# Large-Scale Machine Learning

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

Department of Computer Science,
National Tsing Hua University, Taiwan

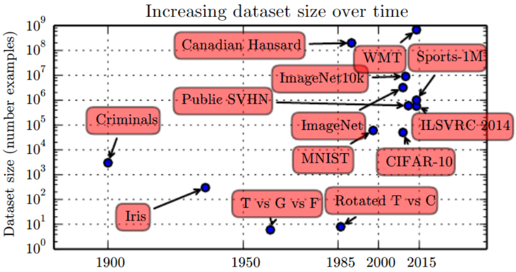
Machine Learning

# Outline

1. When ML Meets Big Data
2. Representation Learning
3. Curse of Dimensionality
4. Trade-Offs in Large-Scale Learning
5. SGD-Based Optimization

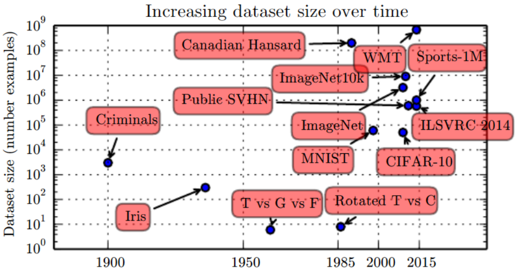
# Outline

# The Big Data Era



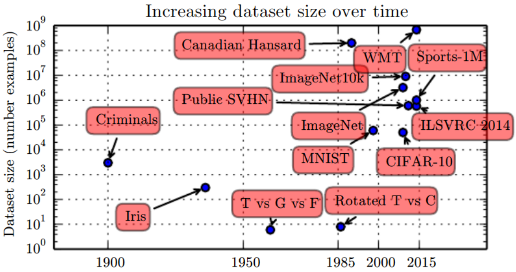

- Today, more and more of our activities are recorded by ubiquitous computing devices

# The Big Data Era



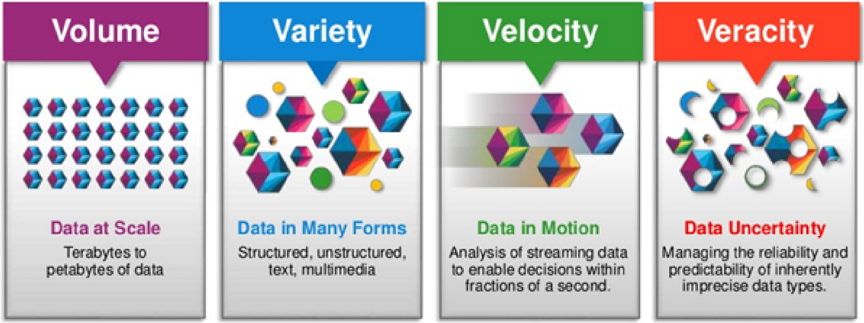

- Today, more and more of our activities are recorded by ubiquitous computing devices
- Networked computers make it easy to centralize these records and curate them into a ***big*** dataset

# 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***
  - 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***
  - Transfer learning
- Volume
  - Large $D$: curse of dimensionality
  - Large $N$: training efficiency

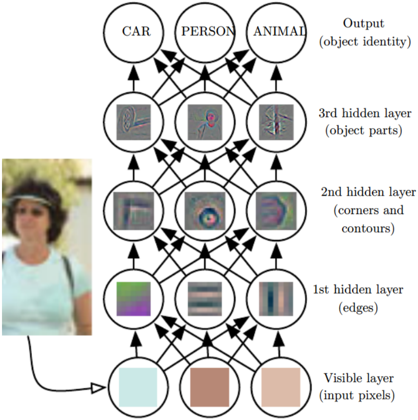
A group of young people playing a game of Frisbee

# Challenges of Large-Scale ML

- Variety and veracity
  - Feature engineering gets ***even harder***
  - Transfer learning
- Volume
  - Large $D$: curse of dimensionality
  - Large $N$: training efficiency
- Velocity
  - Online learning

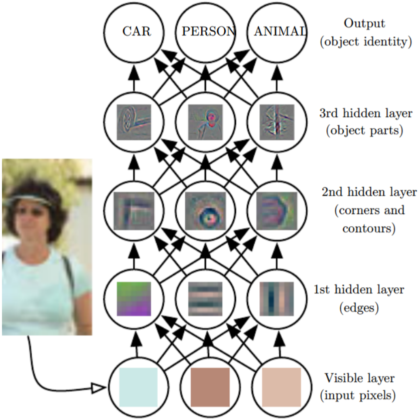
A group of young people playing a game of Frisbee

# The Rise of Deep Learning



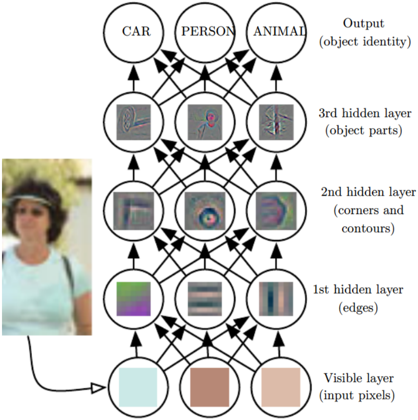

- *Neural Networks* (NNs) that go deep

# The Rise of Deep Learning



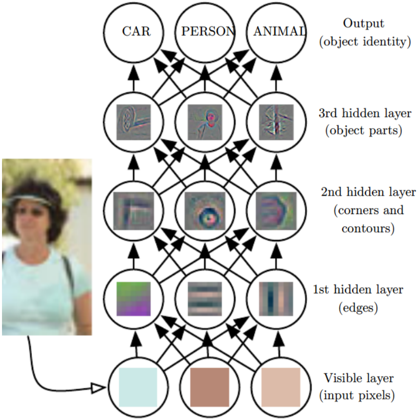

- ***Neural Networks*** (NNs) that go deep
- Feature engineering:
  - Learned automatically (a kind of ***representation learning***)

# The Rise of Deep Learning



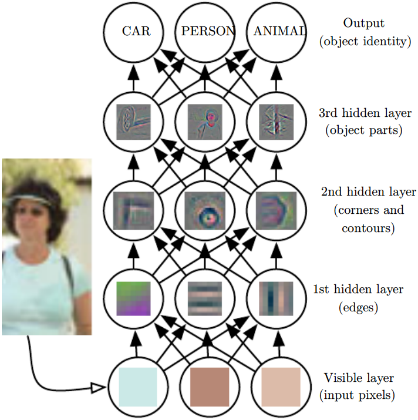

- ***Neural Networks*** (NNs) that go deep
- Feature engineering:
  - Learned automatically (a kind of ***representation learning***)
- Curse of dimensionality:
  - Countered by the exponential gains of deep, distributed representations

# The Rise of Deep Learning




- *Neural Networks* (NNs) that go deep
- Feature engineering:
  - Learned automatically (a kind of ***representation learning***)
- Curse of dimensionality:
  - Countered by the exponential gains of deep, distributed representations
- Training efficiency:
  - SGD
  - GPU-based parallelism

# The Rise of Deep Learning




- *Neural Networks* (NNs) that go deep
- Feature engineering:
  - Learned automatically (a kind of ***representation learning***)
- Curse of dimensionality:
  - Countered by the exponential gains of deep, distributed representations
- Training efficiency:
  - SGD
  - GPU-based parallelism
- Supports 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?

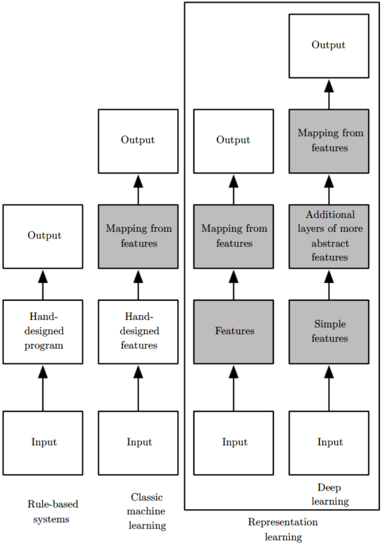
- I have big data, so I have to use deep learning
- ***Wrong!*** No free launch theorem: there is no single ML algorithm that outperforms others in every domain
- Deep learning is more useful when the function $f$ to learn is ***complex*** (nonlinear to the input dimension) and has ***repeating patterns***
  - 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 ***repeating patterns***
  - E.g., image recognition, natural language processing
- For simple (linear) $f$, there are specialized large-scale ML techniques (e.g., LIBLINEAR [4]) that are much more efficient
  - E.g., text classification

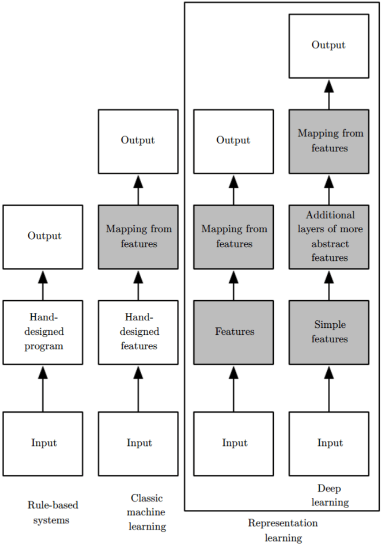
# Outline

# Representation Learning



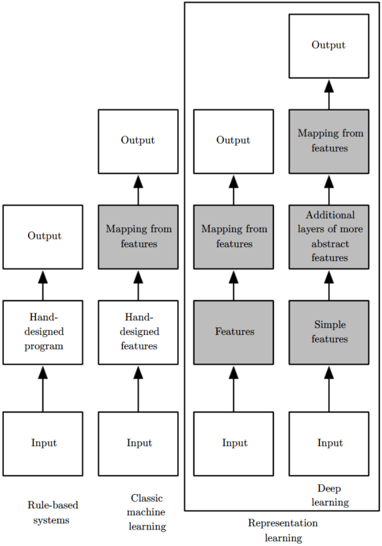

- Gray boxes are learned automatically

# Representation Learning



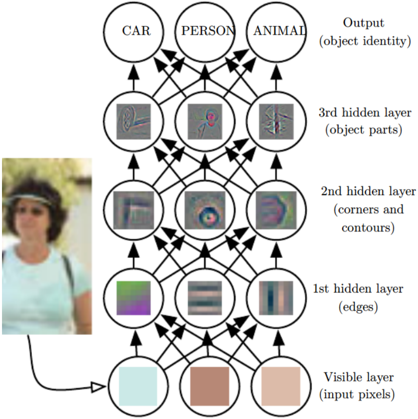

- Gray boxes are learned automatically
- Deep learning maps the ***most abstract*** (deepest) features to the output
  - Usually, a simple linear function suffices

# Representation Learning



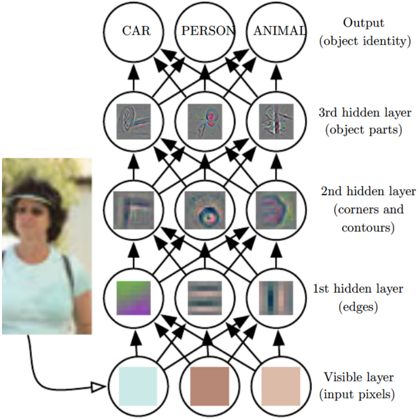

- Gray boxes are learned automatically
- Deep learning maps the ***most abstract*** (deepest) features to the output
  - Usually, a simple linear function suffices
- In deep learning, features/presentations are ***distributed***

# Distributed Representations of Data



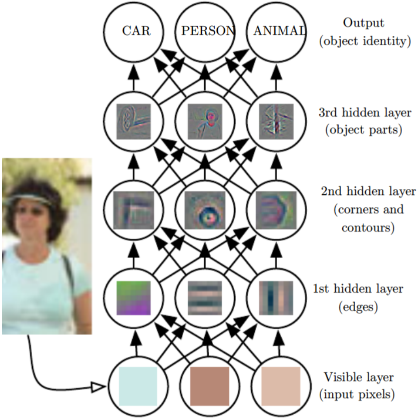

- In deep learning, we assume that $\boldsymbol{x}$'s were generated by the ***composition of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]

# Distributed Representations of Data



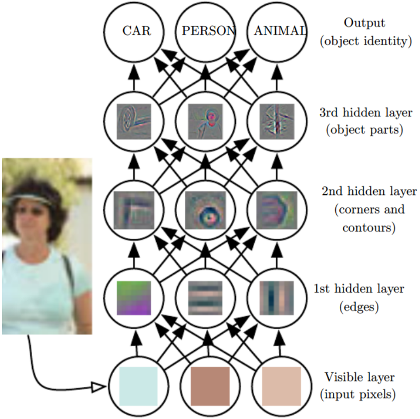

- In deep learning, we assume that $\boldsymbol{x}$'s were generated by the ***composition of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]
  - [.] a predefined non-linear function
  - Weights (arrows) learned from training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

# Distributed Representations of Data



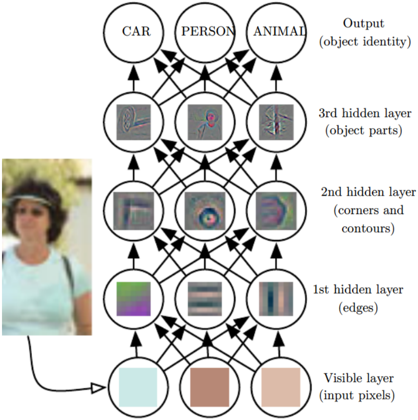

- In deep learning, we assume that $\boldsymbol{x}$'s were generated by the ***composition of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]
  - [.] a predefined non-linear function
  - Weights (arrows) learned from training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



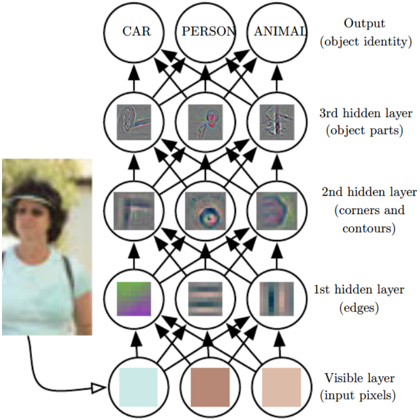

- In deep learning, we assume that $\boldsymbol{x}$'s were generated by the ***composition of factors***, potentially ***at multiple levels*** in a hierarchy
  - E.g., layer 3: face = 0.3 [corner] + 0.7 [circle] + 0 [curve]
  - [.] a predefined non-linear function
  - Weights (arrows) learned from training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

# Transfer Learning




- Transfer learning: to reuse the knowledge learned from a task to help another task

# Transfer Learning




- Transfer learning: to reuse the knowledge learned from a task to help another task
- In deep learning, it's common to reuse the feature extraction network from one task in another
    - Weights may be further updated when training model in a new task

# Outline

# Curse of Dimensionality

- Most classic nonlinear ML models find $\theta$ 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 $\theta$ 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 nonlinear SVM predicts the label $\hat{y}$ of $\boldsymbol{x}$ by simply interpolating the labels of support vectors $\boldsymbol{x}^{(i)}$'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 $\theta$ by assuming function smoothness:

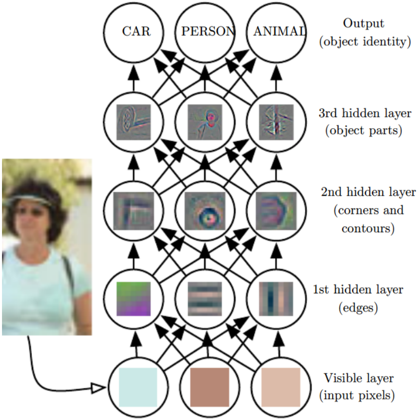
$$\text{if } \boldsymbol{x} \sim \boldsymbol{x}^{(i)} \in \mathbb{X}, \text{ then } f(\boldsymbol{x};\boldsymbol{w}) \sim f(\boldsymbol{x}^{(i)};\boldsymbol{w})$$

- E.g., the nonlinear SVM predicts the label $\hat{y}$ of $\boldsymbol{x}$ by simply interpolating the labels of support vectors $\boldsymbol{x}^{(i)}$'s ***close to $\boldsymbol{x}$***:

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

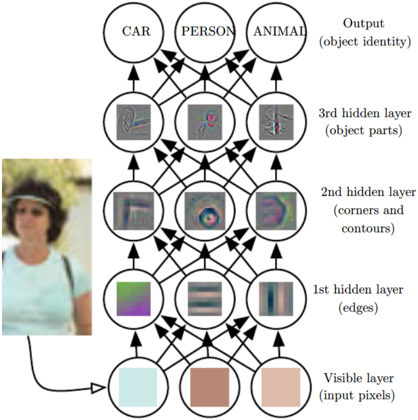
- Suppose $f$ is smooth within a bin, we need ***exponentially more examples*** to get a good interpolation as $D$ increases

# Exponential Gains from Depth



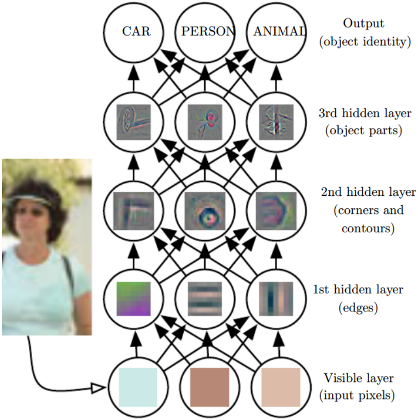

- In deep learning, a deep factor is defined by "reusing" the shallow ones
  - Face $=$ 0.3 [corner] $+$ 0.7 [circle]

# Exponential Gains from Depth



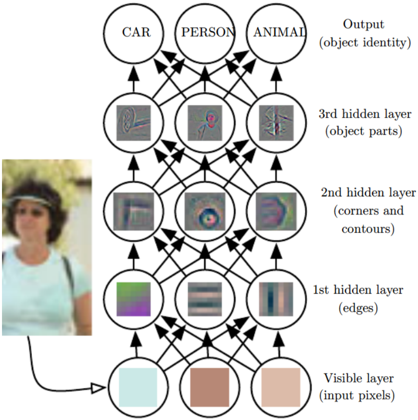

- In deep learning, a deep factor is defined by "reusing" the shallow ones
  - Face = 0.3 [corner] + 0.7 [circle]
- With a shallow structure, a deep factor needs to be replaced by ***exponentially many*** factors
  - Face = 0.3 [0.5 [vertical] + 0.5 [horizontal] ] + 0.7 [ ... ]

# Exponential Gains from Depth




- 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 [ ... ]
  - Exponentially more weights to learn
  - More training data needed

# Exponential Gains from Depth




- 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 [ ... ]
  - Exponentially more weights to learn
  - More training data needed
- Exponential gains from depth counter the exponential challenges posed by the curse of dimensionality

# Outline

# Learning Theory Revisited

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

# Learning Theory Revisited

- 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 Revisited

- 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 Revisited

- 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 Revisited

- 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 Revisited

- 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 $\mathscr{E} = C[\tilde{f}_N] - C[f^*]$:

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

- ***Approximation error $\mathscr{E}_{\mathrm{app}}$***: 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 [2, 5, 7]

# 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 [2, 5, 7]
- ***Optimization error $\mathscr{E}_{opt}$***: reduced by
  1. Running optimization alg. longer (with smaller $\rho$)
  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

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

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

# 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 $\rho$

# 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 $\rho$
  - 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 $\rho$
  - 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 $\rho$
  - 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

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

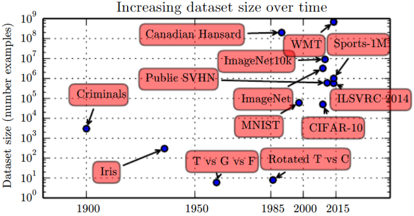
- Small-scale ML tasks:
  - Mainly constrained by $N$
  - Computing time is not an issue, so $\mathscr{E}_{\text{opt}}$ can be insignificant by choosing small $\rho$
  - Size of hypothesis is important to balance the trade-off between $\mathscr{E}_{\text{app}}$ and $\mathscr{E}_{\text{est}}$
- Large-scale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

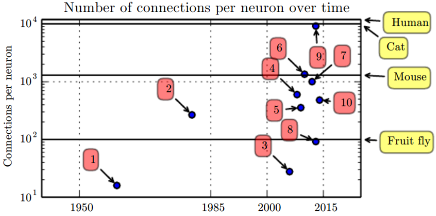
# Minimizing Excess Error

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

- 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 $\rho$
  - 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 $\mathscr{E}_{\text{app}}$

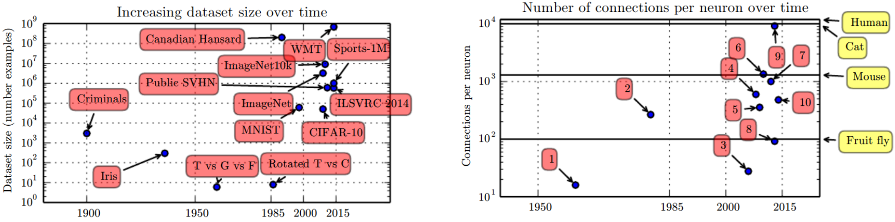
# Big Data + Big Models




9. COTS HPC unsupervised convolutional network [3]
10. GoogleLeNet [6]

# Big Data + Big Models




9. COTS HPC unsupervised convolutional network [3]
10. GoogleLeNet [6]

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

# 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

# Stochastic Gradient Descent

## Gradient Descent (GD)

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

- Needs to scan the entire dataset to descent

## (Mini-Batched) Stochastic Gradient Descent (SGD)

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

# Stochastic Gradient Descent

## Gradient Descent (GD)

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

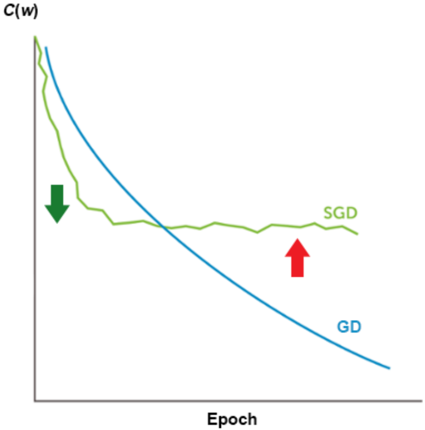
- Needs to scan the entire dataset to descent

## (Mini-Batched) Stochastic Gradient Descent (SGD)

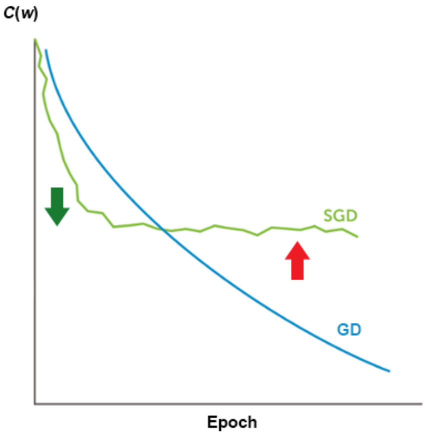
$\boldsymbol{w}^{(0)} \leftarrow$ a randon vector;
Repeat until convergence {
Randomly partition the training set $\mathbb{X}$ into ***minibatches*** $\{\mathbb{X}^{(j)}\}_j$;
$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} C(\boldsymbol{w}^{(t)}; \mathbb{X}^{(j)})$;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

- Supports ***online*** training

# GD vs. SGD

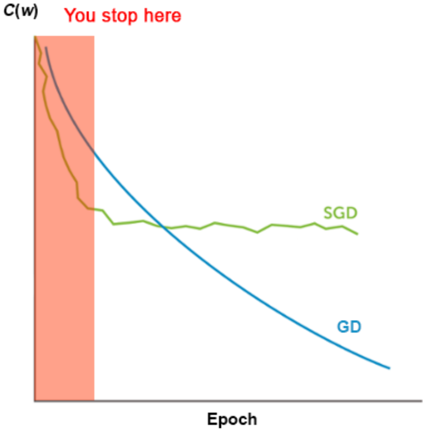


# GD vs. SGD




- Is SGD really a better algorithm?

# Yes, If You Have Big Data



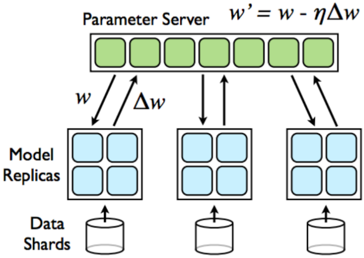

- Performance is limited by ***training time***

# Asymptotic Analysis [1]

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

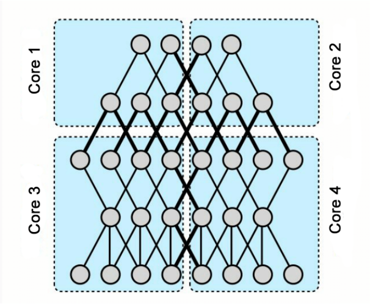
# Parallelizing SGD

## Data Parallelism




Every core trains the full model given partitioned data.

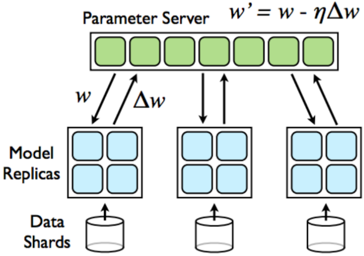
## Model Parallelism




Every core train a partitioned model partition given full data.

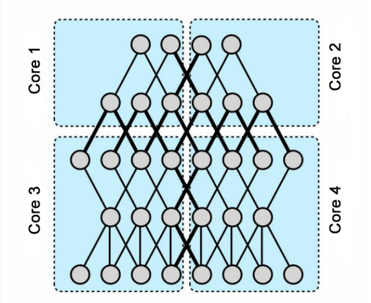
# Parallelizing SGD

## Data Parallelism




Every core trains the full model given partitioned data.

## Model Parallelism

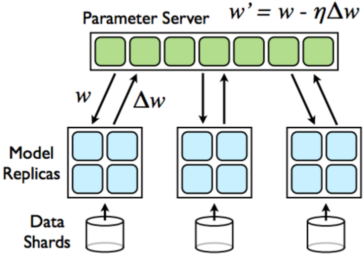



Every core train a partitioned model partition given full data.

- Normally, model parallelism exchange less parameters in a large NN and can support more cores

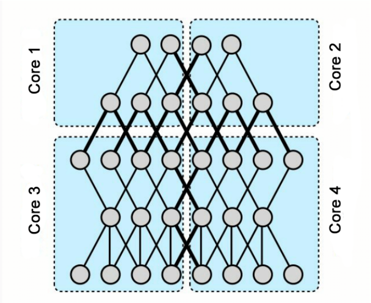
# Parallelizing SGD

**Data Parallelism**




Every core trains the full model given partitioned data.

**Model Parallelism**




Every core train a partitioned model partition given full data.

- Normally, model parallelism exchange less parameters in a large NN and can support more cores
- The effectiveness depends on settings such as CPU speed, communication latency, and bandwidth, etc.

# Reference I


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

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

[3] 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 II


[4] 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.

[5] 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.

[6] 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.

# Reference III

[7] 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.