Towards understanding the hierarchical learning in multi-layer (over-parameterized) neural networks

Yuanzhi Li

Assistant Professor, Carnegie Mellon University
Consulting Researcher, Microsoft Research

date: can be changed to a custom date

Hierarchical learning in multi-layer networks

- During the training process of a deep neural network:

Hierarchical learning in multi-layer networks

- During the training process of a deep neural network:
  - The first layer learns features (F_1) representing the most basic knowledge of the data set.

Hierarchical learning in multi-layer networks

- During the training process of a deep neural network:
  - The first layer learns features (F_1) representing the most basic knowledge of the data set.

  - The second layer learns features (F_2) representing the simple ways of compositing F_1 to reason about the data set.

Hierarchical learning in multi-layer networks

- During the training process of a deep neural network:
  - The first layer learns features (F_1) representing the most basic knowledge of the data set.

  - The second layer learns features (F_2) representing the simple ways of compositing F_1 to reason about the data set.

  - The third layer learns features (F_3) that compose F_2 in the simple way.

Hierarchical learning in multi-layer networks

- During the training process of a deep neural network:
  - The first layer learns features (F_1) representing the most basic knowledge of the data set.

  - The second layer learns features (F_2) representing the simple ways of compositing F_1 to reason about the data set.

  - The third layer learns features (F_3) that compose F_2 in the simple way.

  - The complexity of the features (as a function of the input) F_i gradually increases for deeper layers.

Hierarchical learning is NOT layer-wise training

- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.

Hierarchical learning is NOT layer-wise training

- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.
  - Layer-wise training: Train first layer, then train second layer, then train third layer... Typically can not match the performance of training all layers simultaneously: The lower level features are not very good if we only train lower level layers: e.g. the quality of F_1 is not very good if we only train the first layer.

Hierarchical learning is NOT layer-wise training

- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.
  - Layer-wise training: Train first layer, then train second layer, then train third layer... Typically can not match the performance of training all layers simultaneously: The lower level features are not very good if we only train lower level layers: e.g. the quality of F_1 is not very good if we only train the first layer.
  - Training all layers together: The quality of F_1 can be improved via learning other layers.

Hierarchical learning is NOT layer-wise training

- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.
  - Layer-wise training: Train first layer, then train second layer, then train third layer... Typically can not match the performance of training all layers simultaneously: The lower level features are not very good if we only train lower level layers: e.g. the quality of F_1 is not very good if we only train the first layer.
  - Training all layers together: The quality of F_1 can be improved via learning other layers.
  - How? Theory?

Hierarchical learning is NOT layer-wise training

- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.
  - Layer-wise training: Train first layer, then train second layer, then train third layer... Typically can not match the performance of training all layers simultaneously: The lower level features are not very good if we only train lower level layers: e.g. the quality of F_1 is not very good if we only train the first layer.
  - Training all layers together: The quality of F_1 can be improved via learning other layers.
  - How? Theory?
- We present a formal proof of this hierarchical learning process, under the model of multi-layer DenseNet with quadratic activation functions (σ).

Hierarchical learning is NOT layer-wise training

- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.
  - Layer-wise training: Train first layer, then train second layer, then train third layer... Typically can not match the performance of training all layers simultaneously: The lower level features are not very good if we only train lower level layers: e.g. the quality of F_1 is not very good if we only train the first layer.
  - Training all layers together: The quality of F_1 can be improved via learning other layers.
  - How? Theory?
- We present a formal proof of this hierarchical learning process, under the model of multi-layer DenseNet with quadratic activation functions (σ).
  - We assume there is a multi-layer (i.e. hierarchical) target network that generates the labels by composing features layer by layer.

Hierarchical learning is NOT layer-wise training

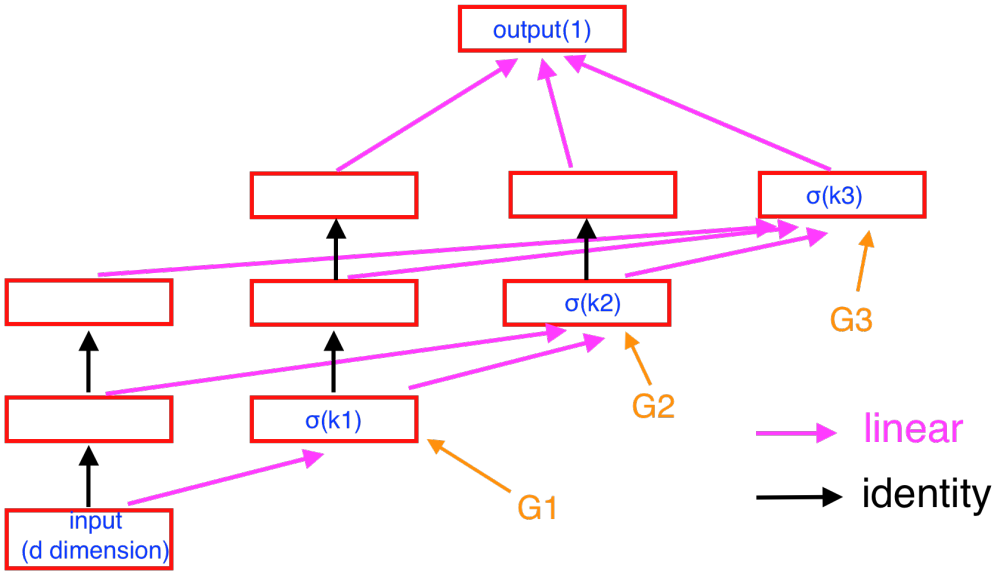
- More importantly, via learning the higher level features $\{F_\ell\}_{\ell>\ell_0}$, it also improves the quality of the lower level features $\{F_\ell\}_{\ell\le\ell_0}$.
  - Layer-wise training: Train first layer, then train second layer, then train third layer... Typically can not match the performance of training all layers simultaneously: The lower level features are not very good if we only train lower level layers: e.g. the quality of F_1 is not very good if we only train the first layer.
  - Training all layers together: The quality of F_1 can be improved via learning other layers.
  - How? Theory?
- We present a formal proof of this hierarchical learning process, under the model of multi-layer DenseNet with quadratic activation functions (σ).
  - We assume there is a multi-layer (i.e. hierarchical) target network that generates the labels by composing features layer by layer.
  - We will show how training a neural network via SGD from random initialization can recover this hierarchical composition of the target network gradually (learning higher level features can also improve the quality of lower ones).

Target network

- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:

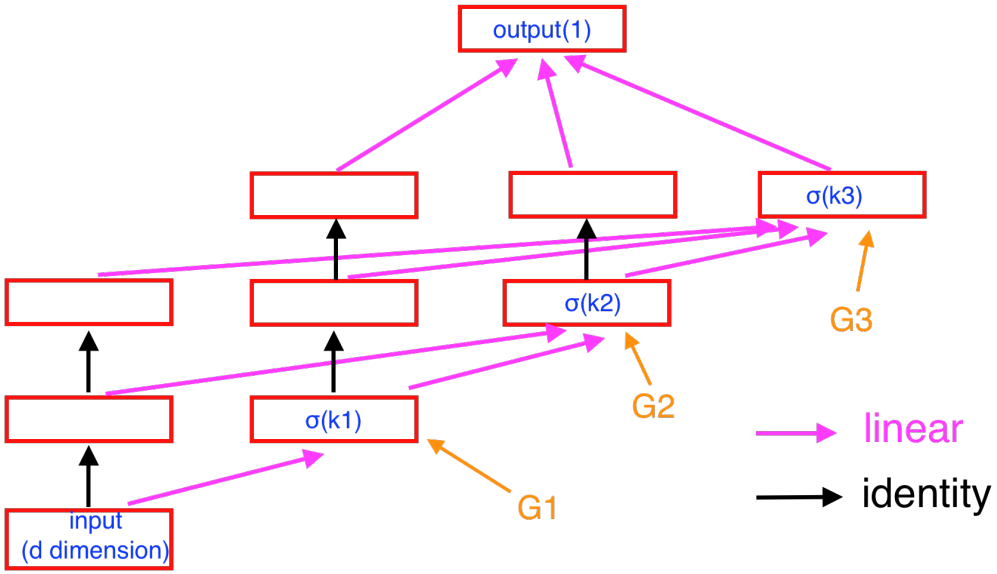
Target network

- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:



Target network

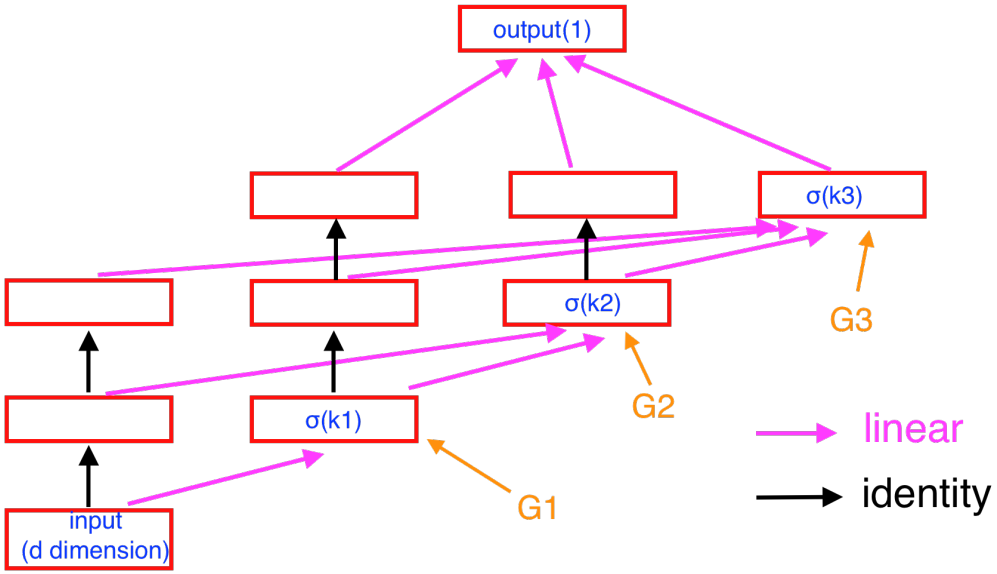
- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:



- $G_1(x) = \sigma(W_{1,0}x)$, $W_{1,0} \in \mathbb{R}^{k_1 \times k_0}$.

Target network

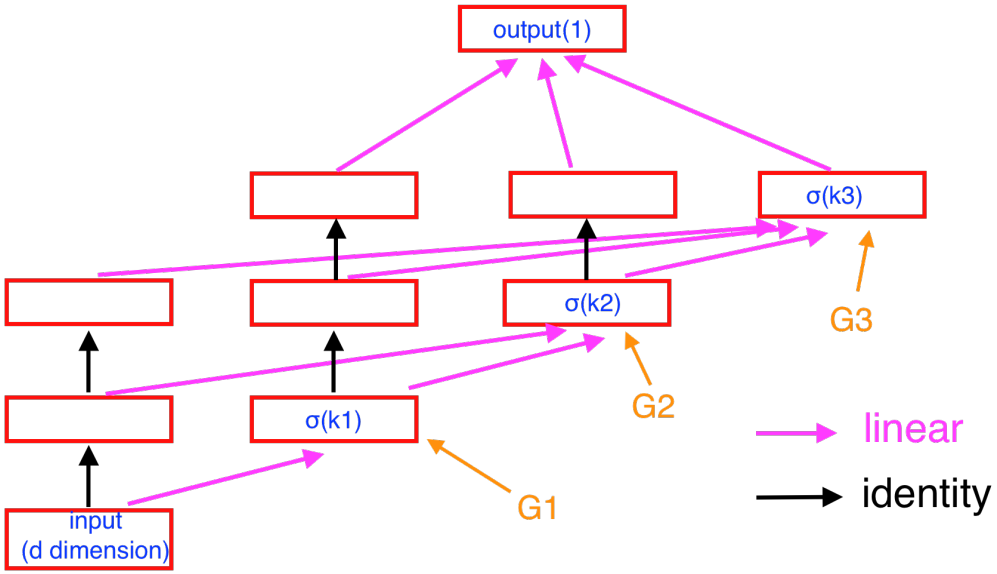
- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:



- $G_1(x) = \sigma(W_{1,0}x),\ W_{1,0} \in \mathbb{R}^{k_1 \times k_0}$.
- $G_2(x) = \sigma(W_{2,0}x + W_{2,1}G_1(x)),\ W_{2,0} \in \mathbb{R}^{k_2 \times k_0}, W_{2,1} \in \mathbb{R}^{k_2 \times k_1}$

Target network

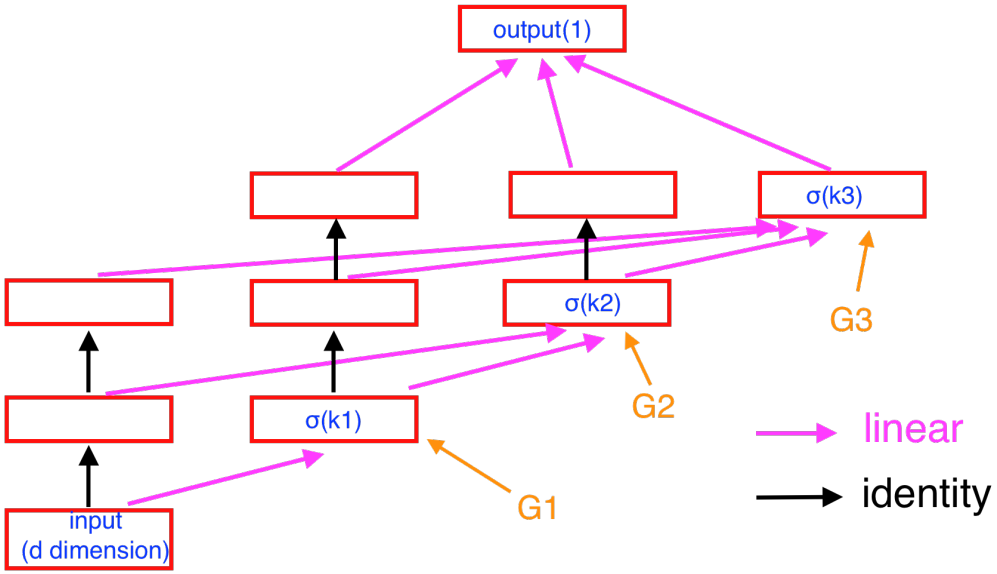
- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:



- $G_1(x) = \sigma(W_{1,0}x),\ W_{1,0} \in \mathbb{R}^{k_1 \times k_0}$.
- $G_2(x) = \sigma(W_{2,0}x + W_{2,1}G_1(x)),\ W_{2,0} \in \mathbb{R}^{k_2 \times k_0}, W_{2,1} \in \mathbb{R}^{k_2 \times k_1}$
- ...

Target network

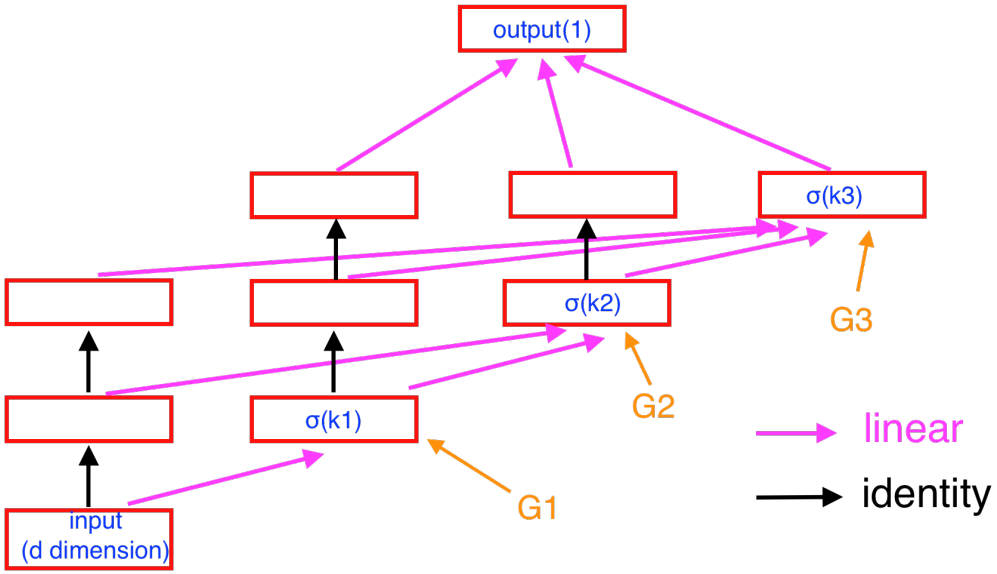
- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:



- $G_1(x) = \sigma(W_{1,0}x)$, $W_{1,0} \in \mathbb{R}^{k_1 \times k_0}$.
- $G_2(x) = \sigma(W_{2,0}x + W_{2,1}G_1(x))$, $W_{2,0} \in \mathbb{R}^{k_2 \times k_0}$, $W_{2,1} \in \mathbb{R}^{k_2 \times k_1}$
- ...
- $G_L(x) = \sigma\left(W_{L,0}x + \sum_{j<L} W_{L,j}G_j(x)\right)$. $W_{L,j} \in \mathbb{R}^{k_L \times k_j}$.

Target network

- Target network that generates the labels (L-layer DenseNet with quadratic activations σ): For $d = k_0 \geq k_1 \geq k_2 \cdots \geq k_L$:



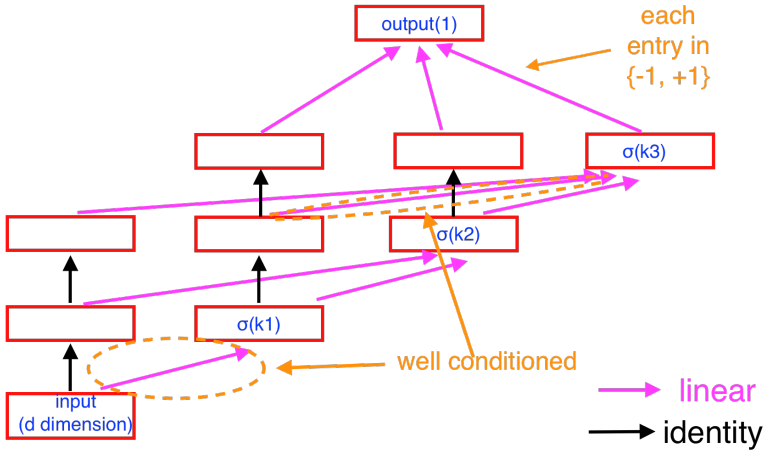
- $G_1(x) = \sigma(W_{1,0}x)$, $W_{1,0} \in \mathbb{R}^{k_1 \times k_0}$.
- $G_2(x) = \sigma(W_{2,0}x + W_{2,1}G_1(x))$, $W_{2,0} \in \mathbb{R}^{k_2 \times k_0}$, $W_{2,1} \in \mathbb{R}^{k_2 \times k_1}$
- ...
- $G_L(x) = \sigma\left(W_{L,0}x + \sum_{j<L} W_{L,j}G_j(x)\right)$. $W_{L,j} \in \mathbb{R}^{k_L \times k_j}$.
- $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$. ($\alpha_\ell > 0$ is a scaler)

Regularity condition target network

- Well conditioned: The singular values of each $W_{\ell,j}$ are constants, each entry of $w_\ell \in \{-1, 1\}$ ($w_\ell \in \mathbb{R}^{k_\ell}$).

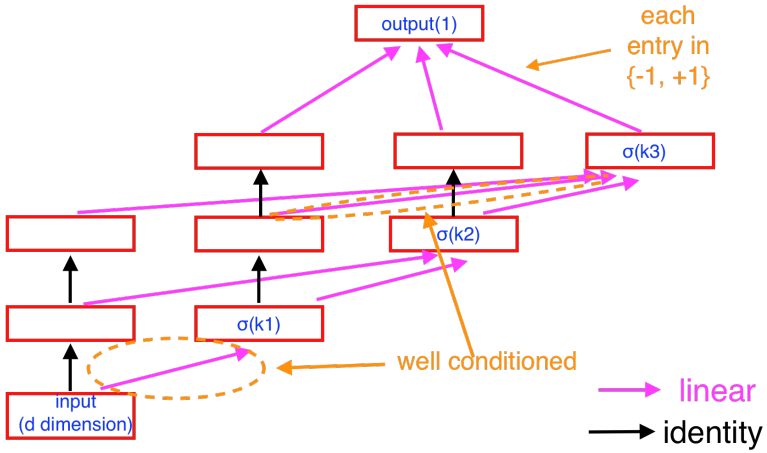
Regularity condition target network

- Well conditioned: The singular values of each $W_{\ell,j}$ are constants, each entry of $w_\ell \in \{-1, 1\}$ ($w_\ell \in \mathbb{R}^{k_\ell}$).



Regularity condition target network

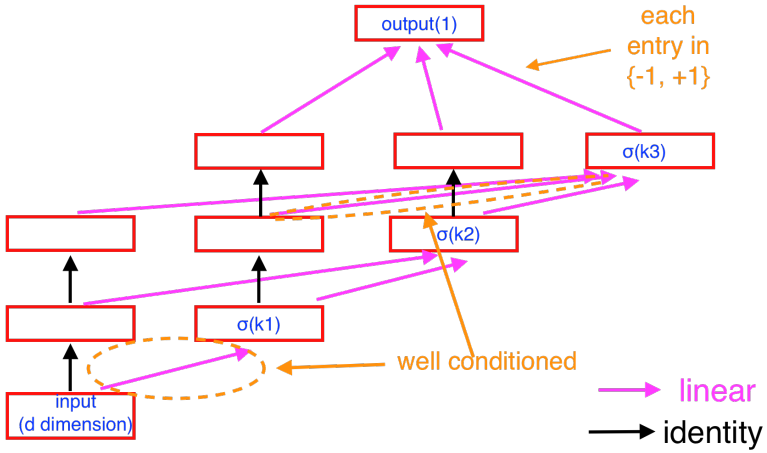
- Well conditioned: The singular values of each $W_{\ell,j}$ are constants, each entry of $w_\ell \in \{-1, 1\}$ ($w_\ell \in \mathbb{R}^{k_\ell}$).



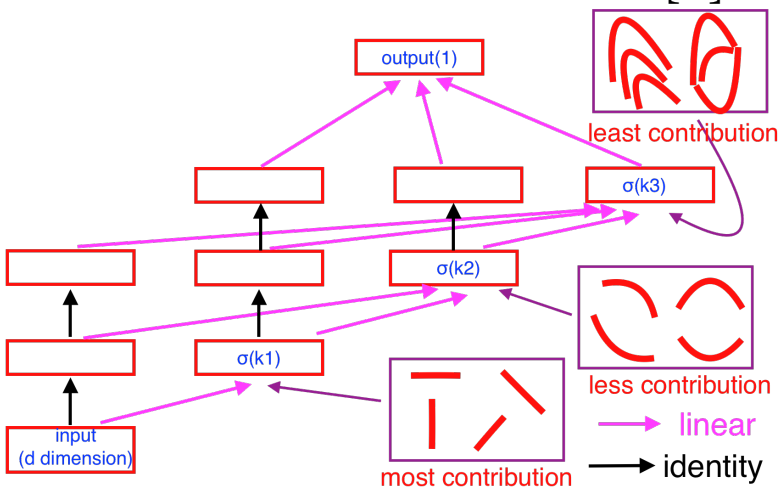
- Information gap: $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$: $\alpha_1 = 1$, $\alpha_{\ell+1} \ll \alpha_\ell$.

Regularity condition target network

- Well conditioned: The singular values of each $W_{\ell,j}$ are constants, each entry of $w_\ell \in \{-1, 1\}$ ($w_\ell \in \mathbb{R}^{k_\ell}$).



- Information gap: $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$: $\alpha_1 = 1$, $\alpha_{\ell+1} \ll \alpha_\ell$.

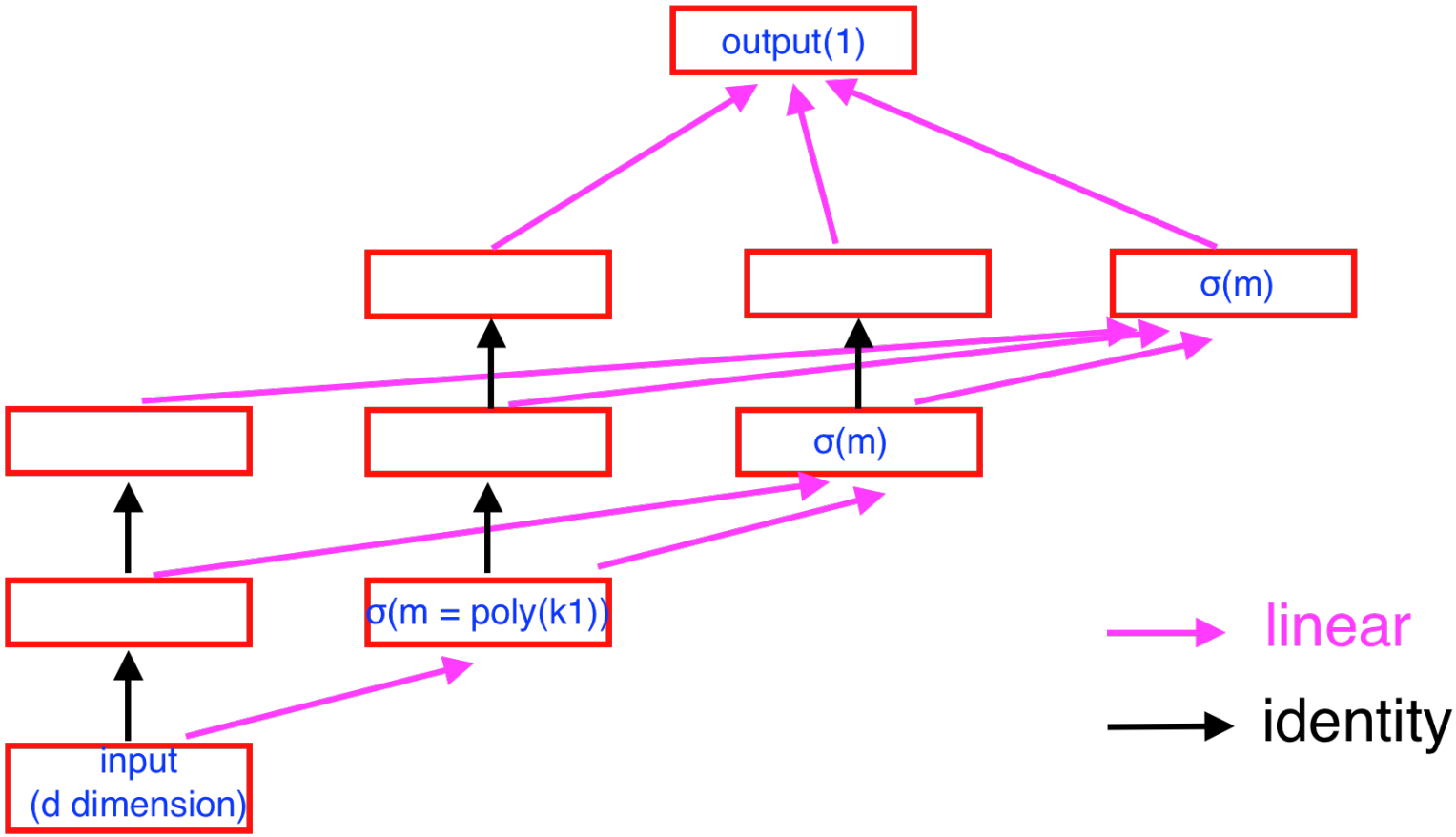


Learner network

- Over-parameterized DenseNet with quadratic activation functions.

Learner network

- Over-parameterized DenseNet with quadratic activation functions.



Main Theorem

Main Theorem

Theorem (AL'19)

When $k_\ell \approx d^{\frac{1}{2^\ell}}$, $\alpha_{\ell+1} \approx \frac{\alpha_\ell}{d^{\frac{1}{2^\ell}}}$: *Over certain input distributions (including non-spherical Gaussian, mixture of non-spherical Gaussian), for every* $L = o(\log \log d)$, *for every* $\varepsilon > 0$, *given* $N = poly(d, 1/\varepsilon)$ *many training examples, SGD* (ℓ_2 *loss* + *regularizer, starting from random initialization) finds a target network* F *with generalization error* ε *in time* $poly(d, 1/\varepsilon)$

Main Theorem

Theorem (AL'19)

When $k_\ell \approx d^{\frac{1}{2^\ell}}$, $\alpha_{\ell+1} \approx \frac{\alpha_\ell}{d^{\frac{1}{2^\ell}}}$: *Over certain input distributions (including non-spherical Gaussian, mixture of non-spherical Gaussian), for every* $L = o(\log \log d)$, *for every* $\varepsilon > 0$, *given* $N = poly(d, 1/\varepsilon)$ *many training examples, SGD* (ℓ_2 *loss* + *regularizer, starting from random initialization) finds a target network* F *with generalization error* ε *in time* $poly(d, 1/\varepsilon)$

-
- The target function is a degree 2^L polynomial over d dimension input, other learning algorithm (e.g. brutal search/matrix sensing/kernel method) would typically require either running time 2^d or sample complexity $d^{2^{L-1}}$. Which is super polynomial as long as $L = \omega(1)$.

Main Theorem

Theorem (AL'19)

When $k_\ell \approx d^{\frac{1}{2^\ell}}$, $\alpha_{\ell+1} \approx \frac{\alpha_\ell}{d^{\frac{1}{2^\ell}}}$: *Over certain input distributions (including non-spherical Gaussian, mixture of non-spherical Gaussian), for every* $L = o(\log \log d)$, *for every* $\varepsilon > 0$, *given* $N = poly(d, 1/\varepsilon)$ *many training examples, SGD* (ℓ_2 *loss* + *regularizer, starting from random initialization) finds a target network* F *with generalization error* ε *in time* $poly(d, 1/\varepsilon)$

-
- The target function is a degree 2^L polynomial over d dimension input, other learning algorithm (e.g. brutal search/matrix sensing/kernel method) would typically require either running time 2^d or sample complexity $d^{2^{L-1}}$. Which is super polynomial as long as $L = \omega(1)$.
- $k_\ell \approx d^{\frac{1}{2^\ell}}$ implies that the hierarchical learning must be done for at least $\omega(1)$ times, to avoid learning in one shot a $\omega(1)$ degree polynomial over $d^{\Omega(1)}$ dimension input.

Implication

- To the best of my knowledge, this is the only theoretical result of learning a deep neural network via SGD, so the target function is not directly learnable by other known algorithms (Kernel methods/Isotonic regression/Tensor decomposition etc).

Implication

- To the best of my knowledge, this is the only theoretical result of learning a deep neural network via SGD, so the target function is not directly learnable by other known algorithms (Kernel methods/Isotonic regression/Tensor decomposition etc).
- Typically (e.g. NTK, Neural network under Gaussian inputs, sparse coding) the theoretical reasoning of how neural network works is by showing that the network is simulating other known algorithms.

Implication

- To the best of my knowledge, this is the only theoretical result of learning a deep neural network via SGD, so the target function is not directly learnable by other known algorithms (Kernel methods/Isotonic regression/Tensor decomposition etc).
- Typically (e.g. NTK, Neural network under Gaussian inputs, sparse coding) the theoretical reasoning of how neural network works is by showing that the network is simulating other known algorithms.
- But if neural networks are simulating those algorithms, then why do we use deep learning in practice?

Implication

- To the best of my knowledge, this is the only theoretical result of learning a deep neural network via SGD, so the target function is not directly learnable by other known algorithms (Kernel methods/Isotonic regression/Tensor decomposition etc).
- Typically (e.g. NTK, Neural network under Gaussian inputs, sparse coding) the theoretical reasoning of how neural network works is by showing that the network is simulating other known algorithms.
- But if neural networks are simulating those algorithms, then why do we use deep learning in practice?
- This work: hierarchical learning in multi-layer networks, to the best of my knowledge, only achievable by training a neural network.

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.
- BUG ALERT! Why the first layer feeds x_1^2, x_2^2 to the second layer? Why can't it learns $\frac{1}{5}(x_1 + 2x_2)^2, \frac{1}{5}(2x_1 - x_2)^2$ at the first layer?

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.
- BUG ALERT! Why the first layer feeds x_1^2, x_2^2 to the second layer? Why can't it learns $\frac{1}{5}(x_1 + 2x_2)^2, \frac{1}{5}(2x_1 - x_2)^2$ at the first layer?
  - $\frac{1}{5}(x_1 + 2x_2)^2 + \frac{1}{5}(2x_1 - x_2)^2 = x_1^2 + x_2^2$.

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.
- BUG ALERT! Why the first layer feeds x_1^2, x_2^2 to the second layer? Why can't it learns $\frac{1}{5}(x_1 + 2x_2)^2, \frac{1}{5}(2x_1 - x_2)^2$ at the first layer?
  - $\frac{1}{5}(x_1 + 2x_2)^2 + \frac{1}{5}(2x_1 - x_2)^2 = x_1^2 + x_2^2$.
  - But quadratic functions of $(x_1 + 2x_2)^2, (2x_1 - x_2)^2$ is not able to reconstruct $x_1^4 + x_2^4$! — LEARNING DEAD.

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.
- BUG ALERT! Why the first layer feeds x_1^2, x_2^2 to the second layer? Why can't it learns $\frac{1}{5}(x_1 + 2x_2)^2, \frac{1}{5}(2x_1 - x_2)^2$ at the first layer?
  - $\frac{1}{5}(x_1 + 2x_2)^2 + \frac{1}{5}(2x_1 - x_2)^2 = x_1^2 + x_2^2$.
  - But quadratic functions of $(x_1 + 2x_2)^2, (2x_1 - x_2)^2$ is not able to reconstruct $x_1^4 + x_2^4$! — LEARNING DEAD.
- The rich representation regularizer of over-parameterization: By over-parameterization + gaussian random initialization, It enforces the $m \gg 1$ neurons to learn:

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.
- BUG ALERT! Why the first layer feeds x_1^2, x_2^2 to the second layer? Why can't it learns $\frac{1}{5}(x_1 + 2x_2)^2, \frac{1}{5}(2x_1 - x_2)^2$ at the first layer?
  - $\frac{1}{5}(x_1 + 2x_2)^2 + \frac{1}{5}(2x_1 - x_2)^2 = x_1^2 + x_2^2$.
  - But quadratic functions of $(x_1 + 2x_2)^2, (2x_1 - x_2)^2$ is not able to reconstruct $x_1^4 + x_2^4$! — LEARNING DEAD.
- The rich representation regularizer of over-parameterization: By over-parameterization + gaussian random initialization, It enforces the $m \gg 1$ neurons to learn:
  - $(\alpha_i x_1 + \beta_i x_2)^2_{i \in [m]}$, α_i, β_i looks like independent standard Gaussian: $E_{\alpha_i, \beta_i}(\alpha_i x_1 + \beta_i x_2)^2 = x_1^2 + x_2^2$. This is enough to construct $x_1^4 + x_2^4$.

Intuition of the proof via a super simple example

- $x \in \mathbb{R}^{1000}$ as a standard Gaussian variable, $G(x) = x_1^2 + x_2^2 + \alpha(x_1^4 + x_2^4)$. Say $\alpha = 0.3$.
- Ideal learning: The first layer learns $x_1^2 + x_2^2$, then it feeds x_1^2, x_2^2 to the second layer, then second layer learns $\alpha(x_1^4 + x_2^4)$.
- BUG ALERT! Why the first layer feeds x_1^2, x_2^2 to the second layer? Why can't it learns $\frac{1}{5}(x_1 + 2x_2)^2, \frac{1}{5}(2x_1 - x_2)^2$ at the first layer?
  - $\frac{1}{5}(x_1 + 2x_2)^2 + \frac{1}{5}(2x_1 - x_2)^2 = x_1^2 + x_2^2$.
  - But quadratic functions of $(x_1 + 2x_2)^2, (2x_1 - x_2)^2$ is not able to reconstruct $x_1^4 + x_2^4$! — LEARNING DEAD.
- The rich representation regularizer of over-parameterization: By over-parameterization + gaussian random initialization, It enforces the $m \gg 1$ neurons to learn:
  - $(\alpha_i x_1 + \beta_i x_2)^2_{i \in [m]}$, α_i, β_i looks like independent standard Gaussian: $E_{\alpha_i, \beta_i}(\alpha_i x_1 + \beta_i x_2)^2 = x_1^2 + x_2^2$. This is enough to construct $x_1^4 + x_2^4$.
- The rich representation does not matter for the current layer, but it is crucial for the next one.

Intuition of the "correction" stage in hierarchical learning via a super simple example

- $x \in \mathbb{R}^{1000}$, $\mathbb{E}[x_i] = 1$; $G(x) = x_1^2 + x_2^2 + \alpha((x_1^2 + x_3)^2 + (x_2^2 + x_4)^2)$. Say $\alpha = 0.3$.

Intuition of the "correction" stage in hierarchical learning via a super simple example

- $x \in \mathbb{R}^{1000}$, $\mathbb{E}[x_i] = 1$; $G(x) = x_1^2 + x_2^2 + \alpha((x_1^2 + x_3)^2 + (x_2^2 + x_4)^2)$. Say $\alpha = 0.3$.
- The first layer learns a quadratic approximation of this function, which could possibly be $(x_1 + \alpha x_3)^2 + (x_2 + \alpha x_4)^2$ due to (over-fitting to) the x_3, x_4 in the second layer. Correct up to accuracy α.

Intuition of the "correction" stage in hierarchical learning via a super simple example

- $x \in \mathbb{R}^{1000}$, $\mathbb{E}[x_i] = 1$; $G(x) = x_1^2 + x_2^2 + \alpha((x_1^2 + x_3)^2 + (x_2^2 + x_4)^2)$. Say $\alpha = 0.3$.
- The first layer learns a quadratic approximation of this function, which could possibly be $(x_1 + \alpha x_3)^2 + (x_2 + \alpha x_4)^2$ due to (over-fitting to) the x_3, x_4 in the second layer. Correct up to accuracy α.
- The second layer then (to fit the degree 3, degree 4 terms) possibly learns $\alpha\left((x_1 + \alpha x_3)^2 + (1-\alpha)x_3\right)^2 + \alpha\left((x_2 + \alpha x_4)^2 + (1-\alpha)x_4\right)^2$: Correct up to accuracy α^2.

Intuition of the "correction" stage in hierarchical learning via a super simple example

- $x \in \mathbb{R}^{1000}$, $\mathbb{E}[x_i] = 1$; $G(x) = x_1^2 + x_2^2 + \alpha((x_1^2 + x_3)^2 + (x_2^2 + x_4)^2)$. Say $\alpha = 0.3$.
- The first layer learns a quadratic approximation of this function, which could possibly be $(x_1 + \alpha x_3)^2 + (x_2 + \alpha x_4)^2$ due to (over-fitting to) the x_3, x_4 in the second layer. Correct up to accuracy α.
- The second layer then (to fit the degree 3, degree 4 terms) possibly learns $\alpha\left((x_1 + \alpha x_3)^2 + (1-\alpha)x_3\right)^2 + \alpha\left((x_2 + \alpha x_4)^2 + (1-\alpha)x_4\right)^2$: Correct up to accuracy α^2.
- The first layer then gets correct to α^2 accuracy: $(x_1 + \alpha^2 x_3)^2 + (x_2 + \alpha^2 x_4)^2$ due to less over-fitting to the x_3, x_4 terms.

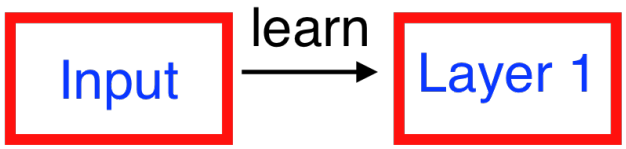
Intuition of the "correction" stage in hierarchical learning via a super simple example

- $x \in \mathbb{R}^{1000}$, $\mathbb{E}[x_i] = 1$; $G(x) = x_1^2 + x_2^2 + \alpha((x_1^2 + x_3)^2 + (x_2^2 + x_4)^2)$. Say $\alpha = 0.3$.
- The first layer learns a quadratic approximation of this function, which could possibly be $(x_1 + \alpha x_3)^2 + (x_2 + \alpha x_4)^2$ due to (over-fitting to) the x_3, x_4 in the second layer. Correct up to accuracy α.
- The second layer then (to fit the degree 3, degree 4 terms) possibly learns $\alpha\left((x_1 + \alpha x_3)^2 + (1-\alpha)x_3\right)^2 + \alpha\left((x_2 + \alpha x_4)^2 + (1-\alpha)x_4\right)^2$: Correct up to accuracy α^2.
- The first layer then gets correct to α^2 accuracy: $(x_1 + \alpha^2 x_3)^2 + (x_2 + \alpha^2 x_4)^2$ due to less over-fitting to the x_3, x_4 terms.
- The second layer then possibly learns $\alpha\left((x_1 + \alpha^2 x_3)^2 + (1-\alpha^2)x_3\right)^2 + \alpha\left((x_2 + \alpha^2 x_4)^2 + (1-\alpha^2)x_4\right)^2$: Correct up to accuracy α^3.

Intuition of the "correction" stage in hierarchical learning via a super simple example

- $x \in \mathbb{R}^{1000}$, $\mathbb{E}[x_i] = 1$; $G(x) = x_1^2 + x_2^2 + \alpha((x_1^2 + x_3)^2 + (x_2^2 + x_4)^2)$. Say $\alpha = 0.3$.
- The first layer learns a quadratic approximation of this function, which could possibly be $(x_1 + \alpha x_3)^2 + (x_2 + \alpha x_4)^2$ due to (over-fitting to) the x_3, x_4 in the second layer. Correct up to accuracy α.
- The second layer then (to fit the degree 3, degree 4 terms) possibly learns $\alpha\left((x_1 + \alpha x_3)^2 + (1-\alpha)x_3\right)^2 + \alpha\left((x_2 + \alpha x_4)^2 + (1-\alpha)x_4\right)^2$: Correct up to accuracy α^2.
- The first layer then gets correct to α^2 accuracy: $(x_1 + \alpha^2 x_3)^2 + (x_2 + \alpha^2 x_4)^2$ due to less over-fitting to the x_3, x_4 terms.
- The second layer then possibly learns $\alpha\left((x_1 + \alpha^2 x_3)^2 + (1-\alpha^2)x_3\right)^2 + \alpha\left((x_2 + \alpha^2 x_4)^2 + (1-\alpha^2)x_4\right)^2$: Correct up to accuracy α^3.
- Keep going...

Multi-layer hierarchical learning

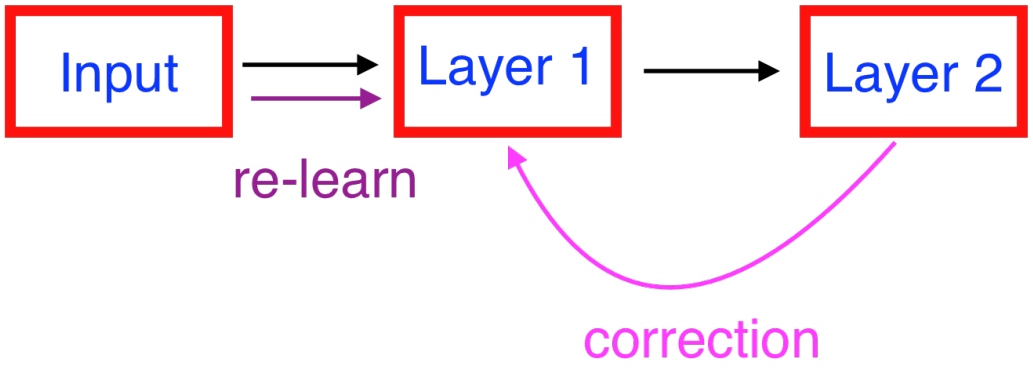


The first layer learns a quadratic approximation of the target function.
Which is not very accurate due to over-fitting to the higher level features.

Multi-layer hierarchical learning

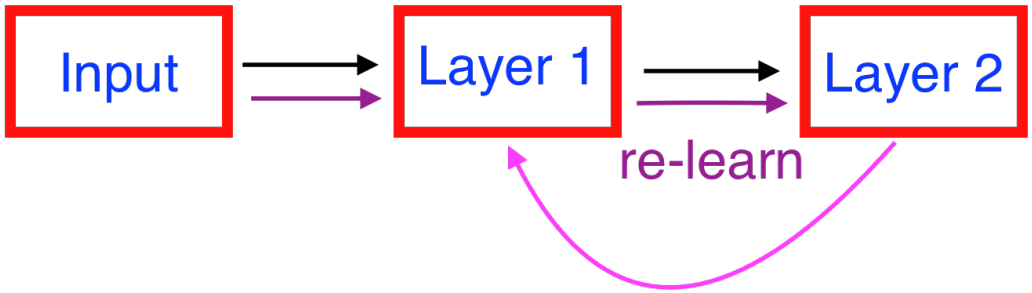
The second layer then learns a quadratic function on top of the first layer (doable via the rich representation given by over-parameterization in the first layer). To approximate the remaining higher degree (3,4) terms in the target function.

Multi-layer hierarchical learning



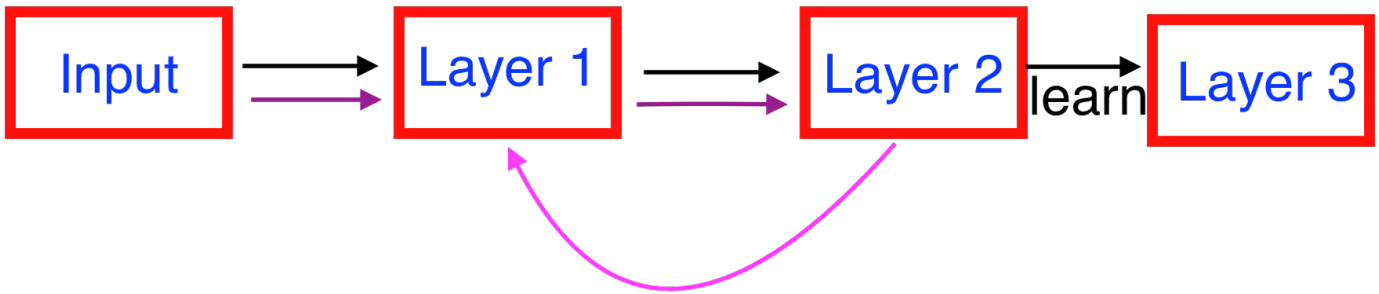
The first layer then "correct" its features due to a reduction of over-fitting in the present of the second layer.

Multi-layer hierarchical learning



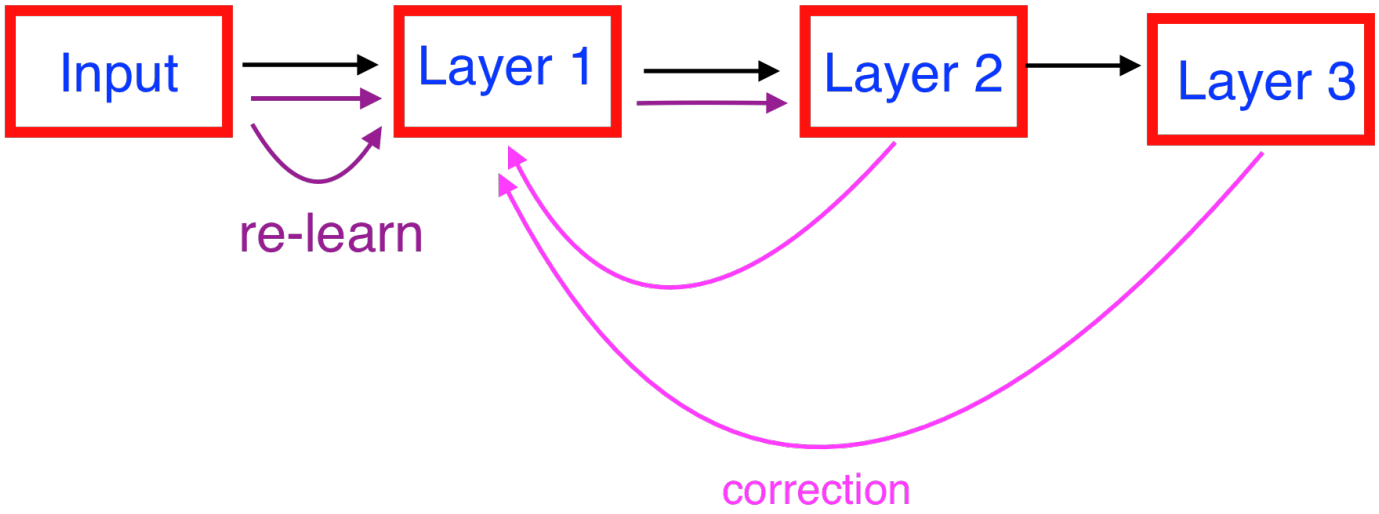
The second layer also learns a more accurate features due to the better input features from the first layer.

Multi-layer hierarchical learning



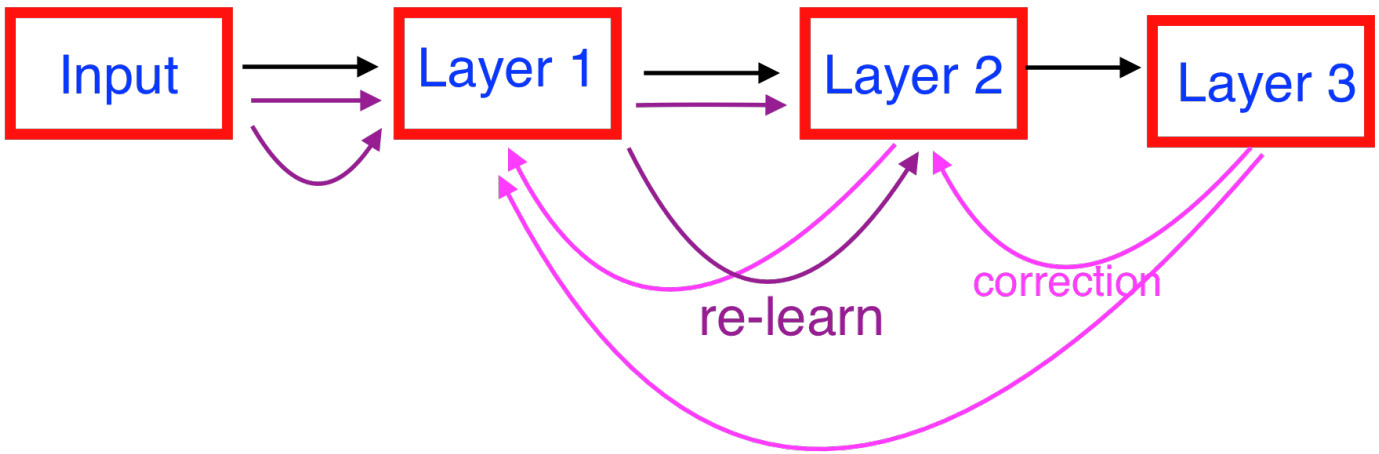
The third layer starts to learn.

Multi-layer hierarchical learning



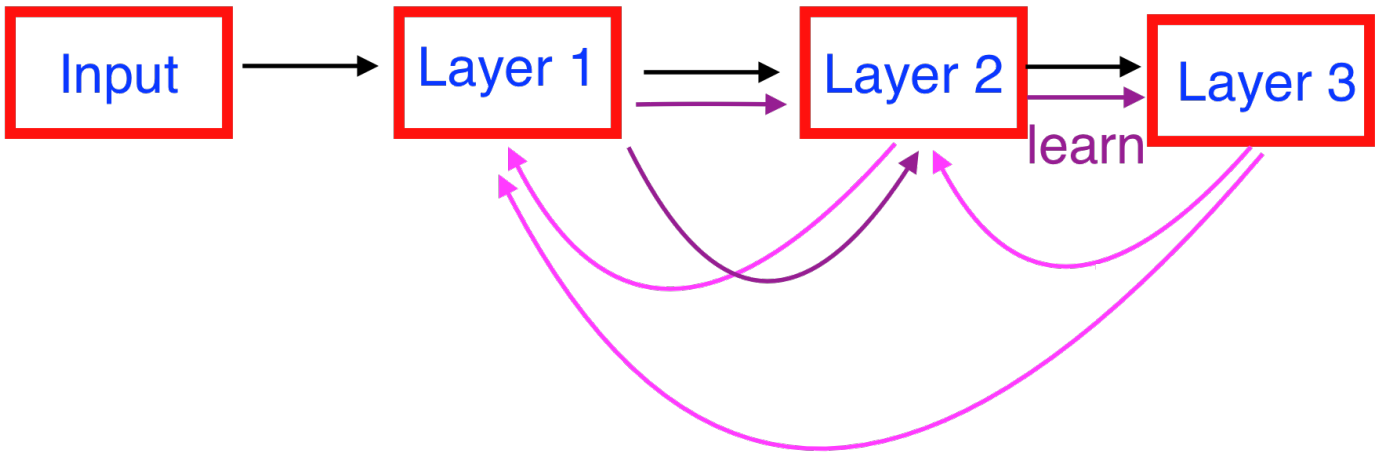
The first layer then further "correct" its features due to a reduction of over-fitting in the present of the third layer.

Multi-layer hierarchical learning



The second layer then further "correct" its features due to a reduction of over-fitting in the present of the third layer and better features from the first layer.

Multi-layer hierarchical learning



KEY MESSAGE: In our work, we do not explicitly train the layers of the network in this order (layers are essentially trained simultaneously), and we can still prove this result (so the hierarchical learning is rather implicit).

Extension to classification and the information gap α

- Our theorem also extends to classification: When the labeling function is given by:

$$y = 1_{\langle w,x\rangle + G(\Pi_{w^\perp} x) \geq 0}$$

Extension to classification and the information gap α

- Our theorem also extends to classification: When the labeling function is given by:

$$y = 1_{\langle w,x\rangle + G(\Pi_{w^\perp} x) \geq 0}$$

Extension to classification and the information gap α

- Our theorem also extends to classification: When the labeling function is given by:

$$y = 1_{\langle w,x\rangle + G(\Pi_{w^\perp} x) \geq 0}$$

Theorem (AL'19)

When $k_\ell \approx d^{\frac{1}{2^\ell}}$, $\alpha_{\ell+1} \approx \frac{\alpha_\ell}{d^{\frac{1}{2^\ell}}}$, $\mathbb{E}[G(\Pi_{w^\perp} x)] \approx 0, \|w\|_2^2 \approx var(G)$*: Over standard gaussian distribution, for every* $L = o(\log\log d)$*, for every* $\varepsilon > 0$*, given* $N = poly(d, 1/\varepsilon)$ *many training examples, SGD (cross entropy loss + regularizer, starting from random initialization) finds a target network* F *with generalization error* ε *in time* $poly(d, 1/\varepsilon)$

-

Extension to classification and the information gap α

- Recall $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$.

Extension to classification and the information gap α

- Recall $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$.
- Critical observation: With probability $\sim \alpha_\ell$ over x,

$$\left| \langle w, x \rangle + \sum_{s \in [\ell-1]} \alpha_s w_s^\top G_s(\Pi_{w^\perp} x) \right| \leq \alpha_\ell$$

Extension to classification and the information gap α

- Recall $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$.
- Critical observation: With probability $\sim \alpha_\ell$ over x,

$$\left| \langle w, x \rangle + \sum_{s \in [\ell-1]} \alpha_s w_s^\top G_s(\Pi_{w^\perp} x) \right| \leq \alpha_\ell$$

- So the label is mostly decided by $\alpha_\ell w_\ell^\top G_\ell(\Pi_{w^\perp} x)$.

Extension to classification and the information gap α

- Recall $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$.
- Critical observation: With probability $\sim \alpha_\ell$ over x,

$$\left| \langle w, x \rangle + \sum_{s \in [\ell-1]} \alpha_s w_s^\top G_s(\Pi_{w^\perp} x) \right| \leq \alpha_\ell$$

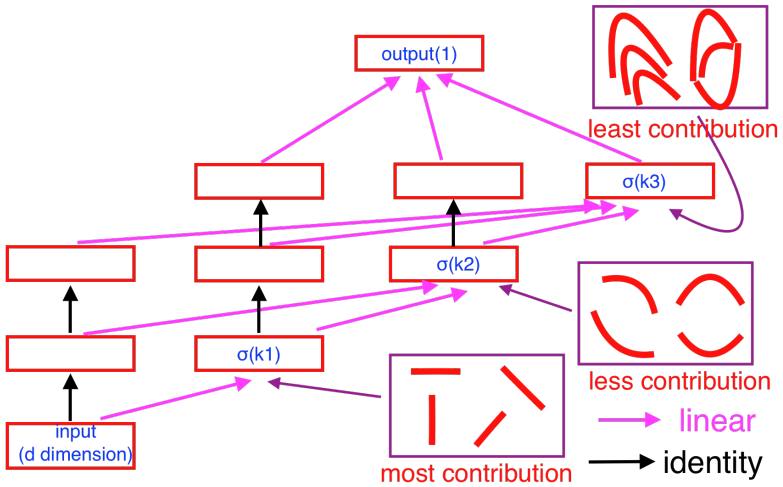
- So the label is mostly decided by $\alpha_\ell w_\ell^\top G_\ell(\Pi_{w^\perp} x)$.
- There are $\sim \alpha_\ell$ many examples that require the ℓ-layer feature to classify.

Extension to classification and the information gap α

- Recall $G(x) = \sum_{\ell \in [L]} \alpha_\ell w_\ell^\top G_\ell(x)$.
- Critical observation: With probability $\sim \alpha_\ell$ over x,

$$\left| \langle w, x \rangle + \sum_{s \in [\ell-1]} \alpha_s w_s^\top G_s(\Pi_{w^\perp} x) \right| \leq \alpha_\ell$$

- So the label is mostly decided by $\alpha_\ell w_\ell^\top G_\ell(\Pi_{w^\perp} x)$.
- There are $\sim \alpha_\ell$ many examples that require the ℓ-layer feature to classify.



Summary and future direction

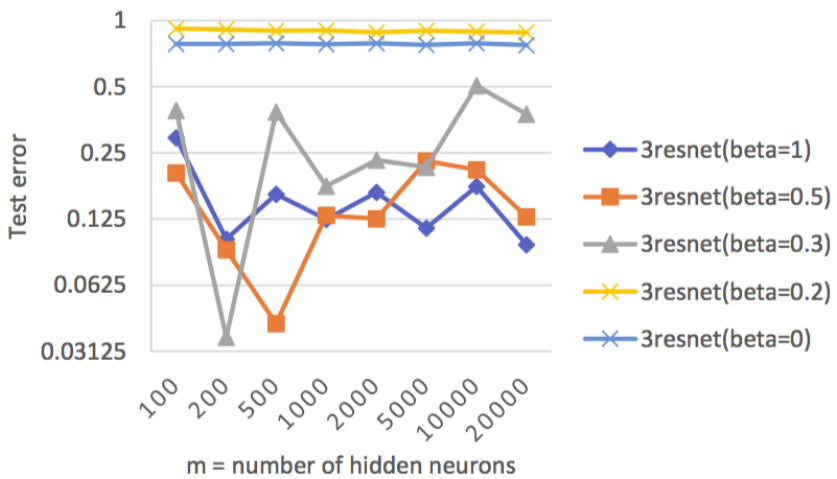
- We studied the process of hierarchical learning in over-parameterized DenseNet with quadratic activation function.

Summary and future direction

- We studied the process of hierarchical learning in over-parameterized DenseNet with quadratic activation function.
- To the best of my knowledge, no known algorithms (NTK, tensor decomposition, SOS etc) can be directly applied to this setting within the same resource (samples + running time).

Summary and future direction

- We studied the process of hierarchical learning in over-parameterized DenseNet with quadratic activation function.
- To the best of my knowledge, no known algorithms (NTK, tensor decomposition, SOS etc) can be directly applied to this setting within the same resource (samples + running time).
- TODO: Generalize to ReLU network? Remove the information gap assumption? – Our earlier paper What can ResNet efficiently learn, Going Beyond Kernels



(b) sensitivity test on $\alpha = 0.3$