Approximation Theory

UNIVERSITY of WASHINGTON

Multivariate Approximation

Theorem: Let g be a continuous function that satisfies $\|x - x'\|_\infty \leq \delta \Rightarrow |g(x) - g(x')| \leq \epsilon$ (Lipschitzness).
Then there exists a 3-layer ReLU neural network with $O(\frac{1}{\delta^d})$ nodes that satisfy

$$\int_{[0,1]^d} |f(x) - g(x)|\, dx = \|f - g\|_1 \leq \epsilon$$

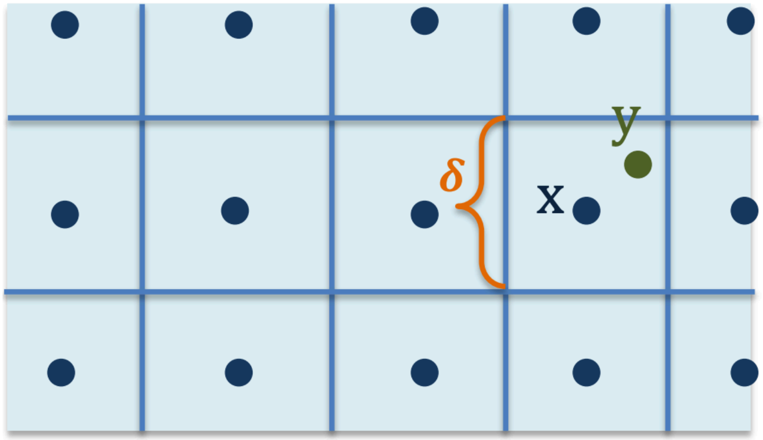
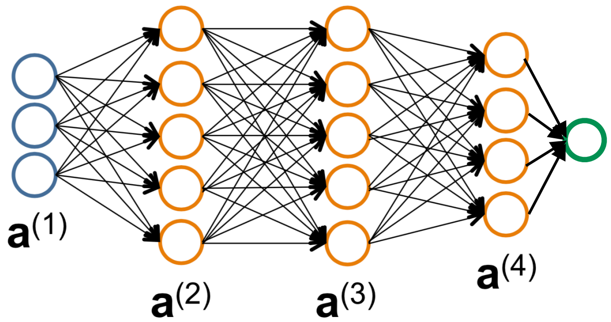


Figure credit to Andrej Risteski



Barron's Theory

- Can we avoid the curse of dimensionality for "nice" functions?
- What are nice functions?
  - Fast decay of the Fourier coefficients

- Fourier basis functions:
  $\{e_w(x) = e^{i\langle w,x\rangle} = \cos(\langle w, x\rangle) + i\sin(\langle w, x\rangle) \mid w \in \mathbb{R}^d\}$

- Fourier coefficient: $\hat{f}(w) = \int_{\mathbb{R}^d} f(x)e^{-i\langle w,x\rangle}dx$
- Fourier integral / representation: $f(x) = \int_{\mathbb{R}^d} \hat{f}(w)e^{i\langle w,x\rangle}dw$

Barron's Theorem

Definition: The Barron constant of a function f is:

$$C \triangleq \int_{\mathbb{R}^d} \|w\|_2 |\hat{f}(w)| \, dw.$$

Theorem (Barron '93): For any $g : \mathbb{B}_1 \to \mathbb{R}$ where $\mathbb{B}_1 = \{x \in \mathbb{R} : \|x\|_2 \leq 1\}$ is the unit ball, there exists a 3-layer neural network f with $O(\frac{C^2}{\epsilon})$ neurons and sigmoid activation function such that

$$\int_{\mathbb{B}_1} (f(x) - g(x))^2 dx \leq \epsilon.$$

Examples

- Gaussian function: $f(x) = (2\pi\sigma^2)^{d/2} \exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right)$

- Other functions:
  - Polynomials
  - Function with bounded derivatives

Proof Ideas for Barron's Theorem

Step 1: show any continuous function can be written as an infinite neural network with cosine-like activation functions.
(Tool: Fourier representation.)

Step 2: Show that a function with small Barron constant can be approximated by a convex combination of a small number of cosine-like activation functions.
(Tool: subsampling / probabilistic method.)

Step 3: Show that the cosine function can be approximated by sigmoid functions.
(Tool: classical approximation theory.)

Simple Infinite Neural Nets

Definition: An infinite-wide neural network is defined by a signed measure ν over neuron weights (w, b)

$$f(x) = \int_{w \in \mathbb{R}^d, b \in \mathbb{R}} \sigma(w^\top x + b) d\nu(w, b).$$

Theorem: Suppose $g : \mathbb{R} \to \mathbb{R}$ is differentiable, if $x \in [0,1]$, then $g(x) = \int_0^1 \mathbf{1}\{x \geq b\} \cdot g'(b) db + g(0)$

Step 1: Infinite Neural Nets

The function can be written as

$$f(x) = f(0) + \int_{\mathbb{R}^d} |\hat{f}(w)| (\cos(b_w + \langle w, x \rangle) - \cos(b_w)) dw.$$

Step 2: Subsampling

Writing the function as the expectation of a random variable:

$$f(x) = f(0) + \int_{\mathbb{R}^d} \frac{|\hat{f}(w)|\|w\|_2}{C} \left(\frac{C}{\|w\|_2} (\cos(b_w + \langle w, x \rangle) - \cos(b_w)) \right) dw.$$

Sample one $w \in \mathbb{R}^d$ with probability $\frac{|\hat{f}(w)|\|w\|_2}{C}$ for r times.

Step 3: Approximating the Cosines

Lemma: Given $g_w(x) = \frac{C}{\|w\|_2}(\cos(b_w + \langle w, x \rangle) - \cos(b_w))$, there exists a 2-layer neural network f_0 of size $O(1/\epsilon)$ with sigmoid activations, such that $\sup_{x \in [-1,1]} |f_0(y) - h_w(y)| \leq \epsilon$.

Depth Separation

So far we only talk about 2-layer or 3-layer neural networks.

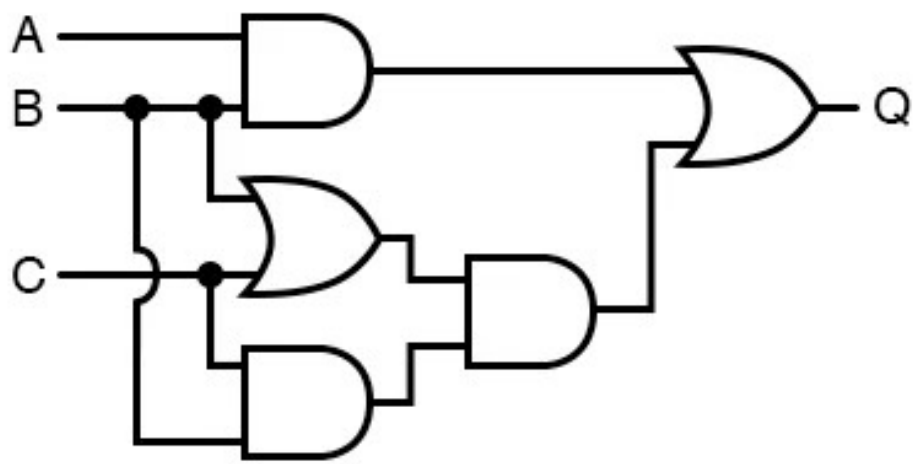
Why we need **Deep** learning?

Can we show deep neural networks are **strictly** better than shallow neural networks?

A brief history of depth separation

Early results from theoretical computer science

Boolean circuits: a directed acyclic graph model for computation over binary inputs; each node ("gate") performs an operation (e.g. OR, AND, NOT) on the inputs from its predecessors.



A brief history of depth separation

Early results from theoretical computer science

Boolean circuits: a directed acyclic graph model for computation over binary inputs; each node (“gate”) performs an operation (e.g. OR, AND, NOT) on the inputs from its predecessors.

Depth separation: the difference of the computation power: shallow vs deep Boolean circuits.

Håstad (’86): parity function cannot be approximated by a small constant-depth circuit with OR and AND gates.

Modern depth-separation in neural networks

- **Related architectures / models of computation**
  - Sum-product networks [Bengio, Delalleau ’11]
- **Heuristic measures of complexity**
  - Bound of number of linear regions for ReLU networks [Montufar, Pascanu, Cho, Bengio ‘14]
- **Approximation error**
  - A small deep network cannot be approximated by a small shallow network [Telgarsky ’15]

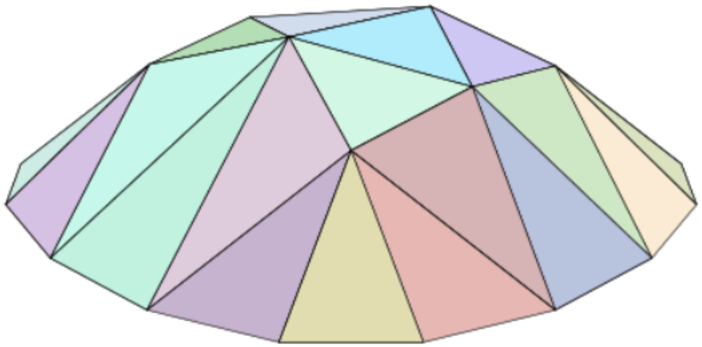
Shallow Nets Cannot Approximate Deep Nets

Theorem (Telgarsky '15): For every $L \in \mathbb{N}$, there exists a function $f : [0,1] \to [0,1]$ representable as a network of depth $O(L^2)$, with $O(L^2)$ nodes, and ReLU activation such that, for every network $g : [0,1] \to \mathbb{R}$ of depth L and $\leq 2^L$ nodes, and ReLU activation, we have

$$\int_{[0,1]} |f(x) - g(x)| \, dx \geq \frac{1}{32}.$$

Intuition

A ReLU network f is piecewise linear, we can subdivide domain into a finite number of polyhedral pieces $(P_1, P_2, \ldots, P_N)$ such that in each piece, f is linear: $\forall x \in P_i, f(x) = A_i x + b_i$.



Deeper neural networks can make exponentially more regions than shallow neural networks.

Make each region has different values, so shallow neural networks cannot approximate.

Benefits of depth for smooth functions

Theorem (Yarotsky '15): Suppose $f : [0,1]^d \to \mathbb{R}$ has all partial derivatives of order r with coordinate-wise bound in $[-1,1]$, and let $\epsilon > 0$ be given. Then there exists a $O(\ln \frac{1}{\epsilon})$ - depth and $\left(\frac{1}{\epsilon}\right)^{O(\frac{d}{r})}$ -size network so that $\sup_{x \in [0,1]^d} |f(x) - g(x)| \leq \epsilon$.

Remarks

- All results discussed are existential: they prove that a good approximator exists. Finding one efficiently (e.g., using gradient descent) is the next topic (optimization).
- The choices of non-linearity are usually very flexible: most results we saw can be re-proven using different non-linearities.
- There are other approximation error results: e.g., deep and narrow networks are universal approximators.
- Depth separation for optimization and generalization is widely open.

Comparing RNN and Transformer

AI Coding Assistants

Tools

- Cursor: VS Code + ChatGPT for coding.
- GitHub Copilot: AI pair programmer
- Amazon CodeWhisperer: AWS-integrated assistant
- Tabnine: Self-hosted AI coding assistant

Key Features

- Natural language code generation
- Context-aware completions
- Debugging
- Compilation

Theory of LLMs for Coding

Key Questions

- How to characterize the expressive power of LLMs for coding?
- Why do transformers outperform RNNs in coding?

Our Focus

The **compilation** capability of LLMs.

Test-Language: Mini-Husky

Overview

Simple yet representative C-like programming language designed to formally assess LLM's capabilities in programming language processing.

```
pub struct Person {
    pub name: String,   // Person's name as a string
    pub age: Int,       // Person's age as an integer
}

pub enum Animal {
    Dog { name: String },   // Dog variant with name field
    Cat { name: String },   // Cat variant with name field
}

pub fn f() {
    let a = 1;          // Initialize local variable
    ...                 // Additional implementation
}

fn g() { f() }         // Private function that calls f()
```

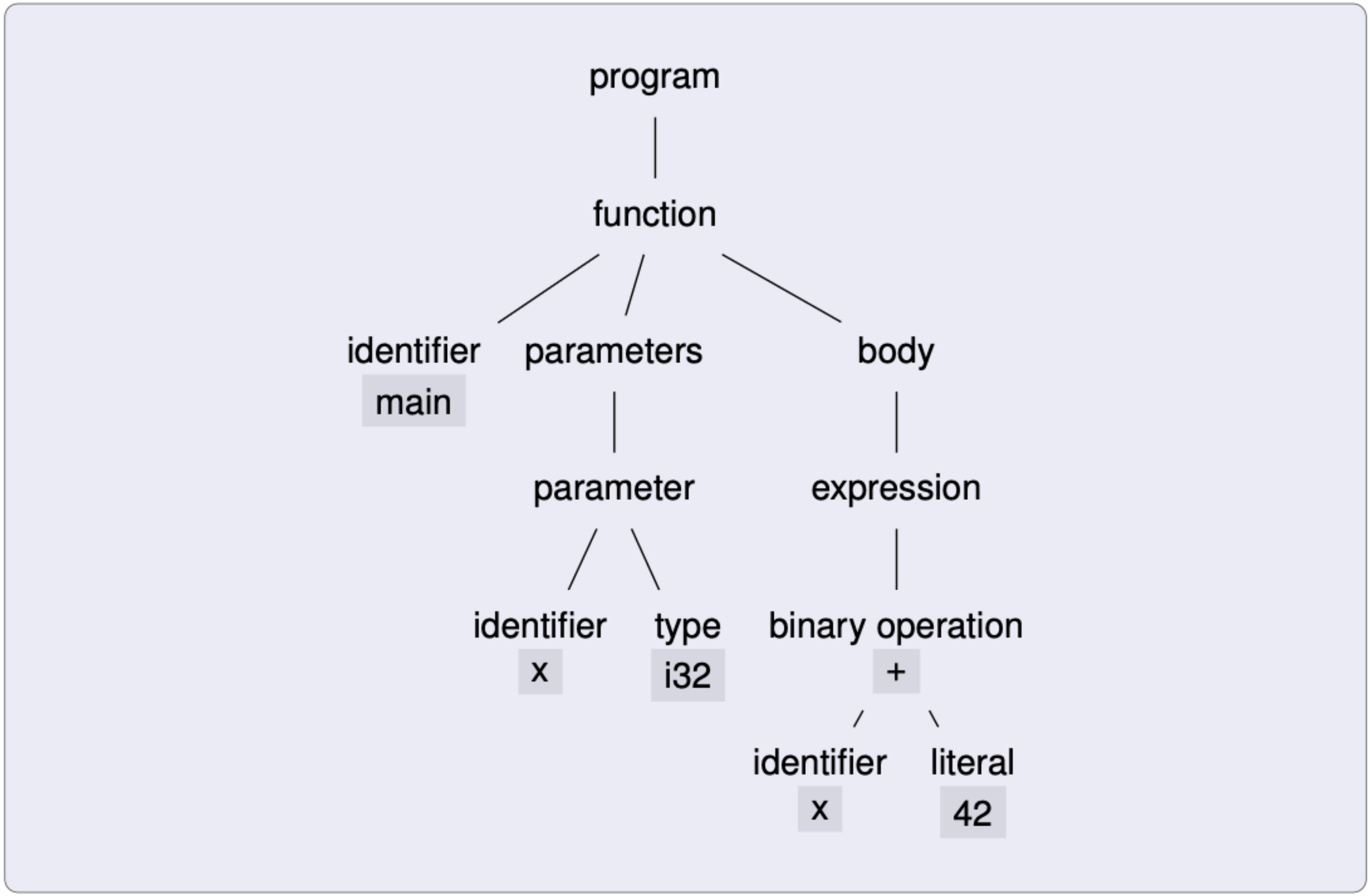
Compilation Pipeline

- Goal: Transform source code into executable code.
- Key compilation stages:
  - **Parsing** $\rightarrow$ Abstract Syntax Tree (AST) construction
    - Tokenization (lexical analysis)
    - Parse tree building (syntax analysis)
  - **Semantic Analysis** $\rightarrow$ Program Verification
    - Resolving symbols
    - Checking types
  - **Code Generation** $\rightarrow$ Executable Output
    - Intermediate representation generation
    - Code optimization
    - Machine code output

AST Example

```
pub fn main(x: i32) { x + 42 }
```

Its corresponding AST (depth = 6):



AST Example

Worse case: depth is proportional to the context length.

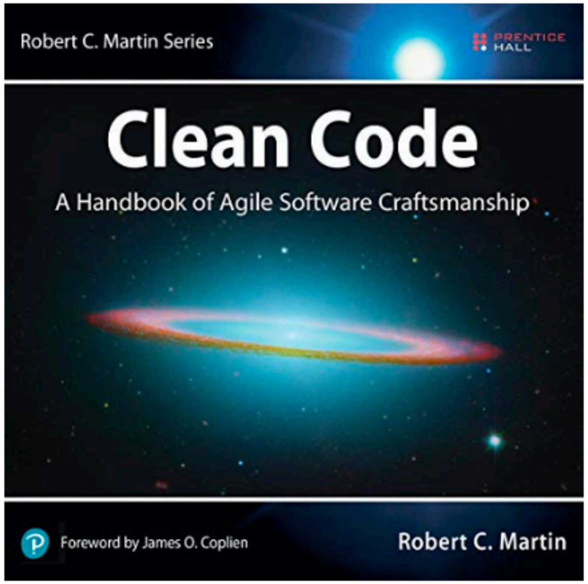
```
pub fn main() {
    f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
        f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
            f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(
                42
            ))))))))))))))))))))
        ))))))))))))))))))))
    ))))))))))))))))))))
}
```

Depth is about 123.

Why does it make sense to assume small depth?

Because human-readable code is naturally bounded.

Clean Code Principle



Principles:

- High cohesion. Do one thing.
- Low coupling.
- Small.
- Use descriptive names.
- Prefer fewer arguments.
- Have no side effects.
- ...

Clean Code Principle

One example from the Linux kernel:

```
static void perf_ctx_unlock(struct perf_cpu_context *cpuctx,
                            struct perf_event_context *ctx) {
    if (ctx)
        __perf_ctx_unlock(ctx);
    __perf_ctx_unlock(&cpuctx->ctx);
}

#define TASK_TOMBSTONE ((void *)-1L)

static bool is_kernel_event(struct perf_event *event) {
    return READ_ONCE(event->owner) == TASK_TOMBSTONE;
}

static DEFINE_PER_CPU(struct perf_cpu_context, perf_cpu_context);

struct perf_event_context *perf_cpu_task_ctx(void) {
    lockdep_assert_irqs_disabled();
    return this_cpu_ptr(&perf_cpu_context)->task_ctx;
}
```

https://github.com/torvalds/linux/blob/master/kernel/events/core.c#L201

Bounded AST Depth

Definition (Codes with Bounded Depth)

Let $\mathsf{MiniHusky}_D$ be the set of token sequences that can be parsed into valid ASTs in **Mini-Husky** with a depth less than D.

Transformer for AST Construction

Theorem (Transformer for AST Construction)

There exists a transformer of model dimension and number of layers at most $O(\log L + D)$ and number of heads at most $O(1)$ that represents a function that maps any token sequence of length L in $\mathsf{MiniHusky}_D$ to its abstract syntax tree.

- **Highlight.** Only $\log L$ dependency: transformer can process long-context sequences.
- **Proof Sketch.** Construct the AST layer by layer. Each layer of AST takes a constant number of transformer layers.

Type Checking

```
//  Type Error: the return type is `i32`, yet the last expression is of type `f32`
fn f(a: i32) -> i32 { return 1.1 }

fn g() {
    // Type Error: `x` is of type f32 but it's assigned by a value of type `i32`
    // Type Error: the first argument of `f` expects be of type `i32` but gets a float literal instead
    let x: f32 = f(1.1);

}
```

Type checking includes:

- checking arguments of function calls against expected types from function declarations
- checking return types of functions against expected types from function declarations
- checking assignment expressions against expected types from variable declarations
- ...

Transformer for Type Checking

Theorem

There exists a transformer of model dimension and number of layers being $O(\log L + D)$ and number of heads being $O(1)$ that represents a function that does type checking for any token sequence token sequence of length L in $\mathsf{MiniHusky}_D$.

- **Highlight.** Only $\log L$ dependency: transformer can process long-context sequences.
- **Proof Sketch.** Use the attention mechanism heavily to fetch type signature information and populate type inference results.

RNN for Type Checking

Theorem

For $L, D \in \mathbb{N}$, for any RNN that represents a function that does type checking for any token sequence of length L in MiniHusky_D with $D = O(1)$, then its state space size is at least $\Omega(L)$.

- **Exponential Separation.** $\Omega(L)$ for RNN vs. $O(\log L)$ for transformer.
- **Proof Sketch.** Type checking requires associative recall: Given a series of key-value pairs as a string, the model is required to recall the value given a key. RNN's memory needs to scale with L [Wen-Dang-Lyu, 2024].

Experiments

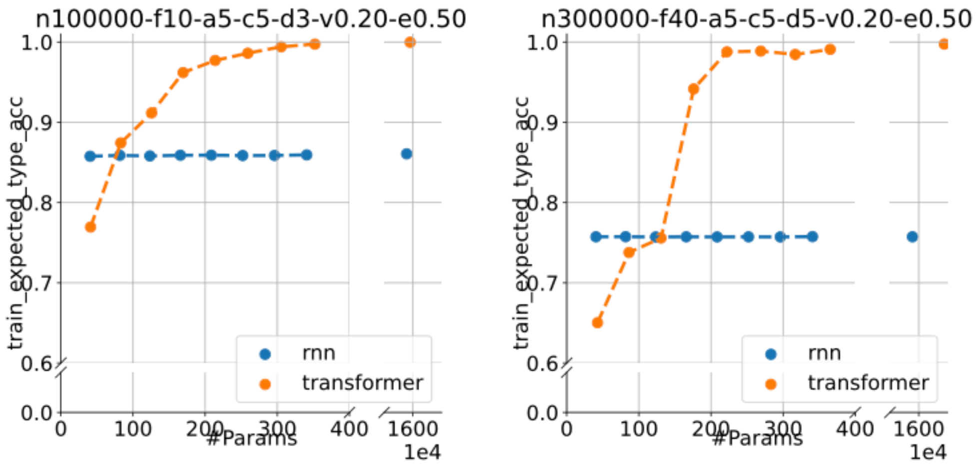


Figure: Training error of two setups with different (1) number of data, (2) number of functions, (3) minimum distance between the declaration and the first call of a function, etc.

Recent Advances in Representation Power

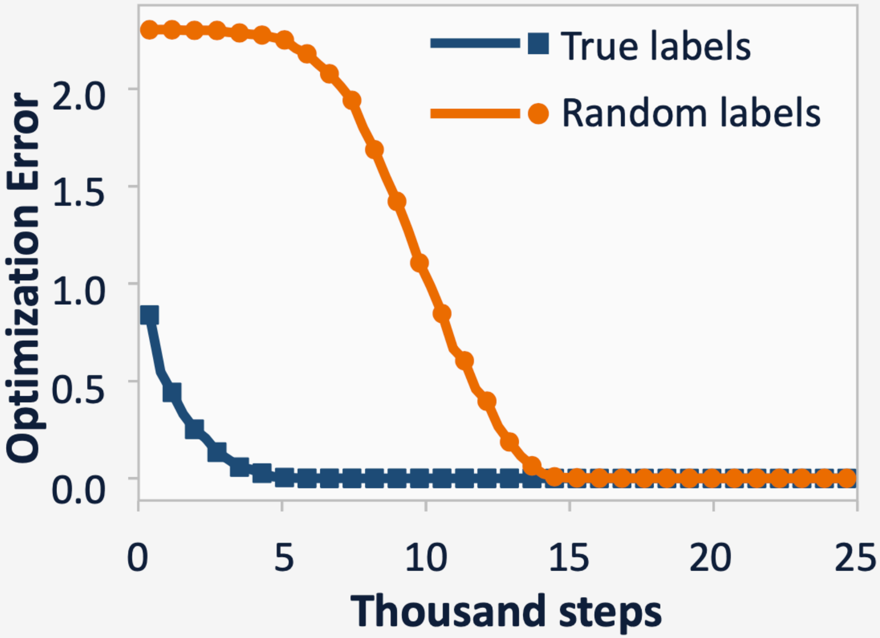
- Analyses of different architectures
  - Graph neural network
  - Attention-based neural network
- Separation between different architectures
- Finite data approximation
- In-context learning for specific tasks
- Chain-of-thought
- …

Optimization Theory of Deep Learning

Gradient descent finds global minima

Practice: gradient descent

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



Optimization error **-> 0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017

Understanding DL Requires Rethinking Generalization

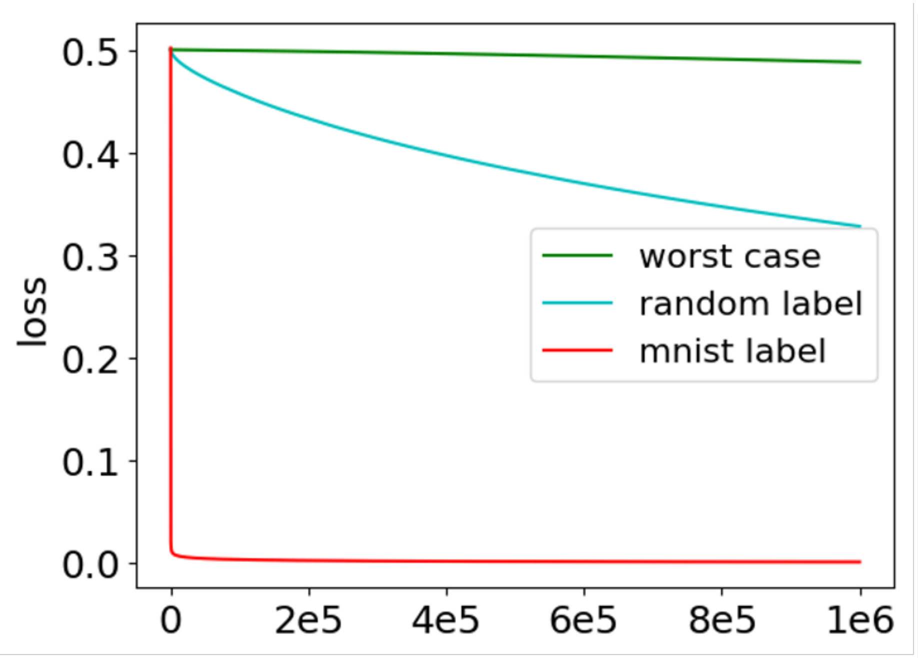
Global convergence of gradient descent

Theorem (Du et al. ’18, Allen-Zhu et al. ’18, Zou et al ’19) If the width of each layer is poly(n) where n is the number of data. Using random initialization with a particular scaling, gradient descent finds an approximate global minimum in polynomial time.

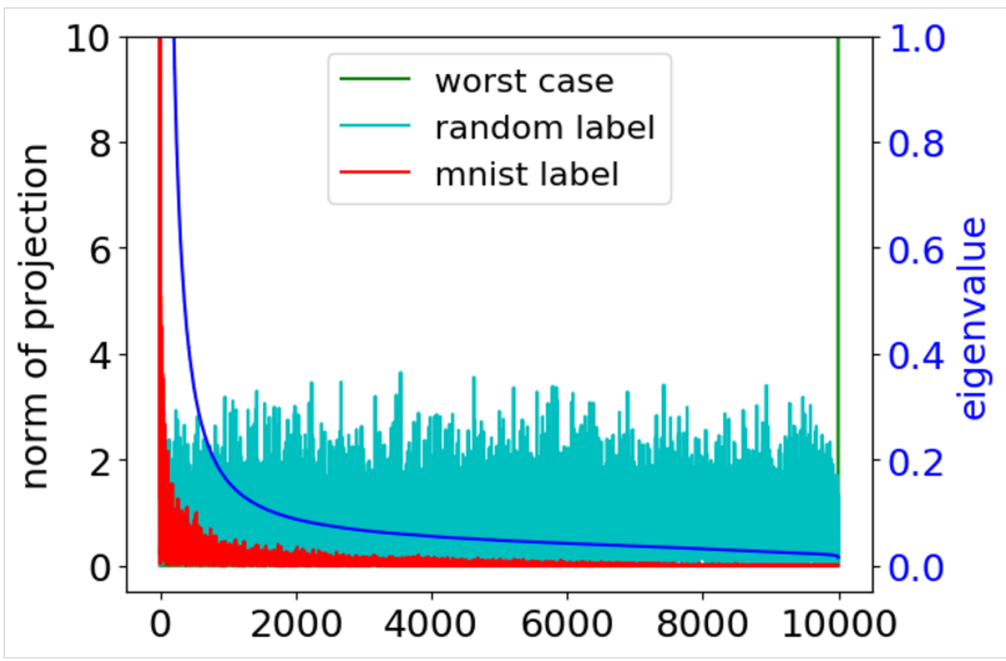
Main Proof Ideas

Main Proof Ideas

What determines the convergence rate?



Convergence Rate



Projections

Neural Tangent Kernel

Recipe for designing new kernels

$$f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x\right) \quad ➤ \quad k\left(x, x'\right)=\mathbb{E}_{\theta_{\mathrm{NN}} \sim \mathcal{W}}\left[\left\langle\frac{\partial f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x\right)}{\partial \theta_{\mathrm{NN}}}, \frac{\partial f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x'\right)}{\partial \theta_{\mathrm{NN}}}\right\rangle\right]$$

Transform a neural network of any architecture to a kernel!

Fully-connected NN → Fully-connected NTK

Convolutional NN → Convolutional NTK

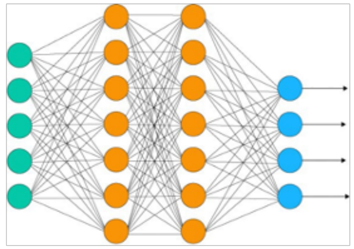
Graph NN → Graph NTK

……

Fully-Connect NTK

$$\begin{pmatrix} -0.1 \\ 0.2 \\ \cdots \\ 0.9 \end{pmatrix}$$

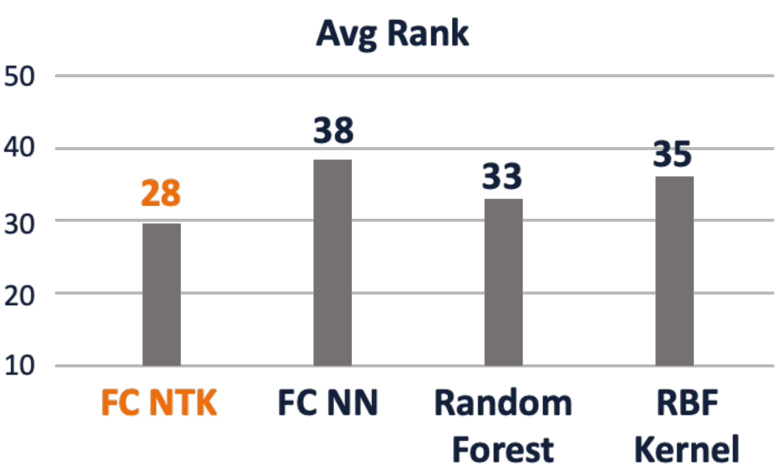
Features



FC NN

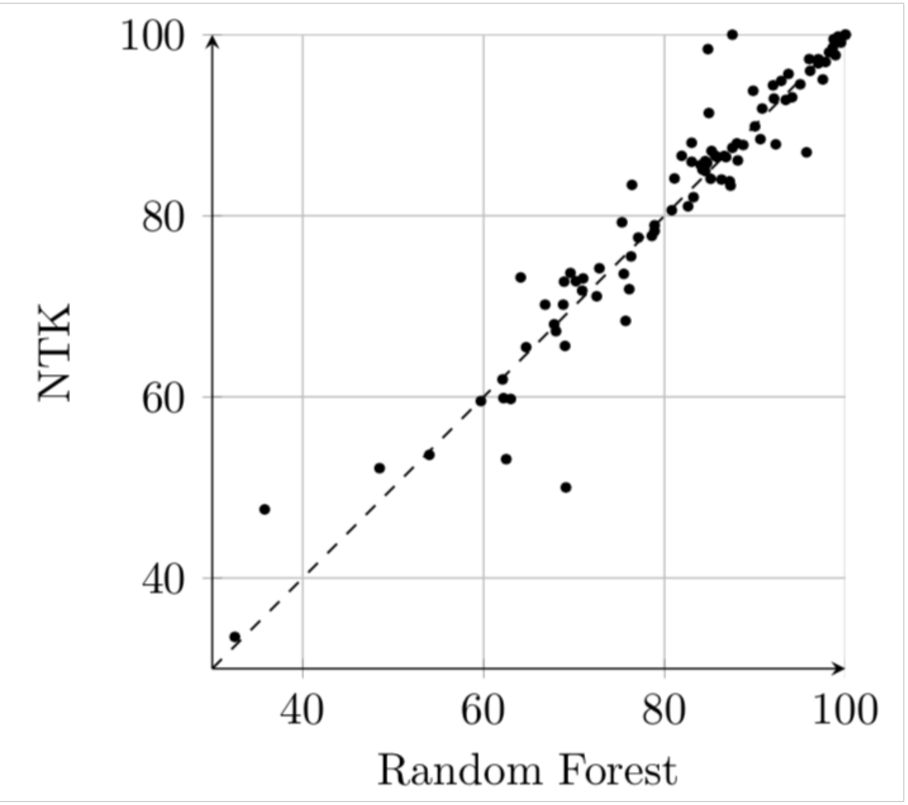
$$k\left(\begin{pmatrix} -0.1 \\ 0.2 \\ \cdots \\ 0.9 \end{pmatrix}, \begin{pmatrix} -0.3 \\ 0.5 \\ \cdots \\ -0.8 \end{pmatrix}\right)$$

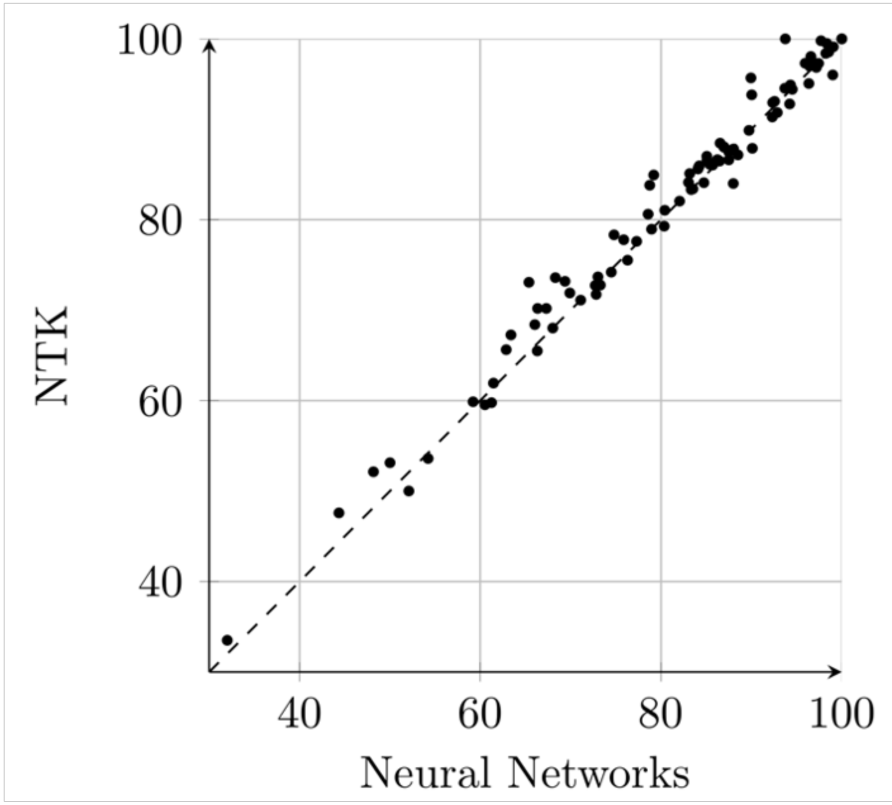
FC NTK



Classifier	Avg Acc	P95	PMA
FC NTK	82%	72%	96%
FC NN	81%	60%	95%
Random Forest	82%	68%	95%
RBF Kernel	81%	72%	94%

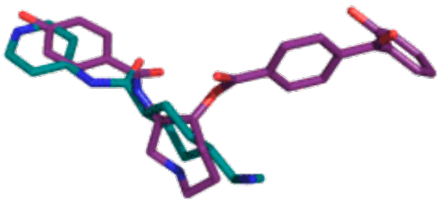
Pairwise Comparisons

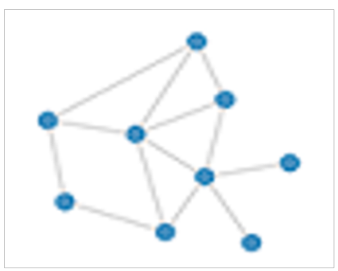




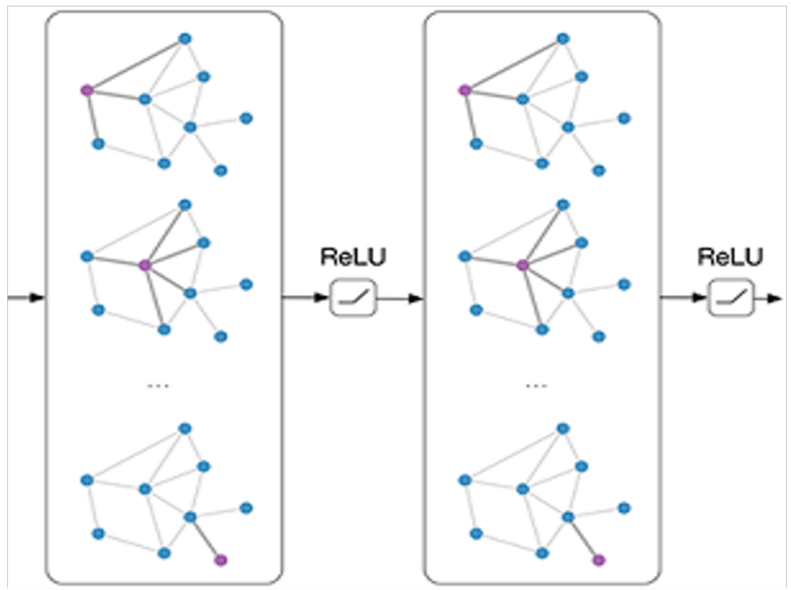
Classification
Accuracy

Graph Neural Network





Graph



Graph Neural Network

Toxicity

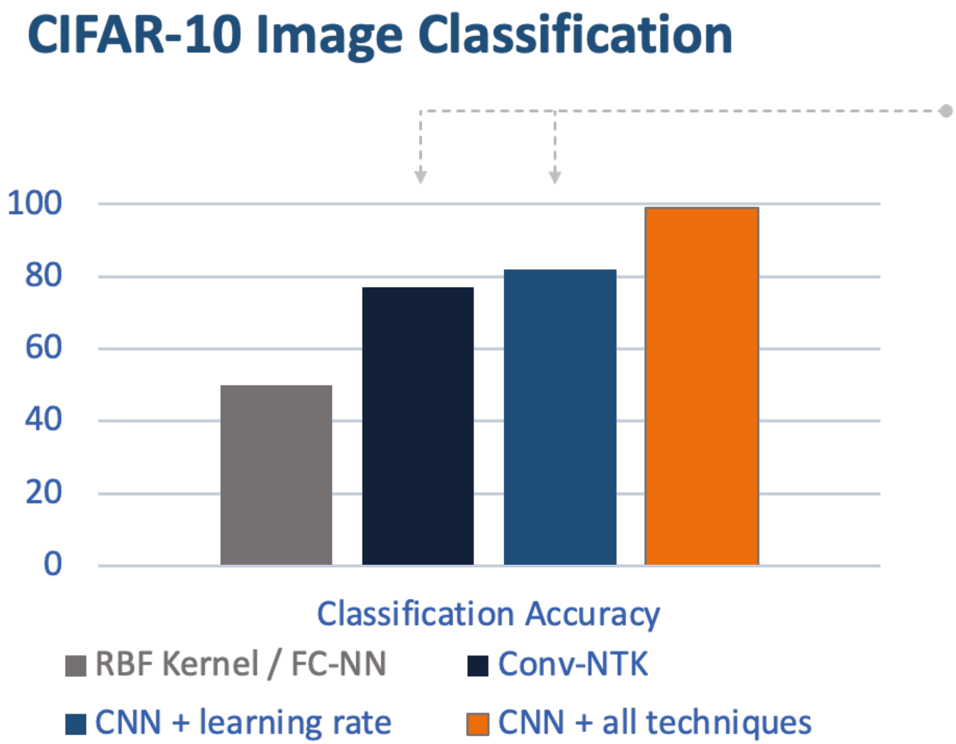
Label

Graph Neural Tangent Kernel

Graph Graph NN Graph NTK

	Method	COLLAB	IMDB-B	IMDB-M	PTC
GNN	GCN	79%	74%	51%	64%
	GIN	80%	75%	52%	65%
GK	WL	79%	74%	51%	60%
	GNTK	84%	77%	53%	68%

What are left open?



Open Problems:

Why there is a gap:
finite-width?
learning rate?

Understanding techniques:
batch-norm
dropout
data-augmentation
…

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

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

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

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

Practical implications: to have a good generalization, make sure $\mathscr{F}$ 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 (ℓ_2 norm), dropout, etc.

Techniques for Improving Generalization

Weight Decay

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

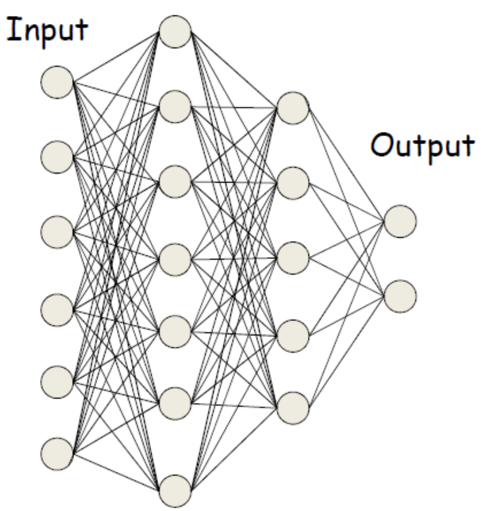
Implementation: $\theta \leftarrow (1 - \eta\lambda)\theta - \eta \nabla f(\theta)$

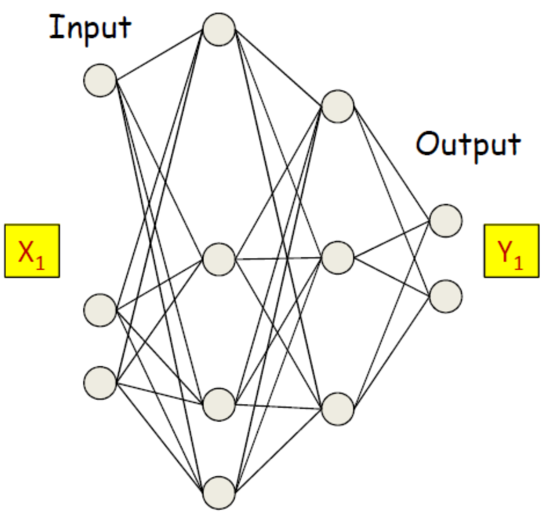
Dropout

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 propogatd from non-zero neurons.



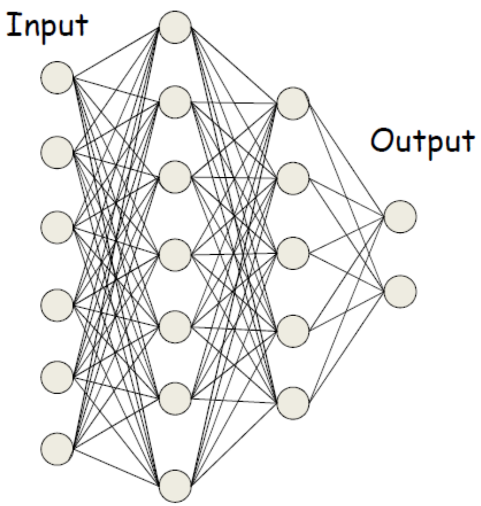


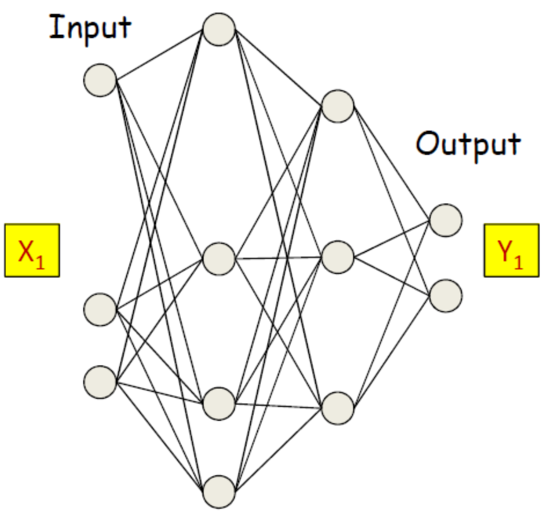
Dropout

Dropout changes the scale of the output neuron:

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

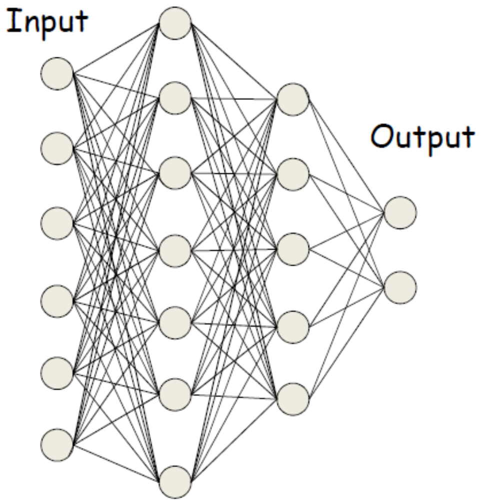
Test time: $y = \alpha\sigma(Wx)$ to match the scale

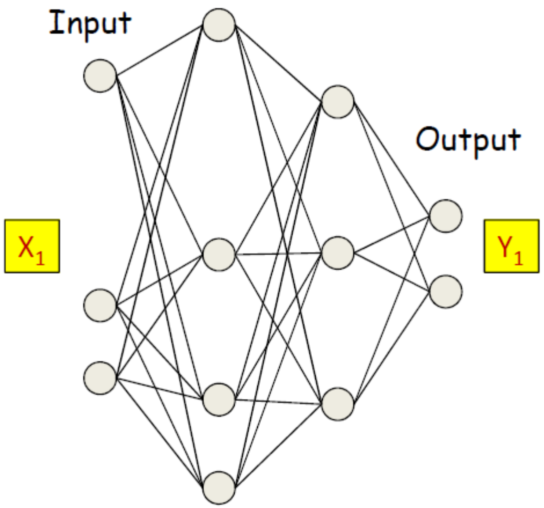




Understanding Dropout

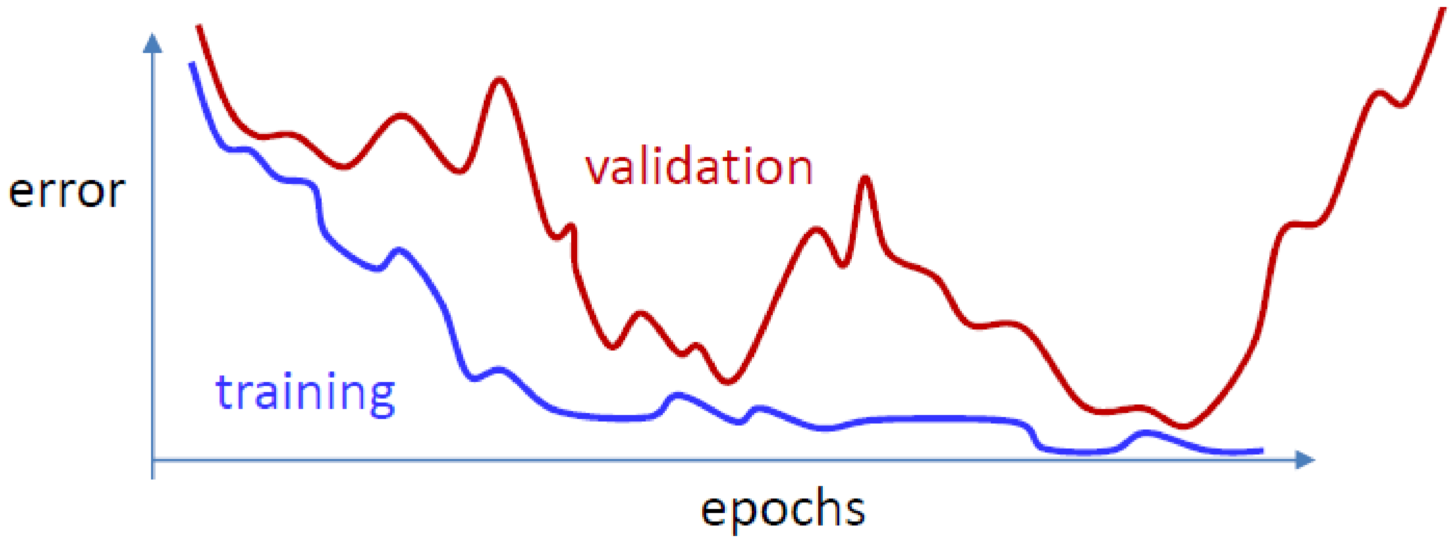
- Dropout forces the neural network to learn redundant patterns.
- Dropout can be viewed as an implicit L2 regularizer (Wager, Wang, Liang '13).





Early Stopping

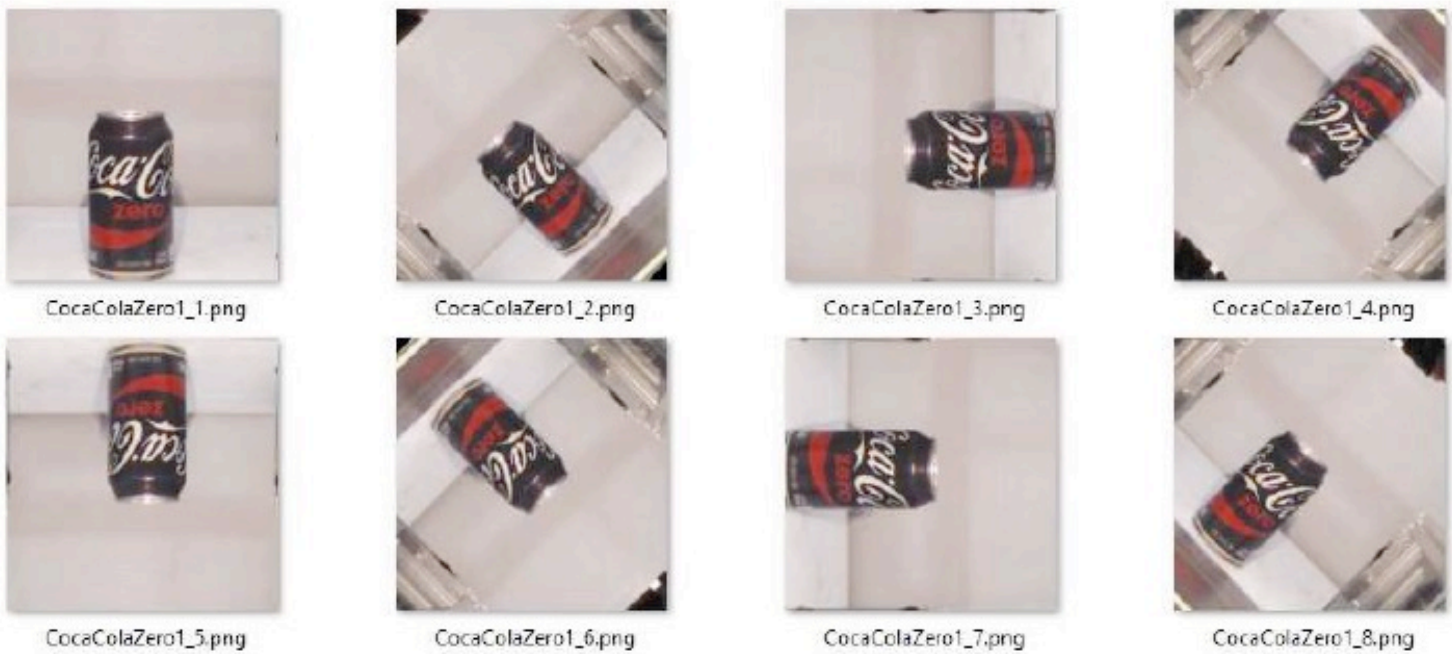
- Continue training may lead to overfitting.
- Track performance on a held-out validation set.
- Theory: for linear models, equivalent to L2 regularization.



Data Augmentation

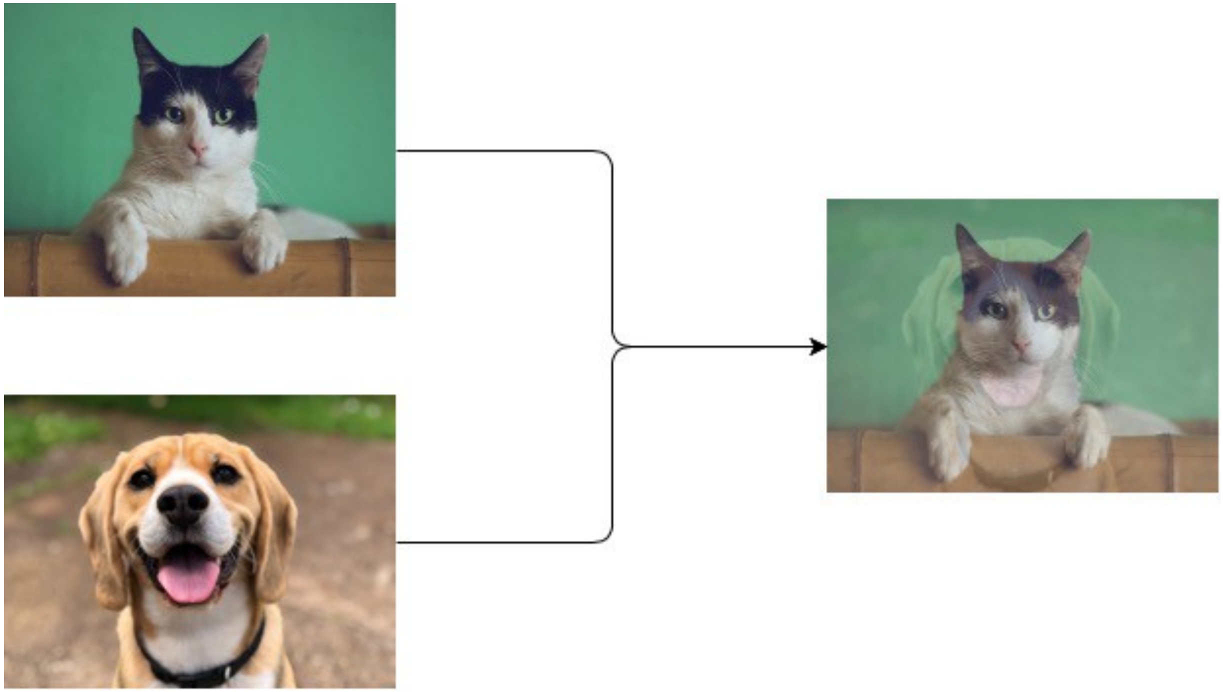
Depend on data types.

Computer vision: rotation, stretching, flipping, etc



Mixup data augmentation

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



Data Augmentation

Depend on data types.

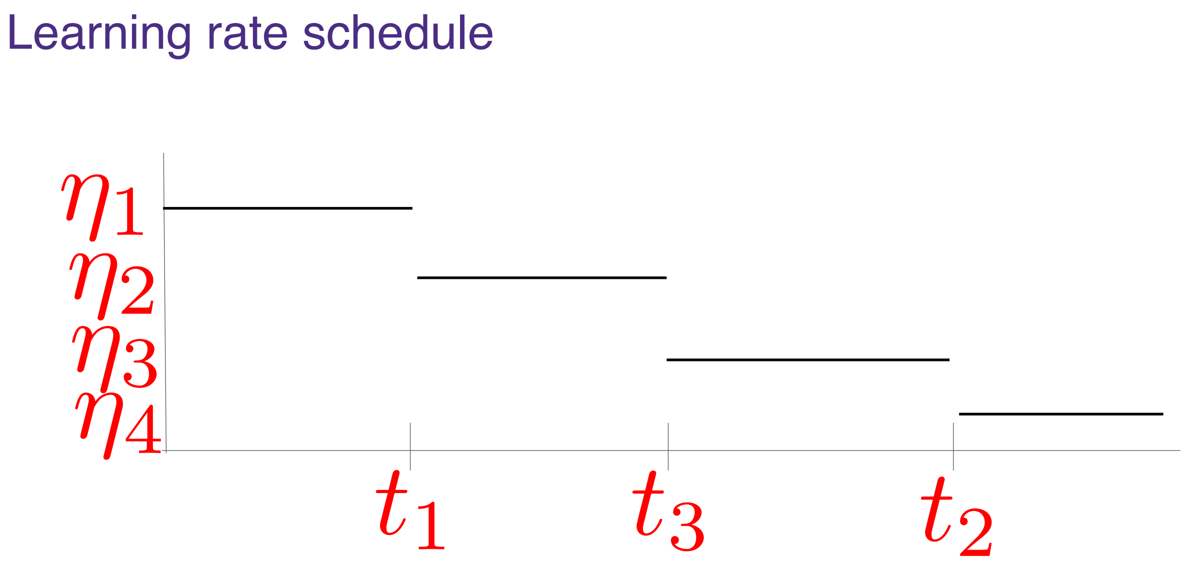
Natural language processing:

- 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.

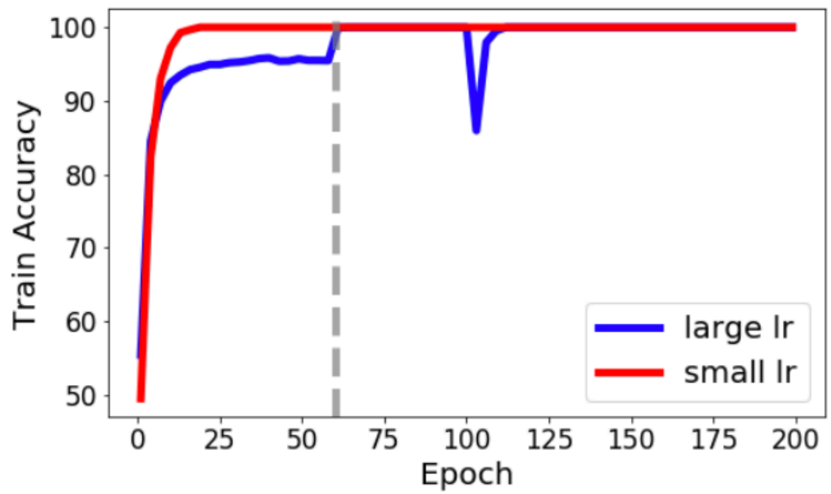


Learning rate scheduling

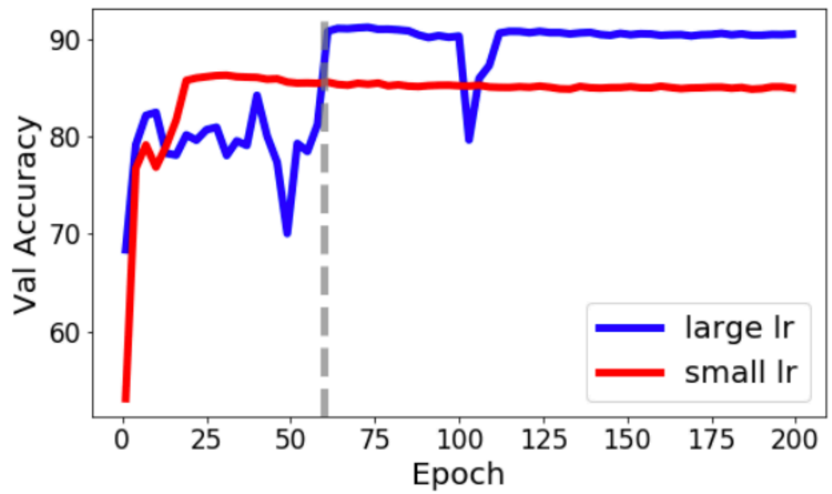
Start with large learning rate. After some epochs, use small learning rate.

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 ℓ, we have

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

VC-Dimension

Motivation: Do we need to consider **every** classifier in $\mathscr{F}$?
Intuitively, **pattern of classifications** on the training set should suffice. (Two predictors that predict identically on the training set should generalize similarly).

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

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

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

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

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

VC-dimension Generalization bound

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

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

Examples:

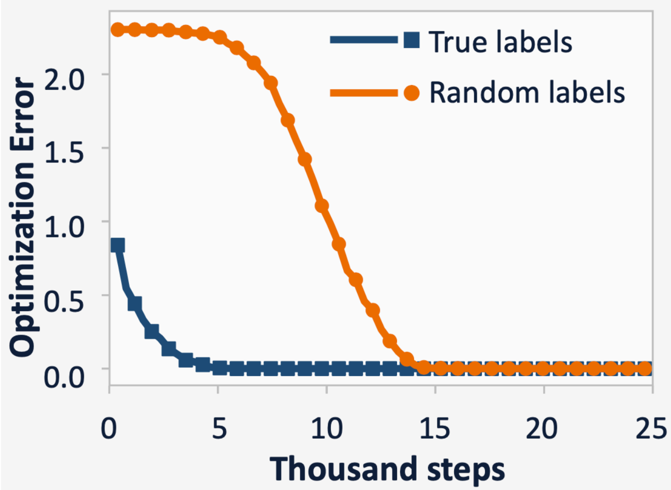
- Linear functions: VC-dim = O(dimension)
- Neural network: VC-dimension of fully-connected net with width W and H layers is $\widetilde{\Theta}(WH)$ (Bartlett et al., '17).

Problems with VC-dimension bound

1. In over-parameterized regime, bound >> 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

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



Optimization error **->** **0** for both ***true labels*** and ***random labels*** !

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 $\mathscr{F}$, 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 ℓ, we have

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

Rademacher Complexity

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

(Empirical) **Rademacher complexity:** For a training set $S = \{x_1, x_2, \ldots, x_n\}$, 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 $\sigma_i \sim$ 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 ℓ, we have

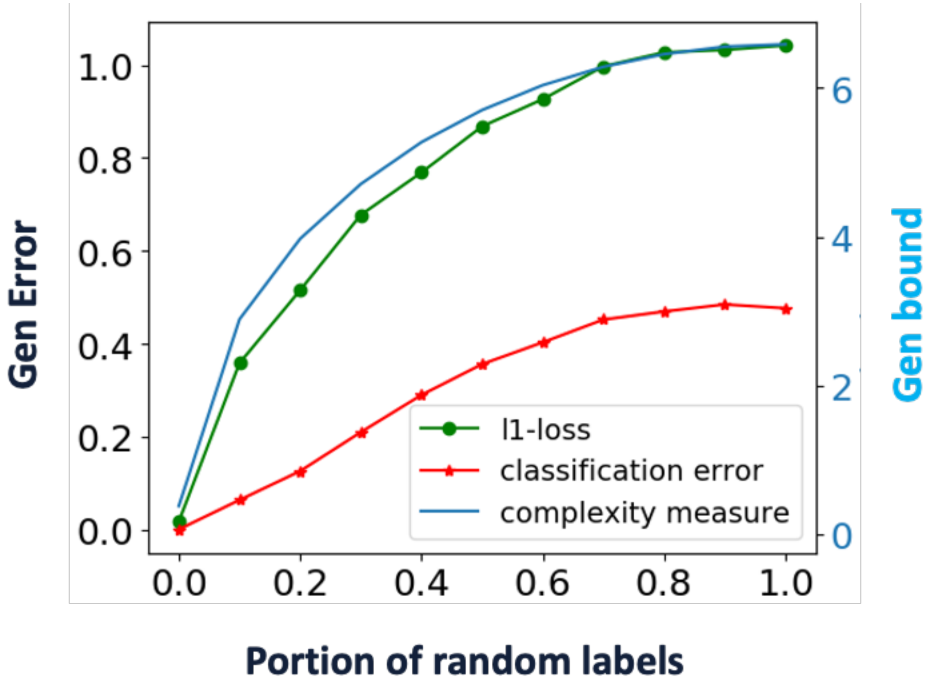
$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim D}\left[\ell(f(x), y)\right] \right| = O\left(\frac{\hat{R}_n}{n} + \sqrt{\frac{\log 1/\delta}{n}} \right)$$

and

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

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 is σ is ρ-Lipschitz. Let
$\mathcal{F} = \{x \mapsto W_{H+1}\sigma(W_h\sigma(\cdots\sigma(W_1x)\cdots)), \|W_h^T\|_{1,\infty} \leq B \forall h \in [H]\}$
then $R_n(\mathcal{S}) \leq \|X^\top\|_{2,\infty}(2\rho B)^{H+1}\sqrt{2\ln d}$ where
$X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$ 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.

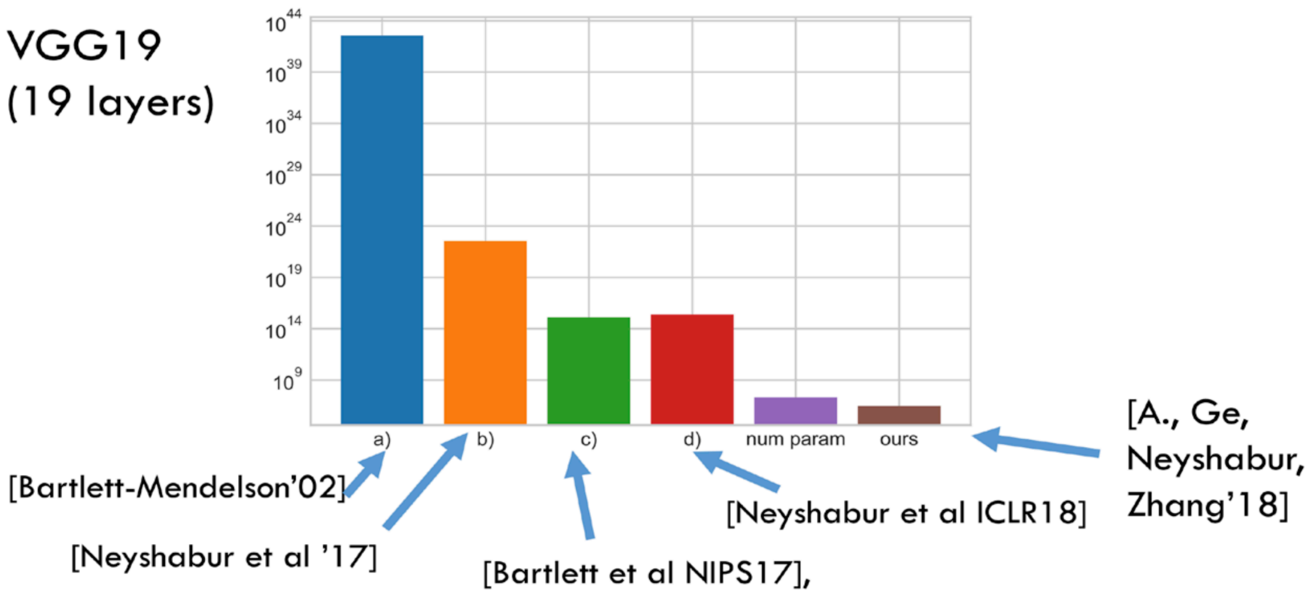


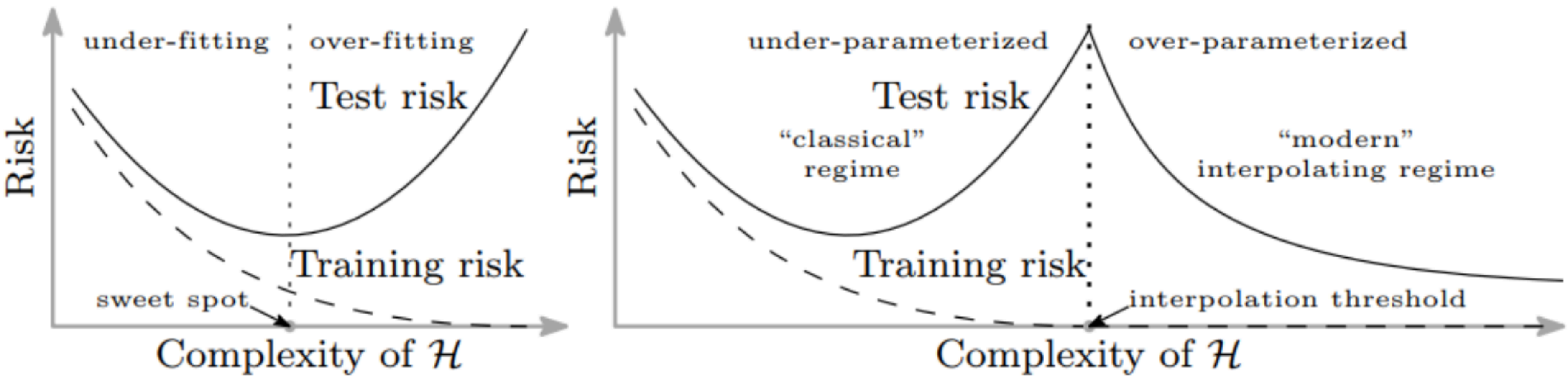
Image credits to Andrej Risteski

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

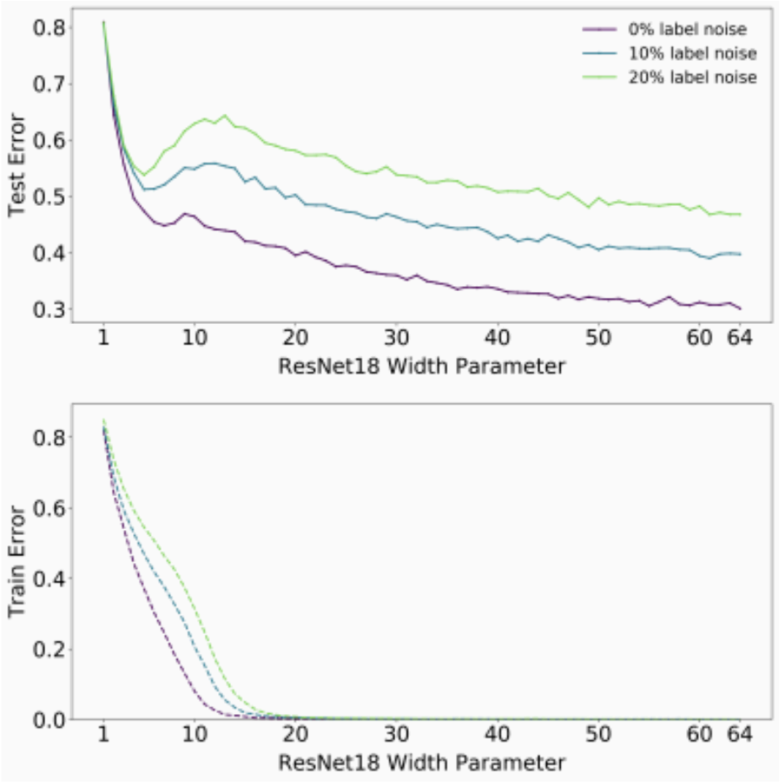
(b) "double descent" risk curve

Belkin, Hsu, Ma, Mandal '18

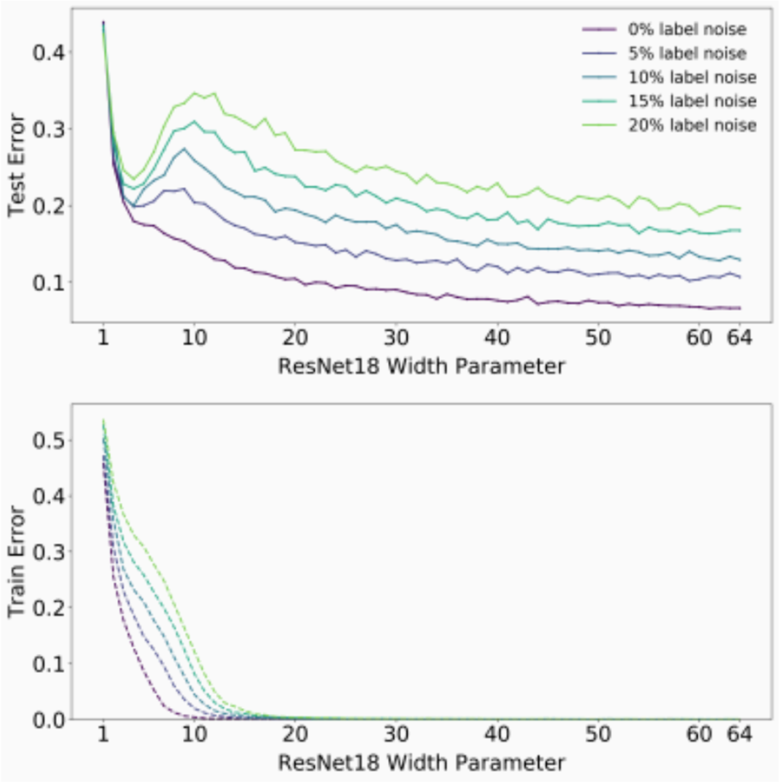
- There are cases where the model gets bigger, yet the (test!) loss goes down, sometimes even lower than in the classical "under-parameterized" regime.
- Complexity: number of parameters.

Double descent

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



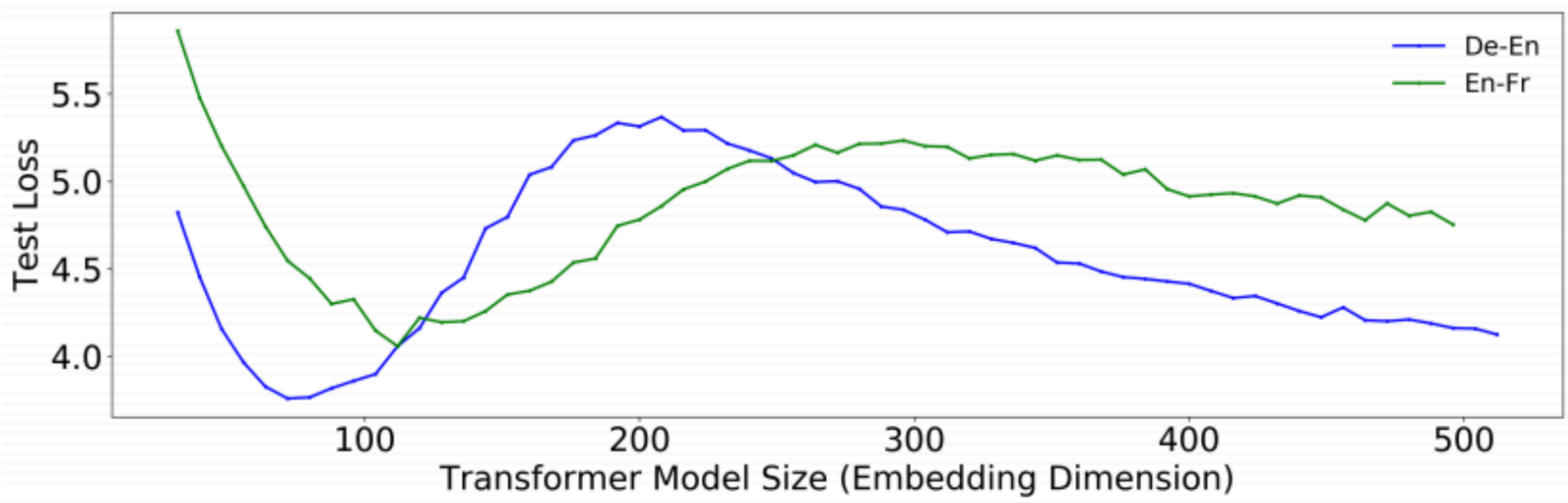
(a) **CIFAR-100.** There is a peak in test error even with no label noise.



(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.

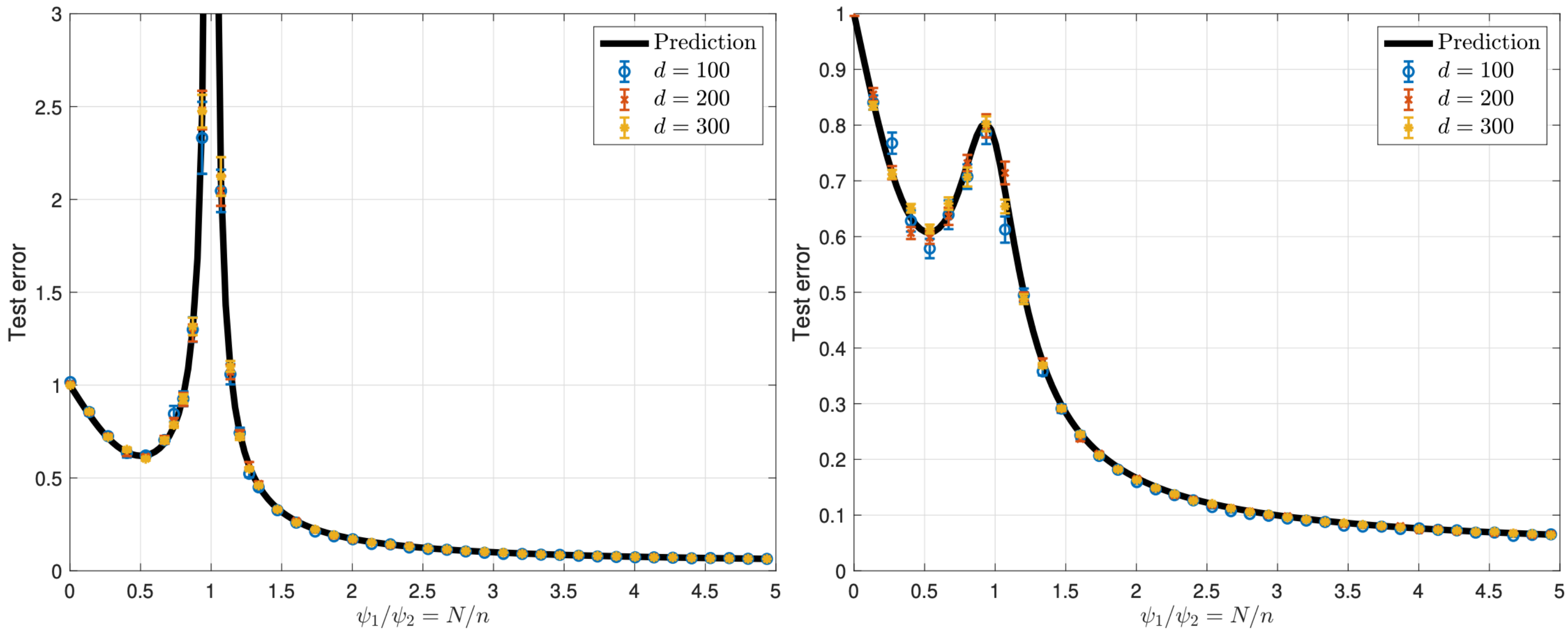
Double descent

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):

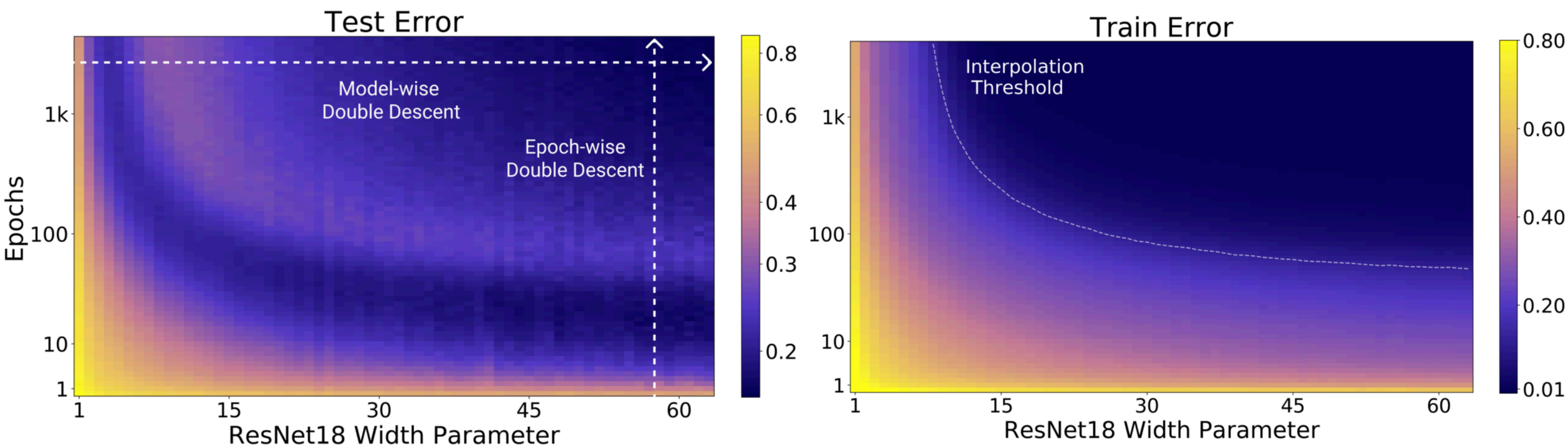
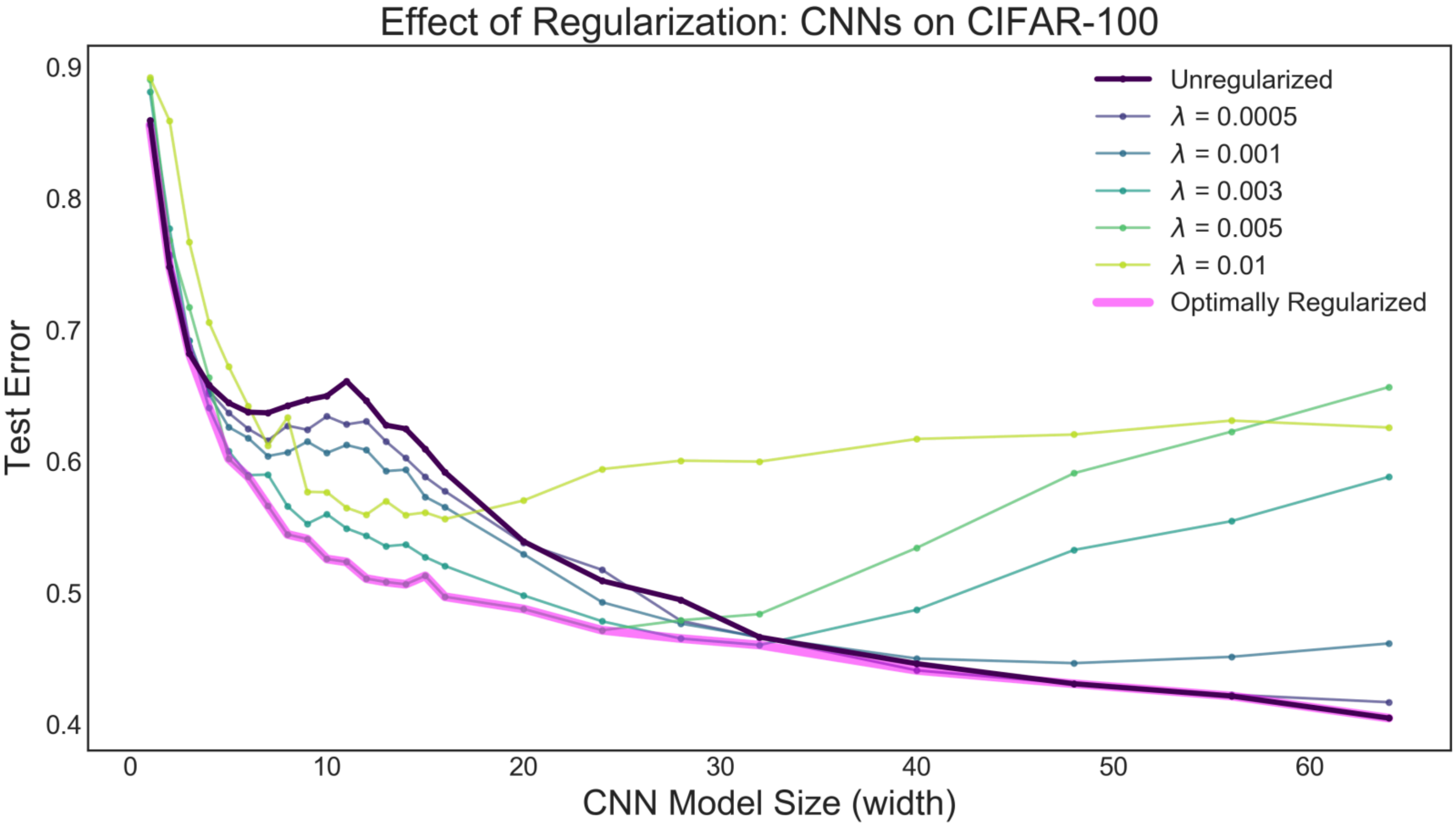


Figure 2: **Left:** Test error as a function of model size and train epochs. The horizontal line corresponds to model-wise double descent–varying model size while training for as long as possible. The vertical line corresponds to epoch-wise double descent, with test error undergoing double-descent as 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.

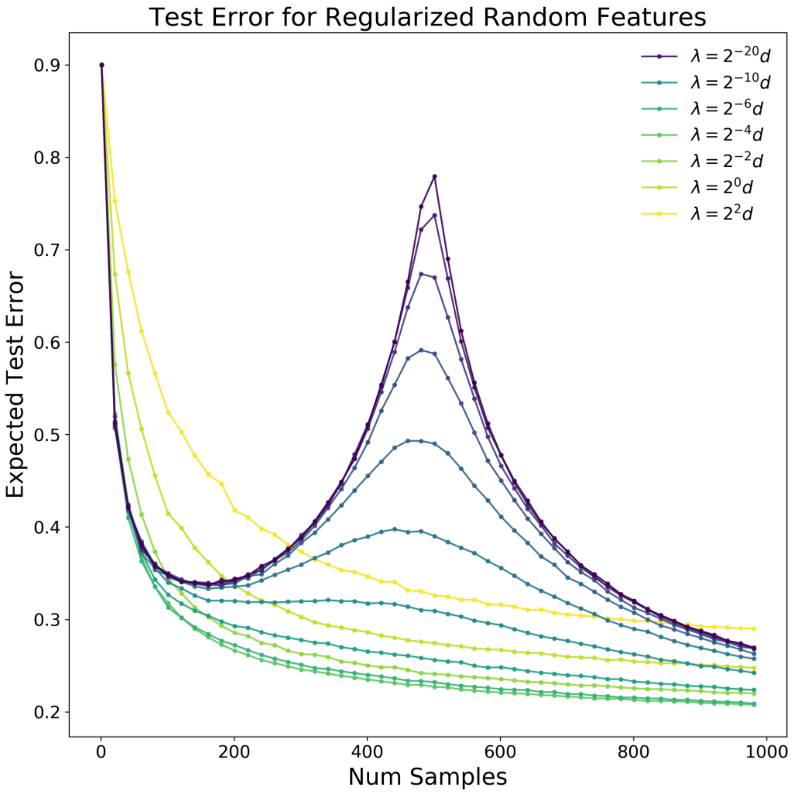
Double descent

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

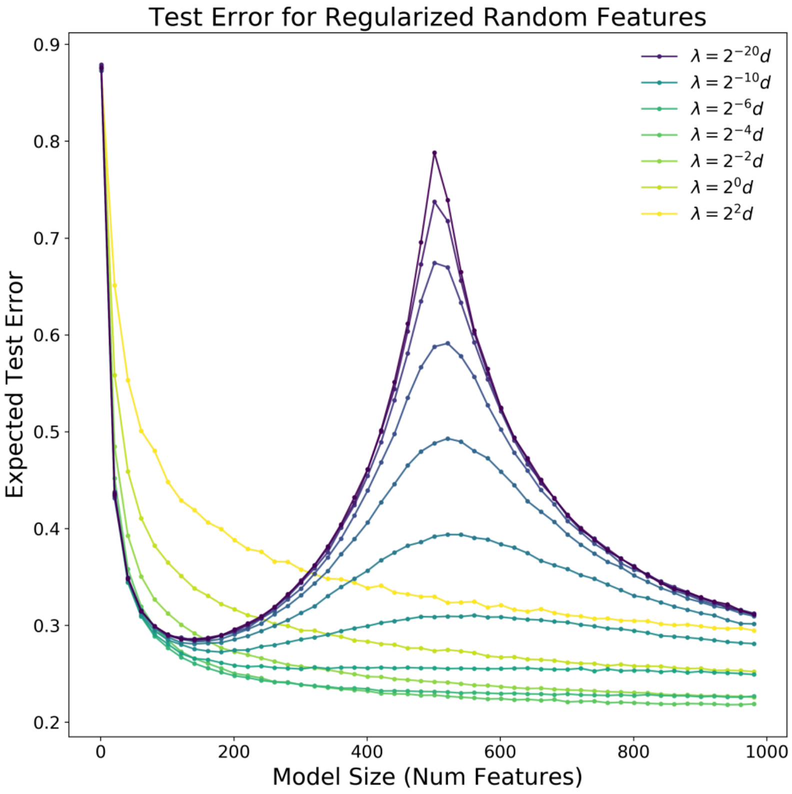


Double descent

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).

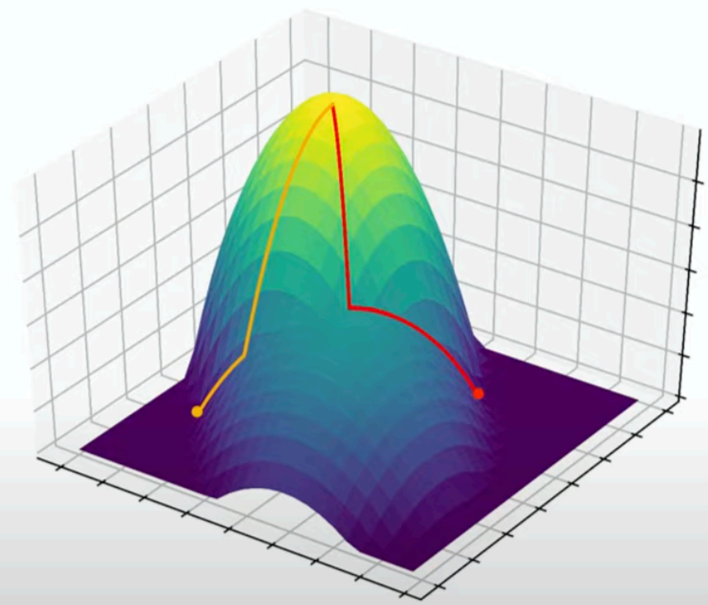
Implicit Regularization

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 $\mathscr{F}$ such that $y = f(x)$ for some $f \in \mathscr{F}$:
  - no kernel is sample-efficient;
  - Exists a neural network that is sample-efficient.