Towards deeper understandings of deep learning

Yuanzhi Li

Stanford University

date: Today

A bit of history of machine learning

$\longrightarrow$ 1940s

- 13.772 billion years BC to 1940s:

A bit of history of machine learning

$\cdot \longrightarrow$ 1940s

- 13.772 billion years BC to 1940s:
  - Nothing happened...

A bit of history of machine learning

$\cdot \longrightarrow$ 1940s

- 13.772 billion years BC to 1940s:
  - Nothing happened...
- 1950s to 2010s:

A bit of history of machine learning

• → 1940s

- 13.772 billion years BC to 1940s:
  - Nothing happened...
- 1950s to 2010s:
  - The age of perceptrons.

A bit of history of machine learning

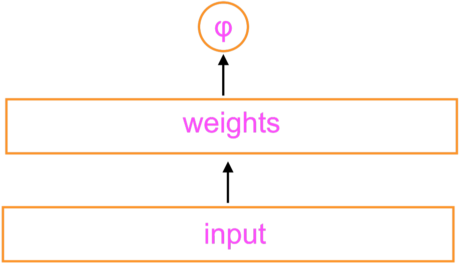
$\cdot \longrightarrow$ 1940s

- 13.772 billion years BC to 1940s:
  - Nothing happened...
- 1950s to 2010s:
  - The age of perceptrons.
  - Single layer learning models:

A bit of history of machine learning

• → 1940s

- 13.772 billion years BC to 1940s:
  - Nothing happened...
- 1950s to 2010s:
  - The age of perceptrons.
  - Single layer learning models:



-

A bit of history of machine learning

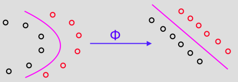
- Many variants of perceptrons: SVM, kernel SVM, products of experts

A bit of history of machine learning

- Many variants of perceptrons: SVM, kernel SVM, products of experts
- There is a lot of theory for them:

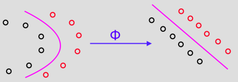
A bit of history of machine learning

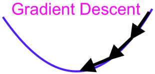
- Many variants of perceptrons: SVM, kernel SVM, products of experts
- There is a lot of theory for them:
  - Theory of kernels (representation power).



A bit of history of machine learning

- Many variants of perceptrons: SVM, kernel SVM, products of experts
- There is a lot of theory for them:
  - Theory of kernels (representation power).

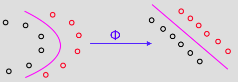


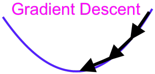


  - Convex optimization (training time bounds).

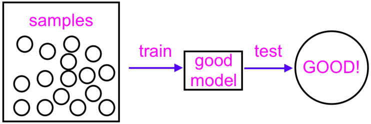
A bit of history of machine learning

- Many variants of perceptrons: SVM, kernel SVM, products of experts
- There is a lot of theory for them:
  - Theory of kernels (representation power).





  - Convex optimization (training time bounds).
  - VC theory (sample complexity and generalization).



A bit of history of machine learning

- In the past few years:

A bit of history of machine learning

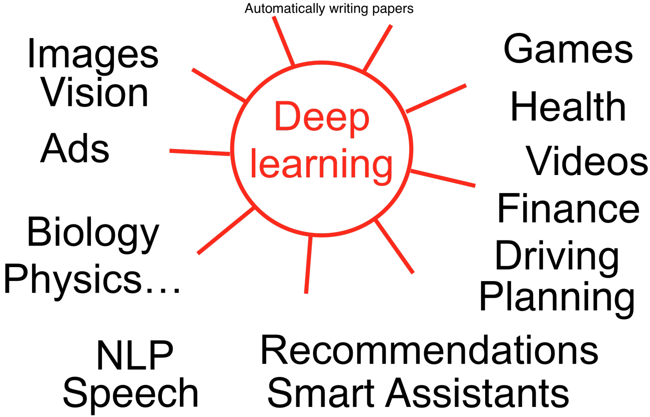
- In the past few years:
- Neural networks (Deep learning).

A bit of history of machine learning

- In the past few years:
- Neural networks (Deep learning).
- Works extremely well in practice, tons of applications.

A bit of history of machine learning

- In the past few years:
- Neural networks (Deep learning).
- Works extremely well in practice, tons of applications.



-

Neural Networks

- Multi-layer models.

Neural Networks

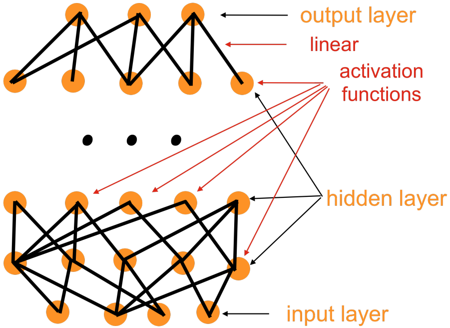
- Multi-layer models.
  - Input layer, output layer, and hidden layers.

Neural Networks

- Multi-layer models.
  - Input layer, output layer, and hidden layers.
    - Each hidden layer is associated with activation function(s).

Neural Networks

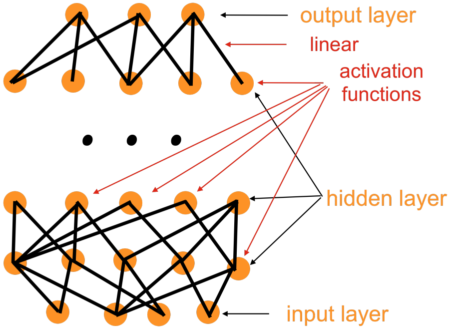
- Multi-layer models.
  - Input layer, output layer, and hidden layers.
    - Each hidden layer is associated with activation function(s).



-

Neural Networks

- Multi-layer models.
  - Input layer, output layer, and hidden layers.
    - Each hidden layer is associated with activation function(s).



  -
- In this talk, I will particularly focus on neural networks with ReLU activations.

Neural Networks work extremely well in practice

- But how do I prove that my neural networks work? How do we reason about their powers?

Neural Networks work extremely well in practice

- But how do I prove that my neural networks work? How do we reason about their powers?
- In the past years, people have reached a satisfying answer to this question:

Neural Networks work extremely well in practice

- But how do I prove that my neural networks work? How do we reason about their powers?
- In the past years, people have reached a satisfying answer to this question:
  - Stop asking about it!

Neural Networks work extremely well in practice

- But how do I prove that my neural networks work? How do we reason about their powers?
- In the past years, people have reached a satisfying answer to this question:
  - Stop asking about it!
  - Fact: Neural networks work in practice although we can not fully prove it.

Neural Networks work extremely well in practice

- But how do I prove that my neural networks work? How do we reason about their powers?
- In the past years, people have reached a satisfying answer to this question:
  - Stop asking about it!
  - Fact: Neural networks work in practice although we can not fully prove it.
  - 

Why stop asking?

- There is a simple reason we might want to stop asking about it:

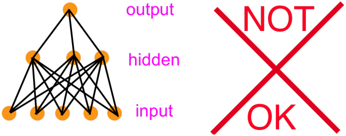
Why stop asking?

- There is a simple reason we might want to stop asking about it:
- In theory, neural networks are also well known as neural notworks.

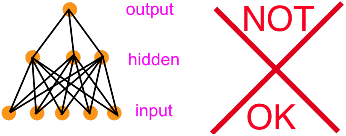
Why stop asking?

- There is a simple reason we might want to stop asking about it:
- In theory, neural networks are also well known as neural notworks.
    - Learning a neural network with three hidden units is NP-hard (BR'98).

Why stop asking?

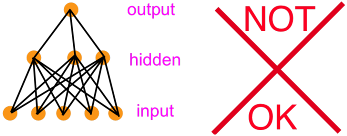
- There is a simple reason we might want to stop asking about it:
- In theory, neural networks are also well known as neural notworks.
    - Learning a neural network with three hidden units is NP-hard (BR'98).
    - 

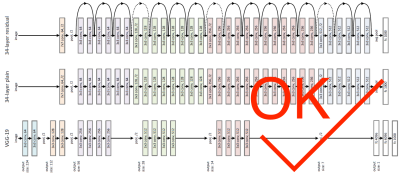
Why stop asking?

- There is a simple reason we might want to stop asking about it:
- In theory, neural networks are also well known as neural notworks.
  - Learning a neural network with three hidden units is NP-hard (BR'98).
  - 

  - In practice, we have millions and they still work.

Why stop asking?

- There is a simple reason we might want to stop asking about it:
- In theory, neural networks are also well known as neural notworks.
    - Learning a neural network with three hidden units is NP-hard (BR'98).
    - 

    - In practice, we have millions and they still work.
    - 

And even harder

- Yet there is another reason:

And even harder

- Yet there is another reason:
- In neural networks, we optimize any neural networks by essentially the same simple simple simple simple simple algorithm: SGD.

And even harder

- Yet there is another reason:
- In neural networks, we optimize any neural networks by essentially the same simple simple simple simple simple algorithm: SGD.
    - Starting from W_0 at random (often Gaussian).

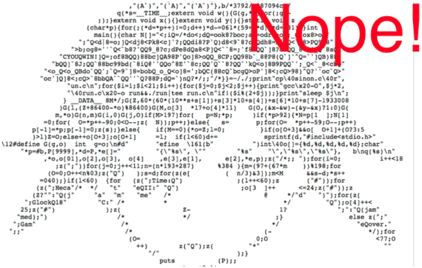
And even harder

- Yet there is another reason:
- In neural networks, we optimize any neural networks by essentially the same simple simple simple simple simple algorithm: SGD.
  - Starting from W_0 at random (often Gaussian).
  - In each iteration, sample an example x from the data set, then update the weights W_t as: $W_{t+1} = W_t - \eta \nabla f(W_t, x)$.

And even harder

- Yet there is another reason:
- In neural networks, we optimize any neural networks by essentially the same simple simple simple simple simple algorithm: SGD.
  - Starting from W_0 at random (often Gaussian).
  - In each iteration, sample an example x from the data set, then update the weights W_t as: $W_{t+1} = W_t - \eta \nabla f(W_t, x)$.
- We use it (or its variants) almost everywhere, regardless of the problem/difficulty.

And even harder

- Yet there is another reason:
- In neural networks, we optimize any neural networks by essentially the same simple simple simple simple simple algorithm: SGD.
    - Starting from W_0 at random (often Gaussian).
    - In each iteration, sample an example x from the data set, then update the weights W_t as: $W_{t+1} = W_t - \eta \nabla f(W_t, x)$.
- We use it (or its variants) almost everywhere, regardless of the problem/difficulty.
- 


```
optimizer = optim.SGD(para, lr)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Yes!

Face recognition

And even harder

- Yet there is another reason:
- In neural networks, we optimize any neural networks by essentially the same simple simple simple simple simple algorithm: SGD.
    - Starting from W_0 at random (often Gaussian).
    - In each iteration, sample an example x from the data set, then update the weights W_t as: $W_{t+1} = W_t - \eta \nabla f(W_t, x)$.
- We use it (or its variants) almost everywhere, regardless of the problem/difficulty.
- 

- Makes the theoretical work even harder (understand the universality of SGD).

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

Graphical Models
Tensor decomposition
Mixture of regressions
k-means
ICA NMF
Deep Learning
Bellman equation
MLE
Sparse Coding
Weighted SVD
Dictionary Learning
Matrix Completion
Topic Modeling
Matrix sensing

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

Graphical Models Tensor decomposition
Mixture of regressions
k-means
ICA NMF Deep Learning
Bellman equation MLE Sparse Coding Weighted SVD
Dictionary Learning Matrix Completion
Topic Modeling Matrix sensing

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.
- In this talk I will focus on principles relevant to (multi-layer) neural networks with ReLU activation functions (NP-hard model).

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.
- In this talk I will focus on principles relevant to (multi-layer) neural networks with ReLU activation functions (NP-hard model).
- Training: SGD with random initializations (simple algorithm).

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.
- In this talk I will focus on principles relevant to (multi-layer) neural networks with ReLU activation functions (NP-hard model).
- Training: SGD with random initializations (simple algorithm).
- We will discuss two principles in this talk.

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

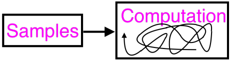
Graphical Models
Tensor decomposition
Mixture of regressions
k-means
ICA NMF
Deep Learning
Bellman equation
MLE
Sparse Coding
Weighted SVD
Dictionary Learning
Matrix Completion
Topic Modeling
Matrix sensing

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.
- In this talk I will focus on principles relevant to (multi-layer) neural networks with ReLU activation functions (NP-hard model).
- Training: SGD with random initializations (simple algorithm).
- We will discuss two principles in this talk.
  - Over-parameterization.

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.
- In this talk I will focus on principles relevant to (multi-layer) neural networks with ReLU activation functions (NP-hard model).
- Training: SGD with random initializations (simple algorithm).
- We will discuss two principles in this talk.
  - Over-parameterization.

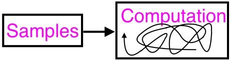


  - Noisy computation.

My work

- In Machine learning, many models are NP-hard, but nevertheless solved efficiently in practice by simple algorithms.

- I work on new, principled ways to understand and enhance the success of deep learning/machine learning in general.
- In this talk I will focus on principles relevant to (multi-layer) neural networks with ReLU activation functions (NP-hard model).
- Training: SGD with random initializations (simple algorithm).
- We will discuss two principles in this talk.
  - Over-parameterization.



  - Noisy computation.
- And how can they help in learning neural networks, provably.

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
    - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (A person who works on deep learning): You should use a larger network.

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (A person who works on deep learning): You should use a larger network.
- (2). My network finds the global optimal on the training set, but it generalizes badly. What should I do?

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (A person who works on deep learning): You should use a larger network.
- (2). My network finds the global optimal on the training set, but it generalizes badly. What should I do?
  - (A person who just learned the VC dimension): Make the total number of parameters smaller so it generalizes better.

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (A person who works on deep learning): You should use a larger network.
- (2). My network finds the global optimal on the training set, but it generalizes badly. What should I do?
  - (A person who just learned the VC dimension): Make the total number of parameters smaller so it generalizes better.
  - (A person who works on deep learning): You should use a larger network.

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (A person who works on deep learning): You should use a larger network.
- (2). My network finds the global optimal on the training set, but it generalizes badly. What should I do?
  - (A person who just learned the VC dimension): Make the total number of parameters smaller so it generalizes better.
  - (A person who works on deep learning): You should use a larger network.
- (3). My network finds a good solution empirically, but I can't prove it. What should I do?

Conversation between deep learning and theory

- (1). My network finds the global optimal on the training set, but it converges slowly. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (A person who works on deep learning): You should use a larger network.
- (2). My network finds the global optimal on the training set, but it generalizes badly. What should I do?
  - (A person who just learned the VC dimension): Make the total number of parameters smaller so it generalizes better.
  - (A person who works on deep learning): You should use a larger network.
- (3). My network finds a good solution empirically, but I can't prove it. What should I do?
  - (In this talk): You should use a larger network!

Principle of over-parameterization

- By building up a network with (much more) parameters than the total number of training examples.

Principle of over-parameterization

- By building up a network with (much more) parameters than the total number of training examples.
- 

Principle of over-parameterization

- By building up a network with (much more) parameters than the total number of training examples.

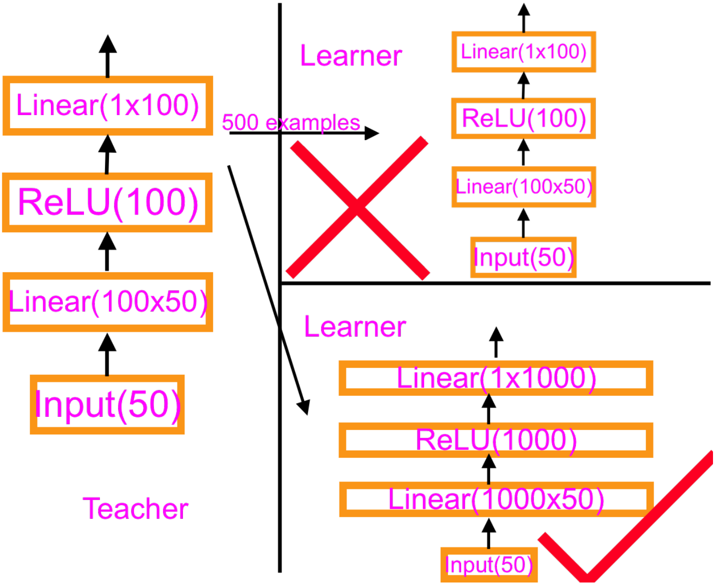
-
- Improves both the training and generalization.

Principle of over-parameterization

- By building up a network with (much more) parameters than the total number of training examples.

-
- Improves both the training and generalization.
- And it improves the theory.

Folklore example for training.



Example for generalization

Widen factor	Number of parameters	Test error
1	0.6M	6.85
2	2.2M	5.33
4	8.9M	4.97
8	36.5M	4.66

Table: Depth 40 WideResNet on CIFAR-10 (0.05M training examples), training errors are all 0.

Example of the Theorem

- Given enough over-parameterization, prove that:

Example of the Theorem

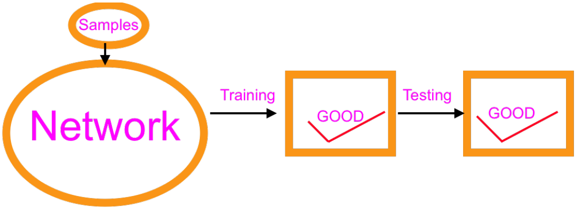
- Given enough over-parameterization, prove that:
- (1). SGD will find a good solution on the training data set (close to zero training error).

Example of the Theorem

- Given enough over-parameterization, prove that:
- (1). SGD will find a good solution on the training data set (close to zero training error).
- (2). And it generalizes to test data set.

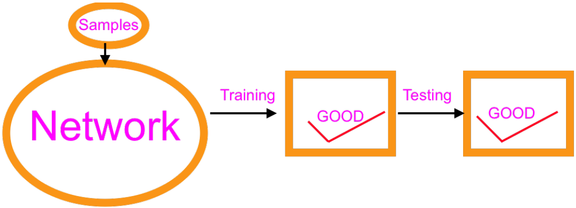
Example of the Theorem

- Given enough over-parameterization, prove that:
- (1). SGD will find a good solution on the training data set (close to zero training error).
- (2). And it generalizes to test data set.



-

Example of the Theorem

- Given enough over-parameterization, prove that:
- (1). SGD will find a good solution on the training data set (close to zero training error).
- (2). And it generalizes to test data set.
- 

- We begin with our theorem for (1), then we will see (2) as well.

Our Theorem

Theorem (Sketched, (LL'18, ALS'18a,b))

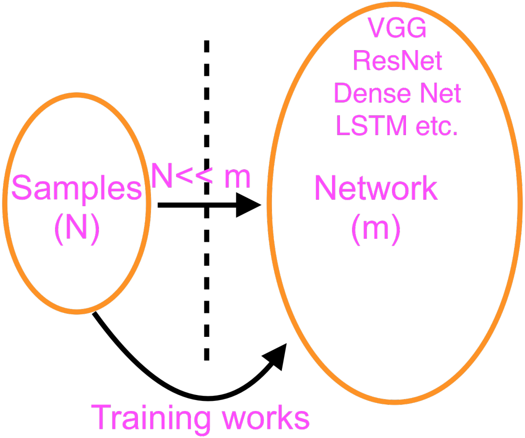
Given N different training examples $x_1, \dots, x_N$ with labels $y_1, \dots, y_N$, then for every $\varepsilon > 0$, as long as the number of neurons (m) in the network satisfies

$$m \geq poly(N \log(1/\varepsilon))$$

then SGD starting from gaussian random initialization finds an ε-approximate global optimal of the training objective in time poly(m/ε).

The theorem holds for multi-layer DNNs, CNNs, ResNet and Recurrent Neural Networks (all with ReLU activation functions), the training loss can be given by ℓ_2 loss, cross entropy etc.

Our theorem by picture



-

Closer look at the Theorem:

- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.

Closer look at the Theorem:

- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Essentially no assumptions on the training examples, in particular training labels can be random.

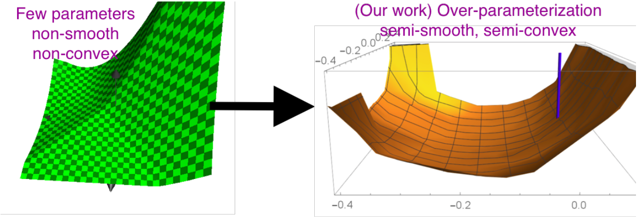
Closer look at the Theorem:

- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Essentially no assumptions on the training examples, in particular training labels can be random.
  - From capacity perspective: Obviously there exists a network that fits the training data.

Closer look at the Theorem:

- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Essentially no assumptions on the training examples, in particular training labels can be random.
  - From capacity perspective: Obviously there exists a network that fits the training data.
  - From optimization perspective: How can SGD find such fitting? The training objective is not only non-convex, but non-smooth as well due to ReLU.

Closer look at the Theorem:

- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Essentially no assumptions on the training examples, in particular training labels can be random.
    - From capacity perspective: Obviously there exists a network that fits the training data.
    - From optimization perspective: How can SGD find such fitting? The training objective is not only non-convex, but non-smooth as well due to ReLU.
    - 

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
  - What about generalization?

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
  - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
  - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.
  - Over-parameterized networks (AlexNet) can fit CIFAR-10 data with random labels.

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
  - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.
  - Over-parameterized networks (AlexNet) can fit CIFAR-10 data with random labels.
  - The capacity of the model is way larger than the total number of training examples.

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
  - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.
  - Over-parameterized networks (AlexNet) can fit CIFAR-10 data with random labels.
  - The capacity of the model is way larger than the total number of training examples.
  - More importantly, SGD can find such (over)fitting.

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
  - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.
  - Over-parameterized networks (AlexNet) can fit CIFAR-10 data with random labels.
  - The capacity of the model is way larger than the total number of training examples.
  - More importantly, SGD can find such (over)fitting.

  -

Training is ok with over-parameterization

- Memorization is not a completely trivial theorem, but much simpler compare to this question:
    - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.
    - Over-parameterized networks (AlexNet) can fit CIFAR-10 data with random labels.
    - The capacity of the model is way larger than the total number of training examples.
    - More importantly, SGD can find such (over)fitting.

    -
    - So why does the solution found by SGD still generalize?

Two main reasons for generalization:

- The labels are not random (necessary).

Two main reasons for generalization:

- The labels are not random (necessary).
  - What about SGD?

Two main reasons for generalization:

- The labels are not random (necessary).
  - What about SGD?
- The inductive bias of SGD: SGD usually biases towards generalizable solutions instead of the solutions that simply memorize the training data.

Two main reasons for generalization:

- The labels are not random (necessary).
  - What about SGD?
- The inductive bias of SGD: SGD usually biases towards generalizable solutions instead of the solutions that simply memorize the training data.
  - We are going to prove these claims for certain neural networks.

Two main reasons for generalization:

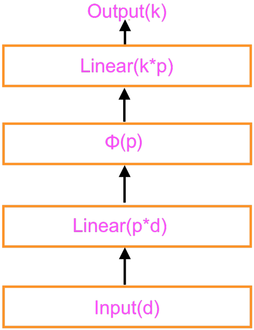
- The labels are not random (necessary).
  - What about SGD?
- The inductive bias of SGD: SGD usually biases towards generalizable solutions instead of the solutions that simply memorize the training data.
  - We are going to prove these claims for certain neural networks.
  - We start with two layer networks, and then multi-layer ones.

Labels are not random? (Two layer networks)

- Assume that the labels are realizable by a simple neural network:

Labels are not random? (Two layer networks)

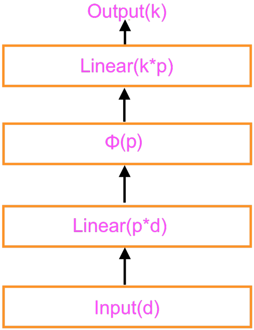
- Assume that the labels are realizable by a simple neural network:



-

Labels are not random? (Two layer networks)

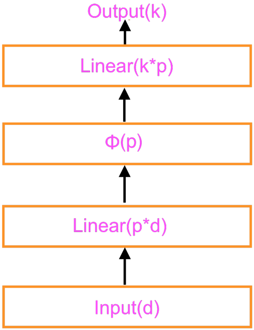
- Assume that the labels are realizable by a simple neural network:



-
- Two-layer network with p hidden neurons, where p is something small (e.g. 100). Φ is a smooth activation function (such as $\sin, \cos, \exp, \tanh, \mathit{sigmoid}$, and low degree polynomials).

Labels are not random? (Two layer networks)

- Assume that the labels are realizable by a simple neural network:



-
- Two-layer network with p hidden neurons, where p is something small (e.g. 100). Φ is a smooth activation function (such as $\sin, \cos, \exp, \tanh, \textit{sigmoid}$, and low degree polynomials).
- Such that the average loss of this network on the training data set is $\leq \varepsilon$.

Our Theorem

Theorem (Sketched (LL'18, ALL'18))

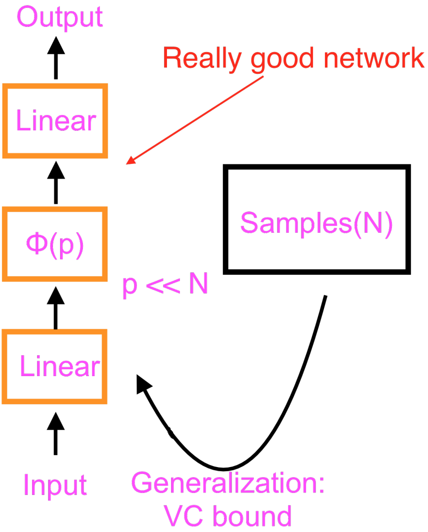
Suppose the labels of the data can be realized as in the previous slides, then as long as the number of training examples N satisfies:

$$N \geq poly(kp/\varepsilon) \times poly(log(m))$$

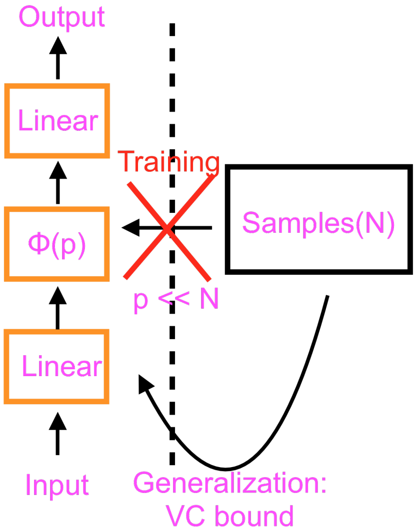
Then SGD on a two layer neural network (with m neurons and ReLU activation functions) finds a solution with generalization gap $\leq \varepsilon$ in time $poly(m/\varepsilon)$.

Generalization gap = average error on test data - average error on training data.

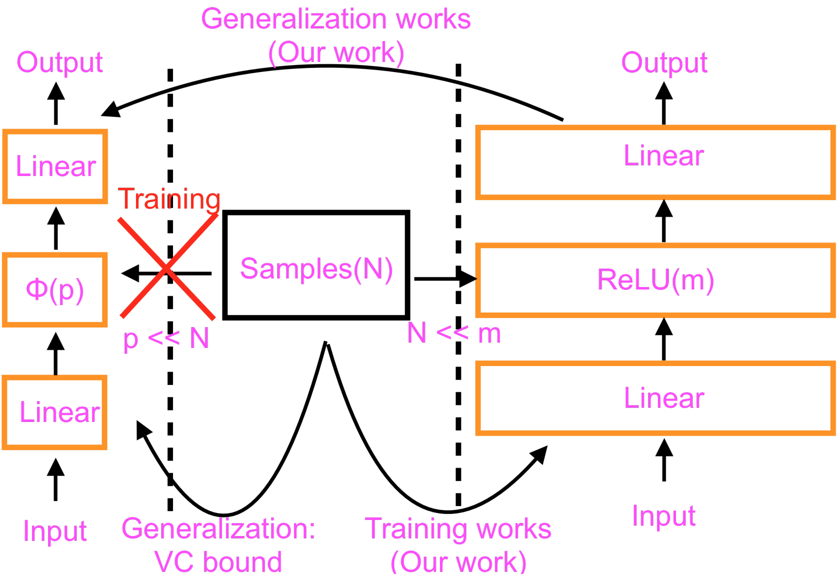
Picture of the Theorem



Picture of the Theorem



Picture of the Theorem



The main message

- Training a two layer network is NP-hard.

The main message

- Training a two layer network is NP-hard.
- But we are not optimizing a network up to optimal, we just want it to do as well as the best (much) smaller networks.

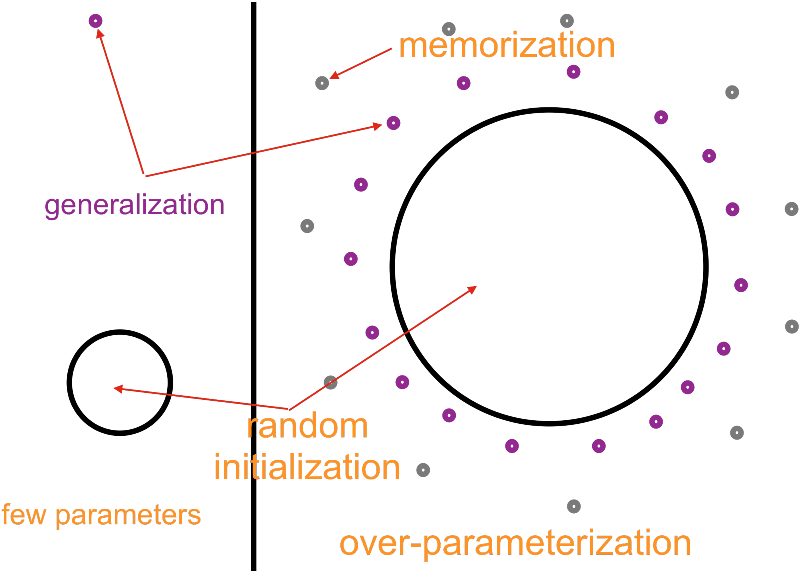
The main message

- Training a two layer network is NP-hard.
- But we are not optimizing a network up to optimal, we just want it to do as well as the best (much) smaller networks.
- We are training a network with $1M$ parameters to match the performance of a network with $1K$ parameters, and that is easy!

The main message

- Training a two layer network is NP-hard.
- But we are not optimizing a network up to optimal, we just want it to do as well as the best (much) smaller networks.
- We are training a network with $1M$ parameters to match the performance of a network with $1K$ parameters, and that is easy!
- This is what behind the principle of over-parameterization.

Intuition



memorization
generalization
random
initialization
few parameters
over-parameterization

Intuition

Generalization

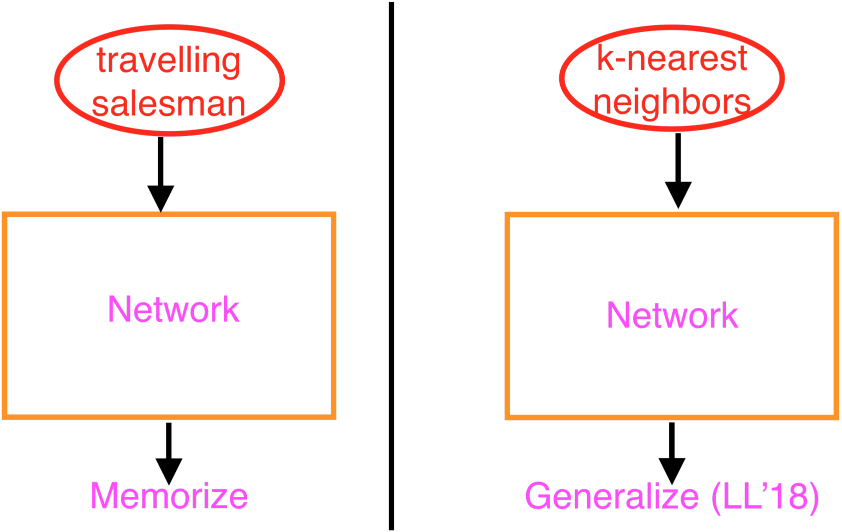
y = cos(x)

Simple!

Memorization

if x = 1, y = 0.54
if x = 2, y = -0.42
if x = 3, y = -0.99
if x = 4, y = -0.65
if x = 5, y = 0.28
if x = 6, y = 0.96
if x = 7, y = 0.75
if x = 8, y = -0.15
if x= 9, y = -0.91
if x = 10, y = -0.84
I gave up typing
those things…
too tired…

Intuition



Existence of small networks is somewhat necessary

Networks with more hidden layers?

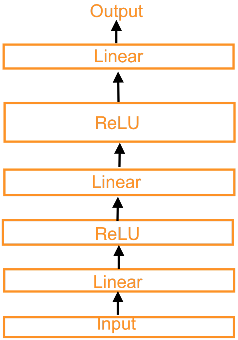
- Our theorem [ALL'18] also works for three layer networks (two hidden layers), and ([ALL'18] Corollary) single skip ResNet with multiple layers.

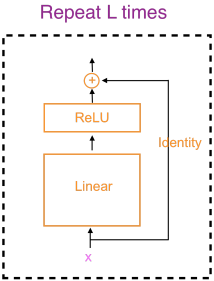
Networks with more hidden layers?

- Our theorem [ALL'18] also works for three layer networks (two hidden layers), and ([ALL'18] Corollary) single skip ResNet with multiple layers.
- Theoretical reasoning on multi-layer networks is much harder due to the non-convex interactions between the hidden layers.

Networks with more hidden layers?

- Our theorem [ALL'18] also works for three layer networks (two hidden layers), and ([ALL'18] Corollary) single skip ResNet with multiple layers.
- Theoretical reasoning on multi-layer networks is much harder due to the non-convex interactions between the hidden layers.





-

Key Messages

- Depth provably matters.

Key Messages

- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.

Key Messages

- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
  - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.

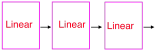
Key Messages

- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
  - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.
  - Can not be learnt by kernel methods efficiently.

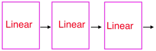
Key Messages

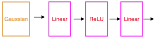
- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
    - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.
    - Can not be learnt by kernel methods efficiently.
- Prior works:

Key Messages

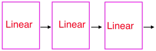
- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
  - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.
  - Can not be learnt by kernel methods efficiently.
- Prior works:
  - Linear networks. 

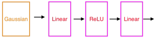
Key Messages

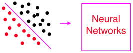
- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
  - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.
  - Can not be learnt by kernel methods efficiently.
- Prior works:
  - Linear networks. 

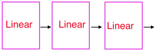
  - Networks on Gaussian inputs (tensor decomposition). 

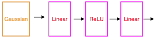
Key Messages

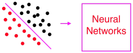
- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
  - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.
  - Can not be learnt by kernel methods efficiently.
- Prior works:
  - Linear networks. 

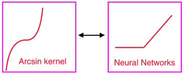
  - Networks on Gaussian inputs (tensor decomposition). 

  - Networks to learn linear separable data. 

- Depth provably matters.
- For the set of functions (super set of three layer networks with smooth activations) efficiently learnable by (over-paramterized) three layer networks.
  - The ONLY efficient learning algorithm we know is to train a (over-paramterized) three layer networks.
  - Can not be learnt by kernel methods efficiently.
- Prior works:
  - Linear networks. 

  - Networks on Gaussian inputs (tensor decomposition). 

  - Networks to learn linear separable data. 

  - Networks to learn functions that are learnable by arcsin kernel. 

Summary

- We have shown that:

Summary

- We have shown that:
    - SGD provably optimizes over-parameterization deep neural networks with ReLU activations, on the training data set.

Summary

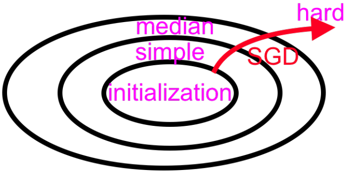
- We have shown that:
  - SGD provably optimizes over-parameterization deep neural networks with ReLU activations, on the training data set.
  - SGD also biases towards generalizable solutions on two/three layer networks and ResNet if the labels are structured.

Summary

- We have shown that:
  - SGD provably optimizes over-parameterization deep neural networks with ReLU activations, on the training data set.
  - SGD also biases towards generalizable solutions on two/three layer networks and ResNet if the labels are structured.
  - Reason: SGD biases towards simple solutions.

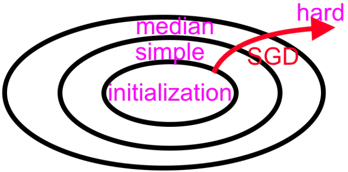
Summary

- We have shown that:
  - SGD provably optimizes over-parameterization deep neural networks with ReLU activations, on the training data set.
  - SGD also biases towards generalizable solutions on two/three layer networks and ResNet if the labels are structured.
  - Reason: SGD biases towards simple solutions.



  -

Summary

- We have shown that:
  - SGD provably optimizes over-parameterization deep neural networks with ReLU activations, on the training data set.
  - SGD also biases towards generalizable solutions on two/three layer networks and ResNet if the labels are structured.
  - Reason: SGD biases towards simple solutions.
  - 

- Question: Does this bias always lead to best generalization? Can we do better?

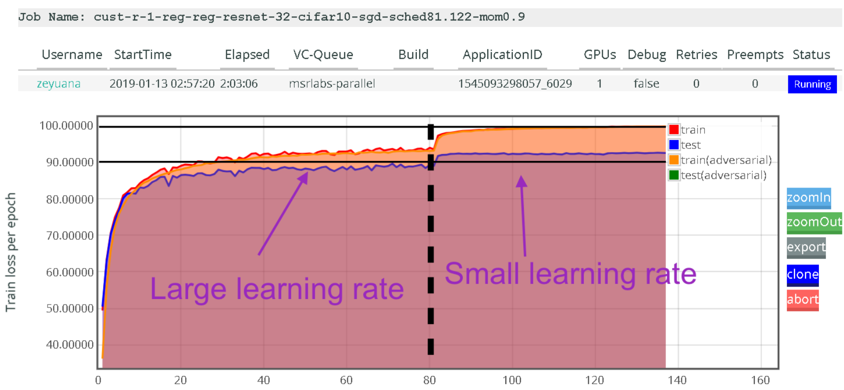
SGD with learning rate decay

- When we train a neural network (VGG, ResNet, Dense Net, etc. for image classification etc.)

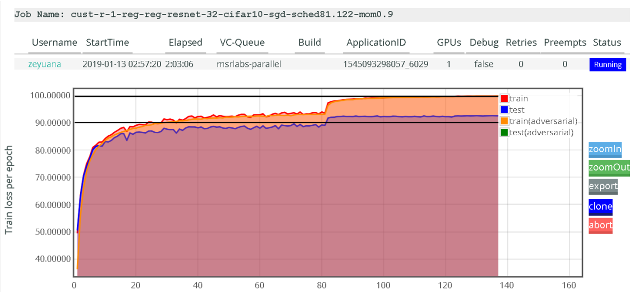
SGD with learning rate decay

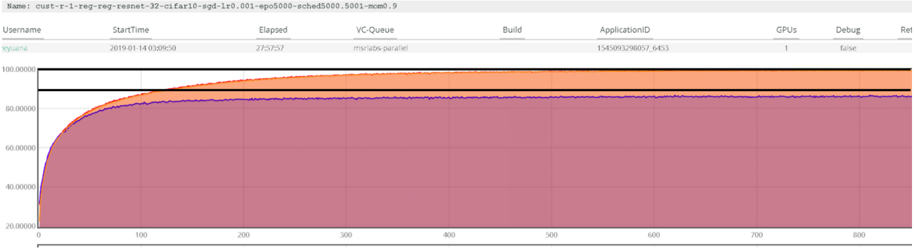
- When we train a neural network (VGG, ResNet, Dense Net, etc. for image classification etc.)
- It is better to use large learning rate first, then decay the learning rate.

SGD with learning rate decay

- When we train a neural network (VGG, ResNet, Dense Net, etc. for image classification etc.)
- It is better to use large learning rate first, then decay the learning rate.
- 

SGD with different learning rates





Principle of “Noisy computation”

- Training loss of using large learning rate + learning rate decay and using small learning rate are all close to zero.

Principle of "Noisy computation"

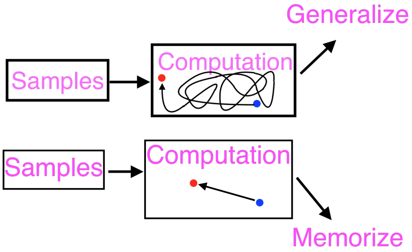
- Training loss of using large learning rate + learning rate decay and using small learning rate are all close to zero.
- But large learning rate generalizes better!

Principle of "Noisy computation"

- Training loss of using large learning rate + learning rate decay and using small learning rate are all close to zero.
- But large learning rate generalizes better!
- Principle: It is better to incorporate noise when training neural networks, as it improves generalization. (SGD + large learning rate = large noise)

Principle of "Noisy computation"

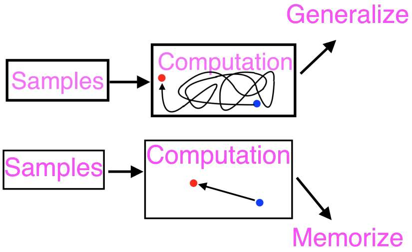
- Training loss of using large learning rate + learning rate decay and using small learning rate are all close to zero.
- But large learning rate generalizes better!
- Principle: It is better to incorporate noise when training neural networks, as it improves generalization. (SGD + large learning rate = large noise)



-

Principle of "Noisy computation"

- Training loss of using large learning rate + learning rate decay and using small learning rate are all close to zero.
- But large learning rate generalizes better!
- Principle: It is better to incorporate noise when training neural networks, as it improves generalization. (SGD + large learning rate = large noise)



-
- But why?

Our current work

- The principle of "noisy computation":

Our current work

- The principle of "noisy computation":
- On certain data sets, when training a neural network using SGD, large learning rate + learning rate decay provably generalizes better than small learning rate.

Example of the data set

- Texts labeled as happy:

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹
  - Feeling sorrowful ☹

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹
  - Feeling sorrowful ☹
  - Very unhappy.

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹
  - Feeling sorrowful ☹
  - Very unhappy.
  - Being wrecked ☹

Example of the data set

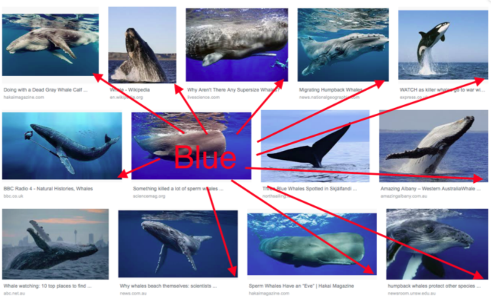
- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹
  - Feeling sorrowful ☹
  - Very unhappy.
  - Being wrecked ☹
  - I am Groot ☹

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹
  - Feeling sorrowful ☹
  - Very unhappy.
  - Being wrecked ☹
  - I am Groot ☹
  - ...

Example of the data set

- Texts labeled as happy:
  - I am so happy ☺
  - Feeling good ☺
  - Excited ☺
  - Today is a good day!
  - I am Groot ☺
  - ...
- Texts labeled as sad:
  - I am so unhappy ☹
  - Feeling sorrowful ☹
  - Very unhappy.
  - Being wrecked ☹
  - I am Groot ☹
  - ...



The intuition behind SGD

- What's wrong with SGD using small learning rate?

The intuition behind SGD

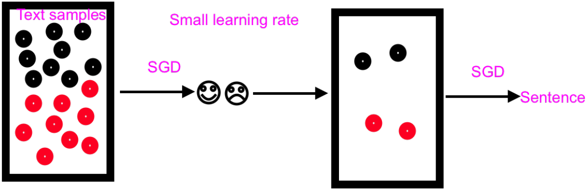
- What's wrong with SGD using small learning rate?
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.

The intuition behind SGD

- What's wrong with SGD using small learning rate?
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- SGD then uses very few examples without these symbols to learn the sentences.

The intuition behind SGD

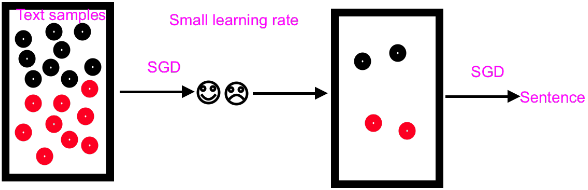
- What's wrong with SGD using small learning rate?
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- SGD then uses very few examples without these symbols to learn the sentences.



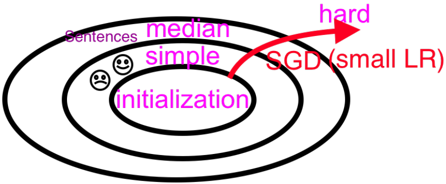
-

The intuition behind SGD

- What's wrong with SGD using small learning rate?
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- SGD then uses very few examples without these symbols to learn the sentences.



-



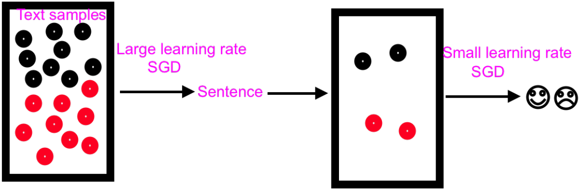
-

The intuition of SGD

- SGD with large learning rate further prevents memorization.

The intuition of SGD

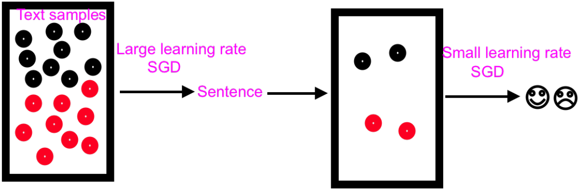
- SGD with large learning rate further prevents memorization.



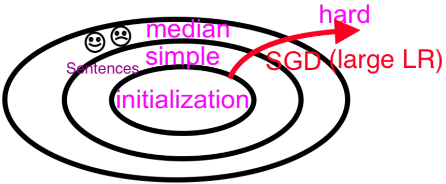
-

The intuition of SGD

- SGD with large learning rate further prevents memorization.



-



-

Memorization vs Generalization

- Memorization is not necessarily bad!

Memorization vs Generalization

- Memorization is not necessarily bad!
- I am Groot ☺ v.s. I am Groot ☹.

Memorization vs Generalization

- Memorization is not necessarily bad!
- I am Groot ☺ v.s. I am Groot ☹.
- 

Memorization vs Generalization

- Memorization is not necessarily bad!
- I am Groot ☺ v.s. I am Groot ☹.
- 

- The order matters: memorization after generalization.

The data set

- Data points in dimension $2p$: $x = (x_1, x_2)$ where $x_1, x_2 \in \mathbb{R}^p$.

The data set

- Data points in dimension $2p$: $x = (x_1, x_2)$ where $x_1, x_2 \in \mathbb{R}^p$.
- $x_1 \sim N(0, I)$ with probability $1 - \tau$, the label of x is $y(x) = 1_{\langle w,x\rangle \geq 0}$, where w is a vector in $\mathbb{R}^p$, otherwise $x_1 = 0$ and $y(x)$ is random ("sentence" signal).

The data set

- Data points in dimension $2p$: $x = (x_1, x_2)$ where $x_1, x_2 \in \mathbb{R}^p$.
- $x_1 \sim N(0, I)$ with probability $1 - \tau$, the label of x is $y(x) = 1_{\langle w,x\rangle \geq 0}$, where w is a vector in $\mathbb{R}^p$, otherwise $x_1 = 0$ and $y(x)$ is random ("sentence" signal).
- With probability $1 - \tau$ (τ is small),

$$x_2 = \begin{cases} z & \text{if } y(x) = 0; \\ z \pm \delta & \text{if } y(x) = 1. \end{cases}$$

("special symbol" signal)

The data set

- Data points in dimension $2p$: $x = (x_1, x_2)$ where $x_1, x_2 \in \mathbb{R}^p$.
- $x_1 \sim N(0, I)$ with probability $1 - \tau$, the label of x is $y(x) = 1_{\langle w, x \rangle \geq 0}$, where w is a vector in $\mathbb{R}^p$, otherwise $x_1 = 0$ and $y(x)$ is random ("sentence" signal).
- With probability $1 - \tau$ (τ is small),

$$x_2 = \begin{cases} z & \text{if } y(x) = 0; \\ z \pm \delta & \text{if } y(x) = 1. \end{cases}$$

  ("special symbol" signal)
- Remark: $\langle w, x \rangle$ can be replaced by two layer networks.

Our Theorem

Our Theorem

Theorem ((LM'19) Sketched)

Given $N \approx p/\tau$ $(\tau \ll 1)$ *training examples, at training error (two layer over-parameterized neural network with ReLU activation)* τ^2 *for the cross entropy loss:*

Our Theorem

Theorem ((LM'19) Sketched)

Given $N \approx p/\tau$ ($\tau \ll 1$) training examples, at training error (two layer over-parameterized neural network with ReLU activation) τ^2 for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $\leq \tau^{1+\Omega(1)}$.*

Our Theorem

Theorem ((LM'19) Sketched)

Given $N \approx p/\tau$ ($\tau \ll 1$) training examples, at training error (two layer over-parameterized neural network with ReLU activation) τ^2 for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $\leq \tau^{1+\Omega(1)}$.*
2. *SGD with small learning rate has generalization error $\approx \tau$.*

Our Theorem

Theorem ((LM'19) Sketched)

Given $N \approx p/\tau$ ($\tau \ll 1$) training examples, at training error (two layer over-parameterized neural network with ReLU activation) τ^2 for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $\leq \tau^{1+\Omega(1)}$.*
2. *SGD with small learning rate has generalization error $\approx \tau$.*

- When $N \gg p$, they both generalize (as shown in the principle of over-parameterization), but SGD with large learning rate + learning rate decay generalizes even better.

Intuition

Generalization:

```
def CMU_talk_example(x):
    x = x^2
    x = x + 10
    x = x*3
    x = x^(1/3)
    x = x - 3
    return x
```

Memorization:

```
def CMU_talk_example(x):
    if x = 1, return 0.2
    if x = 2, return 0.48
    if x = 3, return 0.84
```

SGD with
small learning rate:
I like this one!

Intuition

Generalization:

```
def CMU_talk_example(x):
    x = x^2.03
    x = x + 10.04
    x = x*3.02
    x = x^(1/3.01)
    x = x - 3.05
    return x
```

SGD with
large learning rate
(noisy)

almost correct

Memorization:

```
def CMU_talk_example(x):
    if x = 1.02 , return 0.2
    if x = 2.04,return 0.48
    if x = 3.01 , return 0.84
```

?????

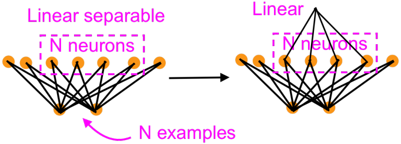
x = 1

Principle of "noisy computation"

- Memorization in neural network: when $m \gg N$, by chance there will be N-neurons in the network that maps all data to linear independent positions. (Geometry of ReLU Lemma in [LL'18, ALS'18(a,b)])

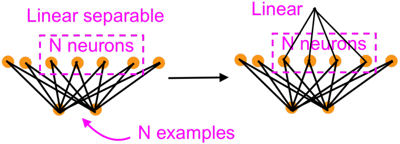
Principle of "noisy computation"

- Memorization in neural network: when $m \gg N$, by chance there will be N-neurons in the network that maps all data to linear independent positions. (Geometry of ReLU Lemma in [LL'18, ALS'18(a,b)])
- Then SGD just lazily finds a linear separator.

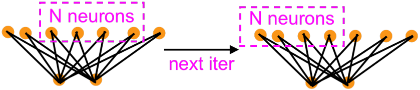


Principle of "noisy computation"

- Memorization in neural network: when $m \gg N$, by chance there will be N-neurons in the network that maps all data to linear independent positions. (Geometry of ReLU Lemma in [LL'18, ALS'18(a,b)])
- Then SGD just lazily finds a linear separator.

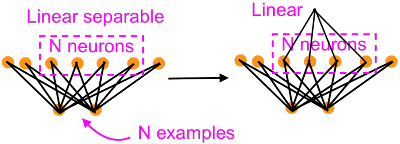


- SGD with large noise will make this set of N-neurons changing rapidly, and destroy the linear separator to prevent memorization.

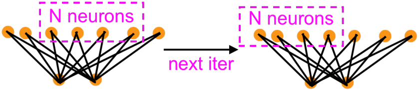


Principle of "noisy computation"

- Memorization in neural network: when $m \gg N$, by chance there will be N-neurons in the network that maps all data to linear independent positions. (Geometry of ReLU Lemma in [LL'18, ALS'18(a,b)])
- Then SGD just lazily finds a linear separator.



- SGD with large noise will make this set of N-neurons changing rapidly, and destroy the linear separator to prevent memorization.



- This is what behind the principle of "noisy computation".

Key Messages

- To the best of my knowledge, this is the ONLY work, that gives both efficiently achievable lower and upper bounds that large learning rate + learning rate decay improves generalization upon small learning rates.

Key Messages

- To the best of my knowledge, this is the ONLY work, that gives both efficiently achievable lower and upper bounds that large learning rate + learning rate decay improves generalization upon small learning rates.
- Even when the training losses of both methods are the same.

Key Messages

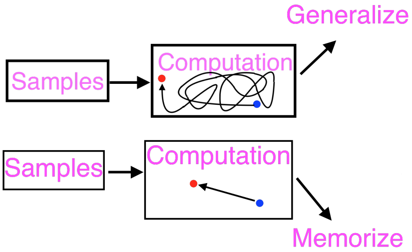
- To the best of my knowledge, this is the ONLY work, that gives both efficiently achievable lower and upper bounds that large learning rate + learning rate decay improves generalization upon small learning rates.
- Even when the training losses of both methods are the same.
- Prior works: small learning rate improves generalization. (More stable, lower training error etc.)

Key Messages

- To the best of my knowledge, this is the ONLY work, that gives both efficiently achievable lower and upper bounds that large learning rate + learning rate decay improves generalization upon small learning rates.
- Even when the training losses of both methods are the same.
- Prior works: small learning rate improves generalization. (More stable, lower training error etc.)
- This work: principle of "noisy computation".

Key Messages

- To the best of my knowledge, this is the ONLY work, that gives both efficiently achievable lower and upper bounds that large learning rate + learning rate decay improves generalization upon small learning rates.
- Even when the training losses of both methods are the same.
- Prior works: small learning rate improves generalization. (More stable, lower training error etc.)
- This work: principle of "noisy computation".



-

Summary of my works

- I have described 10 percent of my works.

Summary of my works

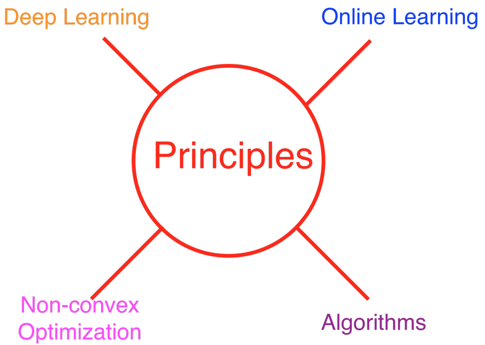
- I have described 10 percent of my works.
- But you might already see the rest from Generalization.

Summary of my works

- I have described 10 percent of my works.
- But you might already see the rest from Generalization.
- I work on the principled methods for machine learning. In the following four areas:

Summary of my works

- I have described 10 percent of my works.
- But you might already see the rest from Generalization.
- I work on the principled methods for machine learning. In the following four areas:



-

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]

Recurrent neural networks: Training/Generalization [ALS'18, AL'18]

ResNet with/without over-para [LY'17, AL'19]

Learning rate[LM'19]

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]

Recurrent neural networks: Training/Generalization [ALS'18, AL'18]

ResNet with/without over-para [LY'17, AL'19]

Learning rate[LM'19]

This talk

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]

Recurrent neural networks: Training/Generalization [ALS'18, AL'18]

ResNet with/without over-para [LY'17, AL'19]

Learning rate[LM'19]

Online Learning

Convex bandit [HL'17]

Chasing convex body[BLLS'18]

First order/second order regret bounds for bandit/contextual bandit [ABL'18, BCL'18, BLLW' 18]

Distributed bandit[BLP'19]

Bandit with approximation oracles [HHLL'18]

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]

Recurrent neural networks: Training/Generalization [ALS'18, AL'18]

ResNet with/without over-para [LY'17, AL'19]

Learning rate[LM'19]

Online Learning

Convex bandit [HL'17]

Chasing convex body[BLLS'18]

First order/second order regret bounds for bandit/contextual bandit [ABL'18, BCL'18, BLLW' 18]

Distributed bandit[BLP'19]

Bandit with approximation oracles [HHLL'18]

Open problem since 2004

Open problem since 1991

Main open problem in contextual bandit (raised in COLT open problem session)

Best student paper(ALT 18)

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]
Recurrent neural networks: Training/Generalization [ALS'18, AL'18]
ResNet with/without over-para [LY'17, AL'19]
Learning rate[LM'19]

Online Learning

Convex bandit [HL'17]
Chasing convex body[BLLS'18]
First order/second order regret bounds for bandit/contextual bandit [ABL'18, BCL'18, BLLW' 18]
Distributed bandit[BLP'19]
Bandit with approximation oracles [HHLL'18]

Non-convex opt.

Topic model/Matrix completion
Matrix sensing/Tensor decomposition
[LL'17, LLR'16(ab), LMZ'18, LZ'19]
Mixture of regressions [LL'18]
Graphical Models [LR'17]
Escaping saddle points/approximate convex optimization[AL'18, LR'16]
Experiments design[ALW'17]

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]

Recurrent neural networks: Training/Generalization [ALS'18, AL'18]

ResNet with/without over-para [LY'17, AL'19]

Learning rate[LM'19]

Online Learning

Convex bandit [HL'17]

Chasing convex body[BLLS'18]

First order/second order regret bounds for bandit/contextual bandit [ABL'18, BCL'18, BLLW' 18]

Distributed bandit[BLP'19]

Bandit with approximation oracles [HHLL'18]

Non-convex opt.

Topic model/Matrix completion

Matrix sensing/Tensor decomposition [LL'17, LLR'16, LLR'16, LMZ'18, LZ'19]

Mixture of regressions [LL'18]

Graphical Models [LR'17]

Escaping saddle points/approximate convex optimization[AL'18, LR'16]

Experiments design[ALW'17]

With practical improvements

Best paper(COLT 18)

In each area

Deep Learning

Over-para: Training/Generalization [LL'18, ALS'18, ALL'18]
Recurrent neural networks: Training/Generalization [ALS'18, AL'18]
ResNet with/without over-para [LY'17, AL'19]
Learning rate[LM'19]

Online Learning

Convex bandit [HL'17]
Chasing convex body[BLLS'18]
First order/second order regret bounds for bandit/contextual bandit [ABL'18, BCL'18, BLLW' 18]
Distributed bandit[BLP'19]
Bandit with approximation oracles [HHLL'18]

Non-convex opt.

Topic model/Matrix completion
Matrix sensing/Tensor decomposition
[LL'17, LLR'16, LLR'16 , LMZ'18, LZ'19]
Mixture of regressions [LL'18]
Graphical Models [LR'17]
Escaping saddle points/approximate convex optimization[AL'18, LR'16]
Experiments design[ALW'17]

Algorithms

SVD(PCA), CCA, PCR, Online PCA [AL'17(abcd)]
Smoothed analysis[LS'18(ab)]
Higher order acceleration/Distributed Convex optimization[BJLLS'19(ab)]
Matrix/Operator Scaling/Orbit Closure Testing[ALOW'17, AGLOW'18]
Lp regression[BCLL'18]

In each area

Deep Learning
Over-para: Training/Generalization
[LL'18, ALS'18, ALL'18]
Recurrent neural networks:
Training/Generalization [ALS'18, AL'18]
ResNet with/without over-para
[LY'17, AL'19]
Learning rate[LM'19]

Online Learning
Convex bandit [HL'17]
Chasing convex body[BLLS'18]
First order/second order regret bounds
for bandit/contextual bandit
[ABL'18, BCL'18, BLLW' 18]
Distributed bandit[BLP'19]
Bandit with approximation oracles
[HHLL'18]

One step towards
derandomizing PIT
(complexity)

Non-convex opt.
Topic model/Matrix completion
Matrix sensing/Tensor decomposition
[LL'17, LLR'16, LLR'16, LMZ'18, LZ'19]
Mixture of regressions [LL'18]
Graphical Models [LR'17]
Escaping saddle points/approximate
convex optimization[AL'18, LR'16]
Experiments design[ALW'17]

Algorithms
$\boxed{\text{SVD}}$(PCA), CCA, PCR, Online PCA
[AL'17(abcd)]
Smoothed analysis[LS'18(ab)]
Higher order acceleration/Distributed
Convex optimization[BJLLS'19(ab)]
Matrix/Operator Scaling/Orbit
Closure Testing[ALOW'17,
AGLOW'18]
Lp regression[BCLL'18]

Principles

Models

Over-parameterization

Algorithms

Inductive bias of SGD

Noisy computation

Connected by principles

Principles

Models

Over-parameterization

Under-parameterization

Re-parameterization

Ensemble(Multiple Objectives)

Random initialization

Algorithms

Inductive bias of SGD

Noisy computation

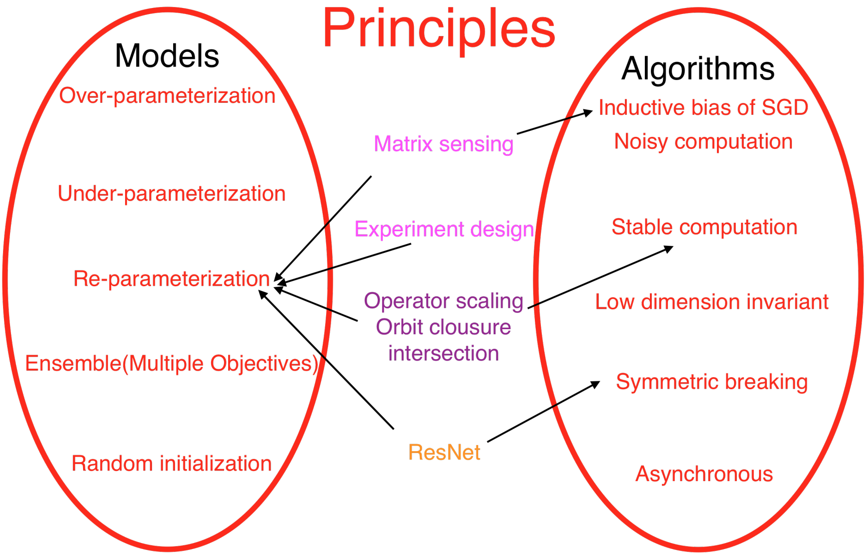
Stable computation

Low dimension invariant

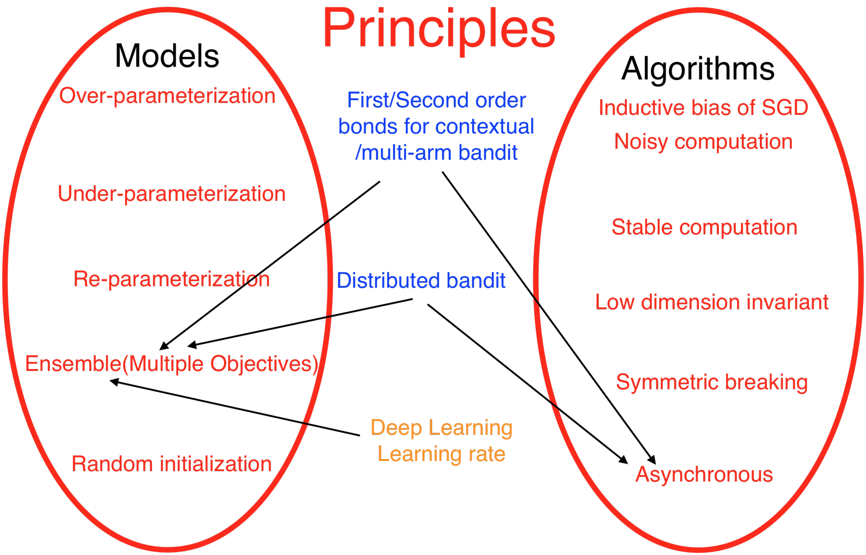
Symmetric breaking

Asynchronous

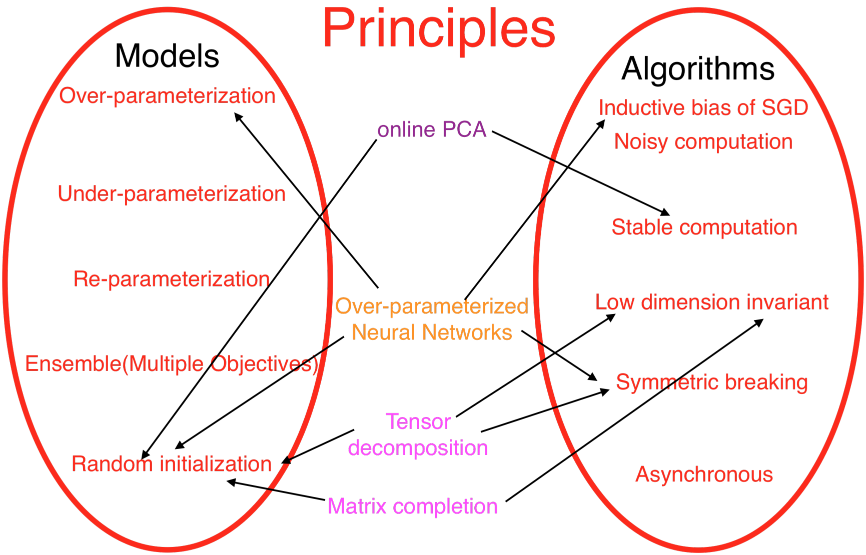
Connected by principles



Connected by principles



Connected by principles

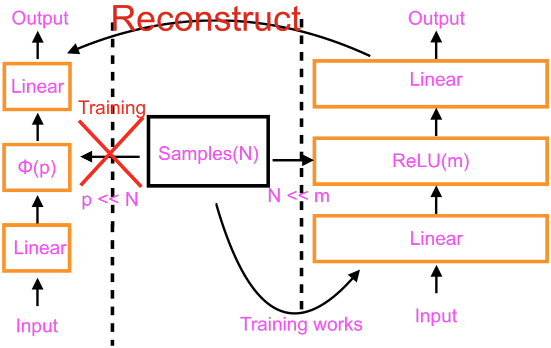


Future directions

- First understand the principles, then implement those principles more principally.

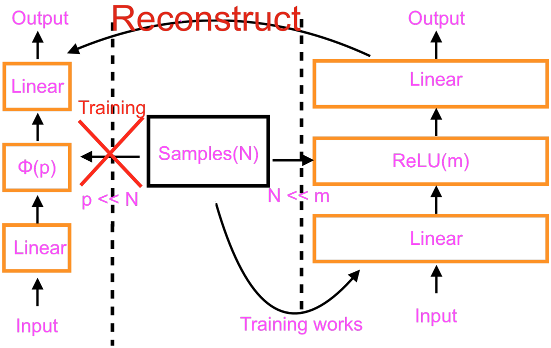
Future directions

- First understand the principles, then implement those principles more principally.
- Deep learning: reconstruct the small networks (Core features).



Future directions

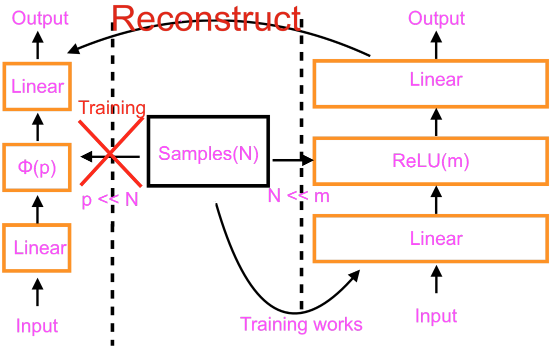
- First understand the principles, then implement those principles more principally.
- Deep learning: reconstruct the small networks (Core features).



- Deep learning: incorporate noise in a more controllable way. (SVRG + noise?)

Future directions

- First understand the principles, then implement those principles more principally.
- Deep learning: reconstruct the small networks (Core features).



- Deep learning: incorporate noise in a more controllable way. (SVRG + noise?)
- Much more exciting directions.

Yelp: Local Food & Services

Restaurant & Delivery Finder

4.1 ★★★★☆

4.42K Ratings

#10

Travel

12+

Age

What's New

Version History

Version 12.27.0

2d ago

We apologize to anyone who had problems with the app this week. We trained a neural net to eliminate all the bugs in the app and it deleted everything. We had to roll everything back. To be fair, we were 100% bug-free... briefly.