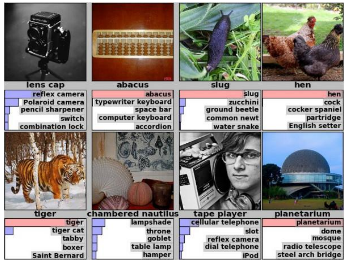
Provable Guarantees for Neural Networks via **Gradient Feature Learning**

Zhenmei Shi*, Junyi Wei*, Yingyu Liang

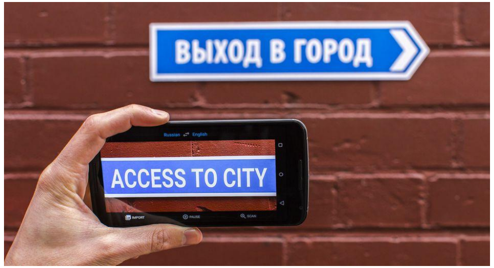
UW-Madison

NeurIPS 2023

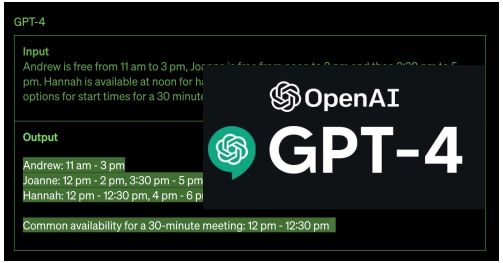
Machine Learning/AI Progress



Computer Vision

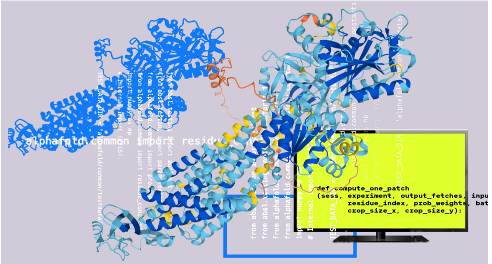


Natural Language Processing



Chatbots

Game Playing

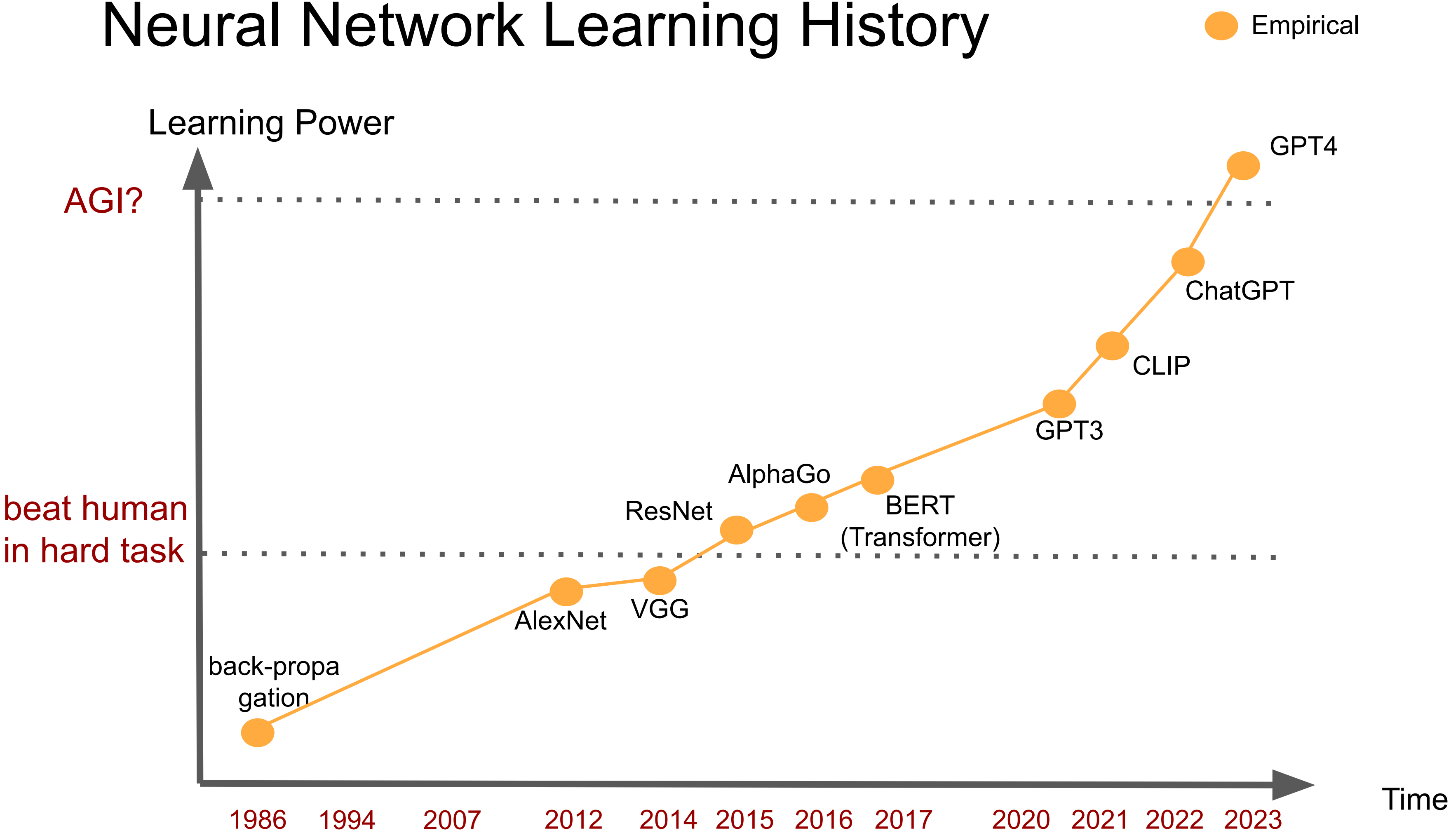


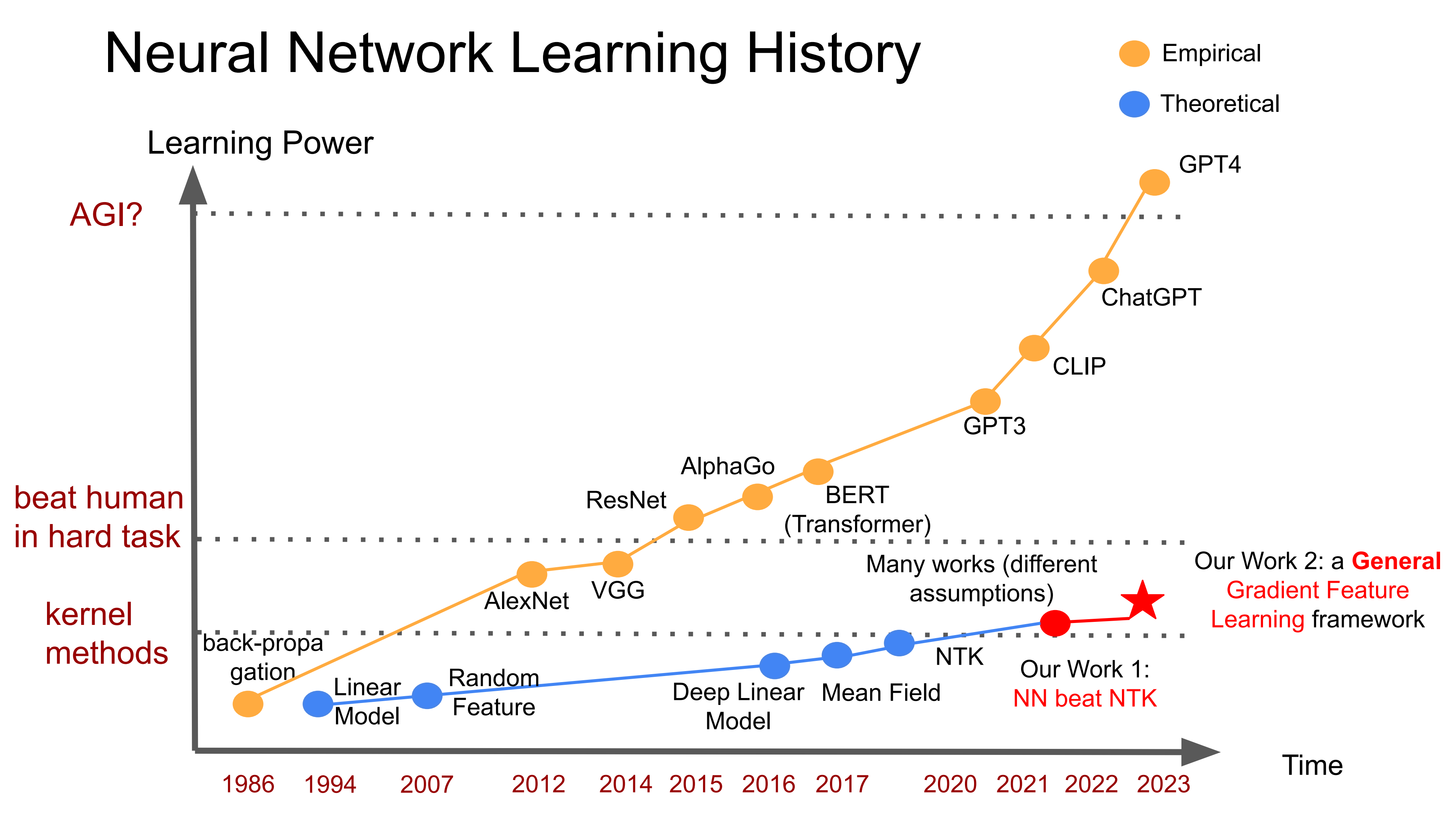
Sciences

Arts

credit: OpenAI, NYTimes, MIT technology review

Neural Network Learning History





Neural Network Learning History
Empirical
Theoretical
Learning Power
AGI?
beat human in hard task
kernel methods
GPT4
ChatGPT
CLIP
GPT3
AlphaGo
ResNet
BERT (Transformer)
AlexNet
VGG
Many works (different assumptions)
Our Work 2: a General Gradient Feature Learning framework
back-propagation
Linear Model
Random Feature
Deep Linear Model
Mean Field
NTK
Our Work 1: NN beat NTK
Time
1986
1994
2007
2012
2014
2015
2016
2017
2020
2021
2022
2023

Statistical Learning Theory Basics

- Given: training data $(x_i, y_i)_{i=1}^n$ i.i.d. from unknown distribution $\mathcal{D}$
- Learn model f from hypothesis class H by minimizing training loss

$$\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

- Goal: f has small test loss on future data (generalization)

$$L(f) = \mathrm{E}_{(x,y)\sim\mathcal{D}}[\ell(f(x), y)]$$

Statistical Learning Theory Basics

- To analyze generalization performance:

$$L(f) = \hat{L}(f) + L(f) - \hat{L}(f)$$

$$\leq \underbrace{\hat{L}(f)}_{\text{Optimization}} + \underbrace{\sup_{h \in H} L(h) - \hat{L}(h)}_{\text{Generalization gap}}$$

Often analyzed via convex optimization

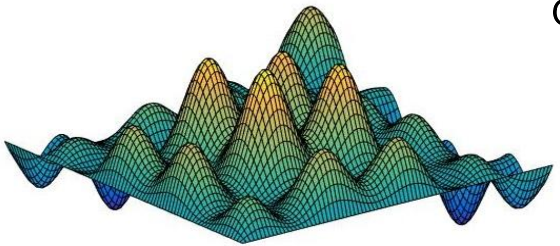
Often analyzed via uniform convergence bounds: $\frac{\text{capacity}(H)}{\sqrt{\#\text{samples}}}$

New Challenges in Analyzing Deep Learning

- To analyze generalization performance:

$$L(f) = \hat{L}(f) + L(f) - \hat{L}(f)$$

$$\le \underbrace{\hat{L}(f)}_{\text{Optimization}} + \underbrace{\sup_{h\in H} L(h) - \hat{L}(h)}_{\text{Generalization gap}}$$



Non-Convex Optimization
Via Gradient Descent

f: low training loss and low test loss

f': low training loss but high test loss

uniform bounds too loose

Fundamental Questions of Neural Networks

Approximation

Two layer-neural-network can approximate any Lipschitz function.

Optimization (challenging, non-convex!)

Why neural-network can find good solution on training data?

Generalization

Why the network also accurate on new test instances?

Recent Work on Optimization of NNs

Why neural networks success?

- Neural Tangent Kernel (NTK regime or lazy learning regime):
  - Key idea: with heavy overpara, in a close neighborhood of random init, the optimization is <u>almost convex</u>
  - Limitation: impractical overpara; generalization only for simple datasets; approximately a kernel method (fixed feature, <u>no feature learning</u>)
- Feature learning beyond NTK
  - Key idea: on data with specific structure, the training algo exploits the structure to <u>learn specific features</u> as the neuron weights
  - Can handle problems that cannot be handled by fixed feature methods
  - Limitation: <u>specific data</u> (mixture of Gaussians, parity etc)

Recent Work on Optimization of NNs

Why neural networks success?

- Neural Tangent Kernel (NTK regime or lazy learning regime):
  - Key idea: with heavy overpara, in a close neighborhood of random init, the optimization is almost convex
  - Limitation: impractical overpara, generalization only for simple datasets; approximately a kernel method (fixed feature, no feature learning)
- Feature learning beyond NTK
  - Key idea: on data with specific structure, the training algo exploits the structure to learn specific features as the neuron weights
  - Can handle problems that cannot be handled by fixed feature methods
  - Limitation: specific data (mixture of Gaussians, parity etc)

Can we provide a more general analysis framework?

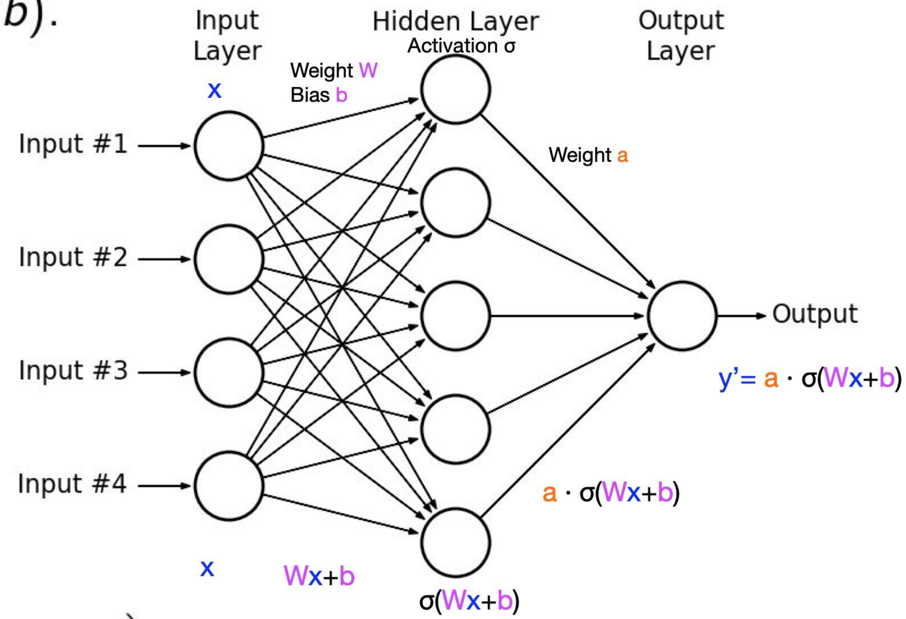
1. for more general data

2. Pin down the principle of feature learning in practical algo

Yes for two-layer networks

Problem Setting

Two-layer network: $y' = g(x) = a^T \sigma(Wx + b)$.

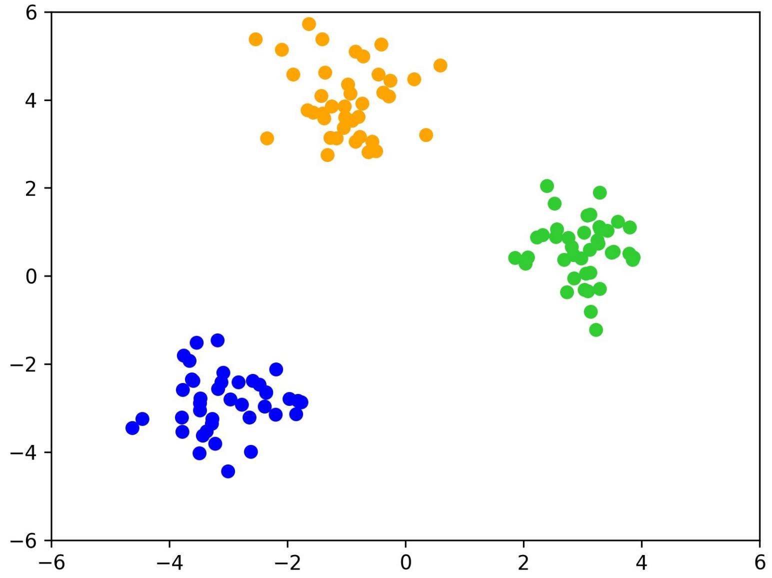


Train: gradient descent

$$\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_\theta \left(L(g^{(t-1)}) \right)$$

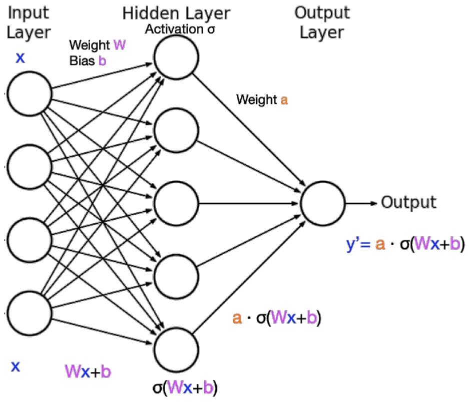
θ denotes W, b and a.

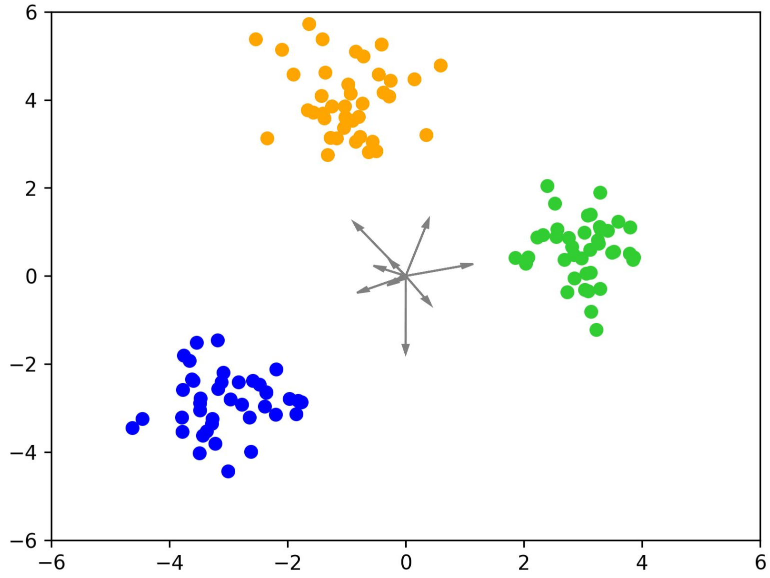
Example: Gradient Descent on Mixture of Gaussians



Mixture of 3 Gaussians, each being a class

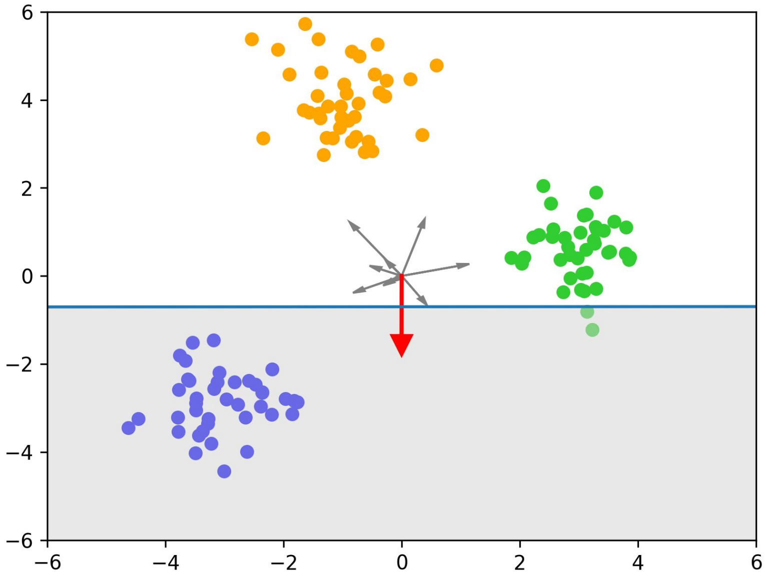
Example: Gradient Descent on Mixture of Gaussians





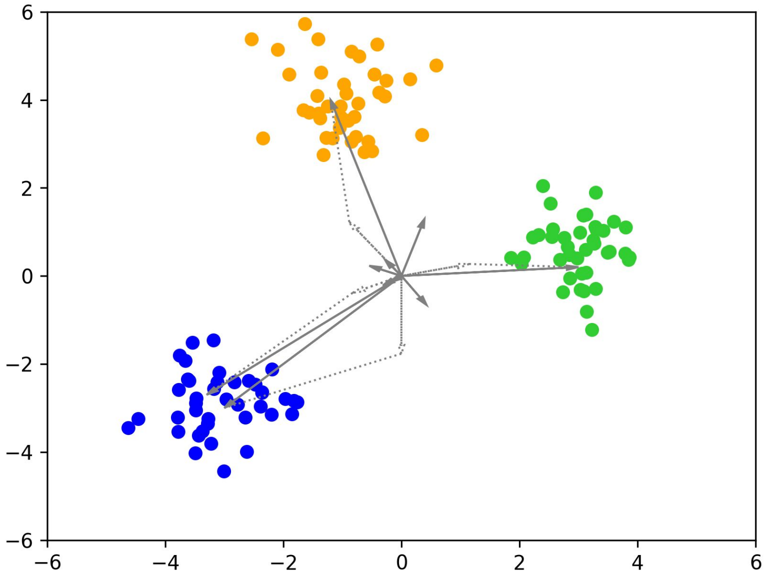
The weight vectors of neurons at random initialization

Example: Gradient Descent on Mixture of Gaussians



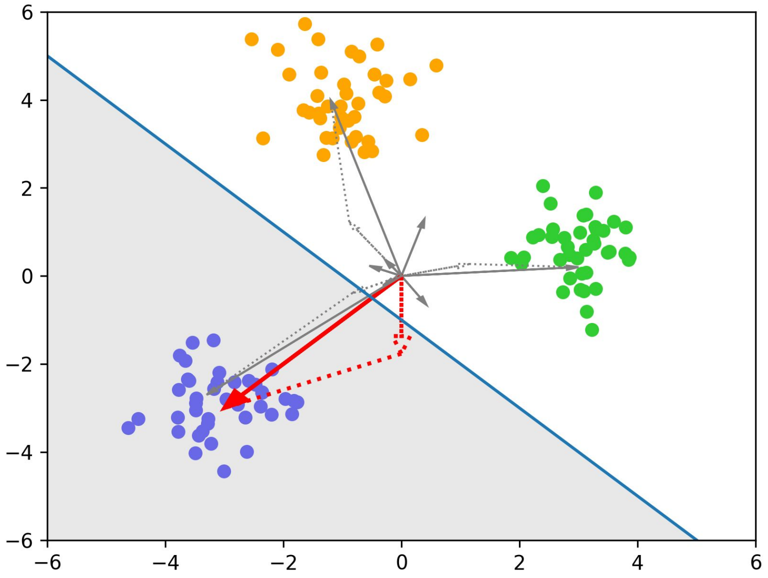
The activation region of one neuron (colored in red)

Example: Gradient Descent on Mixture of Gaussians



The weight vectors of neurons after one gradient update

Example: Gradient Descent on Mixture of Gaussians



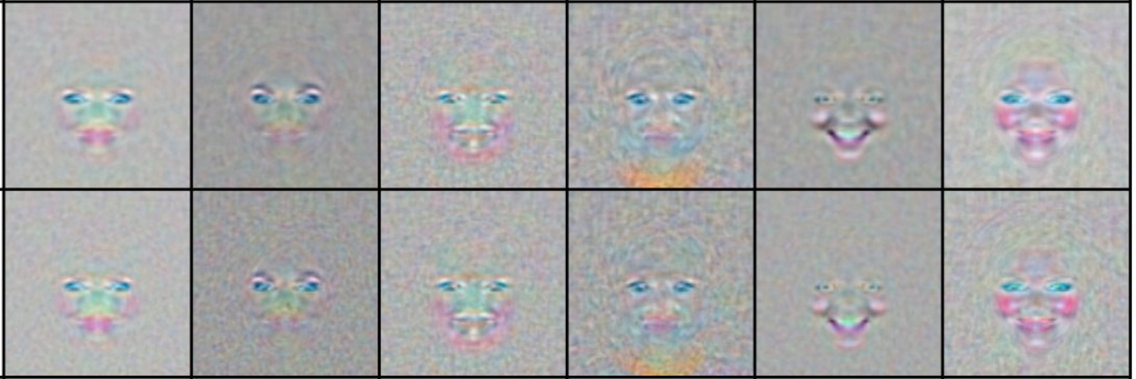
The activation region of one red neuron after one gradient update

Intuition

Gradient captures useful features of data

These features as neuron weights can lead to good performance

(c) Top Eigenvector of Feature Matrices on CelebA Prediction Tasks



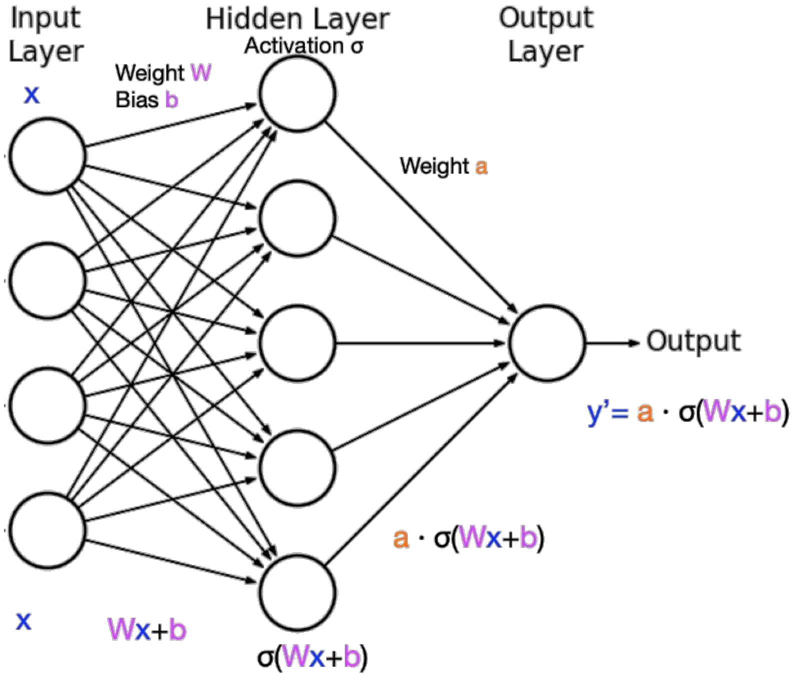
	Lipstick	Eyebrows	5 o'clock shadow	Necktie	Smiling	Rosy Cheeks
Deep Network Feature Matrix: $W_1^T W_1$						
RFM Feature Matrix: $\sum_{i=1}^{n} \nabla f(x_i) \nabla f(x_i)^T$						
Correlation	0.999	0.999	0.999	0.999	0.999	0.999
Deep Network Test Acc.	90.53%	75.71%	85.88%	88.77%	89.83%	87.22%
RFM-T Test Acc.	91.62%	78.11%	88.18%	90.39%	91.24%	88.72%

Visualization of Gradient Feature vs Weight Feature. From: *Feature learning in neural networks and kernel machines that recursively learn features (2022).*

Gradient Features

- Consider network $f(x) = \sum_i a_i \, \mathrm{ReLU}(\mathbf{w_i}^\top \mathbf{x} > b_i)$
- Binary classification, loss $\ell(yf(x))$
- Gradient of a neuron

$$\propto \; \mathrm{E}[\ell'(yf(x)) \cdot y \, \mathrm{I}[\mathbf{w_i}^\top \mathbf{x} > b_i] \mathbf{x}]$$



Gradient Features

- Consider network $f(x) = \sum_i a_i \, \mathrm{ReLU}(\mathbf{w_i}^\top \mathbf{x} > b_i)$
- Binary classification, loss $\ell(yf(x))$
- Gradient of a neuron

$$\propto \mathrm{E}[\, \ell'\big(yf(x)\big) \cdot y \, \mathrm{I}\big[\mathbf{w_i}^\top \mathbf{x} > b_i\big]\mathbf{x} \,]$$

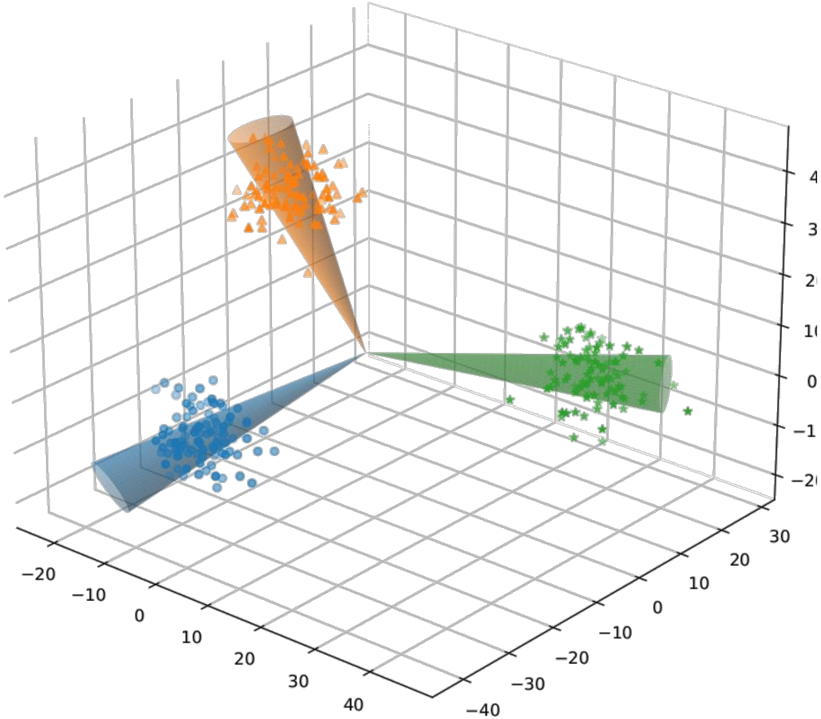
Definition (Simplified Gradient Vector)

For any $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, a Simplified Gradient Vector is defined as

$$G(\mathbf{w}, b) := \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[y\mathbf{x}\mathbb{I}[\mathbf{w}^\top \mathbf{x} > b]]. \quad (2)$$

Gradient Features

- Gradient Features: directions close to the Simplified Gradient Vectors



Gradient Feature Induced Networks

- Gradient Features as neuron weights can lead to good performance
- Induced networks: networks using gradient features as neuron weights
- Optimal approximation using induced networks:

Definition (Optimal Approximation via Gradient Features)

The Optimal Approximation network and loss using gradient feature induced networks $\mathcal{F}_{d,r,B_F,S}$ are defined as:

$$f^* := \text{argmin}_{f \in \mathcal{F}_{d,r,B_F,S}} L_{\mathcal{D}}(f), \qquad \text{OPT}_{d,r,B_F,S} := \min_{f \in \mathcal{F}_{d,r,B_F,S}} L_{\mathcal{D}}(f). \tag{6}$$

Unified Analysis Framework for Two-Layer NNs

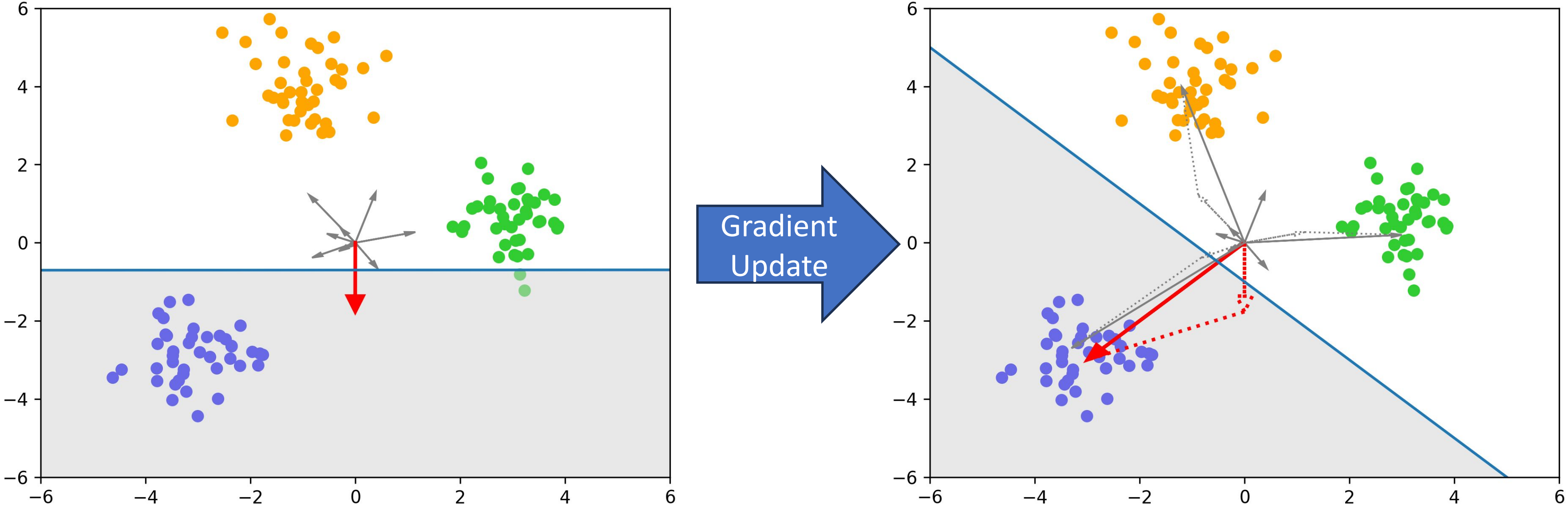
Main Theorem (informal)
Two-layer networks can in poly-time w.h.p. achieve test loss close to the optimal approximation loss by Gradient Feature Induced Networks.

General framework that

1. **captures the feature learning from gradients, and**
2. **gives poly error bounds for prototypical problems**

Proof Sketch

- First learns good features s.t. there is a good classifier on the neurons



Proof Sketch

- First learns good features s.t. there is a good classifier on the neurons
- Then learns a good classifier

- For illustration, suppose freezing the neurons after learning features
- Only need to learn second layer weights: Convex!

- In general, the neuron weights do not change significantly
- Learning second layer weights is similar to convex (Online convex opt)

Fundamental Questions of Neural Networks

Approximation

Two layer-neural-network can approximate any Lipschitz function.

Our focus:

Optimization (challenging, non-convex!)

Why neural-network can find good solution on training data?

Feature Learning (Beyond NTK) + Online Convex Optimization

Generalization

Why the network also accurate on new test instances?

Implicit Regularization / Simplicity Bias

Applications of the Framework

- Case studies on prototypical problems:
  - Mixtures of Gaussians
  - Parity functions
  - Linear data
  - Multiple-index data models + polynomial labeling functions

- Explaining some intriguing empirical phenomena
  - Beyond the Kernel Regime
  - Simplicity Bias
  - Lottery Ticket Hypothesis
  - Learning over Different Data Distributions
  - New Perspectives about Roadmaps Forward

Take-home Message

A general analysis framework for two-layer networks; unifies several recent work, provides new results

Key ideas:

- Gradient captures useful features of data
  - Exploits the structure of the data
  - Needs overparameterization to hit useful gradient features
- These features as neuron weights can lead to good performance
  - Strong approximation power of neural networks: traditionally viewed as an obstacle for optimization; here it's an advantage for learning

Future directions:

- Multiple layers
- Now only early-stage feature learning: 1 step gradient + almost convex optimization

Thanks!