shallow vs deep: the great watershed in learning.

Vikas Sindhwani

Google

Tuesday 2nd May, 2017

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.
- Theme: Optimization Problems in Machine Learning

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.
- Theme: Optimization Problems in Machine Learning
  - Self-driving cars, AlphaGo, Detecting Cancers, Machine Translation

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.
- Theme: Optimization Problems in Machine Learning
  - Self-driving cars, AlphaGo, Detecting Cancers, Machine Translation
  - Is the great watershed in learning between shallow and deep architectures?

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.
- Theme: Optimization Problems in Machine Learning
  - Self-driving cars, AlphaGo, Detecting Cancers, Machine Translation
  - Is the great watershed in learning between shallow and deep architectures?
  - Nonlinear techniques at the opposite ends of Rockafellars watershed:

Topics

- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the “great watershed” (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.
- Theme: Optimization Problems in Machine Learning
  - Self-driving cars, AlphaGo, Detecting Cancers, Machine Translation
  - Is the great watershed in learning between shallow and deep architectures?
  - Nonlinear techniques at the opposite ends of Rockafellars watershed:
    - **Kernel Methods**: Convex Optimization, Learning polynomials
    - **Random embeddings**: scalable kernel methods, shallow networks.
    - **Deep Learning**: Nonconvex Optimization; Architectures; TensorFlow

Topics

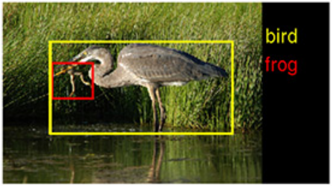
- ORF523: Convex and Conic Optimization
  - Convex Optimization: LPs, QPs, SOCPs, SDPs
  - When is an optimization problem hard? Is the "great watershed" (Rockafellar, 1993) between convexity and nonconvexity?
  - Polynomial Optimization and Sum-of-squares Programming.
- Theme: Optimization Problems in Machine Learning
  - Self-driving cars, AlphaGo, Detecting Cancers, Machine Translation
  - Is the great watershed in learning between shallow and deep architectures?
  - Nonlinear techniques at the opposite ends of Rockafellars watershed:
    - **Kernel Methods**: Convex Optimization, Learning polynomials
    - **Random embeddings**: scalable kernel methods, shallow networks.
    - **Deep Learning**: Nonconvex Optimization; Architectures; TensorFlow
  - Some intriguing vignettes: empirical observations, open questions.
    - How expressive are Deep Nets?
    - Why do Deep Nets generalize?
    - How hard is it to train Deep Nets?

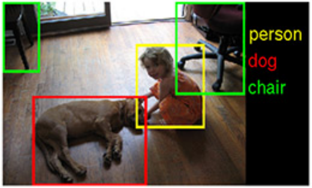
Setting

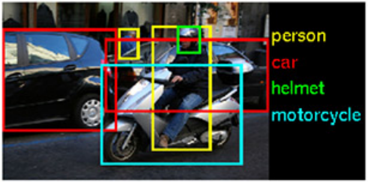
Estimate $f : \mathcal{X} \mapsto \mathcal{Y}$ from $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{l} \sim p$, $\mathbf{x}_i \in \mathcal{X}, \mathbf{y}_i \in \mathcal{Y}$.

Setting

Estimate $f : \mathcal{X} \mapsto \mathcal{Y}$ from $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{l} \sim p$, $\mathbf{x}_i \in \mathcal{X}, \mathbf{y}_i \in \mathcal{Y}$.



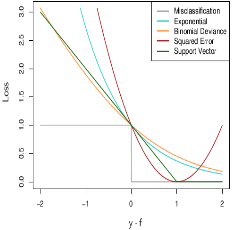


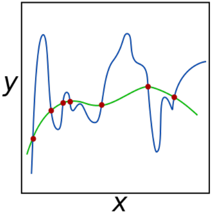


Regularized Loss Minimization

- Regularized Loss Minimization (GD, SGD) in a suitable $\mathcal{H}$,

$$\arg\min_{f \in \mathcal{H}} \sum_{i=1}^{l} V(f(\mathbf{x}_i), \mathbf{y}_i) + \Omega(f)$$

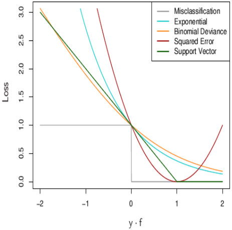


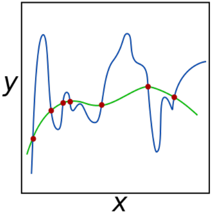


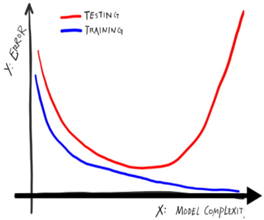
Regularized Loss Minimization

- Regularized Loss Minimization (GD, SGD) in a suitable $\mathcal{H}$,

$$\arg\min_{f \in \mathcal{H}} \sum_{i=1}^{l} V(f(\mathbf{x}_i), \mathbf{y}_i) + \Omega(f)$$







Deep
Learning
★

- *Understanding deep learning requires rethinking generalization*, Zhang et al., 2017.

Optimization in Learning

- **Optimal predictor**

$$f^\star = \arg\min_f \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

Optimization in Learning

- **Optimal predictor**

$$f^\star = \arg\min_f \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Approximation Error**: due to finite domain knowledge (*Expressivity*)

$$f^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

Optimization in Learning

- **Optimal predictor**

$$f^\star = \arg\min_f \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Approximation Error**: due to finite domain knowledge (*Expressivity*)

$$f^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Estimation Error**: due to finite data (*Generalization*)

$$\hat{f}^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \sum_{i=1}^{l} V(f(\mathbf{x}_i), \mathbf{y}_i) + \Omega(f)$$

Optimization in Learning

- **Optimal predictor**

$$f^\star = \arg\min_f \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Approximation Error**: due to finite domain knowledge (*Expressivity*)

$$f^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Estimation Error**: due to finite data (*Generalization*)

$$\hat{f}^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \sum_{i=1}^{l} V(f(\mathbf{x}_i), \mathbf{y}_i) + \Omega(f)$$

- **Optimization Error**: finite computation (*Optimization Landscape*)

$$\text{return:} \quad \hat{f}^{(i)\star}_{\mathcal{H}}$$

Optimization in Learning

- **Optimal predictor**

$$f^\star = \arg\min_f \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Approximation Error**: due to finite domain knowledge (*Expressivity*)

$$f^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p} V(f(\mathbf{x}), \mathbf{y})$$

- **Estimation Error**: due to finite data (*Generalization*)

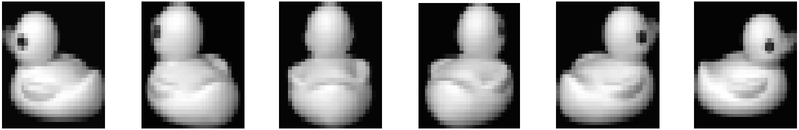
$$\hat{f}^\star_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}} \sum_{i=1}^{l} V(f(\mathbf{x}_i), \mathbf{y}_i) + \Omega(f)$$

- **Optimization Error**: finite computation (*Optimization Landscape*)

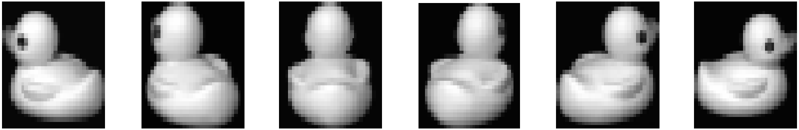
$$\text{return:} \quad \hat{f}^{(i)\star}_{\mathcal{H}}$$

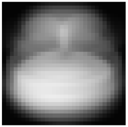
Tractable but bad model, or intractable but good model?

Nonlinearities Everywhere!

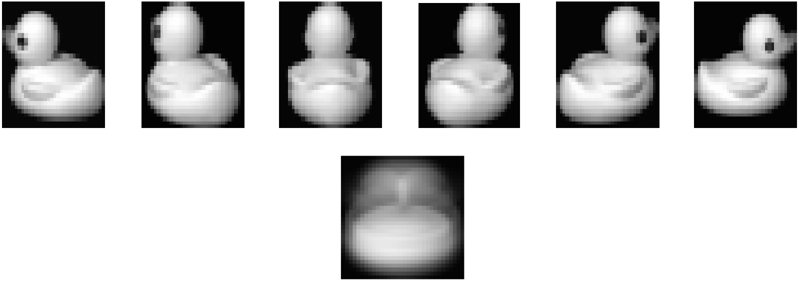


Nonlinearities Everywhere!



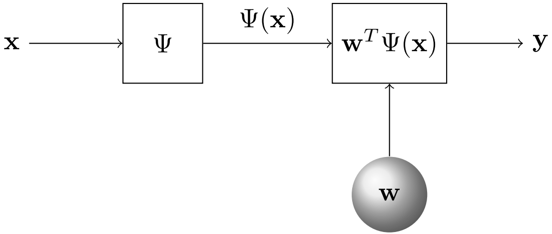


Nonlinearities Everywhere!



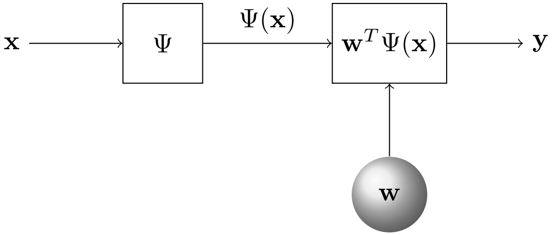
Large $l \implies$ Big models: $\mathcal{H}$ "rich" /non-parametric/nonlinear.

Choice of Nonlinear Hypothesis Space: Kernels



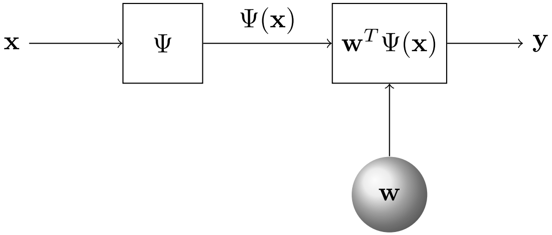
- e.g., Linear functions and Polynomials.
- Infinite-dimensional nonlinear embeddings. Fully non-parameteric.

Choice of Nonlinear Hypothesis Space: Kernels



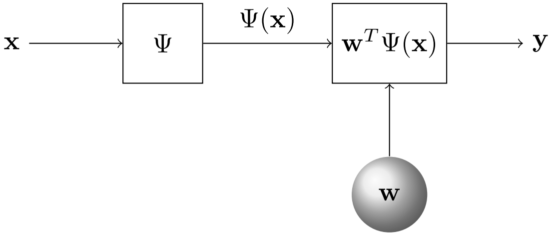
- e.g., Linear functions and Polynomials.
- Infinite-dimensional nonlinear embeddings. Fully non-parameteric.
- Functional Analysis (Aronszajn, Bergman (1950s)); Statistics (Parzen (1960s)); PDEs; Numerical Analysis...

Choice of Nonlinear Hypothesis Space: Kernels



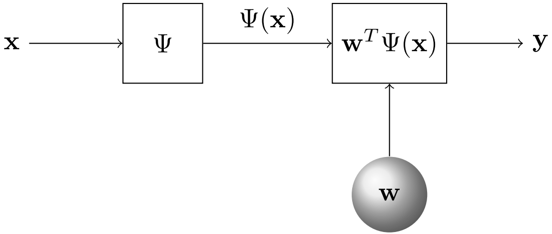
- e.g., Linear functions and Polynomials.
- Infinite-dimensional nonlinear embeddings. Fully non-parameteric.
- Functional Analysis (Aronszajn, Bergman (1950s)); Statistics (Parzen (1960s)); PDEs; Numerical Analysis...
- ML in 1990's: Nonlinear classification, regression, clustering, time-series analysis, dynamical systems, hypothesis testing, causal modeling, ...

Choice of Nonlinear Hypothesis Space: Kernels



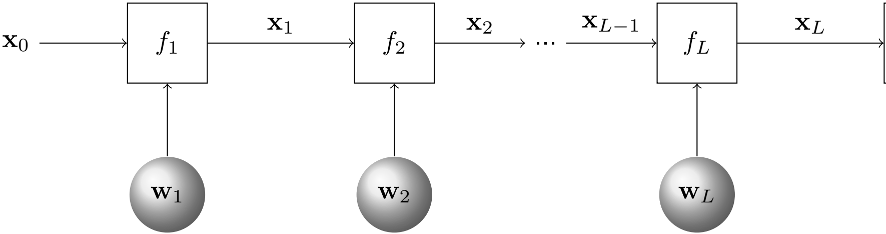
- e.g., Linear functions and Polynomials.
- Infinite-dimensional nonlinear embeddings. Fully non-parameteric.
- Functional Analysis (Aronszajn, Bergman (1950s)); Statistics (Parzen (1960s)); PDEs; Numerical Analysis. . .
- ML in 1990's: Nonlinear classification, regression, clustering, time-series analysis, dynamical systems, hypothesis testing, causal modeling, . . .
- Convex Optimization.

Choice of Nonlinear Hypothesis Space: Kernels



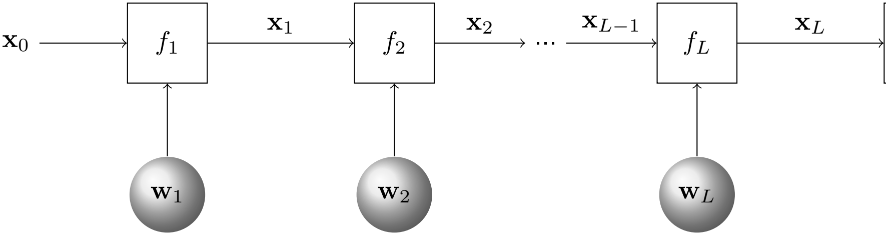
- e.g., Linear functions and Polynomials.
- Infinite-dimensional nonlinear embeddings. Fully non-parameteric.
- Functional Analysis (Aronszajn, Bergman (1950s)); Statistics (Parzen (1960s)); PDEs; Numerical Analysis. . .
- ML in 1990's: Nonlinear classification, regression, clustering, time-series analysis, dynamical systems, hypothesis testing, causal modeling, . . .
- Convex Optimization.

Choice of Nonlinear Hypothesis Space: Deep Learning



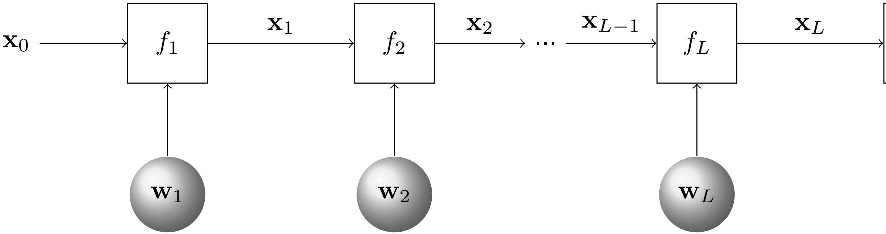
- Compositions: Raw data to higher abstractions (representation learning)

Choice of Nonlinear Hypothesis Space: Deep Learning



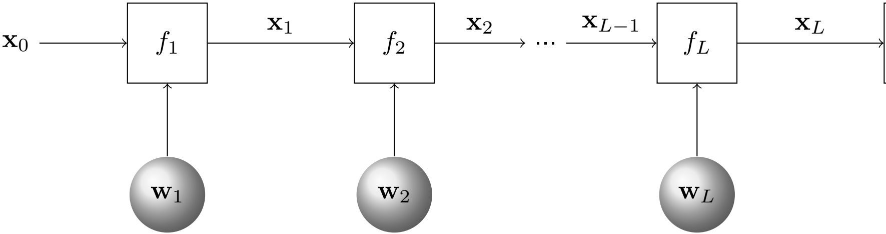
- Compositions: Raw data to higher abstractions (representation learning)
- Multilayer Perceptrons: $f_l(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$ (sigmoids, ReLU)

Choice of Nonlinear Hypothesis Space: Deep Learning



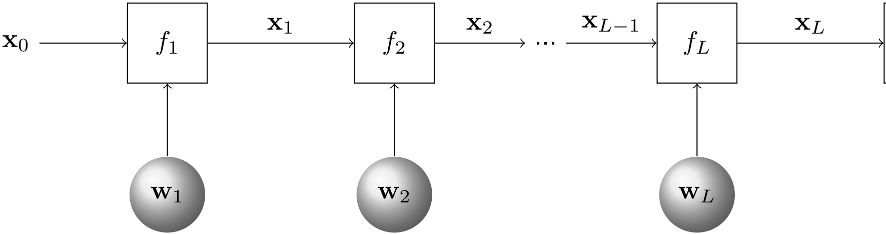
- Compositions: Raw data to higher abstractions (representation learning)
- Multilayer Perceptrons: $f_l(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$ (sigmoids, ReLU)
- Turing (1950): In considering the functions of the mind or the brain we find certain operations which we can explain in purely mechanical terms. This we say does not correspond to the real mind: it is a sort of skin which we must strip off if we are to find the real mind. Proceeding in this way do we ever come to the "real" mind, or do we eventually come to the skin which has nothing in it? In the latter case the whole mind is mechanical.

Choice of Nonlinear Hypothesis Space: Deep Learning



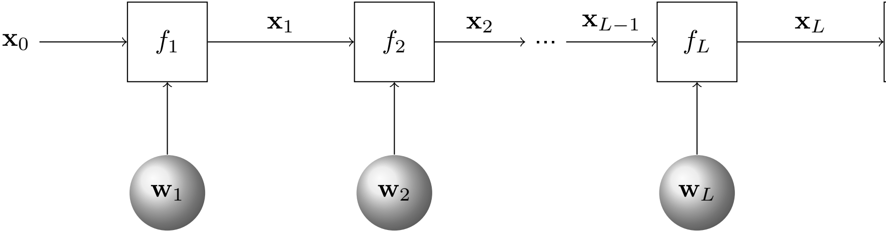
- Compositions: Raw data to higher abstractions (representation learning)
- Multilayer Perceptrons: $f_l(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$ (sigmoids, ReLU)
- Turing (1950): In considering the functions of the mind or the brain we find certain operations which we can explain in purely mechanical terms. This we say does not correspond to the real mind: it is a sort of skin which we must strip off if we are to find the real mind. Proceeding in this way do we ever come to the "real" mind, or do we eventually come to the skin which has nothing in it? In the latter case the whole mind is mechanical.
- Rosenblatt (1957). NYT, 1958: New Navy Device Learns By Doing: The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.

Choice of Nonlinear Hypothesis Space: Deep Learning



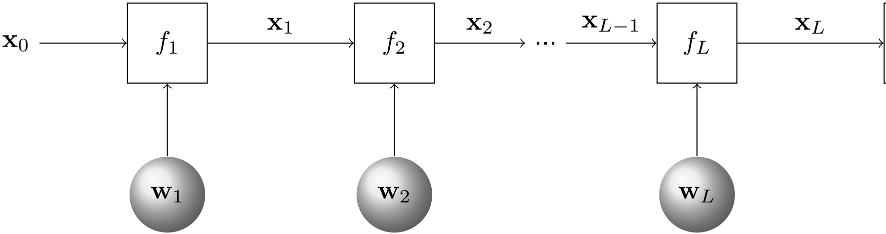
- Compositions: Raw data to higher abstractions (representation learning)
- Multilayer Perceptrons: $f_l(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$ (sigmoids, ReLU)
- Turing (1950): In considering the functions of the mind or the brain we find certain operations which we can explain in purely mechanical terms. This we say does not correspond to the real mind: it is a sort of skin which we must strip off if we are to find the real mind. Proceeding in this way do we ever come to the "real" mind, or do we eventually come to the skin which has nothing in it? In the latter case the whole mind is mechanical.
- Rosenblatt (1957). NYT, 1958: New Navy Device Learns By Doing: The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.
- Perceptron (Minski and Pappert, 1969) [Olarzan, A Sociological Study.., 1996]
- Backprop (1986), Reverse-mode differentiation (1960s-70s), CNNs (1980s, 2012-)

Choice of Nonlinear Hypothesis Space: Deep Learning



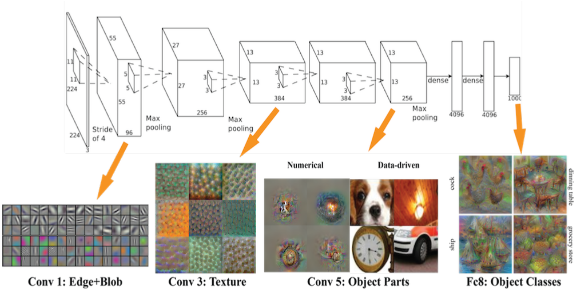
- Compositions: Raw data to higher abstractions (representation learning)
- Multilayer Perceptrons: $f_l(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$ (sigmoids, ReLU)
- Turing (1950): In considering the functions of the mind or the brain we find certain operations which we can explain in purely mechanical terms. This we say does not correspond to the real mind: it is a sort of skin which we must strip off if we are to find the real mind. Proceeding in this way do we ever come to the "real" mind, or do we eventually come to the skin which has nothing in it? In the latter case the whole mind is mechanical.
- Rosenblatt (1957). NYT, 1958: New Navy Device Learns By Doing: The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.
- Perceptron (Minski and Pappert, 1969) [Olarzan, A Sociological Study.., 1996]
- Backprop (1986), Reverse-mode differentiation (1960s-70s), CNNs (1980s, 2012-)
- Breadth-Width tradeoffs
- Non-convex Optimization. Domain-agnostic recipe.

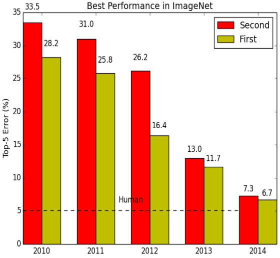
Choice of Nonlinear Hypothesis Space: Deep Learning



- Compositions: Raw data to higher abstractions (representation learning)
- Multilayer Perceptrons: $f_l(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$ (sigmoids, ReLU)
- Turing (1950): In considering the functions of the mind or the brain we find certain operations which we can explain in purely mechanical terms. This we say does not correspond to the real mind: it is a sort of skin which we must strip off if we are to find the real mind. Proceeding in this way do we ever come to the "real" mind, or do we eventually come to the skin which has nothing in it? In the latter case the whole mind is mechanical.
- Rosenblatt (1957). NYT, 1958: New Navy Device Learns By Doing: The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.
- Perceptron (Minski and Pappert, 1969) [Olarzan, A Sociological Study.., 1996]
- Backprop (1986), Reverse-mode differentiation (1960s-70s), CNNs (1980s, 2012-)
- Breadth-Width tradeoffs
- Non-convex Optimization. Domain-agnostic recipe.

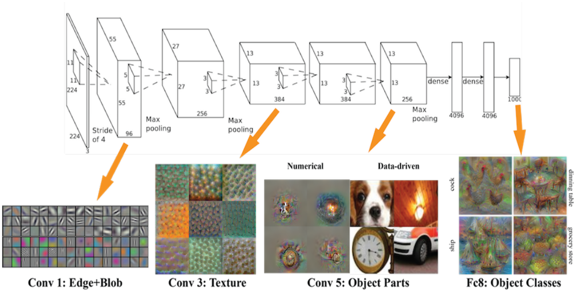
The Watershed Moment: Imagenet, 2012

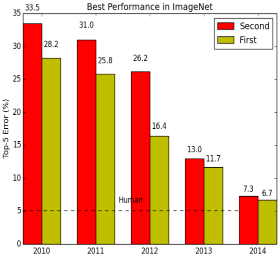




- Many statistical and computational ingredients:
  - Large datasets (ILSVRC since 2010)

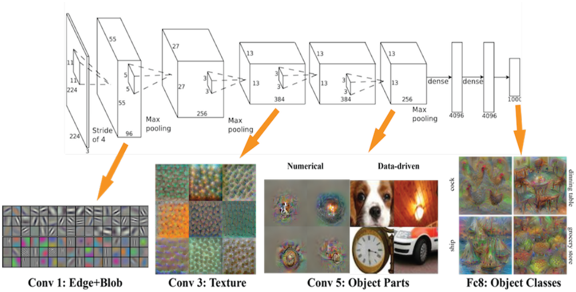
The Watershed Moment: Imagenet, 2012

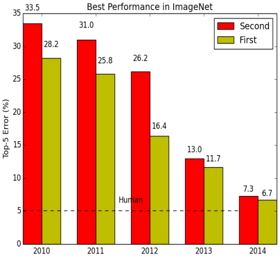




- Many statistical and computational ingredients:
  - Large datasets (ILSVRC since 2010)
  - Large statistical capacity ($1.2M$ images, $60M$ params)

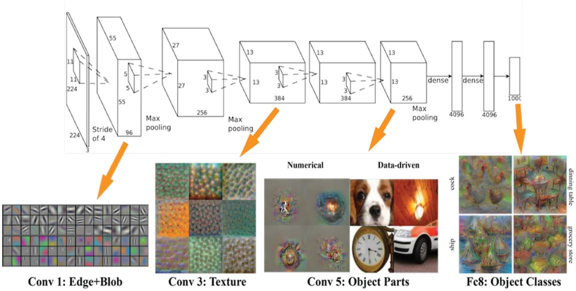
The Watershed Moment: Imagenet, 2012

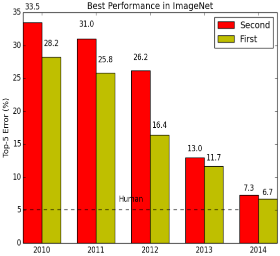




- Many statistical and computational ingredients:
  - Large datasets (ILSVRC since 2010)
  - Large statistical capacity ($1.2M$ images, $60M$ params)
  - Distributed computation

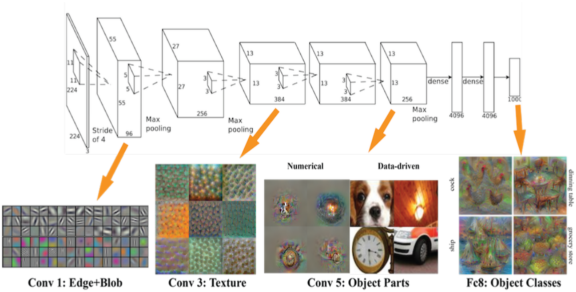
The Watershed Moment: Imagenet, 2012

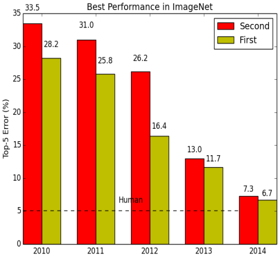




- Many statistical and computational ingredients:
  - Large datasets (ILSVRC since 2010)
  - Large statistical capacity ($1.2M$ images, $60M$ params)
  - Distributed computation
  - Depth, Invariant feature learning (transferrable to other tasks)

The Watershed Moment: Imagenet, 2012





- Many statistical and computational ingredients:
  - Large datasets (ILSVRC since 2010)
  - Large statistical capacity ($1.2M$ images, $60M$ params)
  - Distributed computation
  - Depth, Invariant feature learning (transferrable to other tasks)
  - Engineering: Dropout, ReLU …
- Many astonishing results since then.

CNNs

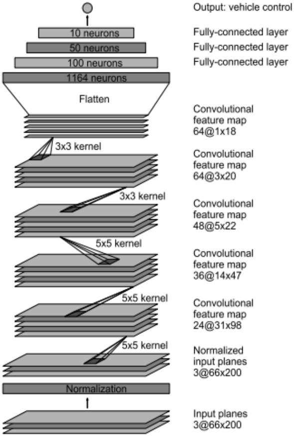

```
conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=None,
    data_format=None,
    name=None
)
```

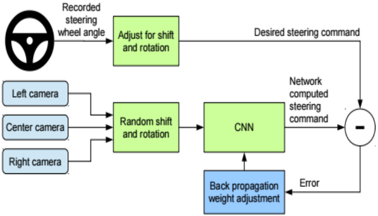
- `input`: A `Tensor`. Must be one of the following types: `half`, `float32`, `float64`. A 4-D tensor. Th order is interpreted according to the value of `data_format`, see below for details.
- `filter`: A `Tensor`. Must have the same type as `input`. A 4-D tensor of shape `[filter_height, filter_width, in_channels, out_channels]`
- `strides`: A list of `ints`. 1-D tensor of length 4. The stride of the sliding window for each dimensio The dimension order is determined by the value of `data_format`, see below for details.
- `padding`: A `string` from: `"SAME", "VALID"`. The type of padding algorithm to use.

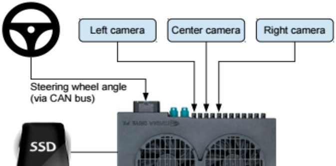
Self-Driving Cars

Figure: End-to-End Learning for Self-driving Cars, Bojarski et al, 2016

Self-Driving Cars







Self-Driving Cars

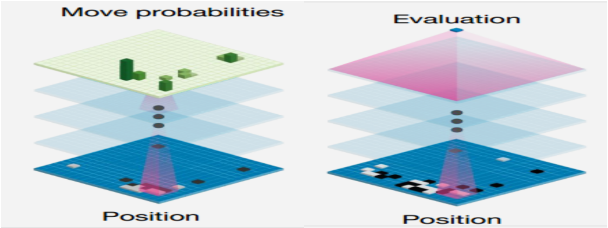
AlphaGo: CNNs, Deep-RL + Tree Search

- Tree complexity b^d: Chess (35^{80}), Go (250^{150})
- Hard to evaluate a mid-position.
- 19x19 board-img (48 planes), player/opponent, 12-layer CNNs.
- 30M human games, 4 weeks, 50 GPUs $\rightarrow$ 57% supervised learning $\rightarrow$ 80% RL [human-level]

D. Silver et. al., Mastering the Game of Go with DNNs and Tree Search, Nature 2016

AlphaGo: CNNs, Deep-RL + Tree Search

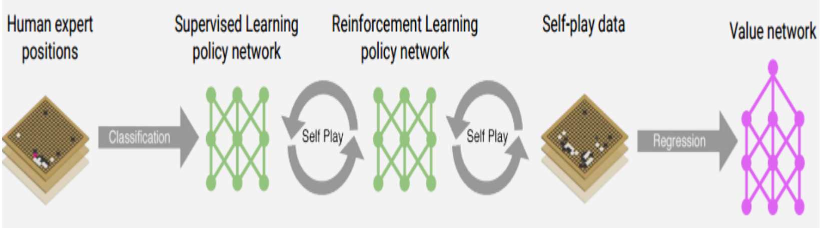
- Tree complexity b^d: Chess (35^{80}), Go (250^{150})
- Hard to evaluate a mid-position.
- 19x19 board-img (48 planes), player/opponent, 12-layer CNNs.
- 30M human games, 4 weeks, 50 GPUs → 57% supervised learning → 80% RL [human-level]



D. Silver et. al., Mastering the Game of Go with DNNs and Tree Search, Nature 2016

AlphaGo: CNNs, Deep-RL + Tree Search

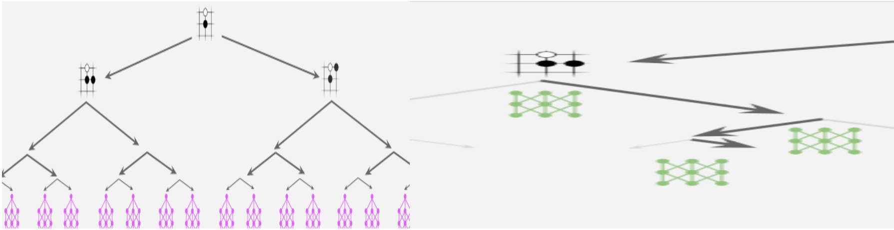
- Tree complexity b^d: Chess (35^{80}), Go (250^{150})
- Hard to evaluate a mid-position.
- 19x19 board-img (48 planes), player/opponent, 12-layer CNNs.
- 30M human games, 4 weeks, 50 GPUs $\rightarrow$ 57% supervised learning $\rightarrow$ 80% RL [human-level]



D. Silver et. al., Mastering the Game of Go with DNNs and Tree Search, Nature 2016

AlphaGo: CNNs, Deep-RL + Tree Search

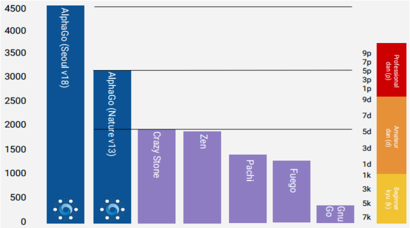
- Tree complexity b^d: Chess (35^{80}), Go (250^{150})
- Hard to evaluate a mid-position.
- 19x19 board-img (48 planes), player/opponent, 12-layer CNNs.
- 30M human games, 4 weeks, 50 GPUs $\rightarrow$ 57% supervised learning $\rightarrow$ 80% RL [human-level]



D. Silver et. al., Mastering the Game of Go with DNNs and Tree Search, Nature 2016

AlphaGo: CNNs, Deep-RL + Tree Search

- Tree complexity b^d: Chess (35^{80}), Go (250^{150})
- Hard to evaluate a mid-position.
- 19x19 board-img (48 planes), player/opponent, 12-layer CNNs.
- 30M human games, 4 weeks, 50 GPUs $\rightarrow$ 57% supervised learning $\rightarrow$ 80% RL [human-level]



D. Silver et. al., Mastering the Game of Go with DNNs and Tree Search, Nature 2016

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  – 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  – 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  – 1999: Introduced to Kernel Methods - by DNN researchers!

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  - 1999: Introduced to Kernel Methods - by DNN researchers!
  - 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  - 1999: Introduced to Kernel Methods - by DNN researchers!
  - 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!
- Why Kernel Methods?

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  – 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  – 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  – 1999: Introduced to Kernel Methods - by DNN researchers!
  – 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!
- Why Kernel Methods?
  – Local Minima free - stronger role of Convex Optimization.

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  - 1999: Introduced to Kernel Methods - by DNN researchers!
  - 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!
- Why Kernel Methods?
  - Local Minima free - stronger role of Convex Optimization.
  - Theoretically appealing

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  - 1999: Introduced to Kernel Methods - by DNN researchers!
  - 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!
- Why Kernel Methods?
  - Local Minima free - stronger role of Convex Optimization.
  - Theoretically appealing
  - Handle non-vectorial data; high-dimensional data
  - Easier model selection via continuous optimization.

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  - 1999: Introduced to Kernel Methods - by DNN researchers!
  - 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!
- Why Kernel Methods?
  - Local Minima free - stronger role of Convex Optimization.
  - Theoretically appealing
  - Handle non-vectorial data; high-dimensional data
  - Easier model selection via continuous optimization.
  - Matched NN in many cases, although didnt scale wrt n as well.
- So what changed?

Machine Learning in 1990s

- Convolutional Neural Networks (Fukushima 1980; Lecun et al 1989)
  - 3 days to train on USPS ($n = 7291$; digit recognition) on Sun Sparcstation 1 (33MHz clock speed, 64MB RAM)
- Personal history:
  - 1998 (Deep Blue, Google, Email): Train DNN on UCI Wine dataset.
  - 1999: Introduced to Kernel Methods - by DNN researchers!
  - 2003-4: NN paper at JMLR rejected; accepted in IEEE Trans. Neural Nets with a kernel methods section!
- Why Kernel Methods?
  - Local Minima free - stronger role of Convex Optimization.
  - Theoretically appealing
  - Handle non-vectorial data; high-dimensional data
  - Easier model selection via continuous optimization.
  - Matched NN in many cases, although didnt scale wrt n as well.
- So what changed?
  - More data, parallel algorithms, hardware? Better DNN training? ...

Kernel Methods vs Neural Networks (Pre-Google)

1. Jackel bets (one fancy dinner) that by March 14, 2000, people will understand quantitatively why big neural nets working on large databases are not so bad. (Understanding means that there will be clear conditions and bounds)

Vapnik bets (one fancy dinner) that Jackel is wrong.

But .. If Vapnik figures out the bounds and conditions, Vapnik still wins the bet.

****

2. Vapnik bets (one fancy dinner) that by March 14, 2005, no one in his right mind will use neural nets that are essentially like those used in 1995.

Jackel bets (one fancy dinner) that Vapnik is wrong

V. Vapnik — 3/14/95

L. Jackel — 3/14//95

Witnessed by Y. LeCun — 3/14/95

Kernel Methods vs Neural Networks

Geoff Hinton facts meme maintained at
`http://yann.lecun.com/ex/fun/`

Kernel Methods vs Neural Networks

Geoff Hinton facts meme maintained at
`http://yann.lecun.com/ex/fun/`

- All **kernels** that ever dared approaching Geoff Hinton woke up convolved.
- The only **kernel** Geoff Hinton has ever used is a **kernel** of truth.
- If you defy Geoff Hinton, he will maximize your entropy in no time. Your free energy will be gone even before you reach equilibrium.

Kernel Methods vs Neural Networks

Geoff Hinton facts meme maintained at
`http://yann.lecun.com/ex/fun/`

- All **kernels** that ever dared approaching Geoff Hinton woke up convolved.
- The only **kernel** Geoff Hinton has ever used is a **kernel** of truth.
- If you defy Geoff Hinton, he will maximize your entropy in no time. Your free energy will be gone even before you reach equilibrium.

Are there synergies between these fields towards design of even better (faster and more accurate) algorithms?

Linear Hypotheses

- $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^n$. Assume $\mathcal{Y} \subset \mathbb{R}$ setting.

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{l} (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_2^2$$

$$\mathbf{w}^\star = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_d)^{-1} \left(\mathbf{X}^T\mathbf{y}\right)$$

$$\mathbf{X} = \begin{pmatrix} \vdots \\ \mathbf{x}_i^T \\ \vdots \end{pmatrix} \in \mathbb{R}^{l\times n}$$

- $n \times n$ linear system $\implies O(ln^2 + n^3)$ training time assuming no structure (e.g., sparsity).
- $O(n)$ prediction time.
- High Approximation error

Polynomials: The expensive way

- Homogeneous degree-d polynomial

$$f(\mathbf{x}) = \mathbf{w}^T \Psi_{n,d}(\mathbf{x}), \quad \mathbf{w} \in \mathbb{R}^s, \quad s = \binom{d+n-1}{d}$$

$$\Psi_{n,d}(\mathbf{x}) = \begin{pmatrix} \vdots \\ \sqrt{\binom{d}{\alpha}}\mathbf{x}^\alpha \\ \vdots \end{pmatrix} \in \mathbb{R}^s$$

$$\alpha = (\alpha_1 \ldots \alpha_n), \sum_i \alpha_i = d, \binom{d}{\alpha} = \frac{d}{\alpha_1! \ldots \alpha_n!}$$

$$\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$$

- Construct $\mathbf{Z} \in \mathbb{R}^{n \times s}$ with rows $\Psi_{n,d}(\mathbf{x}_i)$ and solve in $O(s^3)$ (!) time:

$$\mathbf{w}^\star = (\mathbf{Z}^T\mathbf{Z} + \lambda \mathbf{I}_d)^{-1} \left(\mathbf{Z}^T \mathbf{y}\right) \tag{1}$$

- Note: $n = 100, d = 4 \implies s > 4M$

Polynomials

- Consider the subspace of $\mathbb{R}^s$ spanned by the data,

$$S = \operatorname{span}\left(\Psi(\mathbf{x}_1)\ldots\Psi(\mathbf{x}_l)\right) = \{\mathbf{v} \in \mathbb{R}^s : \mathbf{v} = \sum_{i=1}^{l} \alpha_i \Psi(\mathbf{x}_i)\}$$

Polynomials

- Consider the subspace of $\mathbb{R}^s$ spanned by the data,

$$S = \mathrm{span}\left(\Psi(\mathbf{x}_1) \ldots \Psi(\mathbf{x}_l)\right) = \{\mathbf{v} \in \mathbb{R}^s : \mathbf{v} = \sum_{i=1}^{l} \alpha_i \Psi(\mathbf{x}_i)\}$$

- So, $\mathbb{R}^s = S + S^{\perp}$. Hence, for any vector $\mathbf{w} \in \mathbb{R}^s$,

$$\mathbf{w} = \mathbf{w}_S + \mathbf{w}_{S^{\perp}}$$

Polynomials

- Consider the subspace of $\mathbb{R}^s$ spanned by the data,

$$S = \operatorname{span}\left(\Psi(\mathbf{x}_1)\ldots\Psi(\mathbf{x}_l)\right) = \{\mathbf{v} \in \mathbb{R}^s : \mathbf{v} = \sum_{i=1}^{l} \alpha_i \Psi(\mathbf{x}_i)\}$$

- So, $\mathbb{R}^s = S + S^{\perp}$. Hence, for any vector $\mathbf{w} \in \mathbb{R}^s$,

$$\mathbf{w} = \mathbf{w}_S + \mathbf{w}_{S^{\perp}}$$

- The search of a minimizer can be reduced to S because,

$$\begin{aligned}\sum_{i=1}^{l} V(\mathbf{w}^T\Psi(\mathbf{x}_i)) + \lambda\|\mathbf{w}\|_2^2 &\geq \sum_{i=1}^{l} V(\mathbf{w}_S^T\Psi(\mathbf{x}_i)) + \lambda\|\mathbf{w}_S\|_2^2 + \|\mathbf{w}_S^{\perp}\|_2^2 \\ &\geq \sum_{i=1}^{l} V(\mathbf{w}_S^T\Psi(\mathbf{x}_i)) + \lambda\|\mathbf{w}_S\|_2^2 \qquad (2)\end{aligned}$$

Polynomials

- Consider the subspace of $\mathbb{R}^s$ spanned by the data,

$$S = \text{span}\left(\Psi(\mathbf{x}_1)\ldots\Psi(\mathbf{x}_l)\right) = \{\mathbf{v}\in\mathbb{R}^s : \mathbf{v} = \sum_{i=1}^{l}\alpha_i\Psi(\mathbf{x}_i)\}$$

- So, $\mathbb{R}^s = S + S^\perp$. Hence, for any vector $\mathbf{w}\in\mathbb{R}^s$,

$$\mathbf{w} = \mathbf{w}_S + \mathbf{w}_{S^\perp}$$

- The search of a minimizer can be reduced to S because,

$$\begin{aligned}\sum_{i=1}^{l} V(\mathbf{w}^T\Psi(\mathbf{x}_i)) + \lambda\|\mathbf{w}\|_2^2 &\geq \sum_{i=1}^{l} V(\mathbf{w}_S^T\Psi(\mathbf{x}_i)) + \lambda\|\mathbf{w}_S\|_2^2 + \|\mathbf{w}_S^\perp\|_2^2 \\ &\geq \sum_{i=1}^{l} V(\mathbf{w}_S^T\Psi(\mathbf{x}_i)) + \lambda\|\mathbf{w}_S\|_2^2 \qquad (2)\end{aligned}$$

- Argument holds for any loss (convex or non-convex, additive or not), but needs orthogonality (l_2 regularizer) [**Representer Theorem**]

Polynomials

- Hence, $\mathbf{w}^\star = \sum_{i=1}^{l} \beta_i \Psi(\mathbf{x}_i) \in S$ for $\beta \in \mathbb{R}^l$, and so we can solve:

$$\arg\min_{\mathbf{w} \in \mathbb{R}^d} \quad \sum_{i=1}^{l} (y_i - \mathbf{w}^T \Psi(\mathbf{x}_i))^2 + \lambda \|\mathbf{w}\|_2^2$$

Polynomials

- Hence, $\mathbf{w}^\star = \sum_{i=1}^{l} \beta_i \Psi(\mathbf{x}_i) \in S$ for $\beta \in \mathbb{R}^l$, and so we can solve:

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \quad \sum_{i=1}^{l} (y_i - \mathbf{w}^T \Psi(\mathbf{x}_i))^2 + \lambda \|\mathbf{w}\|_2^2$$

$$\arg\min_{\beta\in\mathbb{R}^l} \quad \sum_{j=1}^{l} \left(y_j - \sum_{i=1}^{l} \beta_i \Psi(\mathbf{x}_i)^T \Psi(\mathbf{x}_j) \right)^2 + \lambda \sum_{i,j=1}^{l} \beta_i \beta_j \Psi(\mathbf{x}_i)^T \Psi(\mathbf{x}_j)$$

Polynomials

- Hence, $\mathbf{w}^\star = \sum_{i=1}^{l} \beta_i \Psi(\mathbf{x}_i) \in S$ for $\beta \in \mathbb{R}^l$, and so we can solve:

$$
\begin{aligned}
\arg\min_{\mathbf{w}\in\mathbb{R}^d} & \quad \sum_{i=1}^{l} (y_i - \mathbf{w}^T \Psi(\mathbf{x}_i))^2 + \lambda \|\mathbf{w}\|_2^2 \\
\arg\min_{\beta\in\mathbb{R}^l} & \quad \sum_{j=1}^{l} \left(y_j - \sum_{i=1}^{l} \beta_i \Psi(\mathbf{x}_i)^T \Psi(\mathbf{x}_j) \right)^2 + \lambda \sum_{i,j=1}^{l} \beta_i \beta_j \Psi(\mathbf{x}_i)^T \Psi(\mathbf{x}_j) \\
\arg\min_{\beta\in\mathbb{R}^l} & \quad \sum_{j=1}^{l} \left(y_j - (\mathbf{G}\beta)_j \right)^2 + \lambda \beta^T \mathbf{G} \beta \\
\mathbf{G}_{ij} & = \Psi(\mathbf{x}_i)^T \Psi(\mathbf{x}_j) \\
\beta^\star & = (\mathbf{G} + \lambda \mathbf{I}_d)^{-1} \mathbf{y}
\end{aligned}
$$

- $O(l^3 + sl^2)$ training time - $O(s^3)$ cost eliminated.
- Inference time ($O(s)$, or $O(ls)$):

$$
f(\mathbf{x}) = \mathbf{w}^T \Psi(\mathbf{x}) = \sum_{i=1}^{l} \beta_i \Psi(\mathbf{x}_i)^T \Psi(\mathbf{x})
$$

Polynomials

- Multinomial Theorem

$$(z_1 + z_2 + \ldots + z_n)^d = \sum_{\alpha : |\alpha| = d} \binom{d}{\alpha} \mathbf{z}^\alpha$$

- Implicit computation of inner products

$$\Psi(\mathbf{x})^T \Psi(\mathbf{x}') \quad = \quad \sum_{i=1}^{s} \binom{d}{\alpha} \mathbf{x}^\alpha \mathbf{x}'^\alpha = \sum_{i=1}^{s} \binom{d}{\alpha} x_1^{\alpha_1} \ldots x_n^{\alpha_n} x_1'^{\alpha_1} \ldots x_n'^{\alpha_n}$$

Polynomials

- Multinomial Theorem

$$(z_1 + z_2 + \ldots + z_n)^d = \sum_{\alpha:|\alpha|=d} \binom{d}{\alpha} \mathbf{z}^\alpha$$

- Implicit computation of inner products

$$\begin{aligned}
\Psi(\mathbf{x})^T \Psi(\mathbf{x}') &= \sum_{i=1}^{s} \binom{d}{\alpha} \mathbf{x}^\alpha \mathbf{x}'^\alpha = \sum_{i=1}^{s} \binom{d}{\alpha} x_1^{\alpha_1} \ldots x_n^{\alpha_n} x_1^{'\alpha_1} \ldots x_n^{'\alpha_n} \\
&= \sum_{i=1}^{s} \binom{d}{\alpha} (x_1 x_1')^{\alpha_1} (x_2 x_2')^{\alpha_2} \ldots (x_n x_n')^{\alpha_n}
\end{aligned}$$

Polynomials

- Multinomial Theorem

$$(z_1 + z_2 + \ldots + z_n)^d = \sum_{\alpha:|\alpha|=d} \binom{d}{\alpha} \mathbf{z}^\alpha$$

- Implicit computation of inner products

$$\begin{aligned}
\Psi(\mathbf{x})^T \Psi(\mathbf{x}') &= \sum_{i=1}^{s} \binom{d}{\alpha} \mathbf{x}^\alpha \mathbf{x}'^\alpha = \sum_{i=1}^{s} \binom{d}{\alpha} x_1^{\alpha_1} \ldots x_n^{\alpha_n} x_1^{'\alpha_1} \ldots x_n^{'\alpha_n} \\
&= \sum_{i=1}^{s} \binom{d}{\alpha} (x_1 x_1')^{\alpha_1} (x_2 x_2')^{\alpha_2} \ldots (x_n x_n')^{\alpha_n} \\
&= (x_1 x_1' + \ldots x_n x_n')^d
\end{aligned}$$

Polynomials

- Multinomial Theorem

$$(z_1 + z_2 + \ldots + z_n)^d = \sum_{\alpha:|\alpha|=d} \binom{d}{\alpha} \mathbf{z}^\alpha$$

- Implicit computation of inner products

$$\begin{aligned}
\Psi(\mathbf{x})^T \Psi(\mathbf{x}') &= \sum_{i=1}^{s} \binom{d}{\alpha} \mathbf{x}^\alpha \mathbf{x}'^\alpha = \sum_{i=1}^{s} \binom{d}{\alpha} x_1^{\alpha_1} \ldots x_n^{\alpha_n} x_1^{'\alpha_1} \ldots x_n^{'\alpha_n} \\
&= \sum_{i=1}^{s} \binom{d}{\alpha} (x_1 x_1')^{\alpha_1} (x_2 x_2')^{\alpha_2} \ldots (x_n x_n')^{\alpha_n} \\
&= (x_1 x_1' + \ldots x_n x_n')^d \\
&= (\mathbf{x}^T \mathbf{x}')^d
\end{aligned}$$

Polynomials

- Multinomial Theorem

$$(z_1 + z_2 + \ldots + z_n)^d = \sum_{\alpha:|\alpha|=d} \binom{d}{\alpha} \mathbf{z}^\alpha$$

- Implicit computation of inner products

$$\begin{aligned}
\Psi(\mathbf{x})^T \Psi(\mathbf{x}') &= \sum_{i=1}^{s} \binom{d}{\alpha} \mathbf{x}^\alpha \mathbf{x}'^\alpha = \sum_{i=1}^{s} \binom{d}{\alpha} x_1^{\alpha_1} \ldots x_n^{\alpha_n} x_1'^{\alpha_1} \ldots x_n'^{\alpha_n} \\
&= \sum_{i=1}^{s} \binom{d}{\alpha} (x_1 x_1')^{\alpha_1} (x_2 x_2')^{\alpha_2} \ldots (x_n x_n')^{\alpha_n} \\
&= (x_1 x_1' + \ldots x_n x_n')^d \\
&= (\mathbf{x}^T \mathbf{x}')^d
\end{aligned}$$

- $O(l^3 + l^2 n)$ training and $O(ln)$ predicting speed.
- Complexity coming from s has been completely eliminated (!). [**Kernel Trick**]

Polynomials: Algorithm

- Algorithm
  - Start with $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$
  - Construct Gram matrix: $\mathbf{G}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ on the training samples.
  - Solve: $\beta^\star = (\mathbf{G} + \lambda \mathbf{I}_d)^{-1} \mathbf{y}$
  - Return $f^\star(\mathbf{x}) = \sum_{i=1}^{l} \beta_i k(\mathbf{x}_i, \mathbf{x})$
- $f^\star$ is the optimal degree-d polynomial solving the learning problem, in complexity independent of d.

Polynomials: Algorithm

- Algorithm
  - Start with $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + 1)^d$
  - Construct Gram matrix: $\mathbf{G}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ on the training samples.
  - Solve: $\beta^\star = (\mathbf{G} + \lambda\mathbf{I}_d)^{-1}\,\mathbf{y}$
  - Return $f^\star(\mathbf{x}) = \sum_{i=1}^{l} \beta_i k(\mathbf{x}_i, \mathbf{x})$
- $f^\star$ is the optimal degree-d polynomial solving the learning problem, in complexity independent of d.
- What other forms of k correspond to linear learning in high-dimensional nonlinear embeddings of the data?

Symmetric, positive semi-definite functions

- **Definition:** A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is p.s.d if for any finite collection of points $\mathbf{x}_1 \ldots \mathbf{x}_l$, the $l \times l$ Gram matrix

$$\mathbf{G}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

is positive semi-definite, i.e. for any vector $\beta \in \mathbb{R}^l$

$$\beta^T \mathbf{G} \beta = \sum_{ij} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Symmetric, positive semi-definite functions

- **Definition:** A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is p.s.d if for any finite collection of points $\mathbf{x}_1 \dots \mathbf{x}_l$, the $l \times l$ Gram matrix

$$\mathbf{G}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

is positive semi-definite, i.e. for any vector $\beta \in \mathbb{R}^l$

$$\beta^T \mathbf{G} \beta = \sum_{ij} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

- **Theorem**[Mercer]: If k is symmetric, p.s.d, $\mathcal{X}$ is compact subset of $\mathbb{R}^n$, then it admits an eigenfunction decomposition:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \sum_{i=1}^{N} \lambda_i \phi_j(\mathbf{x}) \phi_j(\mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{l_2} \\ \Psi(\mathbf{x}) &= [\dots, \sqrt{\lambda_j} \phi(\mathbf{x}) \dots]^T \end{aligned} \tag{3}$$

- Feature map associated with a kernel is not unique.
- Functional generalization of positive semi-definite matrices.

Kernels

- Linear
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$
- Polynomial
$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$$
- Gaussian: $s = \infty$, Universal
$$k(\mathbf{x}, \mathbf{x}') = e^{\frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$

Kernels

- Linear

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- Polynomial

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$$

- Gaussian: $s = \infty$, Universal

$$k(\mathbf{x}, \mathbf{x}') = e^{\frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$

- Conic combinations:

$$k(\mathbf{x}, \mathbf{x}') = \alpha_1 k_1(\mathbf{x}, \mathbf{x}') + \alpha_2 k_2(\mathbf{x}, \mathbf{x}')$$

Kernels

- Linear
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$
- Polynomial
$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$$
- Gaussian: $s = \infty$, Universal
$$k(\mathbf{x}, \mathbf{x}') = e^{\frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$
- Conic combinations:
$$k(\mathbf{x}, \mathbf{x}') = \alpha_1 k_1(\mathbf{x}, \mathbf{x}') + \alpha_2 k_2(\mathbf{x}, \mathbf{x}')$$
- Elementwise products:
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$$
- Kernels on discrete sets: strings, graphs, sequences, shapes

Gaussian Kernel is Positive Definite

- Exponential

$$e^{\beta \mathbf{x}^T \mathbf{x}'} = 1 + \beta \mathbf{x}^T \mathbf{x}' + \frac{\beta^2}{2!}(\mathbf{x}^T \mathbf{y})^2 + \frac{\beta^3}{3!}(\mathbf{x}^T \mathbf{y})^3 + \dots$$

Gaussian Kernel is Positive Definite

- Exponential

$$e^{\beta \mathbf{x}^T \mathbf{x}'} = 1 + \beta \mathbf{x}^T \mathbf{x}' + \frac{\beta^2}{2!}(\mathbf{x}^T \mathbf{y})^2 + \frac{\beta^3}{3!}(\mathbf{x}^T \mathbf{y})^3 + \dots$$

- For any function $f : \mathbb{R}^n \mapsto \mathbb{R}$, p.s.d function k, the following kernel is p.s.d.

$$k'(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\,(k(\mathbf{x}, \mathbf{x}'))\,f(\mathbf{x}')$$

Proof:

$$\sum_{ij} \beta_i \beta_j k'(\mathbf{x}_i, \mathbf{x}_j) = \sum_{ij} \beta_i \beta_j f(\mathbf{x}_i) k(\mathbf{x}_i, \mathbf{x}_j) f(\mathbf{x}_j) = \sum_{ij} \beta'_i \beta'_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Gaussian Kernel is Positive Definite

- Exponential

$$e^{\beta \mathbf{x}^T \mathbf{x}'} = 1 + \beta \mathbf{x}^T \mathbf{x}' + \frac{\beta^2}{2!} (\mathbf{x}^T \mathbf{y})^2 + \frac{\beta^3}{3!} (\mathbf{x}^T \mathbf{y})^3 + \ldots$$

- For any function $f : \mathbb{R}^n \mapsto \mathbb{R}$, p.s.d function k, the following kernel is p.s.d.

$$k'(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) \left(k(\mathbf{x}, \mathbf{x}') \right) f(\mathbf{x}')$$

  Proof:

$$\sum_{ij} \beta_i \beta_j k'(\mathbf{x}_i, \mathbf{x}_j) = \sum_{ij} \beta_i \beta_j f(\mathbf{x}_i) k(\mathbf{x}_i, \mathbf{x}_j) f(\mathbf{x}_j) = \sum_{ij} \beta'_i \beta'_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

- Gaussian Kernel:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= e^{-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}} \\ &= e^{-\frac{\|\mathbf{x}\|_2^2}{2\sigma^2}} \left(e^{\frac{2\mathbf{x}^T \mathbf{x}'}{2\sigma^2}} \right) e^{-\frac{\|\mathbf{x}'\|_2^2}{2\sigma^2}} \end{aligned} \qquad (4)$$

The Mathematical Naturalness of Kernel Methods

- Data $\mathcal{X} \in \mathbb{R}^d$, Models $\in \mathcal{H} : \mathcal{X} \mapsto \mathbb{R}$

The Mathematical Naturalness of Kernel Methods

- Data $\mathcal{X} \in \mathbb{R}^d$, Models $\in \mathcal{H} : \mathcal{X} \mapsto \mathbb{R}$
- Geometry in $\mathcal{H}$: inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, norm $\| \cdot \|_{\mathcal{H}}$ (Hilbert Spaces)

The Mathematical Naturalness of Kernel Methods

- Data $\mathcal{X} \in \mathbb{R}^d$, Models $\in \mathcal{H} : \mathcal{X} \mapsto \mathbb{R}$
- Geometry in $\mathcal{H}$: inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, norm $\| \cdot \|_{\mathcal{H}}$ (Hilbert Spaces)
- **Theorem** *All* nice Hilbert spaces are generated by a symmetric positive definite function (the kernel) $k(\mathbf{x}, \mathbf{x}')$ on $\mathcal{X} \times \mathcal{X}$
- if $f, g \in \mathcal{H}$ close i.e. $\|f - g\|_{\mathcal{H}}$ small, then $f(\mathbf{x}), g(\mathbf{x})$ close $\forall \mathbf{x} \in \mathcal{X}$.

The Mathematical Naturalness of Kernel Methods

- Data $\mathcal{X} \in \mathbb{R}^d$, Models $\in \mathcal{H} : \mathcal{X} \mapsto \mathbb{R}$
- Geometry in $\mathcal{H}$: inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, norm $\| \cdot \|_{\mathcal{H}}$ (Hilbert Spaces)
- **Theorem** *All* nice Hilbert spaces are generated by a symmetric positive definite function (the kernel) $k(\mathbf{x}, \mathbf{x}')$ on $\mathcal{X} \times \mathcal{X}$
- if $f, g \in \mathcal{H}$ close i.e. $\|f - g\|_{\mathcal{H}}$ small, then $f(\mathbf{x}), g(\mathbf{x})$ close $\forall \mathbf{x} \in \mathcal{X}$.
- **Reproducing Kernel Hilbert Spaces** (RKHSs): Hilbert space $\mathcal{H}$ that admit a *reproducing kernel*, i.e., a function $k(\mathbf{x}, \mathbf{x}')$ satisfying:

$$
\begin{aligned}
(i) &\quad \forall \mathbf{x} \in \mathcal{X} : k(\cdot, \mathbf{x}) \in \mathcal{H} \\
(ii) &\quad \langle f, k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = f(\mathbf{x})
\end{aligned}
$$

The Mathematical Naturalness of Kernel Methods

- Data $\mathcal{X} \in \mathbb{R}^d$, Models $\in \mathcal{H} : \mathcal{X} \mapsto \mathbb{R}$
- Geometry in $\mathcal{H}$: inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, norm $\| \cdot \|_{\mathcal{H}}$ (Hilbert Spaces)
- **Theorem** *All* nice Hilbert spaces are generated by a symmetric positive definite function (the kernel) $k(\mathbf{x}, \mathbf{x}')$ on $\mathcal{X} \times \mathcal{X}$
- if $f, g \in \mathcal{H}$ close i.e. $\|f - g\|_{\mathcal{H}}$ small, then $f(\mathbf{x}), g(\mathbf{x})$ close $\forall \mathbf{x} \in \mathcal{X}$.
- **Reproducing Kernel Hilbert Spaces** (RKHSs): Hilbert space $\mathcal{H}$ that admit a *reproducing kernel*, i.e., a function $k(\mathbf{x}, \mathbf{x}')$ satisfying:

$$
\begin{aligned}
&(i) \quad \forall \mathbf{x} \in \mathcal{X} : k(\cdot, \mathbf{x}) \in \mathcal{H} \\
&(ii) \quad \langle f, k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = f(\mathbf{x})
\end{aligned}
$$

- Reproducing kernels are symmetric, p.s.d:

$$
\begin{aligned}
\sum_{i,j} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) &= \langle \sum_i \beta_i k(\cdot, \mathbf{x}_i), \sum_i \beta_i k(\cdot, \mathbf{x}_i) \rangle_{\mathcal{H}} \\
&= \| \sum_i \beta_i k(\cdot, \mathbf{x}_i) \|_{\mathcal{H}}^2 \geq 0 \qquad (5)
\end{aligned}
$$

- **(Moore-Aronsjan, 1950)** Symmetric, p.s.d functions are reproducing kernels for some (unique) RKHS $\mathcal{H}_k$.

Kernel Methods: Summary

- Symm. pos. def. function $k(\mathbf{x}, \mathbf{z})$ on input domain $\mathcal{X} \subset \mathbb{R}^d$
- $k \Leftrightarrow$ rich Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ of real-valued functions, with inner product $\langle \cdot, \cdot \rangle_k$ and norm $\| \cdot \|_k$
- Regularized Risk Minimization $\Leftrightarrow$ Linear models in an *implicit* high-dimensional (often infinite-dimensional) feature space.

$$f^\star = \arg\min_{f \in \mathcal{H}_k} \frac{1}{n} \sum_{i=1}^{n} V(y_i, f(\mathbf{x}_i)) + \lambda \|f\|^2_{\mathcal{H}_k}, \ \mathbf{x}_i \in \mathbb{R}^d$$

- **Representer Theorem**: $f^\star(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$

Shallow and Deep Function Spaces

- What does it mean to understand a function space?
  - Linear $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$

Shallow and Deep Function Spaces

- What does it mean to understand a function space?
  - Linear $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}, \mathbf{w} \in \mathbb{R}^d$
  - Polynomial $f(\mathbf{x}) = \mathbf{w}^T \Psi(\mathbf{x}), \mathbf{w} \in \mathbb{R}^s$

Shallow and Deep Function Spaces

- What does it mean to understand a function space?
  - Linear $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$
  - Polynomial $f(\mathbf{x}) = \mathbf{w}^T\Psi(\mathbf{x}), \mathbf{w} \in \mathbb{R}^s$
  - RKHS balls $f \in \mathcal{H}_k : \|f\|_k < \gamma$ [like $\|\mathbf{v}\|_{\mathbf{M}} \leq 1$]

Shallow and Deep Function Spaces

- What does it mean to understand a function space?
  - Linear $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$
  - Polynomial $f(\mathbf{x}) = \mathbf{w}^T\Psi(\mathbf{x}), \mathbf{w} \in \mathbb{R}^s$
  - RKHS balls $f \in \mathcal{H}_k : \|f\|_k < \gamma$ [like $\|\mathbf{v}\|_{\mathbf{M}} \leq 1$]
- Shallow nets: $f(\mathbf{x}) = \mathbf{w}^T\sigma(\mathbf{W}\mathbf{x})$ ($\mathbf{w} \in \mathbb{R}^d, \mathbf{W} \in \mathbb{R}^s$)

Shallow and Deep Function Spaces

- What does it mean to understand a function space?
  - Linear $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$
  - Polynomial $f(\mathbf{x}) = \mathbf{w}^T\Psi(\mathbf{x}), \mathbf{w} \in \mathbb{R}^s$
  - RKHS balls $f \in \mathcal{H}_k : \|f\|_k < \gamma$ [like $\|\mathbf{v}\|_{\mathbf{M}} \leq 1$]
- Shallow nets: $f(\mathbf{x}) = \mathbf{w}^T\sigma(\mathbf{W}\mathbf{x})$ $(\mathbf{w} \in \mathbb{R}^d, \mathbf{W} \in \mathbb{R}^s)$
  - For $s \to \infty$, $\mathbf{W} \sim p, \sigma(u) = e^{-iu}$, approximates Gaussian kernel methods.
  - Theorem [Bochner, 1937]: One-to-one correspondence between k and a density p such that,

$$k(\mathbf{x}, \mathbf{z}) = \psi(\mathbf{x} - \mathbf{z}) = \int_{\mathbb{R}^d} e^{-i(\mathbf{x}-\mathbf{z})^T\mathbf{w}} p(\mathbf{w}) d\mathbf{w} \approx \frac{1}{s}\sum_{j=1}^{s} e^{-i(\mathbf{x}-\mathbf{z})^T\mathbf{w}_j}$$

    Gaussian kernel: $k(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|_2^2}{2\sigma^2}} \iff p = \mathcal{N}(0, \sigma^{-2}\mathbf{I}_d)$
  - Similar integral approx relates ReLU to the arc-cosine kernel.
  - Optimization versus Randomization

Expressivity of Deep Nets: Linear Regions

- The ReLU map $f(\mathbf{x}) = \max(\mathbf{W}\mathbf{x}, 0), \mathbf{W} \in \mathbb{R}^{w \times n}$ is piecewise linear.

Expressivity of Deep Nets: Linear Regions

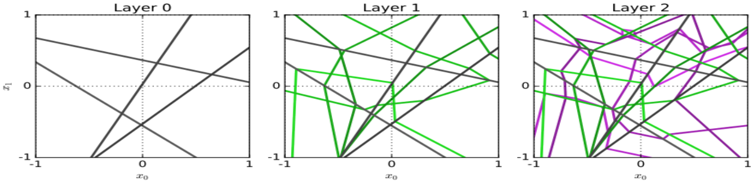
- The ReLU map $f(\mathbf{x}) = \max(\mathbf{W}\mathbf{x}, 0), \mathbf{W} \in \mathbb{R}^{w \times n}$ is piecewise linear.
- A piece is a connected region (a convex polytope) of $\mathbb{R}^d$ with shared sparsity pattern in activations.

Expressivity of Deep Nets: Linear Regions

- The ReLU map $f(\mathbf{x}) = \max(\mathbf{W}\mathbf{x}, 0), \mathbf{W} \in \mathbb{R}^{w \times n}$ is piecewise linear.
- A piece is a connected region (a convex polytope) of $\mathbb{R}^d$ with shared sparsity pattern in activations.
- (Zaslavsky, 1975): Number of linear regions formed by an arrangement of s hyperplanes in d-dimensions in general position is $\sum_{i=0}^{d} \begin{pmatrix} s \\ i \end{pmatrix}$

Expressivity of Deep Nets: Linear Regions

- The ReLU map $f(\mathbf{x}) = \max(\mathbf{W}\mathbf{x}, 0), \mathbf{W} \in \mathbb{R}^{w \times n}$ is piecewise linear.
- A piece is a connected region (a convex polytope) of $\mathbb{R}^d$ with shared sparsity pattern in activations.
- (Zaslavsky, 1975): Number of linear regions formed by an arrangement of s hyperplanes in d-dimensions in general position is $\sum_{i=0}^{d} \binom{s}{i}$
- Composition of ReLU maps is also piecewise linear.



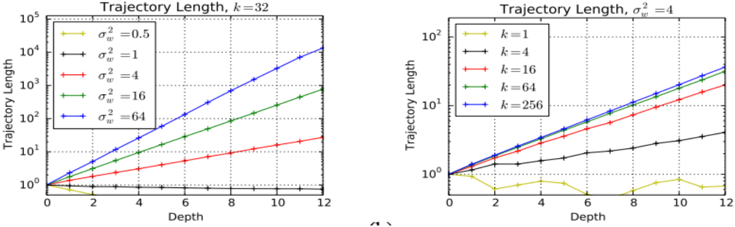
- (Raghu et al, 2017; Pascano et. al. 2014; Montufar, 2014): Number of linear regions grows as $O(w^{nd})$, for a composition of d ReLU maps ($n \to b \to b \to b \ldots \to b$), each of width w.
- (Safran and Shamir, 2017)
  - $1[\|\mathbf{x}\| \leq 1], \mathbf{x} \in \mathbb{R}^d$: ϵ-accurate 3-layer network with $O(d/\epsilon)$ neurons, but cannot be approximated with accuracy higher than $O(1/d^4)$ using 2-layer networks unless width is exponential in d.

Expressivity of Deep Nets: Trajectory Length

- (Raghu et. al. 2017): Trajectory Length ("generalized operator norm"): How much is a curve in the input space stretched by a layer in the network?

Expressivity of Deep Nets: Trajectory Length

- (Raghu et. al. 2017): Trajectory Length ("generalized operator norm"): How much is a curve in the input space stretched by a layer in the network?

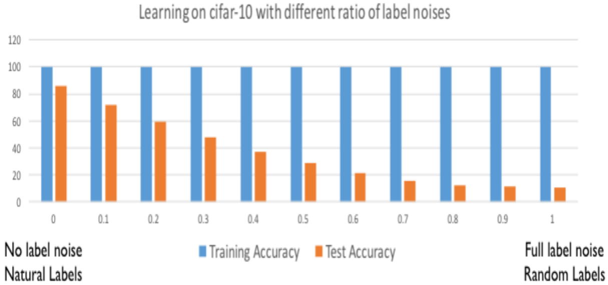


Theorem 1. Bound on Growth of Trajectory Length *Let F_W be a hard tanh random neural network and $x(t)$ a one dimensional trajectory in input space. Define $z^{(d)}(x(t)) = z^{(d)}(t)$ to be the image of the trajectory in layer d of F_W, and let $l(z^{(d)}(t)) = \int_t \left|\left|\frac{dz^{(d)}(t)}{dt}\right|\right| dt$ be the arc length of $z^{(d)}(t)$. Then*

$$\mathbb{E}\left[l(z^{(d)}(t))\right] \geq O\left(\left(\frac{\sigma_w}{(\sigma_w^2+\sigma_b^2)^{1/4}} \cdot \frac{\sqrt{k}}{\sqrt{\sqrt{\sigma_w^2+\sigma_b^2}+k}}\right)^d\right) l(x(t))$$

Expressivity and Generalization

- Expressiveness $\equiv$ capacity to fit random noise $\implies$ susceptibility to overfitting. (VC dimension, Rademacher Complexity)
- Zhang et. al, 2017: Famous CNNs easily fit random labels.
  - Over-parameterized (n=1.2M, p$\sim$ 60M), but structured.
  - Domain-knowledge baked in does not constraint expressiveness.
- CNNs tend to generalize even without regularization (explicit and implicit)



Expressivity and Generalization

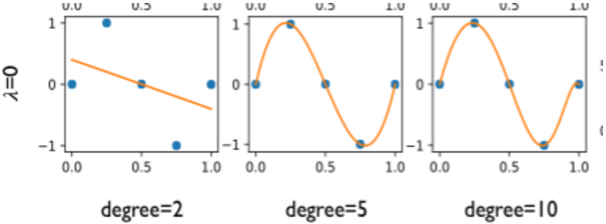
- Expressiveness $\equiv$ capacity to fit random noise $\implies$ susceptibility to overfitting. (VC dimension, Rademacher Complexity)
- Zhang et. al, 2017: Famous CNNs easily fit random labels.
  - Over-parameterized (n=1.2M, p$\sim$ 60M), but structured.
  - Domain-knowledge baked in does not constraint expressiveness.
- CNNs tend to generalize even without regularization (explicit and implicit)

Config	Augmentation	Dropout	Weight decay	Train top-5	Test top-5
Inception on ImageNet	Yes	Yes	Yes	99.21%	93.92%
	Yes	No	No	99.17%	90.43%
	No	No	Yes	100%	86.44%
	No	No	No	100%	80.38% (84.49%)

- For linear regression, SGD returns minimum norm solution.

Expressivity and Generalization

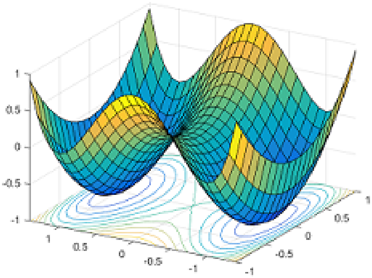
- Expressiveness $\equiv$ capacity to fit random noise $\implies$ susceptibility to overfitting. (VC dimension, Rademacher Complexity)
- Zhang et. al, 2017: Famous CNNs easily fit random labels.
  - Over-parameterized (n=1.2M, p~ 60M), but structured.
  - Domain-knowledge baked in does not constraint expressiveness.
- CNNs tend to generalize even without regularization (explicit and implicit)



- For linear regression, SGD returns minimum norm solution.

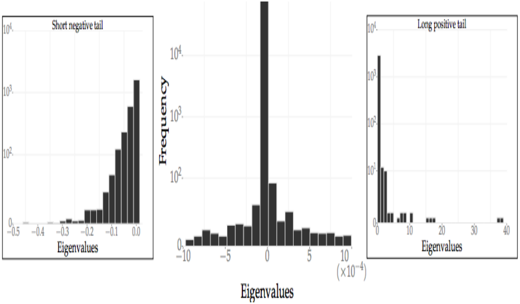
Optimization Landscape: Symmetry and Saddle Points

- Folklore: lots of equivalent good local min.
- Classification of critical points based on Hessian spectrum.
- Permutation and Scaling symmetries.



Optimization Landscape: Symmetry and Saddle Points

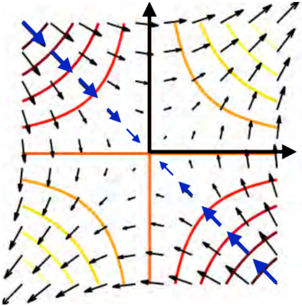
- Folklore: lots of equivalent good local min.
- Classification of critical points based on Hessian spectrum.
- Permutation and Scaling symmetries.



- Thm [Lee et al, 2016; Anadkumar, Ge]: For the short-step gradient method, the basin of attraction of strict saddle points (atleast one negative eigenvalue) has measure zero.

Optimization Landscape: Symmetry and Saddle Points

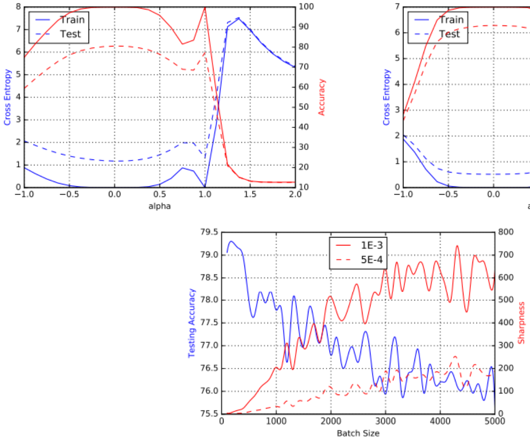
- Folklore: lots of equivalent good local min.
- Classification of critical points based on Hessian spectrum.
- Permutation and Scaling symmetries.



- Thm [Lee et al, 2016; Anadkumar, Ge]: For the short-step gradient method, the basin of attraction of strict saddle points (atleast one negative eigenvalue) has measure zero.

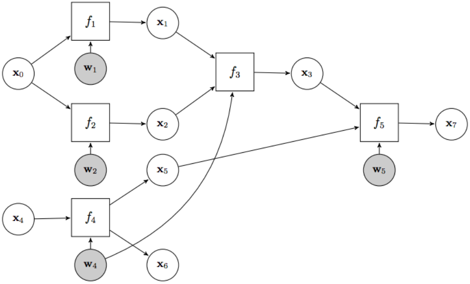
Optimization Landscape: Sharp Minima and Stability

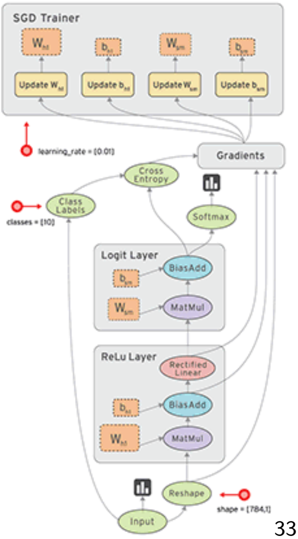
Keskar et. al., 2017, On large-batch training for DL: Generalization Gap and Sharp Minima



Deep Learning, Computational Graphs, TensorFlow

tensorflow.org/





- Tensors-in, Tensors-out
- Evergrowing list of Tensor-Ops (Python/C++)

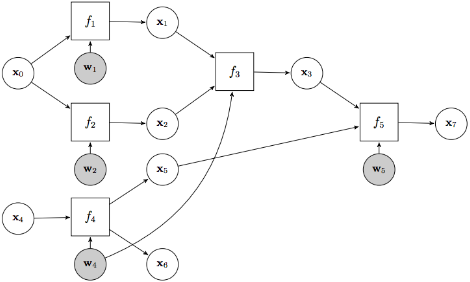
$$f(\mathbf{x}, \mathbf{w}), \left[\frac{\partial f}{\partial \mathbf{x}}\right]^T \star \mathbf{v}, \left[\frac{\partial f}{\partial \mathbf{w}}\right]^T \star \mathbf{v}$$

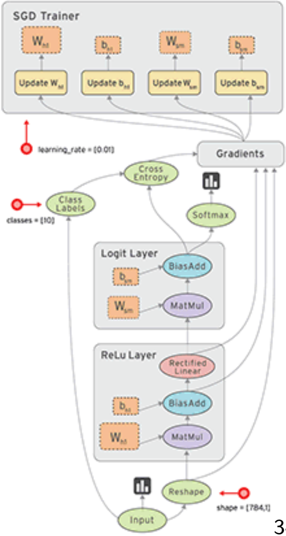
- Automatic Differentiation: Reverse mode

$$\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_i} = \sum_{i \rightarrow j} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_j} \frac{\partial \mathbf{x}_j}{\partial \mathbf{x}_i}$$

Computational Graphs and TensorFlow

tensorflow.org/





- SGD with minibatches

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \sum_{i=1}^{l} \frac{\partial l_i}{\partial \mathbf{w}} \Big|_{\mathbf{w_t}}$$

- Model Parallelism: Assign independent paths to threads, GPUs.
- Data Parallelism: create Graph replicas on different machines.
- Mobile TensorFlow

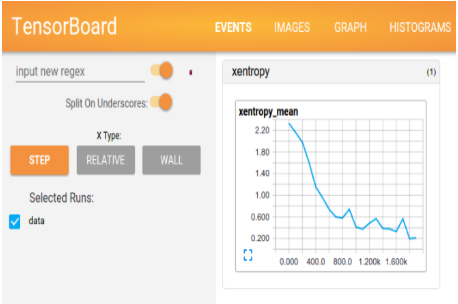
Computational Graphs and TensorFlow

tensorflow.org/

```
def cnn_model_fn(features, labels, mode):
  """Model function for CNN."""
  # Input Layer
  input_layer = tf.reshape(features, [-1, 28, 28, 1])

  # Convolutional Layer #1
  conv1 = tf.layers.conv2d(
      inputs=input_layer,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)

  # Pooling Layer #1
  pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)

  # Convolutional Layer #2 and Pooling Layer #2
  conv2 = tf.layers.conv2d(
      inputs=pool1,
      filters=64,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)
  pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

  # Dense Layer
  pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])
  dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.relu
  dropout = tf.layers.dropout(
      inputs=dense, rate=0.4, training=mode == learn.ModeKeys.TRAIN)

  # Logits Layer
  logits = tf.layers.dense(inputs=dropout, units=10)

  loss = None
  train_op = None

  # Calculate Loss (for both TRAIN and EVAL modes)
  if mode != learn.ModeKeys.INFER:
    onehot_labels = tf.one_hot(indices=tf.cast(labels, tf.int32), depth=10)
    loss = tf.losses.softmax_cross_entropy(
        onehot_labels=onehot_labels, logits=logits)

  # Configure the Training Op (for TRAIN mode)
  if mode == learn.ModeKeys.TRAIN:
    train_op = tf.contrib.layers.optimize_loss(
        loss=loss,
        global_step=tf.contrib.framework.get_global_step(),
        learning_rate=0.001,
        optimizer="SGD")

  # Generate Predictions
  predictions = {
      "classes": tf.argmax(
          input=logits, axis=1),
      "probabilities": tf.nn.softmax(
          logits, name="softmax_tensor")
  }

  # Return a ModelFnOps object
  return model_fn_lib.ModelFnOps(
```

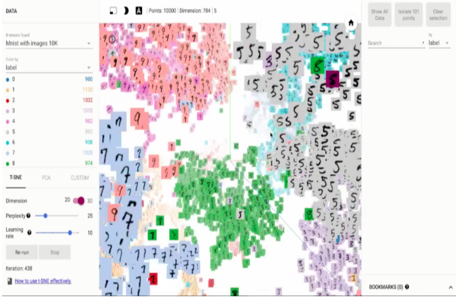
Computational Graphs and TensorFlow

tensorflow.org/



Computational Graphs and TensorFlow

tensorflow.org/



Challenges and Opportunities

- Lots of fascinating unsolved problems.
- Can you win Imagenet with Convex optimization?
- Tools from polynomial optimization (global optimization, deep-RL)
- Supervised to Semi-supervised to Unsupervised Learning.
- Compact, efficient, real-time models (eliminate over-parametrization) for new form factors.
- Exciting, emerging world of Robotics, Wearable Computers, Intelligent home devices, Personal digital assistants!

Google Internship Program.