Kernel Methods for Deep Learning

Speaker: Siyu Gao

New York University Shanghai

MATH-SHU 236 Math Foundation for DS Presentation 1

Outline

- Background
  - Motivation
  - Knowledge Recap
- Arc-cosine Kernels
  - Definition and Properties
  - Single-layer Threshold Networks Approximation
  - Multilayer Threshold Networks Approximation
- Combined with Shallow Machine: SVM
- Deep Architecture: Multilayer Kernel Machines (MKMs)
- Summary
- Q&A

Background-Motivation

1. Models with deep architectures (e.g. Neural Networks) have been proved to be a powerful in many application, but they're generally difficult to train.
2. Shallow architectures with kernel trick proposed to solve non-linear separable problems.
3. Intrigued by success of deep architectures,applying deep learning in kernel machines:
    - Arc-cosine kernel to mimic computation
    - multilayer kernel machine

Background - Knowledge Recap - Kernel

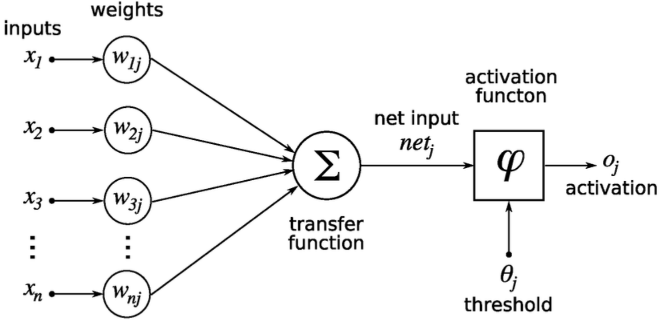
A **Kernel function** K is to define a "comparison function":
$K : X \times X \longmapsto \mathbb{R}$ such that

$$K \langle \mathbf{x}, \mathbf{y} \rangle = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$$

where Φ is a feature map and an explicit representation for Φ is not necessary.
Examples:

- linear Kernel: $K \langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle$
- polynomial Kernel: $K \langle \mathbf{x}, \mathbf{y} \rangle = (\mathbf{x}^T \mathbf{y} + c)^d$
- RBF (Gaussian) Kernel: $K \langle \mathbf{x}, \mathbf{y} \rangle = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$

Each **Neural Network** is composed by **Neurons**.



Taking the activation function σ, weights w_i and bias b

Neuron : output $= \sigma(\sum_i w_i x_i + b)$

Changed to weight matrix W and bias vector $\mathbf{b}$:

Single layer Neural Network: output $= \sigma(\mathbf{Wx} + \mathbf{b})$

Arc-cosine Kernel

Definition: For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the **arc-cosine** function via the integral representation is :

$$k_n(\mathbf{x}, \mathbf{y}) \;=\; 2 \int d\mathbf{w} \, \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}} \, \Theta(\mathbf{w} \cdot \mathbf{x}) \, \Theta(\mathbf{w} \cdot \mathbf{y}) \, (\mathbf{w} \cdot \mathbf{x})^n \, (\mathbf{w} \cdot \mathbf{y})^n$$

To better understand the influence of inputs, it can be rewrite as

$$k_n(\mathbf{x}, \mathbf{y}) \;=\; \frac{1}{\pi} \|\mathbf{x}\|^n \|\mathbf{y}\|^n J_n(\theta)$$

where

$$J_n(\theta) \;=\; (-1)^n (\sin\theta)^{2n+1} \left(\frac{1}{\sin\theta} \frac{\partial}{\partial\theta} \right)^n \left(\frac{\pi - \theta}{\sin\theta} \right).$$

Arc-cosine Kernel Properties

Observing that:

$$\begin{aligned} J_0(\theta) &= \pi - \theta \\ J_1(\theta) &= \sin\theta + (\pi - \theta)\cos\theta \\ J_2(\theta) &= 3\sin\theta\cos\theta + (\pi - \theta)(1 + 2\cos^2\theta) \end{aligned}$$

1. when $n = 0, k_0(\mathbf{x}, \mathbf{x}) = 1$
2. when $n = 1, k_1(\mathbf{x}, \mathbf{x}) = \|\mathbf{x}\|^2$
3. when $n > 1,\ k_n(\mathbf{x}, \mathbf{x}) \sim \|\mathbf{x}\|^{2}n$

Compared with other kernels:

- RBF, linear and polynomial kernels all share these three properties
- Without continuous tuning parameters.

Computation in Single Layer Threshold Networks

Setup for the networks:

- Weight matrix $\mathbf{W}$ is Gaussian distributed with $\mathrm{N}(0, I_d)$: W_{ij} connects jth output to ith input
- Activation function family: $g_n(z) = \Theta(z)z^n$
- The output is a vector: $\mathbf{f}(\mathbf{x}) = g(\mathbf{Wx})$

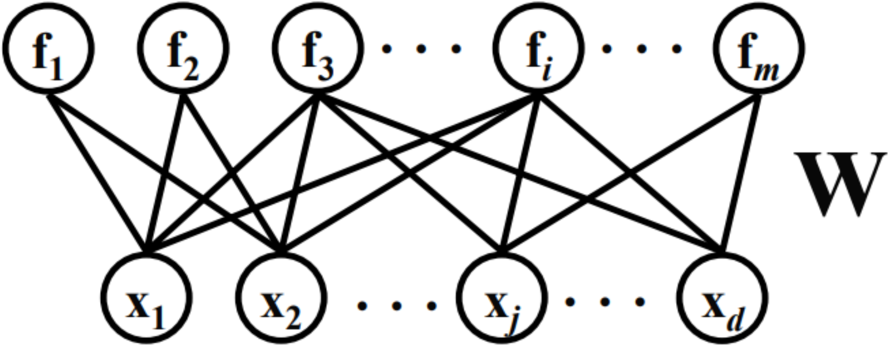
Computing the inner product of two different outputs we have:

$$\mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) = \sum_{i=1}^{m} \Theta(\mathbf{w}_i \cdot \mathbf{x})\Theta(\mathbf{w}_i \cdot \mathbf{y})(\mathbf{w}_i \cdot \mathbf{x})^n(\mathbf{w}_i \cdot \mathbf{y})^n$$

When the network has infinite number of outputs:

$$\lim_{m \to \infty} \frac{2}{m} \mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) = k_n(\mathbf{x}, \mathbf{y})$$

Single Layer Threshold Networks



Computation in Single Layer Threshold Networks

Trick is on weight matrix W:

$$
\begin{aligned}
&\lim_{m\to\infty}\left[\frac{2}{m}\,\mathbf{f}(\mathbf{x})\cdot\mathbf{f}(\mathbf{y})\right] \\
&= \lim_{m\to\infty}\left[\frac{2}{m}\sum_{i=1}^{m}\Theta(\mathbf{w}_i\cdot\mathbf{x})\Theta(\mathbf{w}_i\cdot\mathbf{y})(\mathbf{w}_i\cdot\mathbf{x})^n(\mathbf{w}_i\cdot\mathbf{y})^n\right] \\
&= 2\,\mathbb{E}_{\mathbf{w}\sim\mathcal{N}(0,I_d)}\left[\,\Theta(\mathbf{w}\cdot\mathbf{x})\;\Theta(\mathbf{w}\cdot\mathbf{y})\;(\mathbf{w}\cdot\mathbf{x})^n\;(\mathbf{w}\cdot\mathbf{y})^n\,\right] \\
&= 2\int d\mathbf{w}\;\frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{(2\pi)^{d/2}}\;\Theta(\mathbf{w}\cdot\mathbf{x})\;\Theta(\mathbf{w}\cdot\mathbf{y})\;(\mathbf{w}\cdot\mathbf{x})^n\;(\mathbf{w}\cdot\mathbf{y})^n \\
&= k_n(\mathbf{x},\mathbf{y}).
\end{aligned}
$$

Some views:

Having f as known feature function to find new kernels.

Hard to find one for all inputs.

Choose the weight matrix as random variable.

At least, we preserve the advantage of neural networks layer at infinity.

Computation in Multilayer Threshold Networks

Composition to mimic multilayer:

$$k^{(\ell)}(\mathbf{x},\mathbf{y}) = \underbrace{\mathbf{\Phi}(\mathbf{\Phi}(\ldots\mathbf{\Phi}}_{\ell\text{ times}}(\mathbf{x}))) \cdot \underbrace{\mathbf{\Phi}(\mathbf{\Phi}(\ldots\mathbf{\Phi}}_{\ell\text{ times}}(\mathbf{y})))$$

Arc-cosine kernel can be expressed in iteration:

$$k_n^{(l+1)}(\mathbf{x},\mathbf{y}) = \frac{1}{\pi}\left[\, k_n^{(l)}(\mathbf{x},\mathbf{x})\, k_n^{(l)}(\mathbf{y},\mathbf{y}) \,\right]^{n/2} J_n\left(\theta_n^{(\ell)}\right)$$

$$\theta_n^{(\ell)} = \cos^{-1}\left(k_n^{(\ell)}(\mathbf{x},\mathbf{y}) \left[k_n^{(\ell)}(\mathbf{x},\mathbf{x})\, k_n^{(\ell)}(\mathbf{y},\mathbf{y}) \right]^{-1/2} \right)$$

The resulting kernels mimic the computation in large multilayer threshold networks.

Multilayer Threshold Networks

$$\boldsymbol{\Phi}_{n_3}(\boldsymbol{\Phi}_{n_2}(\boldsymbol{\Phi}_{n_1}(\mathbf{x})))$$

$$\uparrow$$

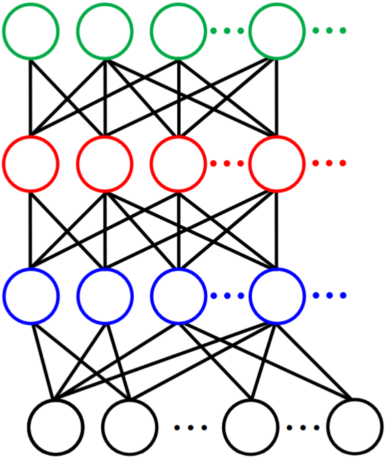
$$\boldsymbol{\Phi}_{n_2}(\boldsymbol{\Phi}_{n_1}(\mathbf{x}))$$

$$\uparrow$$

$$\boldsymbol{\Phi}_{n_1}(\mathbf{x})$$

$$\uparrow$$

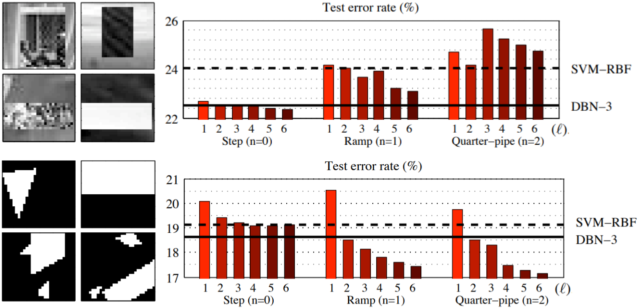
$$\mathbf{x}$$



Combined with Shallow Machine: SVM

Experiment setup:

- Datasets: *rectangle-image*,*convex*
- Benchmark: SVM-RBF, Deep Belief Net (DBN-3)
- Variate parameter: kernel degree (n=1), layer,activation degree



SVM-arc cosine: Result Analysis

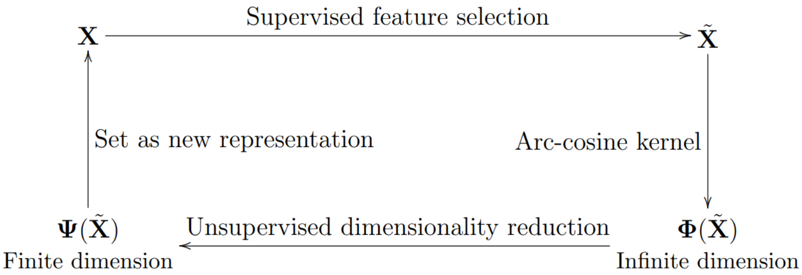
Observation:

- SVM only performed well when kernel degree n=1 for multiple layers. $\rightarrow$ k_1 better interpret magnitude.
- Multilayers tends to outperform single-layer. $\rightarrow$ SVM-arc cosine benefit from the deep architecture.

Deep Architecture: Multilayer Kernel Machines (MKMs)

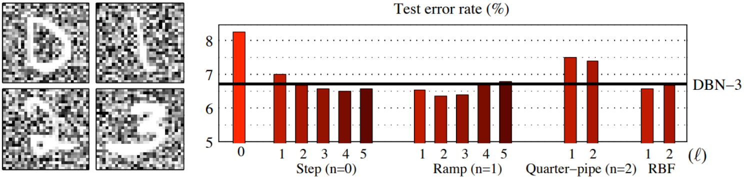
Algorithm:

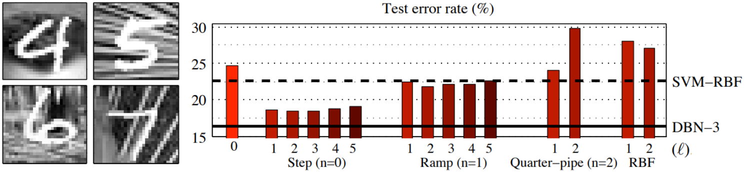
- Prune uninformative features from input space
- Applying kernel PCA
- repeat the first two steps l times
- The last layer: Distance Metric Learning for classification.



Experiment setup:

- Datasets: *MNIST-rand,MNIST-image*
- Benchmark: SVM-RBF, Deep Belief Net (DBN-3)
- Variate parameter: kernel degree , layer,activation degree





MKMs: Result Analysis

Observation:

- MKM perform significantly better than shallow architectures and have competitive result compared with DBN-3.
- kernel degree 0 and 1 is better than 2. $\rightarrow$ kernel PCA distorted the dynamic range of inputs.
- Less time consuming than RBF. $\rightarrow$ Without tuning parameters.

Summary

Significance:

- Develop a new family of kernel functions and mimic the computation of large multilayer neural nets.
- Validate the basic intuition behind deep learning.

Further Discussion:

- Experiments on larger kernel degree n?
- Other random matrix W.
- Other operation to approximate multilayer NNs.
- Improve the scheme of MKM.

Q & A

Welcome any related questions and we can learn together!

Thank you for listening!

References

- Lecture 1 on kernel methods: Positive definite kernels URL: $https://www.youtube.com/watch?v=IzGS8uKc5E4list=P$ LD93kGj6$_E drkNj27AZMecbRlQ1SMkp_o$
- Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, Advances in Neural Information Processing Systems, volume 22.Curran Associates, Inc., 2009.
- Youngmin Cho and Lawrence K. Saul Department of Computer Science and Engineering.Kernel Methods for Deep Learning.
  Advances in Neural Information Processing Systems 22 (NIPS 2009)