Towards a deeper understanding of deep learning

Yuanzhi Li

Stanford University

January 13, 2019

A bit of histroy

- 13.772 billion years BC to 1940s

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...

A bit of histroy

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

A bit of histroy

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

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...
- 1950s to 2010s:
  - The age of perceptrons.
    - Single layer learning models: SVM, kernel SVM, products of experts...

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...
- 1950s to 2010s:
  - The age of perceptrons.
    - Single layer learning models: SVM, kernel SVM, products of experts...
  - There are lots of theory for them: theory of kernels (representation power), convex optimization (training time bound), VC theory (sample complexity and generalization).

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...
- 1950s to 2010s:
  - The age of perceptrons.
    - Single layer learning models: SVM, kernel SVM, products of experts...
  - There are lots of theory for them: theory of kernels (representation power), convex optimization (training time bound), VC theory (sample complexity and generalization).
- In the past few years:

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...
- 1950s to 2010s:
  - The age of perceptrons.
    - Single layer learning models: SVM, kernel SVM, products of experts...
  - There are lots of theory for them: theory of kernels (representation power), convex optimization (training time bound), VC theory (sample complexity and generalization).
- In the past few years:
  - Neural networks (Deep learning).

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...
- 1950s to 2010s:
  - The age of perceptrons.
    - Single layer learning models: SVM, kernel SVM, products of experts...
  - There are lots of theory for them: theory of kernels (representation power), convex optimization (training time bound), VC theory (sample complexity and generalization).
- In the past few years:
  - Neural networks (Deep learning).
    - Works extremely well in practice, tons of applications.

A bit of histroy

- 13.772 billion years BC to 1940s
  - Nothing happens...
- 1950s to 2010s:
  - The age of perceptrons.
    - Single layer learning models: SVM, kernel SVM, products of experts...
  - There are lots of theory for them: theory of kernels (representation power), convex optimization (training time bound), VC theory (sample complexity and generalization).
- In the past few years:
  - Neural networks (Deep learning).
    - Works extremely well in practice, tons of applications.
    - (Please imagine some fancy pictures here)

Neural Networks

- Multi-layer models.

Neural Networks

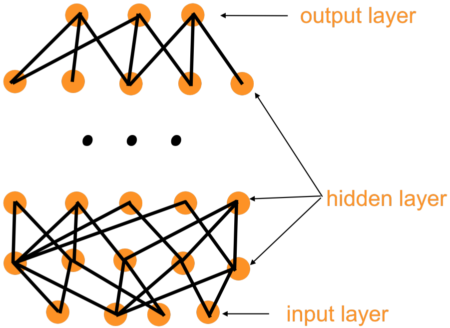
- Multi-layer models.
  - Input layer, output layer, and hidden layers.

Neural Networks

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

Neural Networks

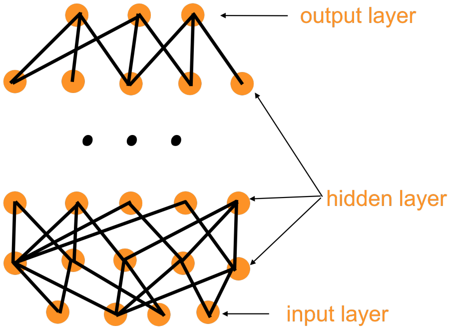
- Multi-layer models.
  - Input layer, output layer, and hidden layers.
    - Each hidden layer is associated with activation function(s).



  -

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 in practice

- But why? How do I prove that my neural networks work?

Neural Networks work in practice

- But why? How do I prove that my neural networks work?
- In the past years, people have reached a satisfying answer to this question:

Neural Networks work in practice

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

Neural Networks work in practice

- But why? How do I prove that my neural networks work?
- In the past years, people have reached a satisfying answer to this question:
  - Stop asking about it!
  - Just deal with the fact that neural networks work in practice although we can not fully prove it.

Neural Networks work in practice

- But why? How do I prove that my neural networks work?
- In the past years, people have reached a satisfying answer to this question:
  - Stop asking about it!
  - Just deal with the fact that 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).
    - In practice, we have millions.

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.
    - Given any practical neural network, we can always come up with examples where we can't learn the network on them. (worst case examples)

Beyond worst case?

- In practice, the data is not worse case.

Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.

Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.
  - A two dimension circle is something of form $x^2 + y^2 = R^2$. Great!

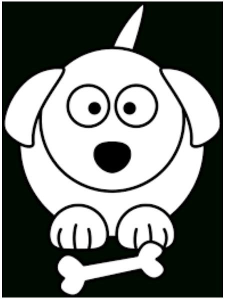
Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.
    - A two dimension circle is something of form $x^2 + y^2 = R^2$. Great!
    - A two dimension picture of a dog is of form?

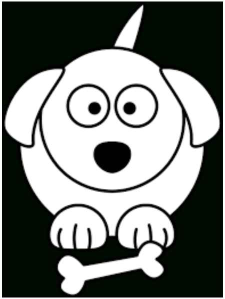
Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.
    - A two dimension circle is something of form $x^2 + y^2 = R^2$. Great!
    - A two dimension picture of a dog is of form?
        - Consists of symbols 'd', 'o', and 'g', and exclusively in that order?

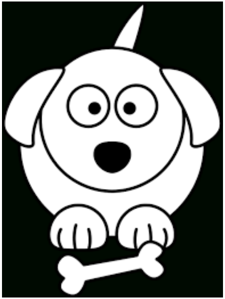
Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.
  - A two dimension circle is something of form $x^2 + y^2 = R^2$. Great!
  - A two dimension picture of a dog is of form?
    - Consists of symbols 'd', 'o', and 'g', and exclusively in that order?
    - 

Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.
    - A two dimension circle is something of form $x^2 + y^2 = R^2$. Great!
    - A two dimension picture of a dog is of form?
        - Consists of symbols 'd', 'o', and 'g', and exclusively in that order?
        - 
        - No clean model.

Beyond worst case?

- In practice, the data is not worse case.
- But it is so hard to model.
  - A two dimension circle is something of form $x^2 + y^2 = R^2$. Great!
  - A two dimension picture of a dog is of form?
    - Consists of symbols 'd', 'o', and 'g', and exclusively in that order?
    - 
    - No clean model.
- So beyond worst case is also hard...

And even harder

- Yet there is another reason:

And even harder

- Yet there is another reason:
- In theory, we usually solve a rather complicated problem with a rather complicated algorithm.

And even harder

- Yet there is another reason:
- In theory, we usually solve a rather complicated problem with a rather complicated algorithm.
    - So we show good respect to the difficulty.

And even harder

- Yet there is another reason:
- In theory, we usually solve a rather complicated problem with a rather complicated algorithm.
  - So we show good respect to the difficulty.
- In neural networks, we optimize super complicated neural networks with an extremely simple algorithm: SGD.

And even harder

- Yet there is another reason:
- In theory, we usually solve a rather complicated problem with a rather complicated algorithm.
  - So we show good respect to the difficulty.
- In neural networks, we optimize super complicated neural networks with an extremely simple algorithm: SGD.
  - Starting from W_0 at random (often Gaussian).

And even harder

- Yet there is another reason:
- In theory, we usually solve a rather complicated problem with a rather complicated algorithm.
  - So we show good respect to the difficulty.
- In neural networks, we optimize super complicated neural networks with an extremely 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 theory, we usually solve a rather complicated problem with a rather complicated algorithm.
  - So we show good respect to the difficulty.
- In neural networks, we optimize super complicated neural networks with an extremely 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.

And even harder

- Yet there is another reason:
- In theory, we usually solve a rather complicated problem with a rather complicated algorithm.
  - So we show good respect to the difficulty.
- In neural networks, we optimize super complicated neural networks with an extremely 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.
- Make the analysis even harder.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.
- This is the end of the talk.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.
- This is the end of the talk.
- If we accept the fact neural networks might not have a complete theorem.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.
- This is the end of the talk.
- If we accept the fact neural networks might not have a complete theorem.
- But of course we won't.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.
- This is the end of the talk.
- If we accept the fact neural networks might not have a complete theorem.
- But of course we won't.
  - Spirit: If we try to build up the theory, we might not be successful immediately.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.
- This is the end of the talk.
- If we accept the fact neural networks might not have a complete theorem.
- But of course we won't.
    - Spirit: If we try to build up the theory, we might not be successful immediately.
    - But if we don't try, we will feel very relaxed immediately.

The end

- Given the above examples, we see that theory for deep learning is indeed quite difficult.
- This is the end of the talk.
- If we accept the fact neural networks might not have a complete theorem.
- But of course we won't.
  - Spirit: If we try to build up the theory, we might not be successful immediately.
  - But if we don't try, we will feel very relaxed immediately.
  - .... we will never have a chance to succeed.

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:
  - Linearization principle: focus on simpler models (linear networks etc.)

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:
    - Linearization principle: focus on simpler models (linear networks etc.)
    - Redesign of the activation functions (exponential, quadratic, special Hermite polynomial etc.)

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:
  - Linearization principle: focus on simpler models (linear networks etc.)
  - Redesign of the activation functions (exponential, quadratic, special Hermite polynomial etc.)
  - Use different algorithms (tensor decomposition etc.)

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:
  - Linearization principle: focus on simpler models (linear networks etc.)
  - Redesign of the activation functions (exponential, quadratic, special Hermite polynomial etc.)
  - Use different algorithms (tensor decomposition etc.)
  - (Please imagine some fancy papers here)

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:
  - Linearization principle: focus on simpler models (linear networks etc.)
  - Redesign of the activation functions (exponential, quadratic, special Hermite polynomial etc.)
  - Use different algorithms (tensor decomposition etc.)
  - (Please imagine some fancy papers here)
- There are lots of amazing (marvelous, thrilling, remarkable, fantastic, magnificent, striking or spectacular) results along these lines and they are really great and impactful...

In past few years

- There are many progress in the theory of deep learning, moving us closer to getting a full explanation.
- Different types of work:
  - Linearization principle: focus on simpler models (linear networks etc.)
  - Redesign of the activation functions (exponential, quadratic, special Hermite polynomial etc.)
  - Use different algorithms (tensor decomposition etc.)
  - (Please imagine some fancy papers here)
- There are lots of amazing (marvelous, thrilling, remarkable, fantastic, magnificent, striking or spectacular) results along these lines and they are really great and impactful...
- But I am going to use a different approach.

In this talk

- I will focus on (multi-layer) neural networks with:

In this talk

- I will focus on (multi-layer) neural networks with:
  - ReLU activation functions.

In this talk

- I will focus on (multi-layer) neural networks with:
  - ReLU activation functions.
- Training algorithm that is:

In this talk

- I will focus on (multi-layer) neural networks with:
  - ReLU activation functions.
- Training algorithm that is:
  - SGD with Gaussian random initialization.

In this talk

- I will focus on (multi-layer) neural networks with:
    - ReLU activation functions.
- Training algorithm that is:
    - SGD with Gaussian random initialization.
- I will do theory with them.

In this talk

- I will focus on (multi-layer) neural networks with:
  - ReLU activation functions.
- Training algorithm that is:
  - SGD with Gaussian random initialization.
- I will do theory with them.
- How?

Simple question before going on

- Some simple questions for you:

Simple question before going on

- Some simple questions for you:
- (1). My network finds the global optimal on the training set, but it runs really slow. What should I do?

Simple question before going on

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

Simple question before going on

- Some simple questions for you:
- (1). My network finds the global optimal on the training set, but it runs really slow. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (Deep learning): You should use a larger network.

Simple question before going on

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

Simple question before going on

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

Simple question before going on

- Some simple questions for you:
- (1). My network finds the global optimal on the training set, but it runs really slow. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (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.
  - (Deep learning): You should use a larger network.

Simple question before going on

- Some simple questions for you:
- (1). My network finds the global optimal on the training set, but it runs really slow. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (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.
  - (Deep learning): You should use a larger network.
- (3). My network finds a good solution, but I can't prove it. What should I do?

Simple question before going on

- Some simple questions for you:
- (1). My network finds the global optimal on the training set, but it runs really slow. What should I do?
  - (A person who just took the convex optimization course): Make the total number of parameters smaller so it runs faster.
  - (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.
  - (Deep learning): You should use a larger network.
- (3). My network finds a good solution, 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.

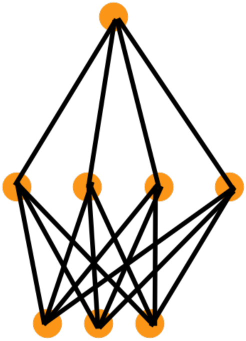
- By building up a network with (much more) parameters than the total number of training examples.
- Improves both the training and generalization.

Principle of over-parameterization.

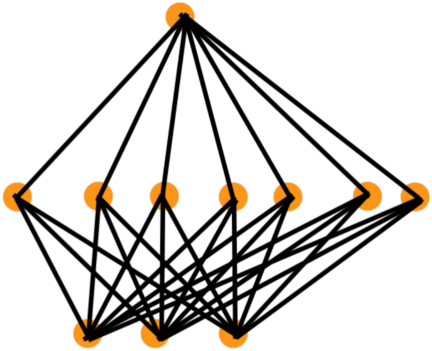
- By building up a network with (much more) parameters than the total number of training examples.
- Improves both the training and generalization.
- And it improves the theory.

Folklore example for training.

teacher
100 neurons



learner
1000 neurons



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)

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.
- 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 $x_1, \ldots, x_N$ with labels $y_1, \ldots, y_N$, 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 optimal of the training objective in time poly(m/ε).

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

Close look at the theorem:

- Only assumption: N different training examples, so no two identical training examples.

Close look at the theorem:

- Only assumption: N different training examples, so no two identical training examples.
- $m \geq \mathrm{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.

Close look at the theorem:

- Only assumption: N different training examples, so no two identical training examples.
- $m \geq \mathrm{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Training labels can be random.

Close look at the theorem:

- Only assumption: N different training examples, so no two identical training examples.
- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Training labels can be random.
  - From capacity view: Obviously there exists a network that fits the training data.

Close look at the theorem:

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

Close look at the theorem:

- Only assumption: N different training examples, so no two identical training examples.
- $m \geq \text{poly}(N \log(1/\varepsilon))$ implies good fitting on the training data set.
- Training labels can be random.
  - From capacity view: Obviously there exists a network that fits the training data.
  - From optimization view: How can SGD finds such fitting? The training objective is not only non-convex, but non-smooth as well due to ReLU.
  - Not a trivial theorem, but much simpler than the next question.

Training is ok with over-parameterization

- The harder question is:

Training is ok with over-parameterization

- The harder question is:
  - What about generalization?

Training is ok with over-parameterization

- The harder question is:
    - What about generalization?
- The empirical example (ZBHR'16), that is also proved by our theorem.

Training is ok with over-parameterization

- The harder question is:
  - 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

- The harder question is:
  - 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

- The harder question is:
  - 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

- The harder question is:
  - 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 SGD it 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 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 biases towards generalizable solutions instead of the solutions that simply memorize the training data.
  - We are going to prove it for certain neural networks.

Labels are not random?

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

$$F^* = (f_1^*, \cdots, f_k^*)$$
$$\text{each } f_r^*(x) = \sum_{i=1}^{p} a_{r,i}^* \phi_i(\langle w_i^*, x \rangle)$$

k is the output dimension, ϕ_i are smooth activation functions (such as $\sin, \cos, \exp$ and low degree polynomials), w_i^* are unit vectors, $|a_{r,i}^*| \leq 1$ and $\|x\|_2 \leq 1$.

Labels are not random?

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

$$F^* = (f_1^*, \cdots, f_k^*)$$
$$\text{each } f_r^*(x) = \sum_{i=1}^{p} a_{r,i}^* \phi_i(\langle w_i^*, x \rangle)$$

  k is the output dimension, ϕ_i are smooth activation functions (such as $\sin, \cos, \exp$ and low degree polynomials), w_i^* are unit vectors, $|a_{r,i}^*| \leq 1$ and $\|x\|_2 \leq 1$.
- Such that the average loss of network F^* on the training data set is $\leq \varepsilon$.

Labels are not random?

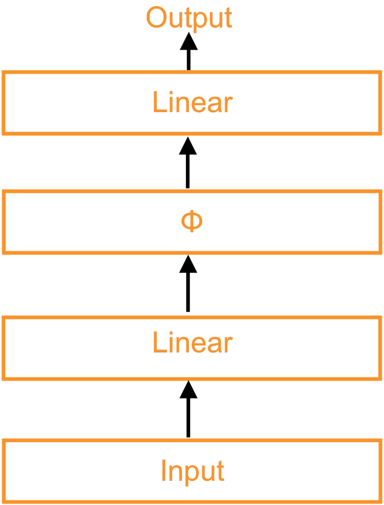
- Assume that the labels are realizable by a simple neural network:

$$F^* = (f_1^*, \cdots, f_k^*)$$

$$\text{each } f_r^*(x) = \sum_{i=1}^{p} a_{r,i}^* \phi_i(\langle w_i^*, x \rangle)$$

  k is the output dimension, ϕ_i are smooth activation functions (such as $\sin, \cos, \exp$ and low degree polynomials), w_i^* are unit vectors, $|a_{r,i}^*| \le 1$ and $\|x\|_2 \le 1$.
- Such that the average loss of network F^* on the training data set is $\le \varepsilon$.
- Two-layer network with p hidden neurons.

Picture of the Network.



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 (one hidden layer) fully-connected neural networks (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.

Close look at the theorem

- Get training error $\leq \varepsilon$:

Close look at the theorem

- Get training error $\leq \varepsilon$:
  - Needs $m \geq poly(N \log(1/\varepsilon))$.

Close look at the theorem

- Get training error $\leq \varepsilon$:
  - Needs $m \geq poly(N \log(1/\varepsilon))$.
- Get generalization gap $\leq \varepsilon$:

Close look at the theorem

- Get training error $\leq \varepsilon$:
  - Needs $m \geq poly(N \log(1/\varepsilon))$.
- Get generalization gap $\leq \varepsilon$:
  - Needs $N \geq poly(1/\varepsilon) \times poly(\log(m))$.

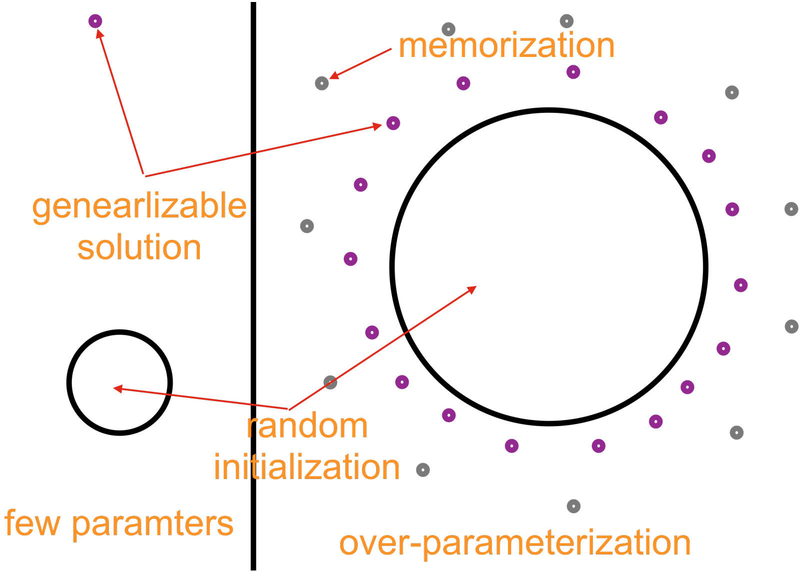
Close look at the theorem

- Get training error $\leq \varepsilon$:
  - Needs $m \geq poly(N \log(1/\varepsilon))$.
- Get generalization gap $\leq \varepsilon$:
  - Needs $N \geq poly(1/\varepsilon) \times poly(\log(m))$.
- We can obtain error on the test data set $\leq 2\varepsilon$ with up to sup-exponential over-parameterization.

Close look at the theorem

- Get training error $\leq \varepsilon$:
  - Needs $m \geq poly(N \log(1/\varepsilon))$.
- Get generalization gap $\leq \varepsilon$:
  - Needs $N \geq poly(1/\varepsilon) \times poly(\log(m))$.
- We can obtain error on the test data set $\leq 2\varepsilon$ with up to sup-exponential over-parameterization.
- The theorem also applies to convolution nets.

Picture of intuition



But wait for a second...

- You just mentioned that learning two-layer network with three neurons is NP-hard, now how can you learn m many efficiently?

But wait for a second...

- You just mentioned that learning two-layer network with three neurons is NP-hard, now how can you learn m many efficiently?
- We are not optimizing a m neurons network up to optimal, we are just making sure it can do as well as the best (much) smaller networks.

But wait for a second...

- You just mentioned that learning two-layer network with three neurons is NP-hard, now how can you learn m many efficiently?
- We are not optimizing a m neurons network up to optimal, we are just making sure it can do as well as the best (much) smaller networks.
  - $m \geq poly(N)$: Training.

But wait for a second...

- You just mentioned that learning two-layer network with three neurons is NP-hard, now how can you learn m many efficiently?
- We are not optimizing a m neurons network up to optimal, we are just making sure it can do as well as the best (much) smaller networks.
  - $m \geq poly(N)$: Training.
  - $N \geq poly(p)$: Testing.

But wait for a second...

- You just mentioned that learning two-layer network with three neurons is NP-hard, now how can you learn m many efficiently?
- We are not optimizing a m neurons network up to optimal, we are just making sure it can do as well as the best (much) smaller networks.
  - $m \geq poly(N)$: Training.
  - $N \geq poly(p)$: Testing.
  - So $m \geq poly(p)$.

But wait for a second...

- You just mentioned that learning two-layer network with three neurons is NP-hard, now how can you learn m many efficiently?
- We are not optimizing a m neurons network up to optimal, we are just making sure it can do as well as the best (much) smaller networks.
  - $m \geq poly(N)$: Training.
  - $N \geq poly(p)$: Testing.
  - So $m \geq poly(p)$.
- We are training a network with $1M$ parameters to match the performance of a network with $1K$ parameters, and that is easy!

Networks with more hidden layers?

- Our theorem also works for three layer networks (two hidden layers).

Networks with more hidden layers?

- Our theorem also works for three layer networks (two hidden layers).
- Theoretical reasoning on three layer networks is much harder due to the extremely non-convex interactions between the two hidden layers.

Label assumption of three-layer networks

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

$$F^* = (f_1^*, \cdots, f_k^*)$$

$$\text{each } f_r^*(x) = \sum_{i \in [p]} a_{r,i}^* \Phi_i \left(\sum_{j \in [p]} v_{i,j}^* \phi_{1,j}(\langle w_j^*, x \rangle) \right)$$

Label assumption of three-layer networks

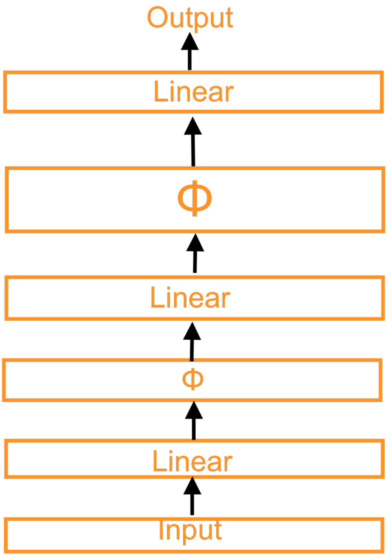
- Assume that the labels are realizable by a simple neural network:

$$F^* = (f_1^*, \cdots, f_k^*)$$

$$\text{each } f_r^*(x) = \sum_{i \in [p]} a_{r,i}^* \Phi_i \left(\sum_{j \in [p]} v_{i,j}^* \phi_{1,j}(\langle w_j^*, x \rangle) \right)$$

- Such that the average loss of network F^* on the training data set is $\leq \varepsilon$.

Picture of the Network.



Our theorem

Theorem (Sketched (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 three layer (two hidden layers) fully-connected neural networks (with m neurons per layer and ReLU activation functions) finds a solution with generalization gap $\leq \varepsilon$ in time $poly(m/\varepsilon)$.

Our theorem

Theorem (Sketched (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 three layer (two hidden layers) fully-connected neural networks (with m neurons per layer and ReLU activation functions) finds a solution with generalization gap $\leq \varepsilon$ in time $poly(m/\varepsilon)$.

- Despite the fact that the paper is 67 pages long (all proofs), the proof is conceptually very simple. Believe me :)

Extending to even deeper?

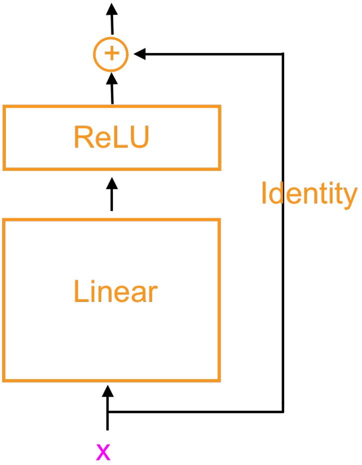
- We could not do it for general network, but we could show it for ResNet.

Extending to even deeper?

- We could not do it for general network, but we could show it for ResNet.
- We first consider single skip ResNet.

Extending to even deeper?

- We could not do it for general network, but we could show it for ResNet.
- We first consider single skip ResNet.



-

Extending to even deeper

- Assume that the labels are realizable by a ResNet:

$$
\begin{aligned}
F^*_\ell &= (f^*_{1,\ell}, \cdots, f^*_{k,\ell}), \text{for } \ell = 1, \cdots, L. \\
f^*_{r,1}(x) &= \phi_{r,1}(\langle w^*_{r,1}, x \rangle) \\
f^*_{r,\ell}(x) &= f^*_{r,\ell-1}(x) + \alpha_\ell \phi_{r,\ell}(\langle w^*_{r,\ell}, F^*_{\ell-1}(x) \rangle) \text{ for } \ell \geq 2
\end{aligned}
$$

Extending to even deeper

- Assume that the labels are realizable by a ResNet:

$$
\begin{aligned}
F_\ell^* &= (f_{1,\ell}^*, \cdots, f_{k,\ell}^*), \text{for } \ell = 1, \cdots, L. \\
f_{r,1}^*(x) &= \phi_{r,1}(\langle w_{r,1}^*, x\rangle) \\
f_{r,\ell}^*(x) &= f_{r,\ell-1}^*(x) + \alpha_\ell \phi_{r,\ell}(\langle w_{r,\ell}^*, F_{\ell-1}^*(x)\rangle) \text{ for } \ell \geq 2
\end{aligned}
$$

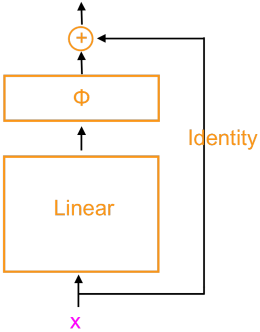
- Where F_0 is the identity mapping and $\alpha_\ell \leq \alpha_{\ell-1}/poly(k)$ with $\alpha_1 = 1$.

Extending to even deeper

- Assume that the labels are realizable by a ResNet:

$$F_\ell^* = (f_{1,\ell}^*, \cdots, f_{k,\ell}^*), \text{for } \ell = 1, \cdots, L.$$
$$f_{r,1}^*(x) = \phi_{r,1}(\langle w_{r,1}^*, x\rangle)$$
$$f_{r,\ell}^*(x) = f_{r,\ell-1}^*(x) + \alpha_\ell \phi_{r,\ell}(\langle w_{r,\ell}^*, F_{\ell-1}^*(x)\rangle) \text{ for } \ell \geq 2$$

- Where F_0 is the identity mapping and $\alpha_\ell \leq \alpha_{\ell-1}/poly(k)$ with $\alpha_1 = 1$.



-

Our theorem

Theorem (Sketched (corollary of 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:

$$N \geq poly(kL/\varepsilon) \times poly(log(mL))$$

Then SGD on a L-layer single-skip ResNet with ReLU activation functions (each layer is fully connected with m neurons) finds a solution with generalization gap $\leq \varepsilon$ in time $poly(mL/\varepsilon)$.

Intuition

- Multi-level features.

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x \rangle)$.

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x\rangle)$.
- Second level features are built on top of first level features by adding simple functions of the first level features.
  $f^*_{r,2}(x) = f^*_{r,1}(x) + \alpha_2 \phi_{r,2}(\langle w^*_{r,2}, F^*_1(x)\rangle)$.

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x\rangle)$.
- Second level features are built on top of first level features by adding simple functions of the first level features.
  $f^*_{r,2}(x) = f^*_{r,1}(x) + \alpha_2 \phi_{r,2}(\langle w^*_{r,2}, F^*_1(x)\rangle)$.
- Third level...

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x \rangle)$.
- Second level features are built on top of first level features by adding simple functions of the first level features.
  $f^*_{r,2}(x) = f^*_{r,1}(x) + \alpha_2 \phi_{r,2}(\langle w^*_{r,2}, F^*_1(x) \rangle)$.
- Third level...
- The 'effect' of the additional features gets smaller as level goes higher ($\alpha_\ell << \alpha_{\ell-1}$).

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x\rangle)$.
- Second level features are built on top of first level features by adding simple functions of the first level features.
  $f^*_{r,2}(x) = f^*_{r,1}(x) + \alpha_2 \phi_{r,2}(\langle w^*_{r,2}, F^*_1(x)\rangle)$.
- Third level...
- The 'effect' of the additional features gets smaller as level goes higher ($\alpha_\ell << \alpha_{\ell-1}$).
- The identity mapping in ResNet allows the network to discover each level gradually to simplify the learning process.

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x\rangle)$.
- Second level features are built on top of first level features by adding simple functions of the first level features.
  $f^*_{r,2}(x) = f^*_{r,1}(x) + \alpha_2\phi_{r,2}(\langle w^*_{r,2}, F^*_1(x)\rangle)$.
- Third level...
- The 'effect' of the additional features gets smaller as level goes higher ($\alpha_\ell << \alpha_{\ell-1}$).
- The identity mapping in ResNet allows the network to discover each level gradually to simplify the learning process.
  - Learn first level features first, then learn second level features, then third...

Intuition

- Multi-level features.
- First level features are simple functions of the input.
  $f^*_{r,1}(x) = \phi_{r,1}(\langle w^*_{r,1}, x\rangle)$.
- Second level features are built on top of first level features by adding simple functions of the first level features.
  $f^*_{r,2}(x) = f^*_{r,1}(x) + \alpha_2 \phi_{r,2}(\langle w^*_{r,2}, F^*_1(x)\rangle)$.
- Third level...
- The 'effect' of the additional features gets smaller as level goes higher ($\alpha_\ell << \alpha_{\ell-1}$).
- The identity mapping in ResNet allows the network to discover each level gradually to simplify the learning process.
  - Learn first level features first, then learn second level features, then third...
- Also applies to ResNet with convolution layers. But not double skip or triple skip yet.

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.
- Question: Why always SGD?

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.
- Question: Why always SGD?
  - Can we modify SGD so it biases towards solutions with even better generalization?

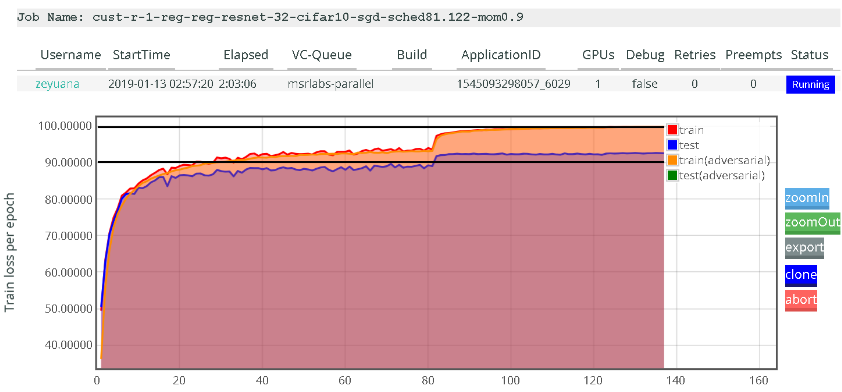
SGD with learning rate decay

- 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.

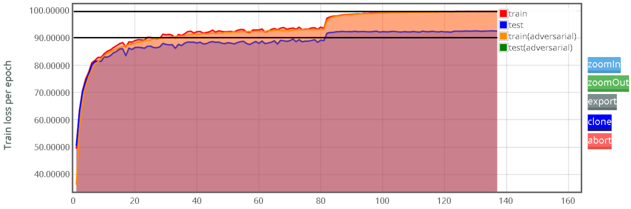
SGD with learning rate decay

- 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 small learning rate

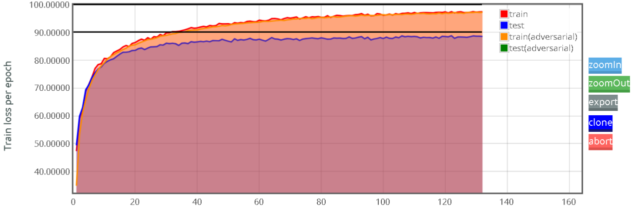
Job Name: cust-r-1-reg-reg-resnet-32-cifar10-sgd-sched81.122-mom0.9

Username	StartTime	Elapsed	VC-Queue	Build	ApplicationID	GPUs	Debug	Retries	Preempts	Status
zeyuana	2019-01-13 02:57:20	2:03:06	msrlabs-parallel		1545093298057_6029	1	false	0	0	Running



Job Name: cust-r-1-reg-reg-resnet-32-cifar10-sgd-lr0.01-sched300.301-mom0.9

Username	StartTime	Elapsed	VC-Queue	Build	ApplicationID	GPUs	Debug	Retries	Preempts	Status
zeyuana	2019-01-13 02:58:56	2:02:01	msrlabs-parallel		1545093298057_6032	1	false	1	0	Running



SGD with learning rate decay

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

SGD with learning rate decay

- 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!

SGD with learning rate decay

- 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!
- SGD with learning rate decay implicitly biases towards solutions with better generalization?

Previous works

- SGD with large learning rate can escape sharp local minimals.

Previous works

- SGD with large learning rate can escape sharp local minimals.
- So after learning rate decay it will converge to a flat local minimal.

Previous works

- SGD with large learning rate can escape sharp local minimals.
- So after learning rate decay it will converge to a flat local minimal.
- Flat local minimal generalize better.

Previous works

- SGD with large learning rate can escape sharp local minimals.
- So after learning rate decay it will converge to a flat local minimal.
- Flat local minimal generalize better.
- But for theory:

Previous works

- SGD with large learning rate can escape sharp local minimals.
- So after learning rate decay it will converge to a flat local minimal.
- Flat local minimal generalize better.
- But for theory:
  - Why neural network has sharp local minimals?

Previous works

- SGD with large learning rate can escape sharp local minimals.
- So after learning rate decay it will converge to a flat local minimal.
- Flat local minimal generalize better.
- But for theory:
  - Why neural network has sharp local minimals?
  - Why flat local minimal in neural network generalizes better?

Our current work

- A concrete example of a data set such that:

Our current work

- A concrete example of a data set such that:
- When training a two layer network using SGD, large learning rate + learning rate decay provably generalizes better than small learning rate.

The intuition of the data set

- Texts labeled as happy:

The intuition of the data set

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

The intuition of the data set

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

The intuition of the data set

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

The intuition of the data set

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

The intuition of the data set

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

The intuition of the data set

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

The intuition 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:

The intuition 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 ☹

The intuition 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 ☹

The intuition 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.

The intuition 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 ☹

The intuition 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 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 of SGD

- Randomly 80 percent of the data has symbols ☺, ☹.

The intuition of SGD

- Randomly 80 percent of the data has symbols ☺, ☹.
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.

The intuition of SGD

- Randomly 80 percent of the data has symbols ☺, ☹.
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- Then the gradient of the examples with these symbols will be close to zero.

The intuition of SGD

- Randomly 80 percent of the data has symbols ☺, ☹.
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- Then the gradient of the examples with these symbols will be close to zero.
- SGD then uses very few examples without these symbols to learn the sentences.

The intuition of SGD

- Randomly 80 percent of the data has symbols ☺, ☹.
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- Then the gradient of the examples with these symbols will be close to zero.
- SGD then uses very few examples without these symbols to learn the sentences.
- SGD with large learning rate will avoid memorizing the special symbols, and learn sentences with all examples.

The intuition of SGD

- Randomly 80 percent of the data has symbols ☺, ☹.
- SGD with small learning rate will quickly memorize the special symbols ☺, ☹.
- Then the gradient of the examples with these symbols will be close to zero.
- SGD then uses very few examples without these symbols to learn the sentences.
- SGD with large learning rate will avoid memorizing the special symbols, and learn sentences with all examples.
- After weight decay it then learns the symbols.

The data set

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

The data set

- Data points in dimension $2d$: $x = (x_1, x_2)$ where $x_1, x_2 \in \mathbb{R}^d$.
- $x_1 \sim N\left(0, \frac{1}{d} I\right)$, the label of x is $y(x) = 1_{\langle w, x \rangle \geq 0}$, where w is a vector in $\mathbb{R}^d$.

The data set

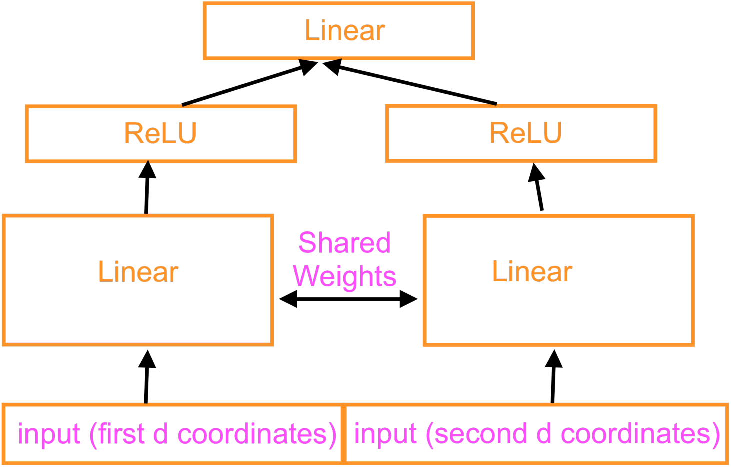
- Data points in dimension $2d$: $x = (x_1, x_2)$ where $x_1, x_2 \in \mathbb{R}^d$.
- $x_1 \sim N\left(0, \frac{1}{d} I\right)$, the label of x is $y(x) = 1_{\langle w,x \rangle \geq 0}$, where w is a vector in $\mathbb{R}^d$.
- With probability $1 - p$,

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

Where z, δ are two vectors in $\mathbb{R}^d$ with $\|z\|_2 = 1$ and $\|\delta\|_2 = \frac{1}{d^{\Theta(1)}}$.

The model

We use the following model:



Our Theorem

Our Theorem

Theorem (Sketched)

Given N training examples, at training error $\frac{p}{N}$ for the cross entropy loss:

Our Theorem

Theorem (Sketched)

Given N training examples, at training error $\frac{p}{N}$ for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $O(p/N)$.*

Our Theorem

Theorem (Sketched)

Given N training examples, at training error $\frac{p}{N}$ for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $O(p/N)$.*
2. *SGD with small learning rate has generalization error at least $\Omega(1/N)$.*

Our Theorem

Theorem (Sketched)

Given N training examples, at training error $\frac{p}{N}$ for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $O(p/N)$.*
2. *SGD with small learning rate has generalization error at least $\Omega(1/N)$.*

- For $p = o(1)$, large learning rate provably generalize better.

Our Theorem

Theorem (Sketched)

Given N training examples, at training error $\frac{p}{N}$ for the cross entropy loss:

1. *SGD with large learning rate + learning rate decay achieves generalization error $O(p/N)$.*
2. *SGD with small learning rate has generalization error at least $\Omega(1/N)$.*

- For $p = o(1)$, large learning rate provably generalize better.
- But in practice, probably $p = 0.9$ (so we ignore 10 percent of the data) would make a huge difference.

Summary

- I have described 10 percent of my works.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.
  - Non-convex optimization.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.
  - Non-convex optimization.
    - Matrix completion, matrix sensing, topic model, graphical model, mixture of regressions, escaping saddle points, experiment design.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.
  - Non-convex optimization.
    - Matrix completion, matrix sensing, topic model, graphical model, mixture of regressions, escaping saddle points, experiment design.
  - Data processing.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.
  - Non-convex optimization.
    - Matrix completion, matrix sensing, topic model, graphical model, mixture of regressions, escaping saddle points, experiment design.
  - Data processing.
    - PCA (SVD), CCA, PCP, PCR, online/stochastic PCA.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.
  - Non-convex optimization.
    - Matrix completion, matrix sensing, topic model, graphical model, mixture of regressions, escaping saddle points, experiment design.
  - Data processing.
    - PCA (SVD), CCA, PCP, PCR, online/stochastic PCA.
  - Convex optimization/Algorithms.

Summary

- I have described 10 percent of my works.
  - In the past 4 years of my Ph.D. I have published 31 papers, with 6 online manuscripts and 6 writing manuscripts.
  - Smoothed analysis of the regularization paths in regression problems.
  - Online learning.
    - Bandit convex optimization; Sparsity, variance and curvature in linear/multi-arm bandit, distributed bandit, chasing convex functions, theory of reinforcement learning.
  - Non-convex optimization.
    - Matrix completion, matrix sensing, topic model, graphical model, mixture of regressions, escaping saddle points, experiment design.
  - Data processing.
    - PCA (SVD), CCA, PCP, PCR, online/stochastic PCA.
  - Convex optimization/Algorithms.
    - Matrix/Operator scaling, ℓ_p regressions, parallel(distributed) optimization.