Large-Scale Machine Learning

Shan-Hung Wu
shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

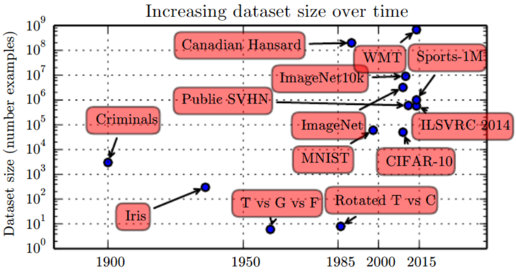
Machine Learning

Outline

1. **When ML Meets Big Data**
2. **Advantages of Deep Learning**
   - Representation Learning
   - Exponential Gain of Expressiveness
   - Memory and GPU Friendliness
   - Online & Transfer Learning
3. **Learning Theory Revisited**
   - Generalizability and Over-Parametrization
   - Wide-and-Deep NN is a Gaussian Process before Training*
   - Gradient Descent is an Affine Transformation*
   - Wide-and-Deep NN is a Gaussian Process after Training*

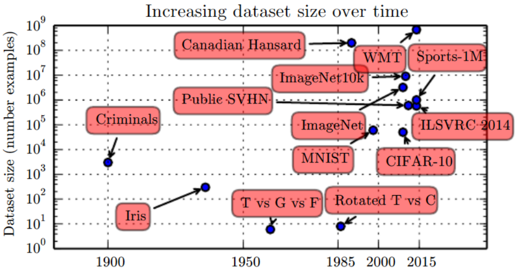
Outline

The Big Data Era



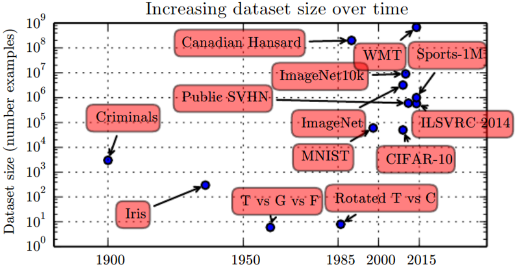
- Today, more and more of our activities are recorded by ubiquitous computing devices

The Big Data Era



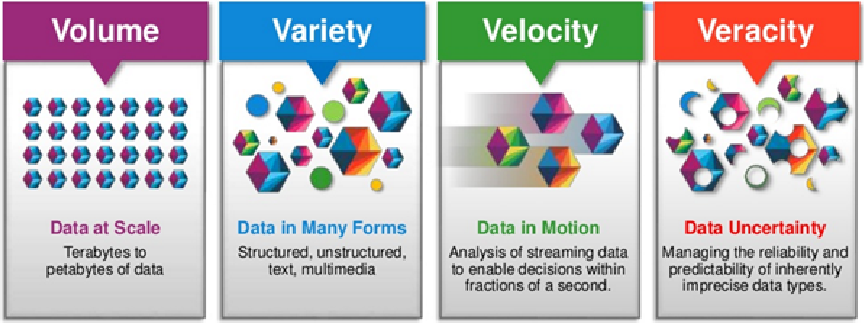
- Today, more and more of our activities are recorded by ubiquitous computing devices
- Networked computers make it easy to centralize these records and curate them into a ***big*** dataset

The Big Data Era



- Today, more and more of our activities are recorded by ubiquitous computing devices
- Networked computers make it easy to centralize these records and curate them into a *big* dataset
- ***Large-scale machine learning*** techniques solve problems by leveraging the posteriori knowledge learned from the big data

Characteristics of Big Data



Challenges of Large-Scale ML

- Variety and veracity
  - Feature engineering gets ***even harder***

Challenges of Large-Scale ML

- Variety and veracity
  - Feature engineering gets ***even harder***
  - Multi-task/transfer learning

A group of young people playing a game of Frisbee

Challenges of Large-Scale ML

- Variety and veracity
  - Feature engineering gets ***even harder***
  - Multi-task/transfer learning
- Volume
  - Large D: curse of dimensionality
  - Large N: training efficiency

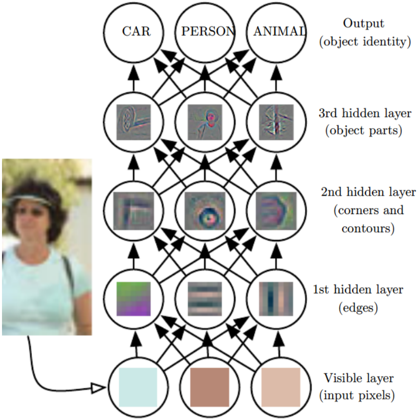
A group of young people playing a game of Frisbee

Challenges of Large-Scale ML

- Variety and veracity
  - Feature engineering gets ***even harder***
  - Multi-task/transfer learning
- Volume
  - Large D: curse of dimensionality
  - Large N: training efficiency
- Velocity
  - Online learning

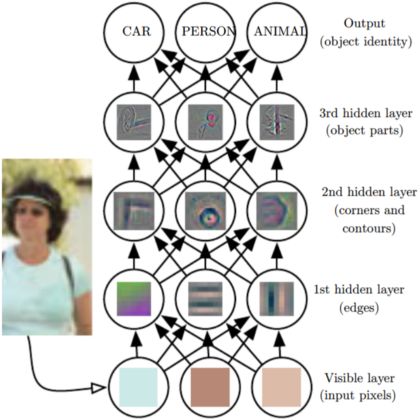
A group of young people playing a game of Frisbee

Advantages of Deep Learning



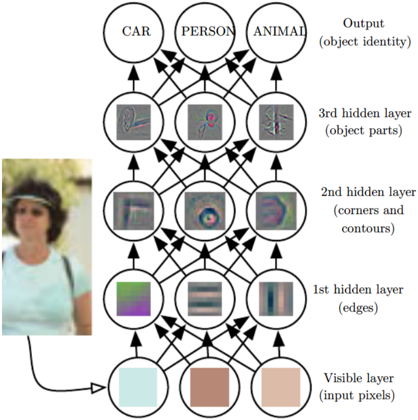
- ***Neural Networks*** (NNs) that go deep

Advantages of Deep Learning



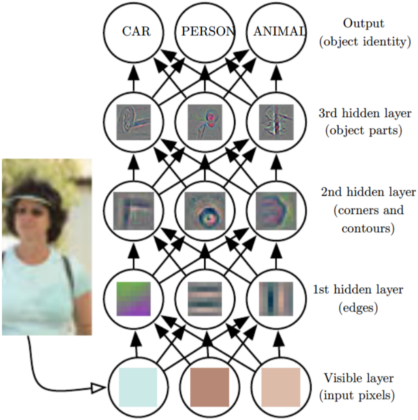
- ***Neural Networks*** (NNs) that go deep
- Automatic feature engineering
  - A kind of ***representation learning***

Advantages of Deep Learning



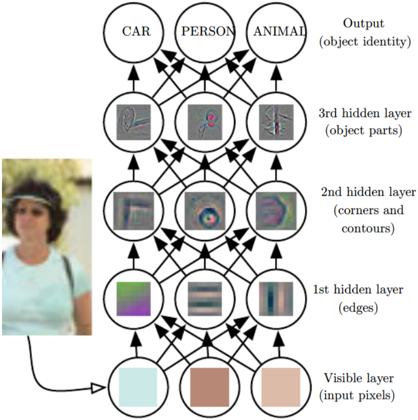
- ***Neural Networks*** (NNs) that go deep
- Automatic feature engineering
  - A kind of ***representation learning***
- Exponential gain of expressiveness
  - Counters the curse of dimensionality

Advantages of Deep Learning



- ***Neural Networks*** (NNs) that go deep
- Automatic feature engineering
  - A kind of ***representation learning***
- Exponential gain of expressiveness
  - Counters the curse of dimensionality
- Memory and GPU friendliness
  - SGD
  - GPU-based parallelism

Advantages of Deep Learning



- *Neural Networks* (NNs) that go deep
- Automatic feature engineering
  - A kind of ***representation learning***
- Exponential gain of expressiveness
  - Counters the curse of dimensionality
- Memory and GPU friendliness
  - SGD
  - GPU-based parallelism
- Supporting online & transfer learning

Is Deep Learning a Panacea?

- I have big data, so I have to use deep learning

Is Deep Learning a Panacea?

- I have big data, so I have to use deep learning
- ***Wrong!*** No free launch theorem: there is no single ML algorithm that outperforms others in every domain

Is Deep Learning a Panacea?

- I have big data, so I have to use deep learning
- ***Wrong!*** No free launch theorem: there is no single ML algorithm that outperforms others in every domain
- Deep learning is more useful when the function f to learn is ***complex*** (nonlinear to the input dimension) and has ***composited patterns***
  - E.g., image recognition, natural language processing

Is Deep Learning a Panacea?

- I have big data, so I have to use deep learning
- ***Wrong!*** No free launch theorem: there is no single ML algorithm that outperforms others in every domain
- Deep learning is more useful when the function f to learn is ***complex*** (nonlinear to the input dimension) and has ***composited patterns***
  - E.g., image recognition, natural language processing
- For simple (linear) f, there are specialized large-scale ML techniques (e.g., LIBLINEAR [7]) that are much more efficient

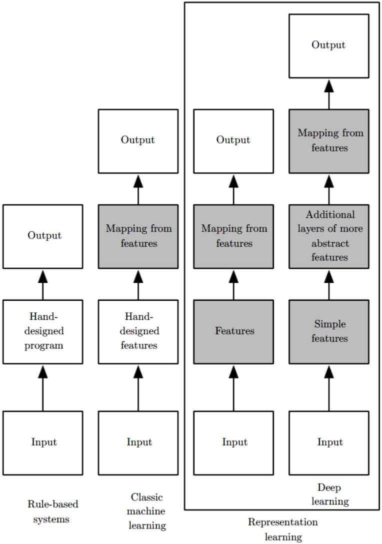
Outline

1. When ML Meets Big Data
2. Advantages of Deep Learning
    - Representation Learning
    - Exponential Gain of Expressiveness
    - Memory and GPU Friendliness
    - Online & Transfer Learning
3. Learning Theory Revisited
    - Generalizability and Over-Parametrization
    - Wide-and-Deep NN is a Gaussian Process before Training*
    - Gradient Descent is an Affine Transformation*
    - Wide-and-Deep NN is a Gaussian Process after Training*

Outline

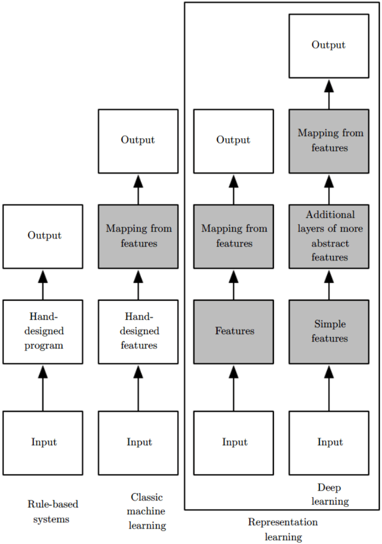
1. When ML Meets Big Data
2. Advantages of Deep Learning
   - Representation Learning
   - Exponential Gain of Expressiveness
   - Memory and GPU Friendliness
   - Online & Transfer Learning
3. Learning Theory Revisited
   - Generalizability and Over-Parametrization
   - Wide-and-Deep NN is a Gaussian Process before Training*
   - Gradient Descent is an Affine Transformation*
   - Wide-and-Deep NN is a Gaussian Process after Training*

Representation Learning



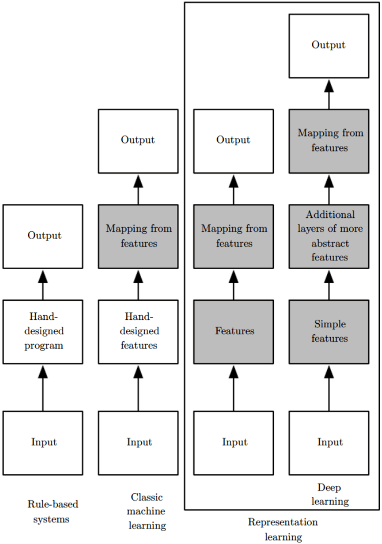
- Gray boxes are learned automatically

Representation Learning



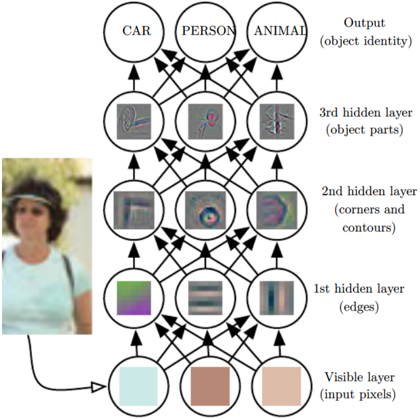
- Gray boxes are learned automatically
- Deep learning maps the ***most abstract*** (deepest) features to the output
  - Usually, a simple linear function suffices

Representation Learning



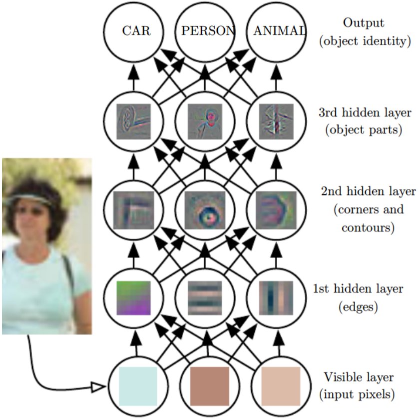
- Gray boxes are learned automatically
- Deep learning maps the ***most abstract*** (deepest) features to the output
  - Usually, a simple linear function suffices
- In deep learning, features/presentations are ***distributed***

Distributed Representations of Data



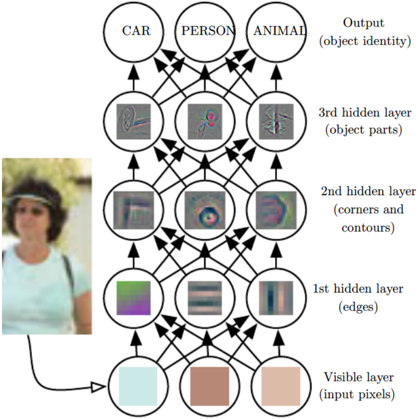
- In deep learning, we assume that $\boldsymbol{x}$'s were generated by ***compositions of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]

Distributed Representations of Data



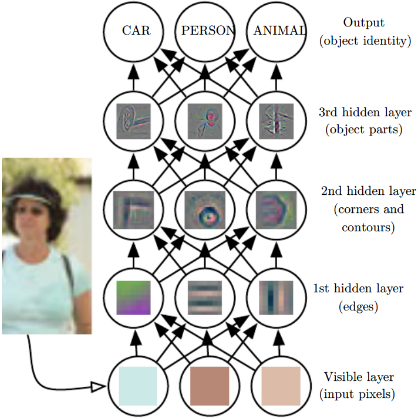
- In deep learning, we assume that $\boldsymbol{x}$'s were generated by ***compositions of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]
  - [.] a predefined non-linear function
  - Weights (arrows) learned from training examples

Distributed Representations of Data



- In deep learning, we assume that $\boldsymbol{x}$'s were generated by ***compositions of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]
  - [.] a predefined non-linear function
  - Weights (arrows) learned from training examples
- Given $\boldsymbol{x}$, factors at the same level output ***a layer of features*** of $\boldsymbol{x}$
  - Layer 2: 1, 2, 0.5 for [corner], [circle], and [curve] respectively

Distributed Representations of Data



- In deep learning, we assume that $\boldsymbol{x}$'s were generated by ***compositions of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]
  - [.] a predefined non-linear function
  - Weights (arrows) learned from training examples
- Given $\boldsymbol{x}$, factors at the same level output ***a layer of features*** of $\boldsymbol{x}$
  - Layer 2: 1, 2, 0.5 for [corner], [circle], and [curve] respectively
- To be fed into the factors in the next (deeper) level
  - Face = 0.3 * 1 + 0.7 * 2

Outline

Curse of Dimensionality

- Most classic nonlinear ML models find θ by assuming function smoothness:

$$\text{if } \boldsymbol{x} \sim \boldsymbol{x}^{(i)} \in \mathbb{X}, \text{ then } f(\boldsymbol{x};\boldsymbol{w}) \sim f(\boldsymbol{x}^{(i)};\boldsymbol{w})$$

Curse of Dimensionality

- Most classic nonlinear ML models find θ by assuming function smoothness:

$$\text{if } \boldsymbol{x} \sim \boldsymbol{x}^{(i)} \in \mathbb{X}, \text{ then } f(\boldsymbol{x};\boldsymbol{w}) \sim f(\boldsymbol{x}^{(i)};\boldsymbol{w})$$

- E.g., the non-parametric methods predict the label $\hat{y}$ of $\boldsymbol{x}$ by simply interpolating the labels of examples $\boldsymbol{x}^{(i)}$'s ***close to $\boldsymbol{x}$***:

$$\hat{y} = \sum_i \alpha_i y^{(i)} k(\boldsymbol{x}^{(i)}, \boldsymbol{x}) + b, \text{ where } k(\boldsymbol{x}^{(i)}, \boldsymbol{x}) = \exp(-\gamma \|\boldsymbol{x}^{(i)} - \boldsymbol{x}\|^2)$$

Curse of Dimensionality

- Most classic nonlinear ML models find θ by assuming function smoothness:

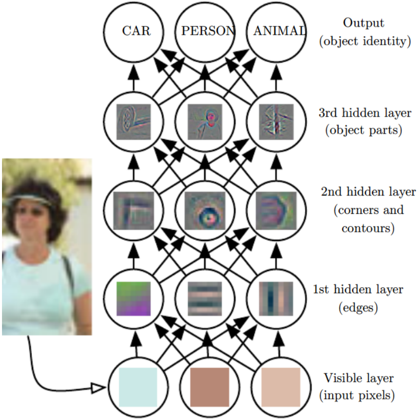
$$\text{if } \boldsymbol{x} \sim \boldsymbol{x}^{(i)} \in \mathbb{X}, \text{ then } f(\boldsymbol{x};\boldsymbol{w}) \sim f(\boldsymbol{x}^{(i)};\boldsymbol{w})$$

- E.g., the non-parametric methods predict the label $\hat{y}$ of $\boldsymbol{x}$ by simply interpolating the labels of examples $\boldsymbol{x}^{(i)}$'s ***close to $\boldsymbol{x}$***:

$$\hat{y} = \sum_i \alpha_i y^{(i)} k(\boldsymbol{x}^{(i)},\boldsymbol{x}) + b, \text{ where } k(\boldsymbol{x}^{(i)},\boldsymbol{x}) = \exp(-\gamma\|\boldsymbol{x}^{(i)} - \boldsymbol{x}\|^2)$$

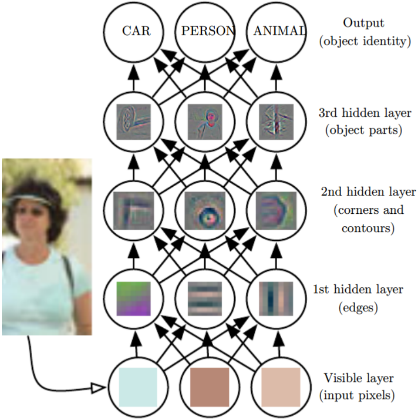
- Suppose f is smooth within a bin, we need ***exponentially more examples*** to get a good interpolation as D increases

Exponential Gains from Depth I



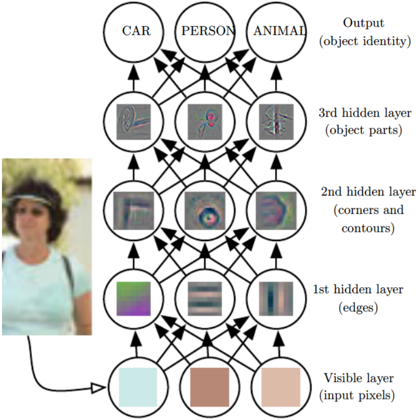
- Functions representable with a deep rectifier NN require an exponential number of hidden units in a shallow NN [13]

Exponential Gains from Depth I



- Functions representable with a deep rectifier NN require an exponential number of hidden units in a shallow NN [13]
- In deep learning, a deep factor is defined by "reusing" the shallow ones
  - Face = 0.3 [corner] + 0.7 [circle]

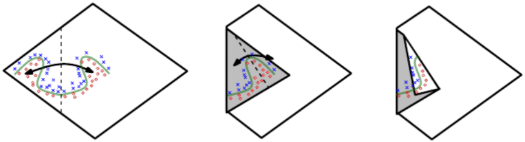
Exponential Gains from Depth I



- Functions representable with a deep rectifier NN require an exponential number of hidden units in a shallow NN [13]
- In deep learning, a deep factor is defined by "reusing" the shallow ones
  - Face = 0.3 [corner] + 0.7 [circle]
- With a shallow structure, a deep factor needs to be replaced by ***exponentially many*** factors
  - Face = 0.3 [0.5 [vertical] + 0.5 [horizontal]] + 0.7 [...]

Exponential Gains from Depth II

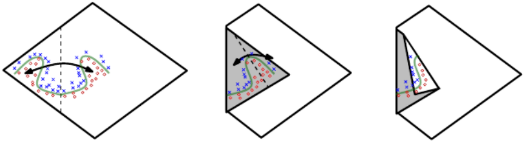
- Another example: an NN with absolute value rectification units



- Each hidden unit specifies where to fold the input space in order to create mirror responses (on both sides of the absolute value)
- A single fold in a deep layer creates an exponentially large number of piecewise linear regions in input space
  - No need to see examples in each linear regions in input space

Exponential Gains from Depth II

- Another example: an NN with absolute value rectification units



- Each hidden unit specifies where to fold the input space in order to create mirror responses (on both sides of the absolute value)
- A single fold in a deep layer creates an exponentially large number of piecewise linear regions in input space
  - No need to see examples in each linear regions in input space
- This exponential gain counters the exponential challenges posed by the curse of dimensionality

Outline

Stochastic Gradient Descent

Gradient Descent (GD)

$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence $\{$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C_N(\boldsymbol{w}^{(t)}; \mathbb{X});$$

$\}$

Stochastic Gradient Descent

Gradient Descent (GD)

$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence {
$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C_N(\boldsymbol{w}^{(t)}; \mathbb{X})$;
}

- Needs to scan the entire dataset to descent (many I/Os)

Stochastic Gradient Descent

Gradient Descent (GD)

$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence {
  $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C_N(\boldsymbol{w}^{(t)}; \mathbb{X})$;
}

- Needs to scan the entire dataset to descent (many I/Os)

(Mini-Batched) Stochastic Gradient Descent (SGD)

$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence {
  Randomly partition the training set $\mathbb{X}$ into ***minibatches*** $\{\mathbb{X}^{(j)}\}_j$;
  $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C(\boldsymbol{w}^{(t)}; \mathbb{X}^{(j)})$;
}

Stochastic Gradient Descent

Gradient Descent (GD)

$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence {
  $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C_N(\boldsymbol{w}^{(t)}; \mathbb{X})$;
}

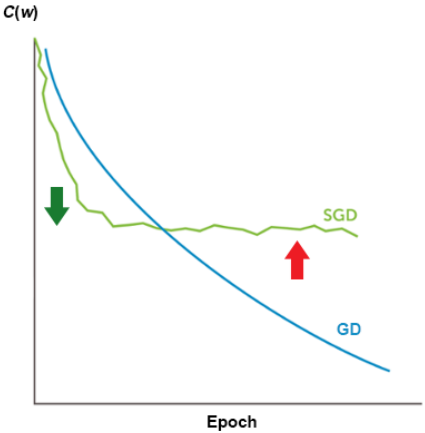
- Needs to scan the entire dataset to descent (many I/Os)

(Mini-Batched) Stochastic Gradient Descent (SGD)

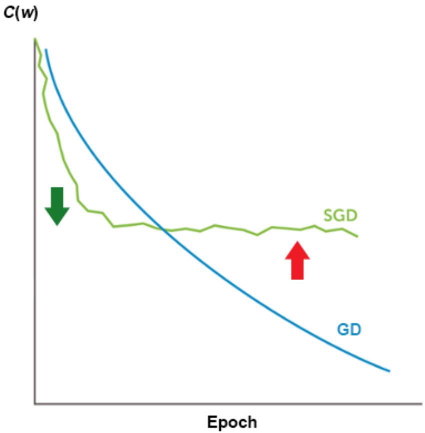
$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence {
  Randomly partition the training set $\mathbb{X}$ into ***minibatches*** $\{\mathbb{X}^{(j)}\}_j$;
  $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C(\boldsymbol{w}^{(t)}; \mathbb{X}^{(j)})$;
}

- No I/O if the next mini-batch can be prefetched

GD vs. SGD

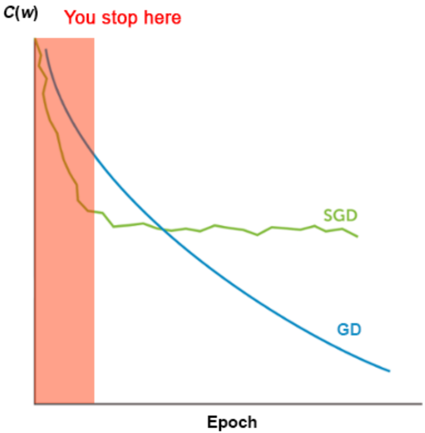


GD vs. SGD



- Is SGD really a better algorithm?

Yes, If You Have Big Data



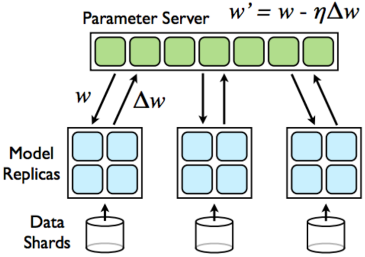
- Performance is limited by ***training time***

Asymptotic Analysis [4]

	GD	SGD
Time per iteration	N	1
#Iterations to opt. error ρ	$\log \frac{1}{\rho}$	$\frac{1}{\rho}$
Time to opt. error ρ	$N \log \frac{1}{\rho}$	$\frac{1}{\rho}$
Time to excess error ε	$\frac{1}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon}$, where $\alpha \in [\frac{1}{2}, 1]$	$\frac{1}{\varepsilon}$

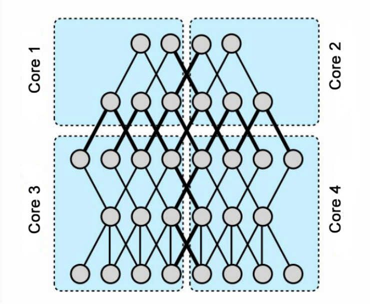
Parallelizing SGD

Data Parallelism



Every core/GPU trains the full model given partitioned data.

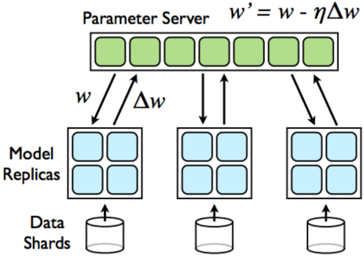
Model Parallelism



Every core/GPU train a partitioned model given full data.

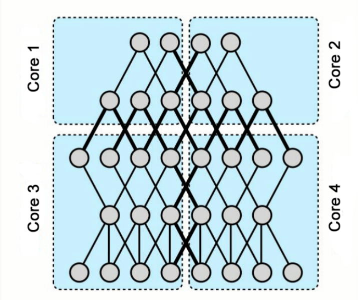
Parallelizing SGD

Data Parallelism



Every core/GPU trains the full model given partitioned data.

Model Parallelism



Every core/GPU train a partitioned model given full data.

- The effectiveness depends on applications and available hardware
  - E.g., CPU/GPU speed, communication latency, bandwidth, etc.

Outline

1. When ML Meets Big Data
2. Advantages of Deep Learning
   - Representation Learning
   - Exponential Gain of Expressiveness
   - Memory and GPU Friendliness
   - Online & Transfer Learning
3. Learning Theory Revisited
   - Generalizability and Over-Parametrization
   - Wide-and-Deep NN is a Gaussian Process before Training*
   - Gradient Descent is an Affine Transformation*
   - Wide-and-Deep NN is a Gaussian Process after Training*

Online Learning

- So far, we assume that the training data $\mathbb{X}$ comes at once
- What if data come sequentially?

Online Learning

- So far, we assume that the training data $\mathbb{X}$ comes at once
- What if data come sequentially?
- ***Online learning***: to update model when new data arrive

Online Learning

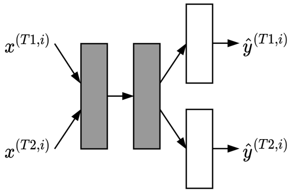
- So far, we assume that the training data $\mathbb{X}$ comes at once
- What if data come sequentially?
- ***Online learning***: to update model when new data arrive
- This a already supported by SGD

Muti-Task and Transfer Learning

- ***Multi-task learning***: to learning a single model for multiple tasks
- ***Transfer learning***: to reuse the knowledge learned from one task to help another

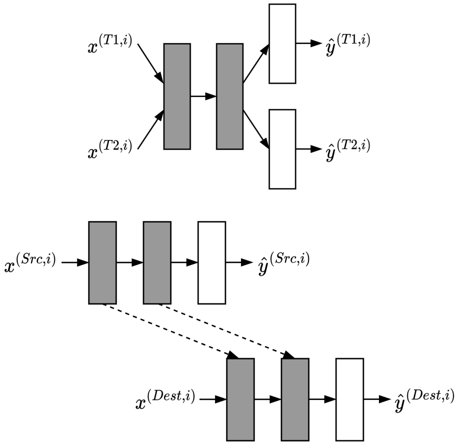
Muti-Task and Transfer Learning

- ***Multi-task learning***: to learning a single model for multiple tasks
  - Via shared layers
- ***Transfer learning***: to reuse the knowledge learned from one task to help another



Muti-Task and Transfer Learning

- ***Multi-task learning***: to learning a single model for multiple tasks
  - Via shared layers
- ***Transfer learning***: to reuse the knowledge learned from one task to help another
  - Via pretrained layers (whose weights may be further updated when a smaller learning rate)



Outline

Learning Theory

- How to learn a function f_N from N examples $\mathbb{X}$ that is close to the true function f^*?

Learning Theory

- How to learn a function f_N from N examples $\mathbb{X}$ that is close to the true function f^*?
- ***Empirical risk***: $C_N[f_N] = \frac{1}{N}\sum_{i=1}^{N} loss(f_N(\boldsymbol{x}^{(i)}), y^{(i)})$
- ***Expected risk***: $C[f_N] = \int loss(f(\boldsymbol{x}), y) dP(\boldsymbol{x}, y)$

Learning Theory

- How to learn a function f_N from N examples $\mathbb{X}$ that is close to the true function f^*?
- ***Empirical risk***: $C_N[f_N] = \frac{1}{N}\sum_{i=1}^{N} loss(f_N(\boldsymbol{x}^{(i)}), y^{(i)})$
- ***Expected risk***: $C[f_N] = \int loss(f(\boldsymbol{x}), y) dP(\boldsymbol{x}, y)$
- Let $f^* = \arg\min_f C[f]$ be the true function (our goal)

Learning Theory

- How to learn a function f_N from N examples $\mathbb{X}$ that is close to the true function f^*?
- ***Empirical risk***: $C_N[f_N] = \frac{1}{N}\sum_{i=1}^{N} loss(f_N(\boldsymbol{x}^{(i)}), y^{(i)})$
- ***Expected risk***: $C[f_N] = \int loss(f(\boldsymbol{x}), y) dP(\boldsymbol{x}, y)$
- Let $f^* = \arg\min_f C[f]$ be the true function (our goal)
- Since we are seeking a function in a model (hypothesis space) $\mathbb{F}$, this is what can have at best: $f^*_{\mathbb{F}} = \arg\min_{f \in \mathbb{F}} C[f]$

Learning Theory

- How to learn a function f_N from N examples $\mathbb{X}$ that is close to the true function f^*?
- ***Empirical risk***: $C_N[f_N] = \frac{1}{N}\sum_{i=1}^{N} loss(f_N(\boldsymbol{x}^{(i)}), y^{(i)})$
- ***Expected risk***: $C[f_N] = \int loss(f(\boldsymbol{x}), y) dP(\boldsymbol{x}, y)$
- Let $f^* = \arg\min_f C[f]$ be the true function (our goal)
- Since we are seeking a function in a model (hypothesis space) $\mathbb{F}$, this is what can have at best: $f^*_{\mathbb{F}} = \arg\min_{f\in\mathbb{F}} C[f]$
- But we only minimizes errors on limited examples in our objective, so we only have $f_N = \arg\min_{f\in\mathbb{F}} C_N[f]$

Learning Theory

- How to learn a function f_N from N examples $\mathbb{X}$ that is close to the true function f^*?
- ***Empirical risk***: $C_N[f_N] = \frac{1}{N}\sum_{i=1}^{N} loss(f_N(\boldsymbol{x}^{(i)}), y^{(i)})$
- ***Expected risk***: $C[f_N] = \int loss(f(\boldsymbol{x}), y) dP(\boldsymbol{x}, y)$
- Let $f^* = \arg\min_f C[f]$ be the true function (our goal)
- Since we are seeking a function in a model (hypothesis space) $\mathbb{F}$, this is what can have at best: $f^*_{\mathbb{F}} = \arg\min_{f\in\mathbb{F}} C[f]$
- But we only minimizes errors on limited examples in our objective, so we only have $f_N = \arg\min_{f\in\mathbb{F}} C_N[f]$
- The ***excess error*** $\mathscr{E} = C[f_N] - C[f^*]$:

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}}$$

Excess Error

- Wait, we may not have enough training time, so we stop iterations early and have $\tilde{f}_N$, where $C_N[\tilde{f}_N] \leq C_N[f_N] + \rho$

Excess Error

- Wait, we may not have enough training time, so we stop iterations early and have $\tilde{f}_N$, where $C_N[\tilde{f}_N] \leq C_N[f_N] + \rho$
- The excess error becomes $\mathscr{E} = C[\tilde{f}_N] - C[f^*]$:

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

Excess Error

- Wait, we may not have enough training time, so we stop iterations early and have $\tilde{f}_N$, where $C_N[\tilde{f}_N] \leq C_N[f_N] + \rho$
- The excess error becomes $\mathscr{E} = C[\tilde{f}_N] - C[f^*]$:

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- ***Approximation error $\mathscr{E}_{\text{app}}$***: reduced by choosing a larger model

Excess Error

- Wait, we may not have enough training time, so we stop iterations early and have $\tilde{f}_N$, where $C_N[\tilde{f}_N] \leq C_N[f_N] + \rho$
- The excess error becomes $\mathscr{E} = C[\tilde{f}_N] - C[f^*]$:

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- ***Approximation error $\mathscr{E}_{app}$***: reduced by choosing a larger model
- ***Estimation error $\mathscr{E}_{est}$***: reduced by
  1. Increasing N, or
  2. Choosing smaller model [5, 12, 15]

Excess Error

- Wait, we may not have enough training time, so we stop iterations early and have $\tilde{f}_N$, where $C_N[\tilde{f}_N] \leq C_N[f_N] + \rho$
- The excess error becomes $\mathscr{E} = C[\tilde{f}_N] - C[f^*]$:

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- ***Approximation error*** $\mathscr{E}_{app}$: reduced by choosing a larger model
- ***Estimation error*** $\mathscr{E}_{est}$: reduced by
  1. Increasing N, or
  2. Choosing smaller model [5, 12, 15]
- ***Optimization error*** $\mathscr{E}_{opt}$: reduced by
  1. Running optimization alg. longer (with smaller ρ)
  2. Choosing more efficient optimization alg.

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\textsf{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\textsf{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\textsf{opt}}}$$

- Small-scale ML tasks:

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\textsf{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\textsf{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\textsf{opt}}}$$

- Small-scale ML tasks:
  - Mainly constrained by N

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\textsf{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\textsf{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\textsf{opt}}}$$

- Small-scale ML tasks:
  - Mainly constrained by N
  - Computing time is not an issue, so $\mathscr{E}_{\textsf{opt}}$ can be insignificant by choosing small ρ

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f_{\mathbb{F}}^*] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f_{\mathbb{F}}^*]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- Small-scale ML tasks:
  - Mainly constrained by N
  - Computing time is not an issue, so $\mathscr{E}_{\text{opt}}$ can be insignificant by choosing small ρ
  - Size of hypothesis is important to balance the trade-off between $\mathscr{E}_{\text{app}}$ and $\mathscr{E}_{\text{est}}$

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- Small-scale ML tasks:
  - Mainly constrained by N
  - Computing time is not an issue, so $\mathscr{E}_{\text{opt}}$ can be insignificant by choosing small ρ
  - Size of hypothesis is important to balance the trade-off between $\mathscr{E}_{\text{app}}$ and $\mathscr{E}_{\text{est}}$
- Large-scale ML tasks:

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- Small-scale ML tasks:
  - Mainly constrained by N
  - Computing time is not an issue, so $\mathscr{E}_{\text{opt}}$ can be insignificant by choosing small ρ
  - Size of hypothesis is important to balance the trade-off between $\mathscr{E}_{\text{app}}$ and $\mathscr{E}_{\text{est}}$
- Large-scale ML tasks:
  - Mainly constrained by time (significant $\mathscr{E}_{\text{opt}}$), so ***SGD*** is preferred

Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

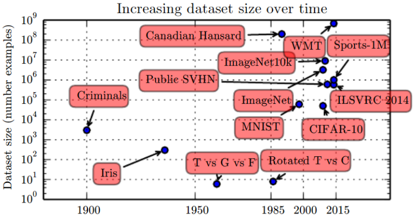
- Small-scale ML tasks:
  - Mainly constrained by N
  - Computing time is not an issue, so $\mathscr{E}_{\text{opt}}$ can be insignificant by choosing small ρ
  - Size of hypothesis is important to balance the trade-off between $\mathscr{E}_{\text{app}}$ and $\mathscr{E}_{\text{est}}$
- Large-scale ML tasks:
  - Mainly constrained by time (significant $\mathscr{E}_{\text{opt}}$), so ***SGD*** is preferred
  - N is large, so $\mathscr{E}_{\text{est}}$ can be reduced

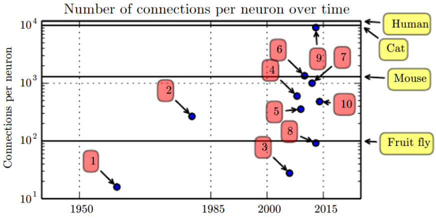
Minimizing Excess Error

$$\mathscr{E} = \underbrace{C[f^*_{\mathbb{F}}] - C[f^*]}_{\mathscr{E}_{\text{app}}} + \underbrace{C[f_N] - C[f^*_{\mathbb{F}}]}_{\mathscr{E}_{\text{est}}} + \underbrace{C[\tilde{f}_N] - C[f_N]}_{\mathscr{E}_{\text{opt}}}$$

- Small-scale ML tasks:
  - Mainly constrained by N
  - Computing time is not an issue, so $\mathscr{E}_{\text{opt}}$ can be insignificant by choosing small ρ
  - Size of hypothesis is important to balance the trade-off between $\mathscr{E}_{\text{app}}$ and $\mathscr{E}_{\text{est}}$
- Large-scale ML tasks:
  - Mainly constrained by time (significant $\mathscr{E}_{\text{opt}}$), so ***SGD*** is preferred
  - N is large, so $\mathscr{E}_{\text{est}}$ can be reduced
  - ***Large model*** is preferred to reduce $\mathscr{E}_{\text{app}}$

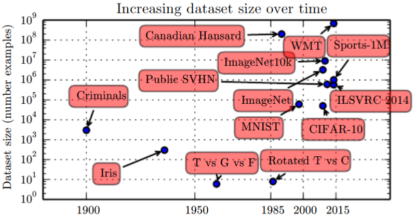
Big Data + Big Models

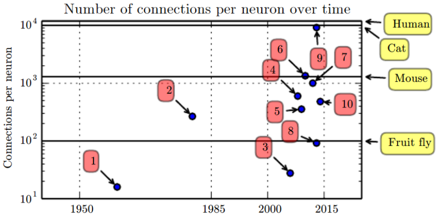




9. COTS HPC unsupervised convolutional network [6]
10. GoogleLeNet [14]

Big Data + Big Models





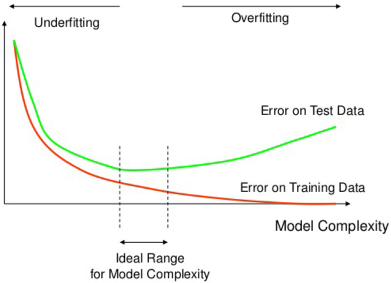
9. COTS HPC unsupervised convolutional network [6]
10. GoogLeNet [14]

- With domain-specific architecture such as ***convolutional NNs*** (CNNs) and ***recurrent NNs*** (RNNs)

Outline

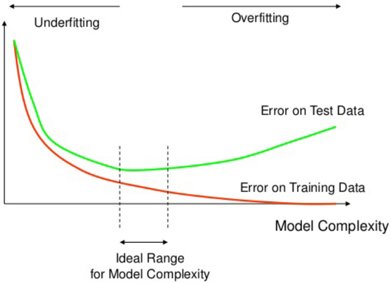
Over-Parametrized NNs

- Let $D^{(l)}$ be the output dimension ("width") of a layer $f^{(l)}(\cdot\,;\boldsymbol{\theta}^{(l)})$ of an NN
  - Input/output dimension: $(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}^{D^{(0)}} \times \mathbb{R}^{D^{(L)}}$
  - $D = \min(D^{(0)},\cdots,D^{(L)})$ the network width
- From the statistical learning theory point of view, the larger the D, the worse the generalizability



Over-Parametrized NNs

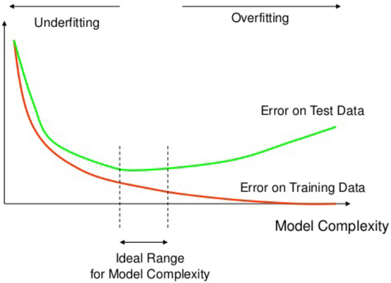
- Let $D^{(l)}$ be the output dimension ("width") of a layer $f^{(l)}(\cdot;\boldsymbol{\theta}^{(l)})$ of an NN
  - Input/output dimension: $(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}^{D^{(0)}} \times \mathbb{R}^{D^{(L)}}$
  - $D = \min(D^{(0)},\cdots,D^{(L)})$ the network width
- From the statistical learning theory point of view, the larger the D, the worse the generalizability



- However, as D grows, the generalizability actually ***increases*** [20]; i.e., over-parametrization leads to better performance

Over-Parametrized NNs

- Let $D^{(l)}$ be the output dimension ("width") of a layer $f^{(l)}(\cdot;\boldsymbol{\theta}^{(l)})$ of an NN
  - Input/output dimension: $(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}^{D^{(0)}} \times \mathbb{R}^{D^{(L)}}$
  - $D = \min(D^{(0)},\cdots,D^{(L)})$ the network width
- From the statistical learning theory point of view, the larger the D, the worse the generalizability



- However, as D grows, the generalizability actually ***increases*** [20]; i.e., over-parametrization leads to better performance
- Why such a paradox?

Wide-and-Deep NNs as Gaussian Processes

- Recent studies [10, 9, 11] show that ***a wide NN of any depth can be approximated by a Gaussian process (GP)***
  - Either before, during, or after training
- Recall that a GP is a non-parametric model whose complexity depends only on the size of training set $|\mathbb{X}|$ and the hyperparameters of kernel function $k(\cdot,\cdot)$:

$$\begin{bmatrix} \boldsymbol{y}_N \\ \boldsymbol{y}_M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}_N \\ \boldsymbol{m}_M \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{N,N} & \boldsymbol{K}_{N,M} \\ \boldsymbol{K}_{M,N} & \boldsymbol{K}_{M,M} \end{bmatrix} \right)$$

with Bayesian inference for test points $\mathbb{X}'$:

$$\mathrm{P}(\boldsymbol{y}_M \,|\, \mathbb{X}', \mathbb{X}) = \mathcal{N}\left(\boldsymbol{K}_{M,N}\boldsymbol{K}_{N,N}^{-1}\boldsymbol{y}_N\,,\, \boldsymbol{K}_{M,M} - \boldsymbol{K}_{M,N}\boldsymbol{K}_{N,N}^{-1}\boldsymbol{K}_{N,M}\right)$$

Wide-and-Deep NNs as Gaussian Processes

- Recent studies [10, 9, 11] show that ***a wide NN of any depth can be approximated by a Gaussian process (GP)***
  - Either before, during, or after training
- Recall that a GP is a non-parametric model whose complexity depends only on the size of training set $|\mathbb{X}|$ and the hyperparameters of kernel function $k(\cdot,\cdot)$:

$$\begin{bmatrix} \boldsymbol{y}_N \\ \boldsymbol{y}_M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}_N \\ \boldsymbol{m}_M \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{N,N} & \boldsymbol{K}_{N,M} \\ \boldsymbol{K}_{M,N} & \boldsymbol{K}_{M,M} \end{bmatrix}\right)$$

with Bayesian inference for test points $\mathbb{X}'$:

$$\mathrm{P}(\boldsymbol{y}_M \,|\, \mathbb{X}', \mathbb{X}) = \mathcal{N}\left(\boldsymbol{K}_{M,N}\boldsymbol{K}_{N,N}^{-1}\boldsymbol{y}_N \,,\, \boldsymbol{K}_{M,M} - \boldsymbol{K}_{M,N}\boldsymbol{K}_{N,N}^{-1}\boldsymbol{K}_{N,M}\right)$$

- Therefore, wide-and-deep NNs do not overfit as one may expect
  - The D, once becoming large, does ***not*** reflect true model complexity

Outline

Example: NN for Regression

- For simplicity, we consider an L-layer NN $f(\cdot;\theta)$ for the regression problem:

$$f(\boldsymbol{x};\theta)=\boldsymbol{a}^{(l)}=\phi^{(l)}(\boldsymbol{W}^{(l)\top}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}),\text{ for } l=1,\ldots,L,$$

where

  - the activation functions $\phi^{(1)}(\cdot)=\cdots=\phi^{(L-1)}(\cdot)\equiv\phi(\cdot)$ and $\phi^{(L-1)}(\cdot)$ is an identify function
  - $\boldsymbol{a}^{(0)}=\boldsymbol{x}$ and $\hat{y}=a^{(L)}=z^{(L)}\in\mathbb{R}$ the mean of a Gaussian
  - $\theta^{(l)}=\mathrm{vec}(\boldsymbol{W}^{(l)},\boldsymbol{b}^{(l)})$ and $\theta=\mathrm{vec}(\theta^{(1)},\cdots,\theta^{(L)})$

Example: NN for Regression

- For simplicity, we consider an L-layer NN $f(\cdot;\theta)$ for the regression problem:

$$f(\boldsymbol{x};\theta) = \boldsymbol{a}^{(l)} = \phi^{(l)}(\boldsymbol{W}^{(l)\top}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}), \text{ for } l = 1,\ldots,L,$$

  where
  - the activation functions $\phi^{(1)}(\cdot) = \cdots = \phi^{(L-1)}(\cdot) \equiv \phi(\cdot)$ and $\phi^{(L-1)}(\cdot)$ is an identify function
  - $\boldsymbol{a}^{(0)} = \boldsymbol{x}$ and $\hat{y} = a^{(L)} = z^{(L)} \in \mathbb{R}$ the mean of a Gaussian
  - $\theta^{(l)} = \mathrm{vec}(\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)})$ and $\theta = \mathrm{vec}(\theta^{(1)}, \cdots, \theta^{(L)})$
- Let $\hat{\boldsymbol{y}}_N = [f(\boldsymbol{x}^{(1)};\theta), \cdots, f(\boldsymbol{x}^{(N)};\theta)]^\top \in \mathbb{R}^N$ be the predictions for the points in training set $\mathbb{X} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^N = \{\boldsymbol{X}_N \in \mathbb{R}^{N\times D^{(0)}}, \boldsymbol{y}_N \in \mathbb{R}^N\}$
- Maximum-likelihood estimation:

$$\arg\max_\theta \mathrm{P}(\mathbb{X}\,|\,\theta) = \arg\min_\theta C(\hat{\boldsymbol{y}}_N, \boldsymbol{y}_N) = \arg\min_\theta \frac{1}{2}\|\hat{\boldsymbol{y}}_N - \boldsymbol{y}_N\|^2$$

Weight Initialization and Normalization

$$\boldsymbol{a}^{(l)} = \phi^{(l)}(\boldsymbol{W}^{(l)\top}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})$$

- Common initialization: $W_{i,j}^{(l)} \sim \mathcal{N}(0, \sigma_w^2)$ and $b_i^{(l)} \sim \mathcal{N}(0, \sigma_b^2)$

Weight Initialization and Normalization

$$\boldsymbol{a}^{(l)} = \phi^{(l)}(\boldsymbol{W}^{(l)\top}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})$$

- Common initialization: $W_{i,j}^{(l)} \sim \mathcal{N}(0, \sigma_w^2)$ and $b_i^{(l)} \sim \mathcal{N}(0, \sigma_b^2)$
- To normalize the forward and backward gradient signals w.r.t. layer width $D^{(l)}$, we can define an equivalent NN:

$$\boldsymbol{a}^{(l)} = \phi^{(l)}(\boldsymbol{W}^{(l)\top}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}),$$

where $W_{i,j}^{(l)} = \frac{\sigma_w}{\sqrt{D^{(l-1)}}}\omega_{i,j}^{(l)}$, $b_i^{(l)} = \sigma_b \beta_i^{(l)}$, and $\omega_{i,j}^{(l)}, \beta_i^{(l)} \sim \mathcal{N}(0,1)$

Distribution of $\hat{y}$

- Given an $\boldsymbol{x}$, what is the distribution of its prediction $\hat{y}$?

Distribution of $\hat{y}$

- Given an $\boldsymbol{x}$, what is the distribution of its prediction $\hat{y}$?
- Recall that

$$\hat{y} = z^{(L)} = \boldsymbol{w}^{(L)\top}\boldsymbol{a}^{(L-1)} + b^{(L)} = \frac{\sigma_w}{\sqrt{D^{(l-1)}}}\Sigma_j \omega_j^{(L)} \phi(z_j^{(L-1)}) + \sigma_b \beta^{(L)}$$

- Since $\omega_j^{(l)}$'s and $\beta^{(l)}$ are Gaussian random variables with zero means, their sum $\hat{y}$ is also a zero-mean Gaussian

Distribution of $\hat{\boldsymbol{y}}$

- Given an $\boldsymbol{x}$, what is the distribution of its prediction $\hat{y}$?
- Recall that

$$\hat{y} = z^{(L)} = \boldsymbol{w}^{(L)\top}\boldsymbol{a}^{(L-1)} + b^{(L)} = \frac{\sigma_w}{\sqrt{D^{(l-1)}}}\Sigma_j \omega_j^{(L)} \phi(z_j^{(L-1)}) + \sigma_b \beta^{(L)}$$

- Since $\omega_j^{(l)}$'s and $\beta^{(l)}$ are Gaussian random variables with zero means, their sum $\hat{y}$ is also a zero-mean Gaussian
- Now consider the predictions $\hat{\boldsymbol{y}}_N = [\hat{y}(\boldsymbol{x}^{(1)}), \cdots, \hat{y}(\boldsymbol{x}^{(N)})]^\top \in \mathbb{R}^N$ for N points, we have

$$\begin{bmatrix} \hat{y}(\boldsymbol{x}^{(1)}) \\ \vdots \\ \hat{y}(\boldsymbol{x}^{(N)}) \end{bmatrix} = \frac{\sigma_w}{\sqrt{D^{(l-1)}}}\Sigma_j \omega_{j,i}^{(l)} \begin{bmatrix} \phi(z_j^{(l-1)}(\boldsymbol{x}^{(1)})) \\ \vdots \\ \phi(z_j^{(l-1)}(\boldsymbol{x}^{(N)})) \end{bmatrix} + \sigma_b \beta_i^{(l)} \mathbf{1}_N$$

- As $D^{(L-1)} \to \infty$, by multidimensional Central Limit Theorem, $\hat{\boldsymbol{y}}$ is a multivariate Gaussian with mean $\mathbf{0}_N$ and covariance Σ

Wide-and-Deep NN as a Gaussian Process

- The covariance Σ completely describes the behavior of our NN $\hat{y}(\cdot) = f(\cdot)$ over N points
- Furthermore, we will show that Σ can be describe by a ***deterministic*** kernel function $k^{(L)}(\cdot,\cdot)$ independent of a particular initialization such that

$$\Sigma = \begin{bmatrix} k^{(L)}(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(1)}) & \cdots & k^{(L)}(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(N)}) \\ \vdots & \ddots & \vdots \\ k^{(L)}(\boldsymbol{x}^{(N)},\boldsymbol{x}^{(1)}) & \cdots & k^{(L)}(\boldsymbol{x}^{(N)},\boldsymbol{x}^{(N)}) \end{bmatrix} \equiv \boldsymbol{K}_{N,N}^{(L)}$$

- This implies that the NN is in correspondent with a GP called ***NN-GP***:

$$\begin{bmatrix} \hat{\boldsymbol{y}}_N \\ \hat{\boldsymbol{y}}_M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{0}_N \\ \boldsymbol{0}_M \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{N,N}^{(L)} & \boldsymbol{K}_{N,M}^{(L)} \\ \boldsymbol{K}_{M,N}^{(L)} & \boldsymbol{K}_{M,M}^{(L)} \end{bmatrix} \right)$$

Wide-and-Deep NN as a Gaussian Process

- The covariance Σ completely describes the behavior of our NN $\hat{y}(\cdot) = f(\cdot)$ over N points
- Furthermore, we will show that Σ can be describe by a ***deterministic*** kernel function $k^{(L)}(\cdot,\cdot)$ independent of a particular initialization such that

$$\Sigma = \begin{bmatrix} k^{(L)}(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(1)}) & \cdots & k^{(L)}(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(N)}) \\ \vdots & \ddots & \vdots \\ k^{(L)}(\boldsymbol{x}^{(N)},\boldsymbol{x}^{(1)}) & \cdots & k^{(L)}(\boldsymbol{x}^{(N)},\boldsymbol{x}^{(N)}) \end{bmatrix} \equiv \boldsymbol{K}^{(L)}_{N,N}$$

- This implies that the NN is in correspondent with a GP called ***NN-GP***:

$$\begin{bmatrix} \hat{\boldsymbol{y}}_N \\ \hat{\boldsymbol{y}}_M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{0}_N \\ \boldsymbol{0}_M \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}^{(L)}_{N,N} & \boldsymbol{K}^{(L)}_{N,M} \\ \boldsymbol{K}^{(L)}_{M,N} & \boldsymbol{K}^{(L)}_{M,M} \end{bmatrix} \right)$$

- What's the $k^{(L)}(\cdot,\cdot)$?

Deriving $k^{(1)}(\cdot,\cdot)$

- We use induction to show that $z_i^{(1)}(\cdot), z_i^{(2)}(\cdot), \cdots, z^{(L)}(\cdot) = \hat{y}(\cdot)$ are GPs, which are govern by kernels $k^{(1)}(\cdot,\cdot), \cdots, k^{(L)}(\cdot,\cdot)$ independent with i, respectively

Deriving $k^{(1)}(\cdot,\cdot)$

- We use induction to show that $z_i^{(1)}(\cdot), z_i^{(2)}(\cdot), \cdots, z^{(L)}(\cdot) = \hat{y}(\cdot)$ are GPs, which are govern by kernels $k^{(1)}(\cdot,\cdot), \cdots, k^{(L)}(\cdot,\cdot)$ independent with i, respectively
- Consider $z_i^{(1)}(\boldsymbol{x}) = \frac{\sigma_w}{\sqrt{D^{(0)}}}\Sigma_j \omega_{j,i}^{(l)} x_j + \sigma_b \beta_i^{(l)}$ a zero-mean Gaussian
- As $D^{(0)} \to \infty$, we have $[z_i^{(1)}(\boldsymbol{x}^{(1)}), \cdots, z_i^{(1)}(\boldsymbol{x}^{(N)})]^\top \sim N(\mathbf{0}_N, \boldsymbol{K}_{N,N}^{(1)})$ by multidimensional Central Limit Theorem, where

$$
\begin{aligned}
k^{(1)}(\boldsymbol{x},\boldsymbol{x}') \quad & = \mathrm{Cov}[z_i^{(1)}(\boldsymbol{x}), z_i^{(1)}(\boldsymbol{x}')] = \mathrm{E}_{\omega_{:,i}^{(l)},\beta_i^{(l)}}[z_i^{(1)}(\boldsymbol{x}) z_i^{(1)}(\boldsymbol{x}')] \\
& = \frac{\sigma_w^2}{D^{(0)}}\mathrm{E}\left[\Sigma_{j,k}\omega_{j,i}^{(l)}\omega_{k,i}^{(l)} x_j x_k'\right] + \frac{\sigma_w\sigma_b}{\sqrt{D^{(0)}}}\mathrm{E}\left[\beta_i^{(l)}\Sigma_j \omega_{j,i}^{(l)} x_j\right] \\
& \quad + \frac{\sigma_w\sigma_b}{\sqrt{D^{(0)}}}\mathrm{E}\left[\beta_i^{(l)}\Sigma_j \omega_{j,i}^{(l)} x_j'\right] + \sigma_b^2 \mathrm{E}\left[\beta_i^{(l)}\beta_i^{(l)}\right] \\
& = \frac{\sigma_w^2}{D^{(0)}}\Sigma_j \mathrm{E}\left[\omega_{j,i}^{(l)}\omega_{j,i}^{(l)}\right] x_j x_j' + \sigma_b^2 \mathrm{E}\left[\beta_i^{(l)}\beta_i^{(l)}\right] \\
& = \frac{\sigma_w^2}{D^{(0)}}\boldsymbol{x}^\top\boldsymbol{x}' + \sigma_b^2,
\end{aligned}
$$

  is independent with i

- Note that $z_i^{(1)}(\cdot)$ and $z_j^{(1)}(\cdot)$ are independent with each other, $\forall i \neq j$

Deriving $k^{(l)}(\cdot,\cdot)$

- Given that $D^{(0)} \to \infty, \cdots, D^{(l-2)} \to \infty$ and
  - $[z_i^{(l-1)}(\boldsymbol{x}^{(1)}), \cdots, z_i^{(l-1)}(\boldsymbol{x}^{(N)})]^\top \sim N(\mathbf{0}_N, \boldsymbol{K}_{N,N}^{(l-1)})$
  - $z_i^{(l-1)}(\cdot)$ and $z_j^{(l-1)}(\cdot)$ are independent with each other, $\forall i \neq j$

Deriving $k^{(l)}(\cdot,\cdot)$

- Given that $D^{(0)} \to \infty, \cdots, D^{(l-2)} \to \infty$ and
  - $[z_i^{(l-1)}(\boldsymbol{x}^{(1)}), \cdots, z_i^{(l-1)}(\boldsymbol{x}^{(N)})]^\top \sim N(\mathbf{0}_N, \boldsymbol{K}_{N,N}^{(l-1)})$
  - $z_i^{(l-1)}(\cdot)$ and $z_j^{(l-1)}(\cdot)$ are independent with each other, $\forall i \neq j$
- Consider $z_i^{(l)}(\boldsymbol{x}) = \frac{\sigma_w}{\sqrt{D^{(l-1)}}} \Sigma_j \omega_{j,i}^{(l)} \phi(z_j^{(l-1)}(\boldsymbol{x})) + \sigma_b \beta_i^{(l)}$ a zero-mean Gaussian
- As $D^{(l-1)} \to \infty$, we have $[z_i^{(l)}(\boldsymbol{x}^{(1)}), \cdots, z_i^{(l)}(\boldsymbol{x}^{(N)})]^\top \sim N(\mathbf{0}_N, \boldsymbol{K}_{N,N}^{(l)})$ by multidimensional Central Limit Theorem, where

$$
\begin{aligned}
k^{(l)}(\boldsymbol{x},\boldsymbol{x}') &= \mathrm{Cov}[z_i^{(l)}(\boldsymbol{x}), z_i^{(l)}(\boldsymbol{x}')] = \mathrm{E}_{\boldsymbol{\omega}_{:,i}^{(l)}, \beta_i^{(l)}, \boldsymbol{z}^{(l-1)}(\boldsymbol{x})}[z_i^{(l)}(\boldsymbol{x}) z_i^{(l)}(\boldsymbol{x}')] \\
&= \frac{\sigma_w^2}{D^{(l-1)}} \mathrm{E}\left[\Sigma_{j,k} \omega_{j,i}^{(l)} \omega_{k,i}^{(l)} \phi(z_j^{(l-1)}(\boldsymbol{x})) \phi(z_k^{(l-1)}(\boldsymbol{x}'))\right] + \sigma_b^2 \mathrm{E}\left[\beta_i^{(l)} \beta_i^{(l)}\right] \\
&\quad + \frac{\sigma_w \sigma_b}{\sqrt{D^{(l-1)}}} \left(\mathrm{E}\left[\beta_i^{(l)} \Sigma_j \omega_{j,i}^{(l)} \phi(z_j^{(l-1)}(\boldsymbol{x}))\right] + \mathrm{E}\left[\beta_i^{(l)} \Sigma_j \omega_{j,i}^{(l)} \phi(z_j^{(l-1)}(\boldsymbol{x}'))\right] \right) \\
&= \frac{\sigma_w^2}{D^{(l-1)}} \Sigma_j \mathrm{E}\left[\omega_{j,i}^{(l)} \omega_{j,i}^{(l)}\right] \mathrm{E}\left[\phi(z_j^{(l-1)}(\boldsymbol{x})) \phi(z_j^{(l-1)}(\boldsymbol{x}'))\right] + \sigma_b^2 \mathrm{E}\left[\beta_i^{(l)} \beta_i^{(l)}\right] \\
&= \sigma_w^2 \mathrm{E}_{(z_i^{(l-1)}(\boldsymbol{x}), z_i^{(l-1)}(\boldsymbol{x}')) \sim \mathcal{N}(\mathbf{0}_2, \boldsymbol{K}_{2,2}^{(l-1)})} \left[\phi(z_i^{(l-1)}(\boldsymbol{x})) \phi(z_i^{(l-1)}(\boldsymbol{x}'))\right] + \sigma_b^2,
\end{aligned}
$$

where

$$
\boldsymbol{K}_{2,2}^{(l-1)} = \begin{bmatrix} k^{(l-1)}(\boldsymbol{x},\boldsymbol{x}) & k^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') \\ k^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') & k^{(l-1)}(\boldsymbol{x}',\boldsymbol{x}') \end{bmatrix}
$$

Evaluating $\boldsymbol{K}^{(l)}$

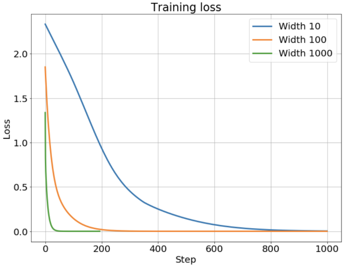
- For certain activation functions $\phi(\cdot)$, such as tanh and ReLU, $k^{(l)}(\boldsymbol{x},\boldsymbol{x}')$ has a closed form [10]
- For other $\phi(\cdot)$'s, Markov Chain Monte Carlo (MCMC) sampling is required to devaluate $k^{(l)}(\boldsymbol{x},\boldsymbol{x}')$

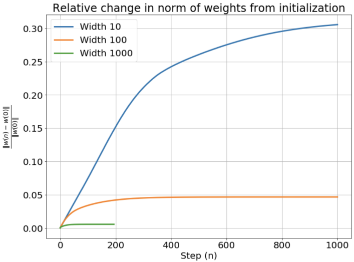
Outline

1. When ML Meets Big Data
2. Advantages of Deep Learning
   - Representation Learning
   - Exponential Gain of Expressiveness
   - Memory and GPU Friendliness
   - Online & Transfer Learning
3. Learning Theory Revisited
   - Generalizability and Over-Parametrization
   - Wide-and-Deep NN is a Gaussian Process before Training*
   - Gradient Descent is an Affine Transformation*
   - Wide-and-Deep NN is a Gaussian Process after Training*

Weight Dynamics

- Observation: the weights of a wide NN do not change much during gradient descent

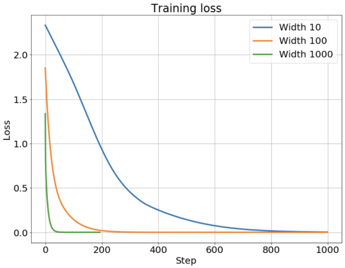


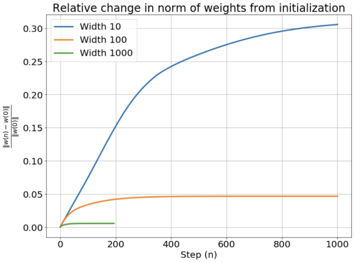


- Why?

Weight Dynamics

- Observation: the weights of a wide NN do not change much during gradient descent





- Why? A small change in a large number of neurons is enough to significantly change the output
- This allows us to approximate an NN $f(\cdot;\theta)$ ***w.r.t. weights*** using the first-order Taylor expansion

Linearization of $f(\cdot;\theta)$

- Let $\theta^{(t)}$ be the parameters of the NN at the t-th step of gradient descent
  - $\hat{\boldsymbol{y}}_N^{(t)} = [f(\boldsymbol{x}^{(1)};\theta^{(t)}),\cdots,f(\boldsymbol{x}^{(N)};\theta^{(t)})]^\top$ be the predictions over training points
- Since $\theta^{(t)}$ is close to $\theta^{(0)}$ at any time t, we can approximate $f(\cdot;\theta^{(t)})$ using the first-order Taylor expansion ***w.r.t.*** $\theta^{(t)}$ around $\theta^{(0)}$:

$$f(\boldsymbol{x},\theta^{(t)}) \approx \bar{f}(\boldsymbol{x},\theta^{(t)}) = f(\boldsymbol{x},\theta^{(0)}) + \nabla_\theta f(\boldsymbol{x},\theta^{(0)})^\top(\theta^{(t)} - \theta^{(0)})$$

  - $\bar{f}$ is still ***non-linear in terms of x***
  - Let $\bar{\boldsymbol{y}}_N^{(t)} = [\bar{f}(\boldsymbol{x}^{(1)};\theta^{(t)}),\cdots,\bar{f}(\boldsymbol{x}^{(N)};\theta^{(t)})]^\top$ be the predictions of $\bar{f}$ at time t

Weight and Prediction Dynamics

$$f(\boldsymbol{x},\boldsymbol{\theta}^{(t)}) \approx \bar{f}(\boldsymbol{x},\boldsymbol{\theta}^{(t)}) = f(\boldsymbol{x},\boldsymbol{\theta}^{(0)}) + \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}^{(0)})^\top(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(0)})$$

- Gradient descent with learning rate η makes the following changes:

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)} &\approx -\eta \nabla_{\boldsymbol{\theta}} C(\bar{\boldsymbol{y}}_N^{(t)}, \boldsymbol{y}_N) \\ &= -\eta \nabla_{\boldsymbol{\theta}} \bar{\boldsymbol{y}}_N^{(t)} \nabla_{\bar{\boldsymbol{y}}_N^{(t)}} C(\bar{\boldsymbol{y}}_N^{(t)}, \boldsymbol{y}_N) \\ &= -\eta \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{y}}_N^{(0)} \nabla_{\bar{\boldsymbol{y}}_N^{(t)}} C(\bar{\boldsymbol{y}}_N^{(t)}, \boldsymbol{y}_N)\end{aligned}$$

and

$$\begin{aligned}\bar{\boldsymbol{y}}_N^{(t+1)} - \bar{\boldsymbol{y}}_N^{(t)} &= \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{y}}_N^{(0)\top}(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) \\ &\approx -\eta \underbrace{\nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{y}}_N^{(0)\top}}_{N\times D} \underbrace{\nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{y}}_N^{(0)}}_{D\times N} \nabla_{\bar{\boldsymbol{y}}_N^{(t)}} C(\bar{\boldsymbol{y}}_N^{(t)}, \boldsymbol{y}_N),\end{aligned}$$

where $\boldsymbol{T}_{N,N}^{(0)} \equiv \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{y}}_N^{(0)\top} \nabla_{\boldsymbol{\theta}} \hat{\boldsymbol{y}}_N^{(0)} \in \mathbb{R}^{N\times N}$ is called the ***Neural Tangent Kernel (NTK)*** matrix

Prediction Dynamics in Regression

- In regression where $C(\bar{\boldsymbol{y}}_N^{(0)}, \boldsymbol{y}_N) = \frac{1}{2}\|\bar{\boldsymbol{y}}_N^{(0)} - \boldsymbol{y}_N\|^2$, we have

$$\bar{\boldsymbol{y}}_N^{(t+1)} - \bar{\boldsymbol{y}}_N^{(t)} \approx -\eta \boldsymbol{T}_{N,N}^{(0)} \nabla_{\bar{\boldsymbol{y}}_N^{(t)}} C(\bar{\boldsymbol{y}}_N^{(t)}, \boldsymbol{y}_N) = -\eta \boldsymbol{T}_{N,N}^{(0)} (\bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N)$$

- With a sufficiently small learning rate η, we can think t as continuous time and each GD step as Δt, where

$$\lim_{\Delta t \to 0} \frac{\bar{\boldsymbol{y}}_N^{(t+\Delta t)} - \bar{\boldsymbol{y}}_N^{(t)}}{\Delta t} = \frac{\partial \bar{\boldsymbol{y}}_N^{(t)}}{\partial t} \approx -\eta \boldsymbol{T}_{N,N}^{(0)} (\bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N)$$

- Letting $\boldsymbol{u}^{(t)} = \bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N$, we have an ordinary differential equation:

$$\begin{gathered} \frac{\partial \bar{\boldsymbol{y}}_N^{(t)}}{\partial t} \approx -\eta \boldsymbol{T}_{N,N}^{(0)} (\bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N) \\ \Rightarrow \frac{\partial \boldsymbol{u}^{(t)}}{\partial t} \approx -\eta \boldsymbol{T}_{N,N}^{(0)} \boldsymbol{u}^{(t)} \end{gathered}$$

Prediction Dynamics in Regression

- In regression where $C(\bar{\boldsymbol{y}}_N^{(0)}, \boldsymbol{y}_N) = \frac{1}{2}\|\bar{\boldsymbol{y}}_N^{(0)} - \boldsymbol{y}_N\|^2$, we have

$$\bar{\boldsymbol{y}}_N^{(t+1)} - \bar{\boldsymbol{y}}_N^{(t)} \approx -\eta \boldsymbol{T}_{N,N}^{(0)} \nabla_{\bar{\boldsymbol{y}}_N^{(t)}} C(\bar{\boldsymbol{y}}_N^{(t)}, \boldsymbol{y}_N) = -\eta \boldsymbol{T}_{N,N}^{(0)}(\bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N)$$

- With a sufficiently small learning rate η, we can think t as continuous time and each GD step as Δt, where

$$\lim_{\Delta t \to 0} \frac{\bar{\boldsymbol{y}}_N^{(t+\Delta t)} - \bar{\boldsymbol{y}}_N^{(t)}}{\Delta t} = \frac{\partial \bar{\boldsymbol{y}}_N^{(t)}}{\partial t} \approx -\eta \boldsymbol{T}_{N,N}^{(0)}(\bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N)$$

- Letting $\boldsymbol{u}^{(t)} = \bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N$, we have an ordinary differential equation:

$$\frac{\partial \bar{\boldsymbol{y}}_N^{(t)}}{\partial t} \approx -\eta \boldsymbol{T}_{N,N}^{(0)}(\bar{\boldsymbol{y}}_N^{(t)} - \boldsymbol{y}_N)$$
$$\Rightarrow \frac{\partial \boldsymbol{u}^{(t)}}{\partial t} \approx -\eta \boldsymbol{T}_{N,N}^{(0)} \boldsymbol{u}^{(t)}$$

- Therefore, $\boldsymbol{u}^{(t)} = e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t} \boldsymbol{u}^{(0)}$
  - Recall that $e^{\boldsymbol{A}t} = \frac{1}{0!}\boldsymbol{I} + \frac{t}{1!}\boldsymbol{A} + \frac{t^2}{2!}\boldsymbol{A}^2 + \cdots$ for a symmetric $\boldsymbol{A}$
  - So, $\frac{\partial e^{\boldsymbol{A}t}}{\partial t} = \frac{1}{0!}\boldsymbol{A} + \frac{t}{1!}\boldsymbol{A}^2 + \cdots = (\frac{1}{0!}\boldsymbol{I} + \frac{t}{1!}\boldsymbol{A} + \cdots)\boldsymbol{A} = \boldsymbol{A}e^{\boldsymbol{A}t}$
- This implies that

$$\bar{\boldsymbol{y}}_N^{(t)} = e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t} \bar{\boldsymbol{y}}_N^{(0)} + (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t}) \boldsymbol{y}_N = e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t} \hat{\boldsymbol{y}}_N^{(0)} + (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t}) \boldsymbol{y}_N$$

Weight Dynamics in Regression

- By definition of $\bar{\boldsymbol{y}}_N^{(t)}$, we also have $\bar{\boldsymbol{y}}_N^{(t)} = \hat{\boldsymbol{y}}_N^{(0)} + \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)\top}(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(0)})$
- Solving $\theta^{(t)}$ in

$$e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t} \hat{\boldsymbol{y}}_N^{(0)} + (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t}) \boldsymbol{y}_N = \hat{\boldsymbol{y}}_N^{(0)} + \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)\top}(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(0)}),$$

we have

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(0)} - \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)} \boldsymbol{T}_{N,N}^{(0)-1} (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t})(\hat{\boldsymbol{y}}_N^{(0)} - \boldsymbol{y}_N)$$

Predictions of Trained NN

- Substituting $\theta^{(t)}$ in $\bar{\boldsymbol{y}}_N^{(t)} = \hat{\boldsymbol{y}}_N^{(0)} + \nabla_{\theta}\hat{\boldsymbol{y}}_N^{(0)\top}(\theta^{(t)} - \theta^{(0)})$, we have that:
- For an arbitrary (training or test) point $\boldsymbol{x}'$, the prediction of trained NN is
$$f(\boldsymbol{x}', \theta^{(t)}) \approx \bar{f}(\boldsymbol{x}'; \theta^{(t)}) = \boldsymbol{p}^\top \begin{bmatrix} \hat{\boldsymbol{y}}_N^{(0)} \\ \hat{y}'^{(0)} \end{bmatrix} + q,$$
where
$$\boldsymbol{p} = [-\boldsymbol{T}_{1',N}^{(0)} \boldsymbol{T}_{N,N}^{(0)-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t}), 1]^\top \in \mathbb{R}^{N+1},$$
$$q = \boldsymbol{T}_{1',N}^{(0)} \boldsymbol{T}_{N,N}^{(0)-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t})\boldsymbol{y}_N$$
  - $\boldsymbol{T}_{N,N}^{(0)} = \nabla_{\theta}\hat{\boldsymbol{y}}_N^{(0)\top} \nabla_{\theta}\hat{\boldsymbol{y}}_N^{(0)} \in \mathbb{R}^{N\times N}$ is the NTK matrix for $\boldsymbol{X}_N$
  - $\boldsymbol{T}_{1',N}^{(0)} = \nabla_{\theta}\hat{y}'^{(0)\top} \nabla_{\theta}\hat{\boldsymbol{y}}_N^{(0)} \in \mathbb{R}^{1\times N}$ is the NTK matrix between $\boldsymbol{x}'$ and $\boldsymbol{X}_N$

Predictions of Trained NN

- Substituting $\theta^{(t)}$ in $\bar{\boldsymbol{y}}_N^{(t)} = \hat{\boldsymbol{y}}_N^{(0)} + \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)\top}(\theta^{(t)} - \theta^{(0)})$, we have that:
- For an arbitrary (training or test) point $\boldsymbol{x}'$, the prediction of trained NN is

$$f(\boldsymbol{x}', \theta^{(t)}) \approx \bar{f}(\boldsymbol{x}'; \theta^{(t)}) = \boldsymbol{p}^\top \begin{bmatrix} \hat{\boldsymbol{y}}_N^{(0)} \\ \hat{y}'^{(0)} \end{bmatrix} + q,$$

  where

$$\boldsymbol{p} = [-\boldsymbol{T}_{1',N}^{(0)} \boldsymbol{T}_{N,N}^{(0)-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t}), 1]^\top \in \mathbb{R}^{N+1},$$
$$q = \boldsymbol{T}_{1',N}^{(0)} \boldsymbol{T}_{N,N}^{(0)-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}^{(0)} t})\boldsymbol{y}_N$$

  - $\boldsymbol{T}_{N,N}^{(0)} = \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)\top} \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)} \in \mathbb{R}^{N\times N}$ is the NTK matrix for $\boldsymbol{X}_N$
  - $\boldsymbol{T}_{1',N}^{(0)} = \nabla_\theta \hat{y}'^{(0)\top} \nabla_\theta \hat{\boldsymbol{y}}_N^{(0)} \in \mathbb{R}^{1\times N}$ is the NTK matrix between $\boldsymbol{x}'$ and $\boldsymbol{X}_N$
- ***No actual training*** needed!

Gradient Descent as an Affine Transformation

Theorem (NTK in infinite width)

As the NN's width goes to infinity, $\boldsymbol{T}^{(0)}_{N,N}$ and $\boldsymbol{T}^{(0)}_{1',N}$ converges to $\boldsymbol{T}_{N,N}$ and $\boldsymbol{T}_{1',N}$, which can be described by a ***deterministic*** *kernel function $\tau^{(L)}(\cdot,\cdot)$ independent of a particular initialization [9, 11].*

- That is, each element $T_{i,j} = \tau^{(L)}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$
- $\tau^{(L)}(\cdot,\cdot)$ depends only on the network structure and hyperparameters of initial weights
- $\tau^{(L)}(\cdot,\cdot)$ has a closed form for certain activation functions $\phi(\cdot)$'s, including erf and ReLU

Gradient Descent as an Affine Transformation

Theorem (NTK in infinite width)

As the NN's width goes to infinity, $\boldsymbol{T}^{(0)}_{N,N}$ and $\boldsymbol{T}^{(0)}_{1',N}$ converges to $\boldsymbol{T}_{N,N}$ and $\boldsymbol{T}_{1',N}$, which can be described by a ***deterministic*** *kernel function $\tau^{(L)}(\cdot,\cdot)$ independent of a particular initialization [9, 11].*

- That is, each element $T_{i,j} = \tau^{(L)}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$
- $\tau^{(L)}(\cdot,\cdot)$ depends only on the network structure and hyperparameters of initial weights
- $\tau^{(L)}(\cdot,\cdot)$ has a closed form for certain activation functions $\phi(\cdot)$'s, including erf and ReLU
- $f(\boldsymbol{x}', \theta^{(t)}) \approx \boldsymbol{p}^\top \begin{bmatrix} \hat{\boldsymbol{y}}^{(0)} \\ \hat{y}'^{(0)} \end{bmatrix} + q$ is an ***affine transformation*** of a random vector $\begin{bmatrix} \hat{\boldsymbol{y}}^{(0)} \\ \hat{y}'^{(0)} \end{bmatrix}$

NTK in Infinite Width

- Consider the pre-activations $z_i^{(1)}(\cdot), z_i^{(2)}(\cdot), \cdots, z^{(L)}(\cdot) = \hat{y}^{(0)}(\cdot)$ at different layers at time 0
- Let $\nabla_{\boldsymbol{\theta}^{(\leq 1)}} z_i^{(1)}(\cdot), \nabla_{\boldsymbol{\theta}^{(\leq 2)}} z_i^{(2)}(\cdot), \cdots, \nabla_{\boldsymbol{\theta}^{(\leq L)}} z^{(L)}(\cdot) = \nabla_{\boldsymbol{\theta}} \hat{y}^{(0)}(\cdot)$ be the corresponding derivatives
  - $\boldsymbol{\theta}^{(\leq l)} \equiv \mathrm{vec}(\boldsymbol{\theta}^{(1)}, \cdots, \boldsymbol{\theta}^{(l)})$
- We use induction to show that, when $D \to \infty$, we have

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}^{(\leq l)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l)}} z_i^{(l)}(\boldsymbol{x}') = \tau^{(l)}(\boldsymbol{x}, \boldsymbol{x}') \\
&\quad = k^{(l)}(\boldsymbol{x}, \boldsymbol{x}') + \\
&\qquad \sigma_w^2 \tau^{(l-1)}(\boldsymbol{x}, \boldsymbol{x}') \mathrm{E}_{(z_i^{(l-1)}(\boldsymbol{x}), z_i^{(l-1)}(\boldsymbol{x}')) \sim \mathcal{N}(\boldsymbol{0}_2, \boldsymbol{K}_{2,2}^{(l-1)})} \left[\phi'(z_i^{(l-1)}(\boldsymbol{x})) \phi'(z_i^{(l-1)}(\boldsymbol{x}')) \right]
\end{aligned}$$

  at any layer l, and $\tau^{(1)}(\boldsymbol{x}, \boldsymbol{x}') = k^{(1)}(\boldsymbol{x}, \boldsymbol{x}')$
  - $\tau^{(l)}(\cdot, \cdot)$ is independent of i

Deriving $\tau^{(1)}(\cdot,\cdot)$

- At the first layer, we have

$$\nabla_{\boldsymbol{\theta}^{(\leq 1)}} z_i^{(1)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq 1)}} z_i^{(1)}(\boldsymbol{x}') = \frac{\sigma_w^2}{D^{(0)}} \boldsymbol{x}^\top \boldsymbol{x}' + \sigma_b^2 = k^{(1)}(\boldsymbol{x}, \boldsymbol{x}')$$

as $D^{(0)} \to \infty$

Deriving $\tau^{(1)}(\cdot,\cdot)$

- At the first layer, we have

$$\nabla_{\boldsymbol{\theta}^{(\leq 1)}} z_i^{(1)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq 1)}} z_i^{(1)}(\boldsymbol{x}') = \frac{\sigma_w^2}{D^{(0)}} \boldsymbol{x}^\top \boldsymbol{x}' + \sigma_b^2 = k^{(1)}(\boldsymbol{x}, \boldsymbol{x}')$$

  as $D^{(0)} \to \infty$

- Now, assume that when $D^{(0)} \to \infty, \cdots, D^{(l-2)} \to \infty$, $\nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l-1)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l-1)}(\boldsymbol{x}') = \tau^{(l-1)}(\boldsymbol{x}, \boldsymbol{x}')$ holds

Deriving $\tau^{(l)}(\cdot,\cdot)$ I

- At the l-th layer, we have

$$
\begin{aligned}
&\nabla_{\boldsymbol{\theta}^{(\leq l)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l)}} z_i^{(l)}(\boldsymbol{x}') \\
&\quad = [\nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}), \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x})][\nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}'), \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x}')]^\top \\
&\quad = \nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}') + \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x}')
\end{aligned}
$$

Deriving $\tau^{(l)}(\cdot,\cdot)$ I

- At the l-th layer, we have

$$
\begin{aligned}
&\nabla_{\boldsymbol{\theta}^{(\leq l)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l)}} z_i^{(l)}(\boldsymbol{x}') \\
&\quad = [\nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}), \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x})][\nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}'), \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x}')]^\top \\
&\quad = \nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}') + \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x}')
\end{aligned}
$$

- As $D^{(l-1)} \to \infty$, the first term

$$
\nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(l)}} z_i^{(l)}(\boldsymbol{x}') = \frac{\sigma_w^2}{D^{(l-1)}} \Sigma_j \phi(z_j^{(l-1)}(\boldsymbol{x}))\phi(z_j^{(l-1)}(\boldsymbol{x}')) + \sigma_b^2
$$

converges to

$$
\begin{aligned}
&\sigma_w^2 \mathrm{E}_{(z_i^{(l-1)}(\boldsymbol{x}), z_i^{(l-1)}(\boldsymbol{x}')) \sim \mathcal{N}(\boldsymbol{0}_2, \boldsymbol{K}_{2,2}^{(l-1)})} \left[\phi(z_i^{(l-1)}(\boldsymbol{x}))\phi(z_i^{(l-1)}(\boldsymbol{x}'))\right] + \sigma_b^2 \\
&\quad = k^{(l)}(\boldsymbol{x}, \boldsymbol{x}')
\end{aligned}
$$

because $z_i^{(l-1)}(\cdot)$ and $z_j^{(l-1)}(\cdot)$ are i.i.d.

Deriving $\tau^{(l)}(\cdot,\cdot)$ II

- Consider the second term

$$
\begin{aligned}
&\nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} z_i^{(l)}(\boldsymbol{x}') \\
&= \nabla_{\boldsymbol{z}^{(l-1)}(\boldsymbol{x})} z_i^{(l)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} \boldsymbol{z}^{(l-1)}(\boldsymbol{x})^\top \nabla_{\boldsymbol{\theta}^{(\leq l-1)}} \boldsymbol{z}^{(l-1)}(\boldsymbol{x}') \nabla_{\boldsymbol{z}^{(l-1)}(\boldsymbol{x})} z_i^{(l)}(\boldsymbol{x}') \\
&= \nabla_{\boldsymbol{z}^{(l-1)}(\boldsymbol{x})} z_i^{(l)}(\boldsymbol{x})^\top \begin{bmatrix} \tau^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \tau^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') \end{bmatrix} \nabla_{\boldsymbol{z}^{(l-1)}(\boldsymbol{x})} z_i^{(l)}(\boldsymbol{x}) \\
&= \tau^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') \Sigma_j \frac{\partial z_i^{(l)}(\boldsymbol{x})}{\partial z_j^{(l-1)}(\boldsymbol{x})} \cdot \frac{\partial z_i^{(l)}(\boldsymbol{x}')}{\partial z_j^{(l-1)}(\boldsymbol{x}')} \\
&= \tau^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') \frac{\sigma_w^2}{D^{(l-1)}} \Sigma_j \omega_{j,i}^{(l)} \omega_{j,i}^{(l)} \phi'(z_j^{(l-1)}(\boldsymbol{x})) \phi'(z_j^{(l-1)}(\boldsymbol{x}'))
\end{aligned}
$$

- As $D^{(l-1)} \to \infty$, it becomes

$$
\sigma_w^2 \tau^{(l-1)}(\boldsymbol{x},\boldsymbol{x}') \mathrm{E}_{(z_i^{(l-1)}(\boldsymbol{x}), z_i^{(l-1)}(\boldsymbol{x}')) \sim \mathcal{N}(\boldsymbol{0}_2, \boldsymbol{K}_{2,2}^{(l-1)})} \left[\phi'(z_i^{(l-1)}(\boldsymbol{x})) \phi'(z_i^{(l-1)}(\boldsymbol{x}')) \right]
$$

Outline

1. When ML Meets Big Data
2. Advantages of Deep Learning
   - Representation Learning
   - Exponential Gain of Expressiveness
   - Memory and GPU Friendliness
   - Online & Transfer Learning
3. **Learning Theory Revisited**
   - Generalizability and Over-Parametrization
   - Wide-and-Deep NN is a Gaussian Process before Training*
   - Gradient Descent is an Affine Transformation*
   - **Wide-and-Deep NN is a Gaussian Process after Training***

Wide-and-Deep NN as a Gaussian Process I

- As $D \to \infty$, randomly initialized NN has a corresponding NN-GP:

$$\begin{bmatrix} \hat{\boldsymbol{y}}_N \\ \hat{\boldsymbol{y}}_M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{0}_N \\ \boldsymbol{0}_M \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{N,N}^{(L)} & \boldsymbol{K}_{N,M}^{(L)} \\ \boldsymbol{K}_{M,N}^{(L)} & \boldsymbol{K}_{M,M}^{(L)} \end{bmatrix} \right)$$

- As $D \to \infty$, GD-based training is an affine transformation:

$$f(\boldsymbol{x}', \theta^{(t)}) \approx \bar{f}(\boldsymbol{x}'; \theta^{(t)}) = \boldsymbol{p}^\top \begin{bmatrix} \hat{\boldsymbol{y}}_N^{(0)} \\ \hat{y}'^{(0)} \end{bmatrix} + q$$

  where
  - $\boldsymbol{p} = [-\boldsymbol{T}_{1',N}\boldsymbol{T}_{N,N}^{-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}t}), 1]^\top \in \mathbb{R}^{N+1}$
  - $q = \boldsymbol{T}_{1',N}\boldsymbol{T}_{N,N}^{-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N}t})\boldsymbol{y}_N$
  - $\boldsymbol{T}_{N,N}$ and $\boldsymbol{T}_{1',N}$ the NTK matrices

Wide-and-Deep NN as a Gaussian Process II

- Therefore, as $D \to \infty$, the trained NN is still in correspondent with a GP, called ***NTK-GP***, whose predictions for M test points are

$$\begin{bmatrix} \hat{\boldsymbol{y}}_N \\ \hat{\boldsymbol{y}}_M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{A}\boldsymbol{y}_N \\ \boldsymbol{B}\boldsymbol{y}_N \end{bmatrix}, \boldsymbol{C}^\top \begin{bmatrix} \boldsymbol{K}^{(L)}_{N,N} & \boldsymbol{K}^{(L)}_{N,M} \\ \boldsymbol{K}^{(L)}_{M,N} & \boldsymbol{K}^{(L)}_{M,M} \end{bmatrix} \boldsymbol{C} \right),$$

  where
  - $\boldsymbol{A} = (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N} t}) \in \mathbb{R}^{N \times N}$
  - $\boldsymbol{B} = \boldsymbol{T}_{M,N} \boldsymbol{T}_{N,N}^{-1} (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N} t}) \in \mathbb{R}^{M \times N}$
  - $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_{N,N} - \boldsymbol{A} & \boldsymbol{O}_{N,M} \\ -\boldsymbol{B} & \boldsymbol{I}_{M,M} \end{bmatrix} \in \mathbb{R}^{(N+M) \times (N+M)}$

Mean Predictions of NTK-GP

- Prior (unconditioned) mean predictions for training set:

$$\hat{\boldsymbol{y}}_N = \boldsymbol{A}\boldsymbol{y}_N = (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N} t})\boldsymbol{y}_N$$

  - As $t \to \infty$, **the $\hat{\boldsymbol{y}}_N$ always approaches true labels $\boldsymbol{y}_N$**
  - This explains why the SGD-based training of large NNs seldom encounters significant obstacles such as local minima [8]

Mean Predictions of NTK-GP

- Prior (unconditioned) mean predictions for training set:

$$\hat{\boldsymbol{y}}_N = \boldsymbol{A}\boldsymbol{y}_N = (\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N} t})\boldsymbol{y}_N$$

  - As $t \to \infty$, **the $\hat{\boldsymbol{y}}_N$ always approaches true labels $\boldsymbol{y}_N$**
  - This explains why the SGD-based training of large NNs seldom encounters significant obstacles such as local minima [8]

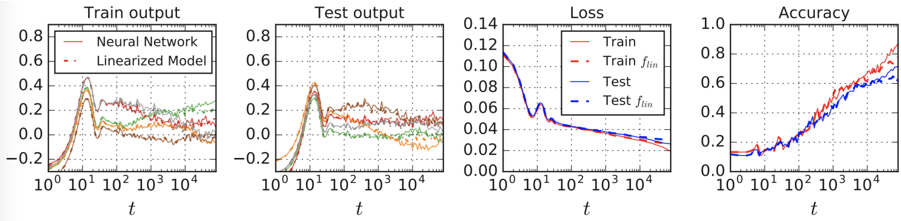
- Prior mean predictions for test set:

$$\hat{\boldsymbol{y}}_M = \boldsymbol{B}\boldsymbol{y} = \boldsymbol{T}_{M,N}\boldsymbol{T}_{N,N}^{-1}(\boldsymbol{I} - e^{-\eta \boldsymbol{T}_{N,N} t})\boldsymbol{y}_N$$

  - As $t \to \infty$, we have $\hat{\boldsymbol{y}}_M = \boldsymbol{T}_{M,N}\boldsymbol{T}_{N,N}^{-1}\boldsymbol{y}_N$
  - ***Weight hyperparameters are important*** because they determines $\boldsymbol{T}_{M,N}\boldsymbol{T}_{N,N}^{-1}$

Analytic vs. Real Predictions

- Wide residual network [19] trained by SGD with momentum on MSE loss on CIFAR-10
  - First two panes shows the output dynamics for a randomly selected subset of train and test points



Remarks

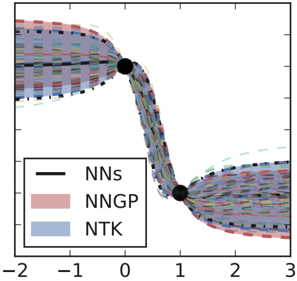
- Wide-and-deep NNs can be approximated by a class of GPs
  - Either before, during, or after training
- Therefore, complexity of wide-and-deep NNs grows with N, not $|\theta|$

Remarks

- Wide-and-deep NNs can be approximated by a class of GPs
  - Either before, during, or after training
- Therefore, complexity of wide-and-deep NNs grows with N, not $|\theta|$
- Applicable to other architectures including CNN [2, 16], RNN [17, 1], and any architecture [18]

Limitations

- Approximation holds only when the NNs have:
  - Infinite width
  - Small learning rate: $\eta < \frac{2}{\lambda_{\max}+\lambda_{\min}}$ where $\lambda_{\max\,/\,\min}$ is the max/min eigenvalue of $\boldsymbol{T}_{N,N}$ [17]
  - Proper initialization (to be discussed next)
- The prior NTK-GP inference $\hat{\boldsymbol{y}}_{\text{NTK-GP}} = \boldsymbol{T}_{M,N}\boldsymbol{T}_{N,N}^{-1}\boldsymbol{y}_N$ is ***inconsistent*** with the Bayesian inference of NN-GP $\hat{\boldsymbol{y}}_{\text{NN-GP}} = \boldsymbol{K}_{M,N}\boldsymbol{K}_{N,N}^{-1}\boldsymbol{y}_N$
  - SGP introduces bias [3]



Reference I

[1] Sina Alemohammad, Zichao Wang, Randall Balestriero, and Richard Baraniuk.
The recurrent neural tangent kernel.
arXiv preprint arXiv:2006.10246, 2020.

[2] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang.
On exact computation with an infinitely wide neural net.
In *Advances in Neural Information Processing Systems*, pages 8141–8150, 2019.

[3] Alberto Bietti and Julien Mairal.
On the inductive bias of neural tangent kernels.
In *Advances in Neural Information Processing Systems*, pages 12893–12904, 2019.

Reference II

[4] Léon Bottou.
Large-scale machine learning with stochastic gradient descent.
In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[5] Olivier Bousquet.
Concentration inequalities and empirical processes theory applied to the analysis of learning algorithms.
Ph.D. thesis, Ecole Polytechnique, Palaiseau, France, 2002.

[6] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew.
Deep learning with cots hpc systems.
In *Proceedings of The 30th International Conference on Machine Learning*, pages 1337–1345, 2013.

Reference III

[7] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin.
Liblinear: A library for large linear classification.
Journal of machine learning research, 9(Aug):1871–1874, 2008.

[8] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe.
Qualitatively characterizing neural network optimization problems.
arXiv preprint arXiv:1412.6544, 2014.

[9] Arthur Jacot, Franck Gabriel, and Clément Hongler.
Neural tangent kernel: Convergence and generalization in neural networks.
In *Advances in neural information processing systems*, pages 8571–8580, 2018.

Reference IV

[10] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein.
Deep neural networks as gaussian processes.
arXiv preprint arXiv:1711.00165, 2017.

[11] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington.
Wide neural networks of any depth evolve as linear models under gradient descent.
In *Advances in neural information processing systems*, pages 8572–8583, 2019.

[12] Pascal Massart.
Some applications of concentration inequalities to statistics.
In *Annales de la Faculté des sciences de Toulouse: Mathématiques*, volume 9, pages 245–303, 2000.

Reference V

[13] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio.
On the number of linear regions of deep neural networks.
In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich.
Going deeper with convolutions.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[15] Vladimir N Vapnik and A Ya Chervonenkis.
On the uniform convergence of relative frequencies of events to their probabilities.
In *Measures of Complexity*, pages 11–30. Springer, 2015.

Reference VI

[16] Greg Yang.
Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation.
arXiv preprint arXiv:1902.04760, 2019.

[17] Greg Yang.
Wide feedforward or recurrent neural networks of any architecture are gaussian processes.
In *Advances in Neural Information Processing Systems*, pages 9951–9960, 2019.

[18] Greg Yang.
Tensor programs ii: Neural tangent kernel for any architecture.
arXiv preprint arXiv:2006.14548, 2020.

Reference VII

[19] Sergey Zagoruyko and Nikos Komodakis.
Wide residual networks.
arXiv preprint arXiv:1605.07146, 2016.

[20] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals.
Understanding deep learning requires rethinking generalization.
arXiv preprint arXiv:1611.03530, 2016.