(WX))$
- $\mathbb{E}[y] = \alpha \mathbb{E}[\sigma(Wx)]$

**Test time:**  $y = \alpha \sigma(Wx)$  to match the scale



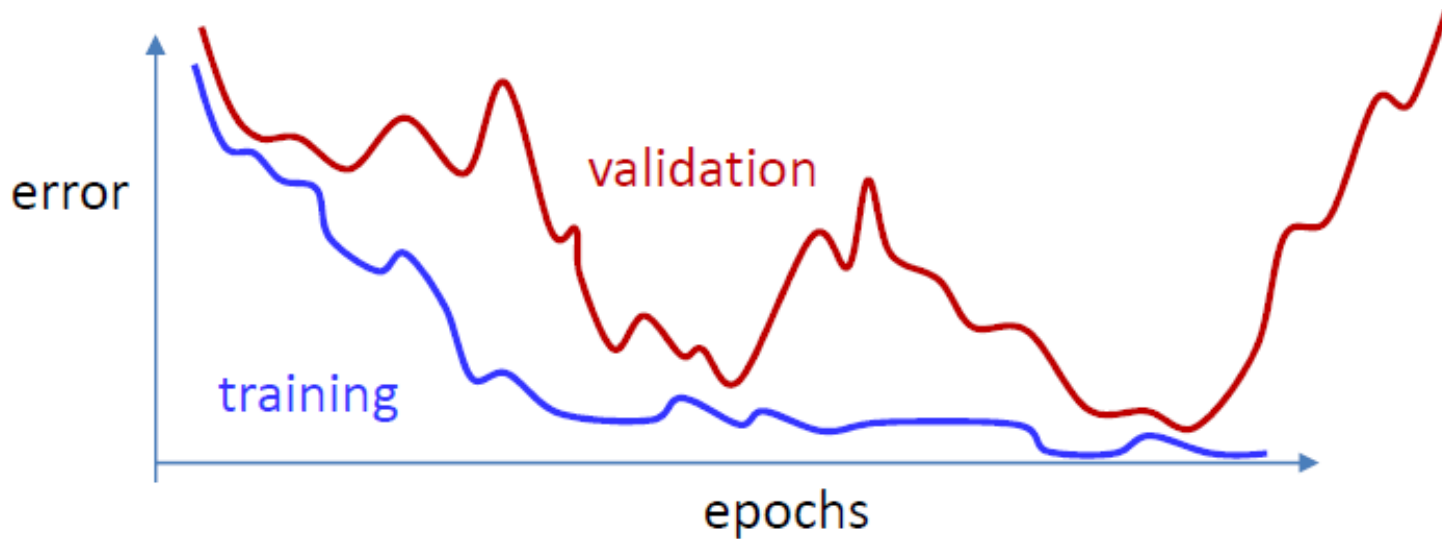
# Understanding Dropout

- Dropout forces the neural network to learn redundant patterns.
- Dropout can be viewed as an implicit L2 regularizer (Wager, Wang, Liang '13).



# Early Stopping

- Continue training may lead to overfitting.
- Track performance on a held-out validation set.
- Theory: for linear models, equivalent to L2 regularization.



# Data Augmentation

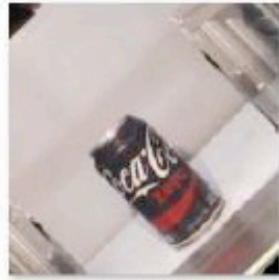
---

Depend on data types.

Computer vision: rotation, stretching, flipping, etc



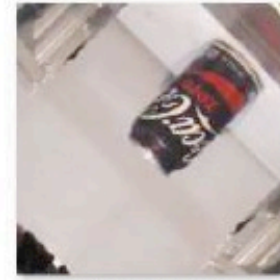
CocaColaZero1\_1.png



CocaColaZero1\_2.png



CocaColaZero1\_3.png



CocaColaZero1\_4.png



CocaColaZero1\_5.png



CocaColaZero1\_6.png



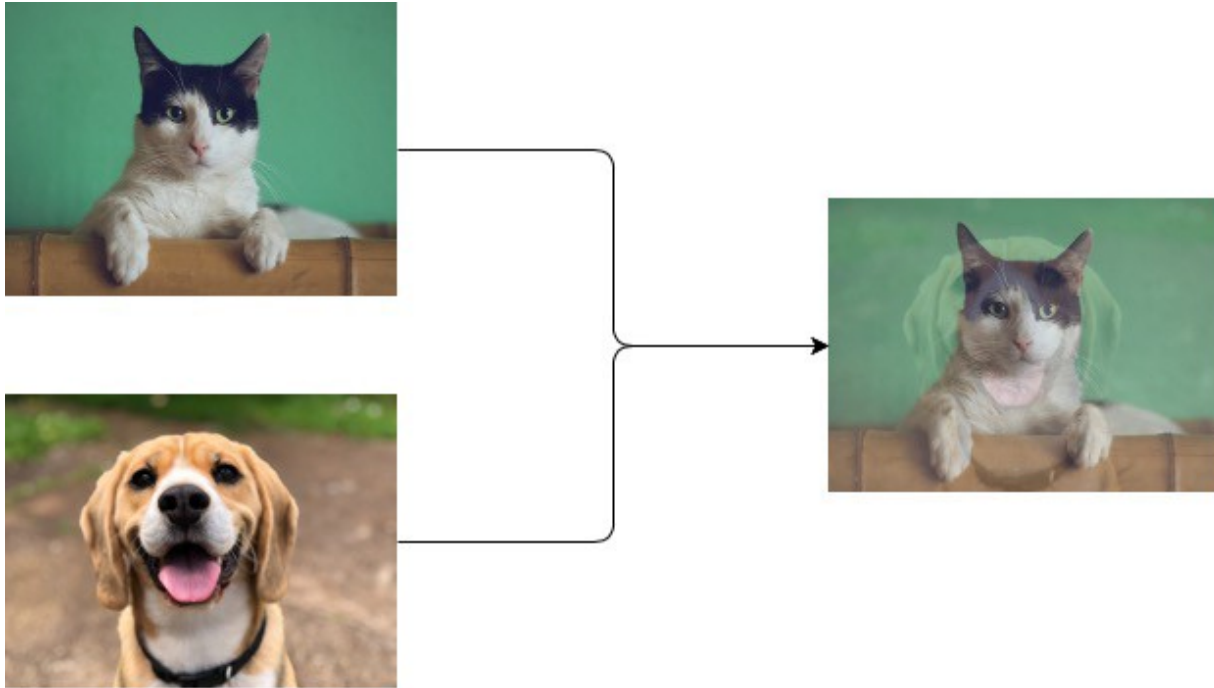
CocaColaZero1\_7.png



CocaColaZero1\_8.png

# Mixup data augmentation

- $\hat{x} = \lambda x_i + (1 - \lambda)x_j$
- $\hat{y} = \lambda y_i + (1 - \lambda)y_j$
- $\lambda \sim \mathbf{Beta}(0.2)$



# Data Augmentation

---

**Depend on data types.**

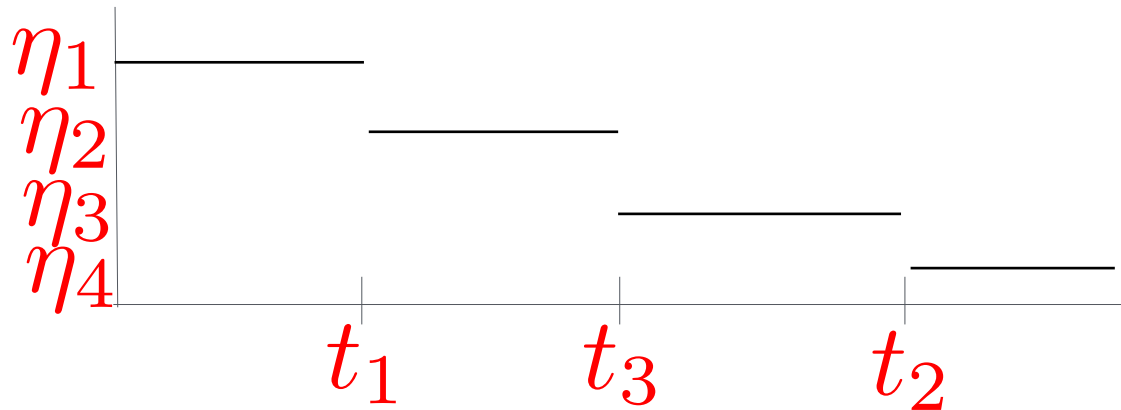
Natural language processing:

- Synonym replacement
  - *This **article** will focus on summarizing data augmentation in NLP.*
  - *This **write-up** will focus on summarizing data augmentation in NLP.*
- Back translation: translate the text data to some language and then translate back
  - *I have no time. -> 我没有时间. -> I do not have time.*

# Learning rate scheduling

Start with large learning rate. After some epochs, use small learning rate.

Learning rate schedule

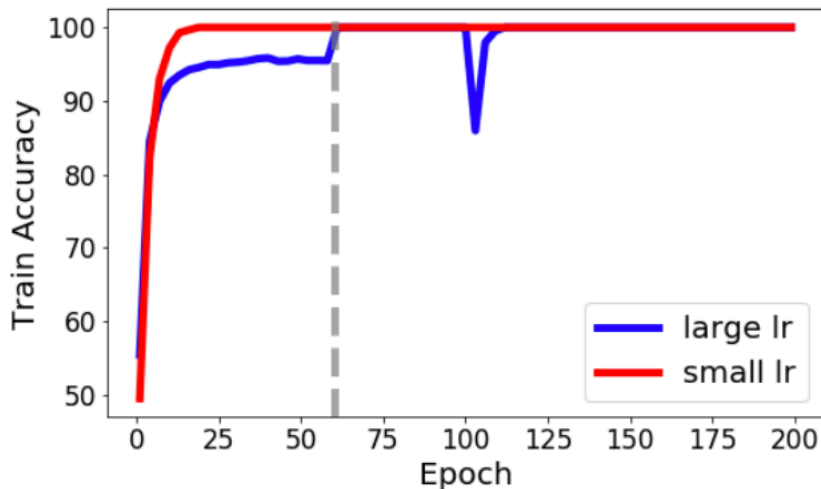


# Learning rate scheduling

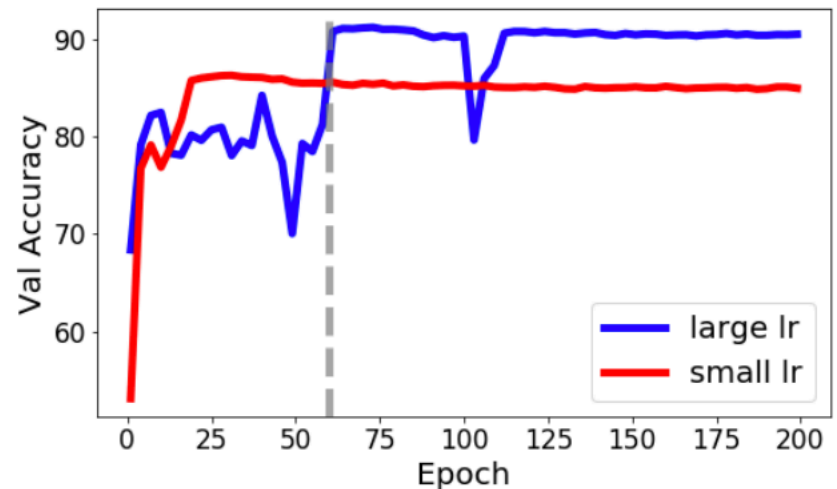
Start with large learning rate. After some epochs, use small learning rate.

Theory:

- Linear model / Kernel: large learning rate first learns eigenvectors with large eigenvalues (Nakkiran, '20).
- Representation learning (Li et al., '19)



Train



Validation



# Normalizations

---

- Batch normalization (Ioffe & Szegedy, '15)
- Layer normalization (Ba, Kiros, Hinton, '16)
- Weight normalization (Salimans, Kingma, '16)
- Instant normalization (Ulyanov, Vedaldi, Lempitsky, '16)
- Group normalization (Wu & He, '18)
- ...

# Generalization Theory for Deep Learning

---



# Basic version: finite hypothesis class

**Finite hypothesis class:** with probability  $1 - \delta$  over the choice of a training set of size  $n$ , for a bounded loss  $\ell$ , we have

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim D} [\ell(f(x), y)] \right| = O \left( \sqrt{\frac{\log |\mathcal{F}| + \log 1/\delta}{n}} \right)$$

# VC-Dimension

**Motivation:** Do we 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 training set should generalize similarly).

Let  $\mathcal{F} = \{f : \mathbb{R}^d \rightarrow \{+1, -1\}\}$  be a class of binary classifiers.

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

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

The **VC dimension** of  $\mathcal{F}$  is defined as:

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

# VC-dimension Generalization bound

**Theorem (Vapnik-Chervonenkis):** with probability  $1 - \delta$  over the choice of a training set, for a bounded loss  $\ell$ , we have

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim D} [\ell(f(x), y)] \right| = O \left( \sqrt{\frac{\text{VCdim}(\mathcal{F}) \log n + \log 1/\delta}{n}} \right)$$

Examples:

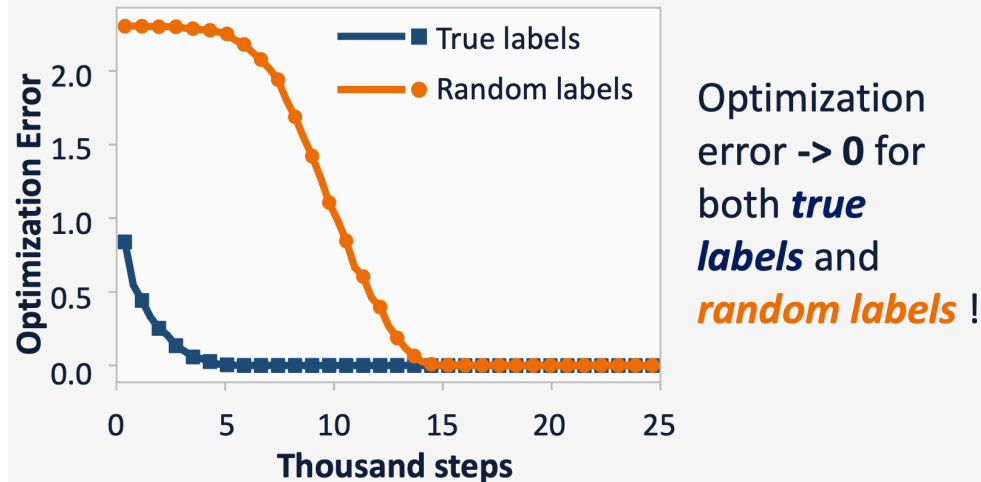
- Linear functions: VC-dim =  $O(\text{dimension})$
- Neural network: VC-dimension of fully-connected net with width  $W$  and  $H$  layers is  $\Theta(WH)$  (Bartlett et al., '17).

# Problems with VC-dimension bound

1. In over-parameterized regime, bound  $\gg 1$ .
2. Cannot explain the random noise phenomenon:
  - Neural networks that fit random labels and that fit true labels have the same VC-dimension.

Practice: gradient descent

$$\theta(t+1) \leftarrow \theta(t) - \eta \frac{\partial L(\theta(t))}{\partial \theta(t)}$$



Zhang Bengio Hardt Recht Vinyals 2017

Understanding DL Requires Rethinking Generalization

# PAC Bayesian Generalization Bounds

**Setup:** Let  $P$  be a prior over function in class  $\mathcal{F}$ , let  $Q$  be the posterior (after algorithm's training).

**Theorem:** with probability  $1 - \delta$  over the choice of a training set, for a bounded loss  $\ell$ , we have

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim D} [\ell(f(x), y)] \right| = O \left( \sqrt{\frac{KL(Q || P) + \log 1/\delta}{n}} \right)$$

# Rademacher Complexity

**Intuition:** how well can a classifier class **fit random noise?**

(Empirical) **Rademacher complexity:** For a training set  $S = \{x_1, x_2, \dots, x_n\}$ , and a class  $\mathcal{F}$ , denote:

$$\hat{R}_n(S) = \mathbb{E}_\sigma \sup_{f \in \mathcal{F}} \sum_{i=1}^n \sigma_i f(x_i) .$$

where  $\sigma_i \sim \text{Unif}\{+1, -1\}$  (Rademacher R.V. ).

(Population) **Rademacher complexity:**

$$R_n = \mathbb{E}_S \left[ \hat{R}_n(S) \right] .$$



# Rademacher Complexity Generalization Bound

**Theorem:** with probability  $1 - \delta$  over the choice of a training set, for a bounded loss  $\ell$ , we have

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim D} [\ell(f(x), y)] \right| = O \left( \frac{\hat{R}_n}{n} + \frac{\log 1/\delta}{n} \right)$$

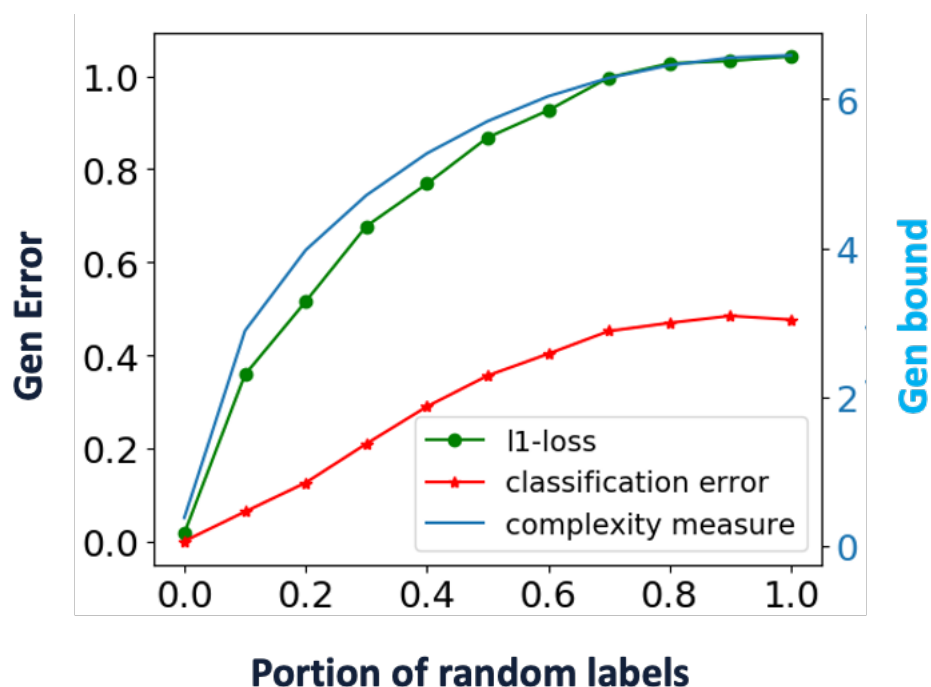
and

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i) - \mathbb{E}_{(x,y) \sim D} [\ell(f(x), y)] \right| = O \left( \frac{R_n}{n} + \frac{\log 1/\delta}{n} \right)$$

# Kernel generalization bound

Use Rademacher complexity theory, we can obtain a

generalization bound  $O(\sqrt{y^\top (H^*)^{-1} y/n})$  where  $y \in \mathbb{R}^n$  are  $n$  labels, and  $H^* \in \mathbb{R}^{n \times n}$  is the kernel (e.g., NTK) matrix.



# Norm-based Rademacher complexity bound

---

**Theorem:** If the activation function  $\sigma$  is  $\rho$ -Lipschitz. Let  $\mathcal{F} = \{x \mapsto W_{H+1}\sigma(W_h\sigma(\dots\sigma(W_1x)\dots)), \|W_h^T\|_{1,\infty} \leq B \forall h \in [H]\}$  then  $R_n(\mathcal{S}) \leq \|X^T\|_{2,\infty} (2\rho B)^{H+1} \sqrt{2 \ln d}$  where  $X = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$  is the input data matrix.

# Comments on generalization bounds

- When plugged in real values, the bounds are rarely non-trivial (i.e., smaller than 1)
- “*Fantastic Generalization Measures and Where to Find them*” by Jiang et al. '19 : large-scale investigation of the correlation of extant generalization measures with true generalization.

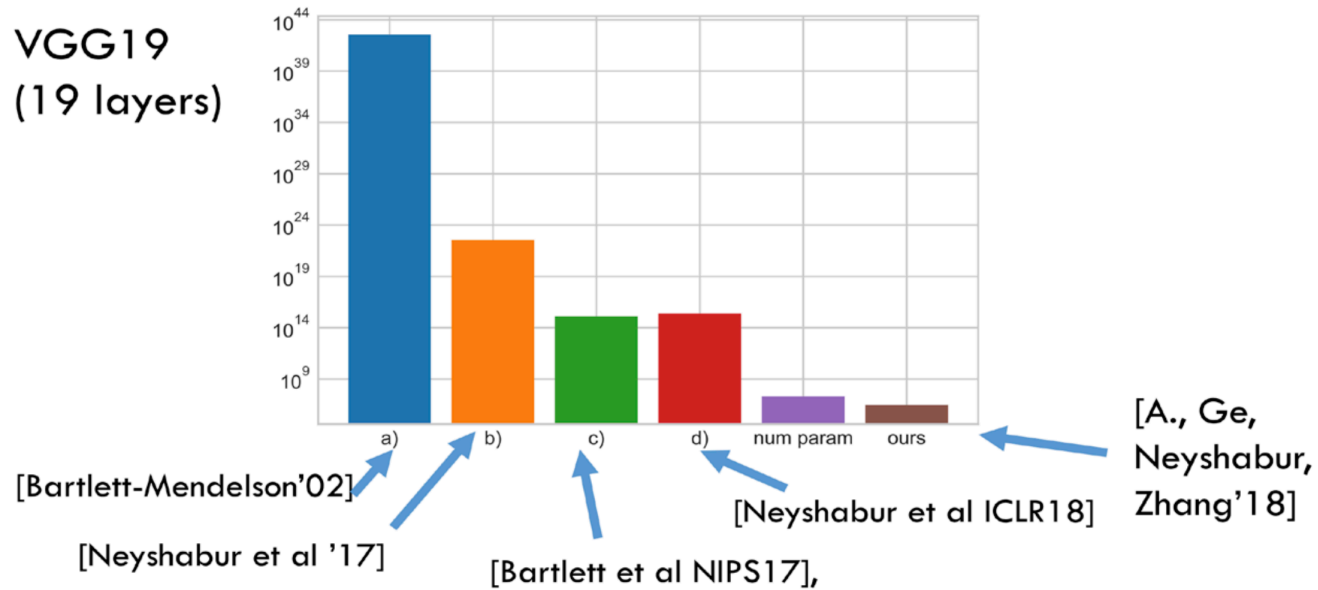


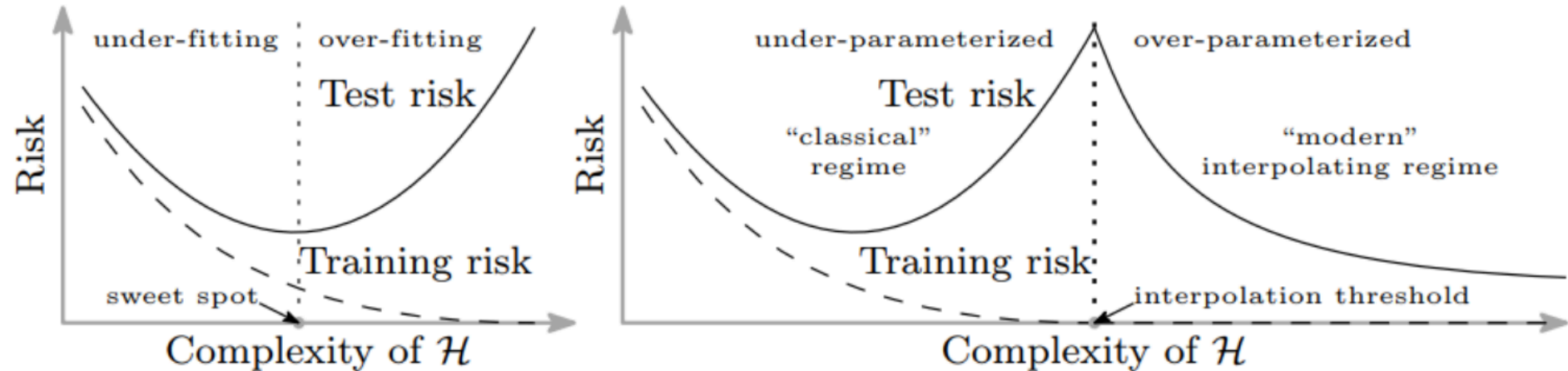
Image credits to Andrej Risteski

# Comments on generalization bounds

---

- Uniform convergence may be unable to explain generalization of deep learning [Nagarajan and Kolter, '19]
  - Uniform convergence: a bound for all  $f \in \mathcal{F}$
  - Exists example that 1) can generalize, 2) uniform convergence fails.
  
- Rates:
  - Most bounds:  $1/\sqrt{n}$ .
  - Local Rademacher complexity:  $1/n$ .

# Double descent



(a) U-shaped “bias-variance” risk curve

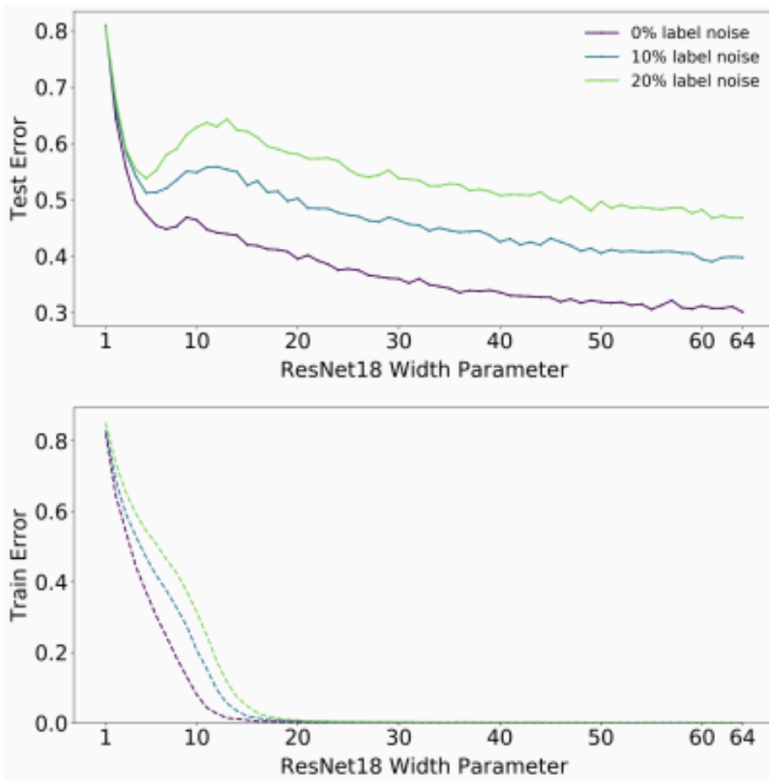
(b) “double descent” risk curve

Belkin, Hsu, Ma, Mandal '18

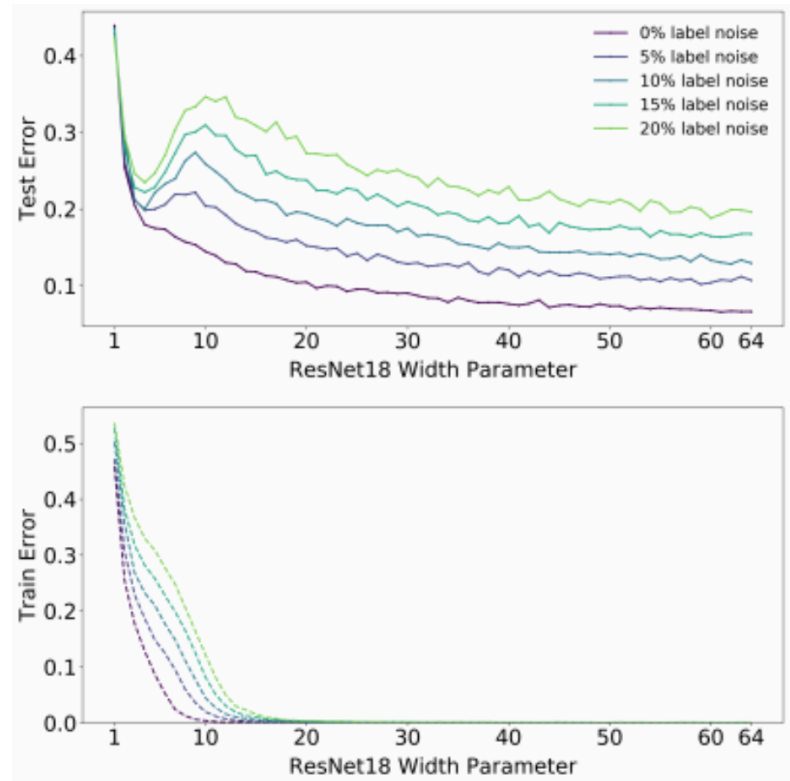
- There are cases where the model gets bigger, yet the (test!) loss goes down, sometimes even lower than in the classical “under-parameterized” regime.
- Complexity: number of parameters.

# Double descent

Widespread phenomenon, across architectures (Nakkiran et al. '19):



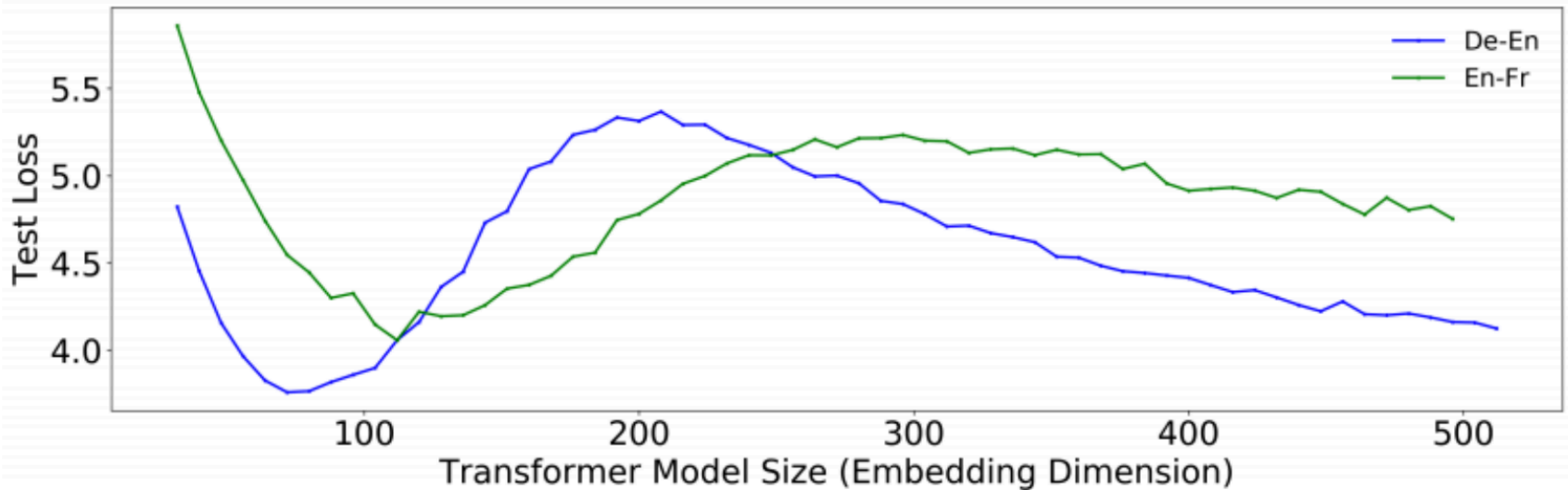
(a) **CIFAR-100.** There is a peak in test error even with no label noise.



(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.

# Double descent

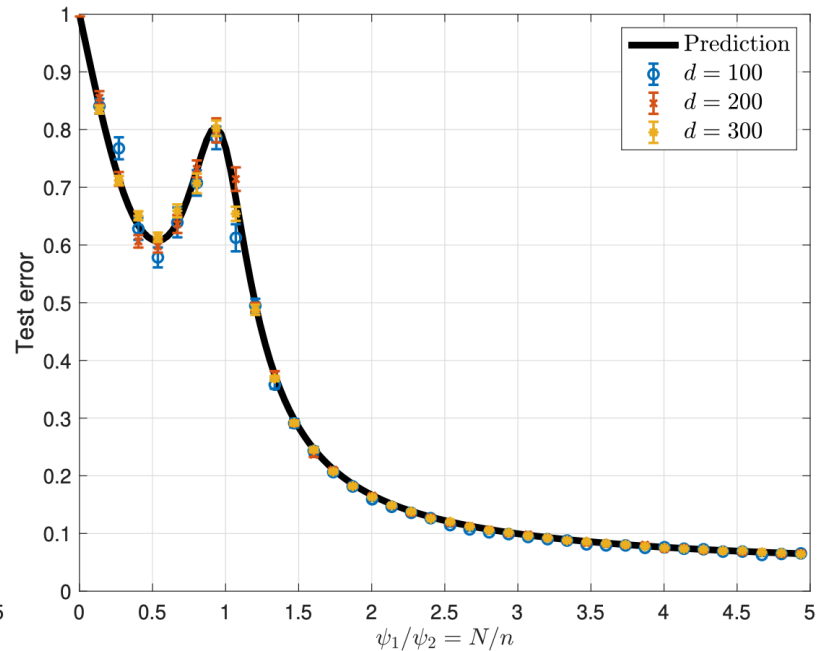
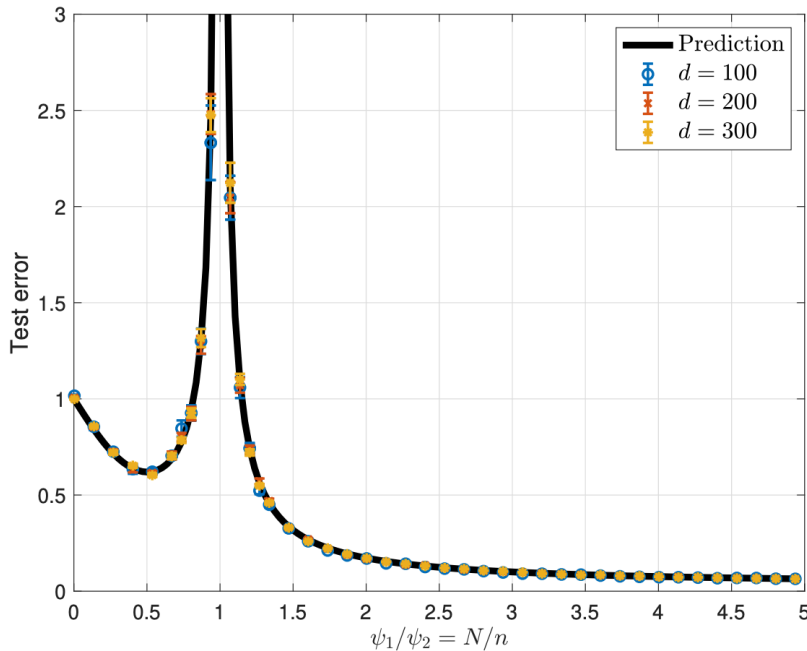
Widespread phenomenon, across architectures (Nakkiran et al. '19):





# Double descent

Widespread phenomenon, also in kernels (can be formally proved in some concrete settings [Mei and Montanari '20]), random forests, etc.



# Double descent

Also in other quantities such as train time, dataset, etc (Nakkiran et al. '19):

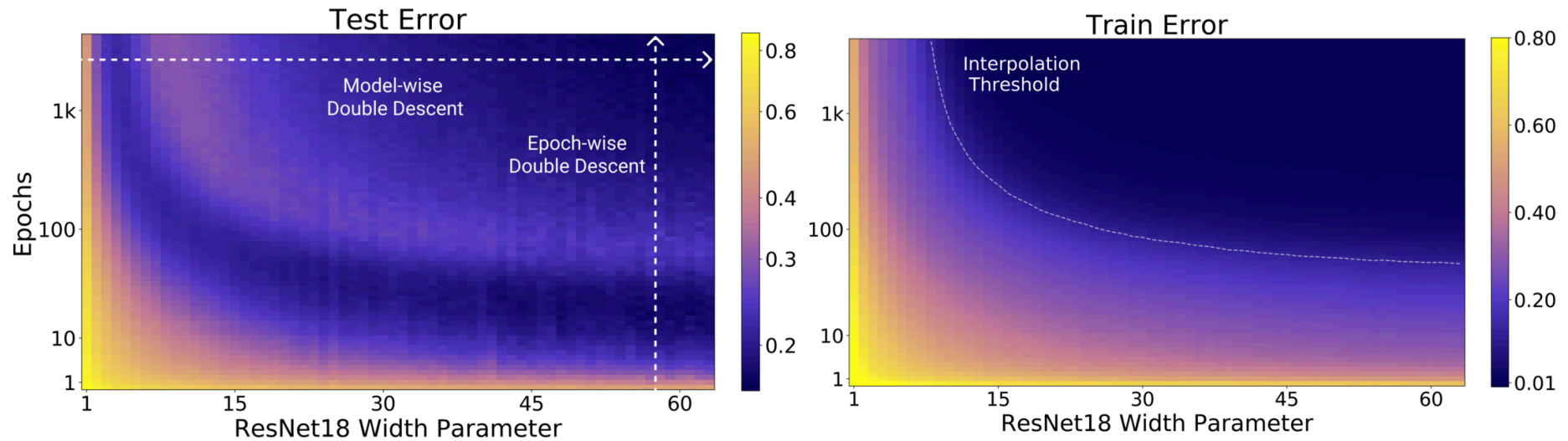
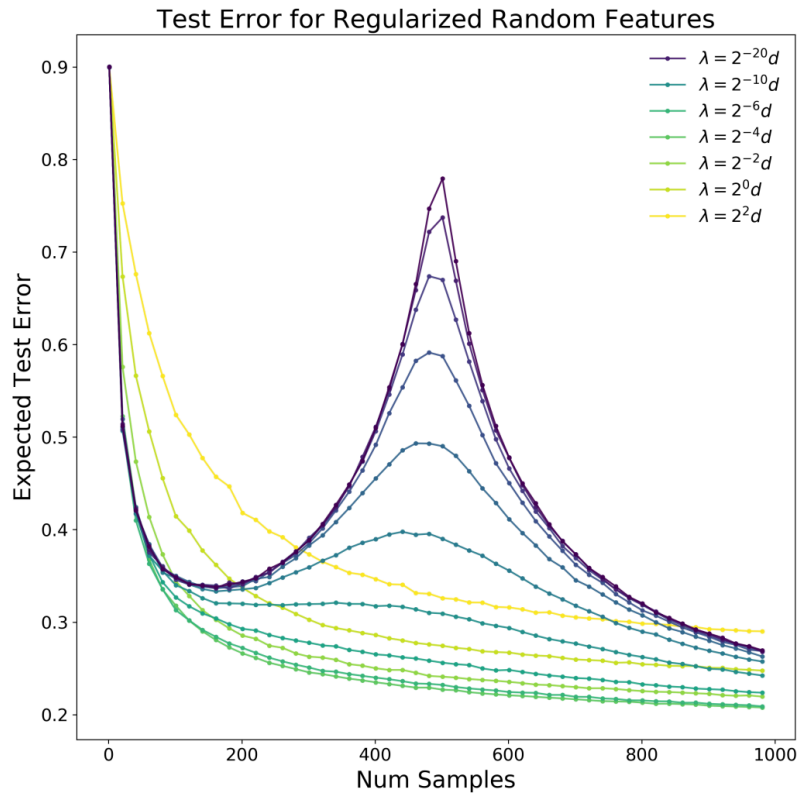


Figure 2: **Left:** Test error as a function of model size and train epochs. The horizontal line corresponds to model-wise double descent—varying model size while training for as long as possible. The vertical line corresponds to epoch-wise double descent, with test error undergoing double-descent as train time increases. **Right** Train error of the corresponding models. All models are Resnet18s trained on CIFAR-10 with 15% label noise, data-augmentation, and Adam for up to 4K epochs.

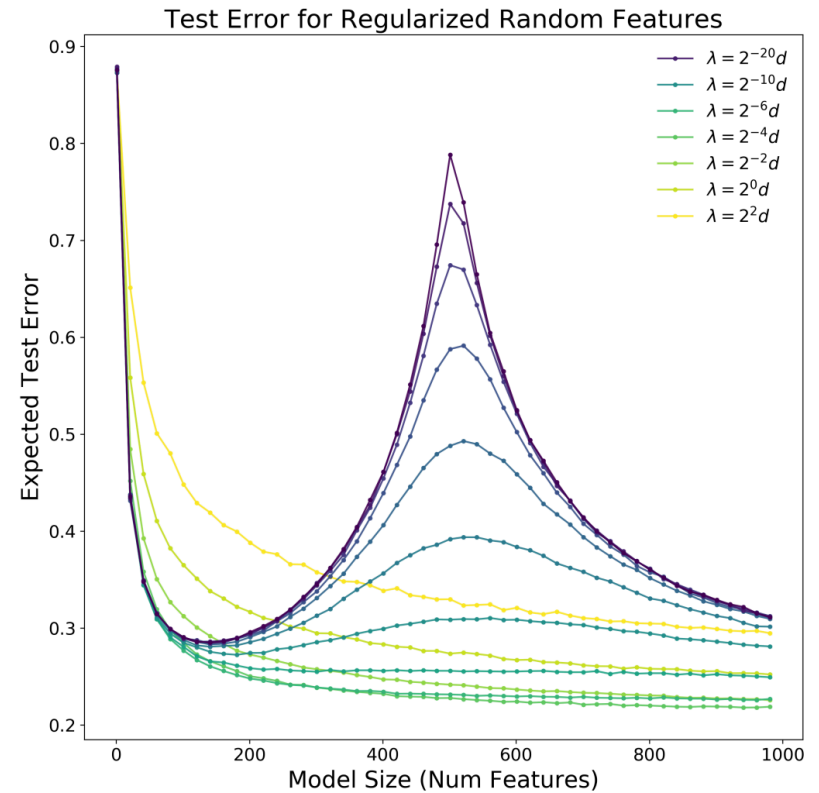


# Double descent

Optimal regularization can mitigate double descent [Nakkiran et al. '21]:



a) Test Classification Error vs. Number of Training Samples.



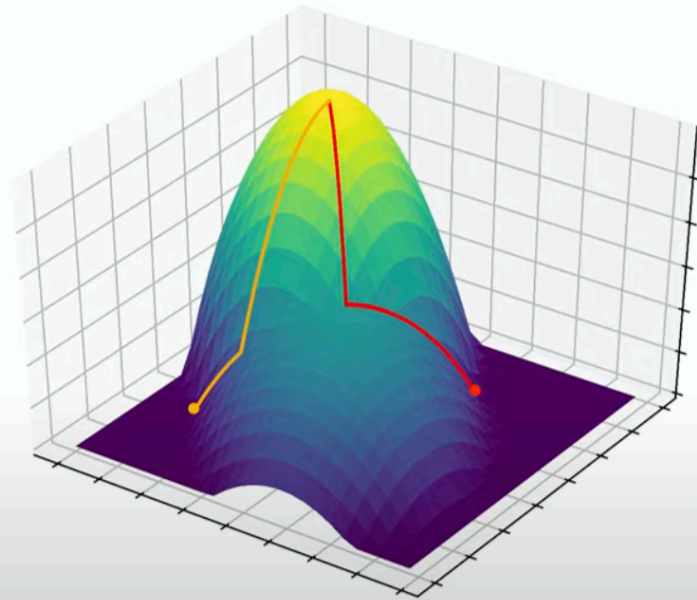
(b) Test Classification Error vs. Model Size (Number of Random Features).

# Implicit Regularization

---

Different optimization algorithm

- Different bias in optimum reached
  - Different Inductive bias
    - Different generalization properties



# Implicit Bias

---

Margin:

- Linear predictors:
  - Gradient descent, mirror descent, natural gradient descent, steepest descent, etc maximize margins with respect to different norms.
- Non-linear:
  - Gradient descent maximizes margin for homogeneous neural networks.
  - Low-rank matrix sensing: gradient descent finds a low-rank solution.

# Separation between NN and kernel

---

- For approximation and optimization, neural network has no advantage over kernel. Why NN gives better performance: **generalization**.
- [Allen-Zhu and Li '20] Construct a class of functions  $\mathcal{F}$  such that  $y = f(x)$  for some  $f \in \mathcal{F}$ :
  - no kernel is sample-efficient;
  - Exists a neural network that is sample-efficient.