Deep Neural Networks

Numerical Methods for Deep Learning

Why Deep Networks?

- Universal approximation theorem of NN suggests that we can approximate **any** function by two layers.
- But - The width of the layer can be very large $\mathcal{O}(n \cdot n_f)$
- Deeper architectures can lead to more efficient descriptions of the problem.
  (No real proof but lots of practical experience)

Learning Objective: Deep Neural Networks

In this module we introduce multilayer deep neural networks.

Learning tasks:

- regression
- classification

Numerical methods:

- non-convex optimization
- probability theory (for initialization)

Deep Neural Networks

Until recently, the standard architecture was

$$
\begin{aligned}
\mathbf{Y}_1 &= \sigma(\mathbf{K}_0 \mathbf{Y}_0 + \mathbf{b}_0) \\
\vdots &= \vdots \\
\mathbf{Y}_N &= \sigma(\mathbf{K}_{N-1} \mathbf{Y}_{N-1} + \mathbf{b}_{N-1})
\end{aligned}
$$

Deep Neural Networks

Until recently, the standard architecture was

$$\begin{aligned}\mathbf{Y}_1 &= \sigma(\mathbf{K}_0\mathbf{Y}_0 + \mathbf{b}_0)\\ \vdots &= \vdots\\ \mathbf{Y}_N &= \sigma(\mathbf{K}_{N-1}\mathbf{Y}_{N-1} + \mathbf{b}_{N-1})\end{aligned}$$

And use $\mathbf{Y}_N$ to classify. This leads to the optimization problem

$$\min_{\mathbf{K}_{0,\ldots,N-1},\mathbf{b}_{0,\ldots,N-1},\mathbf{W}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_1,\ldots,\mathbf{K}_{N-1},\mathbf{b}_1,\ldots,\mathbf{b}_{N-1}),\mathbf{C}^{\mathrm{obs}}\right)$$

Deep Neural Networks

Until recently, the standard architecture was

$$\begin{aligned} \mathbf{Y}_1 &= \sigma(\mathbf{K}_0\mathbf{Y}_0 + \mathbf{b}_0) \\ \vdots &= \vdots \\ \mathbf{Y}_N &= \sigma(\mathbf{K}_{N-1}\mathbf{Y}_{N-1} + \mathbf{b}_{N-1}) \end{aligned}$$

And use $\mathbf{Y}_N$ to classify. This leads to the optimization problem

$$\min_{\mathbf{K}_{0,\ldots,N-1},\mathbf{b}_{0,\ldots,N-1},\mathbf{W}} \; E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_1,\ldots,\mathbf{K}_{N-1},\mathbf{b}_1,\ldots,\mathbf{b}_{N-1}),\mathbf{C}^{\mathrm{obs}}\right)$$

How deep is deep? For now, let's say $N > 1$.

Example: Hand-written digit recognition [4]

layer 1:
- input features: images of size 28×28
- $\mathbf{K}_1$: four 5×5 convolution stencils.
- $\mathbf{b}_1 \in \mathbb{R}^4$ are biases
- $\sigma = \tanh$
- output features: four images of size 24×24

layer 2: average pooling (no trainable weights)

layer 3:
- input features: four images of size 12×12
- $\mathbf{K}_2$: 48 5×5 convolution stencils.
- $\mathbf{b}_2 \in \mathbb{R}^4$ are biases
- $\sigma = \tanh$
- output features: four images of size 12×12

layer 4: average pooling (no trainable weights)

Example: Hand-written digit recognition [4]

layer 1:
- input features: images of size 28×28
- $\mathbf{K}_1$: four 5×5 convolution stencils.
- $\mathbf{b}_1 \in \mathbb{R}^4$ are biases
- $\sigma = \tanh$
- output features: four images of size 24×24

layer 2: average pooling (no trainable weights)

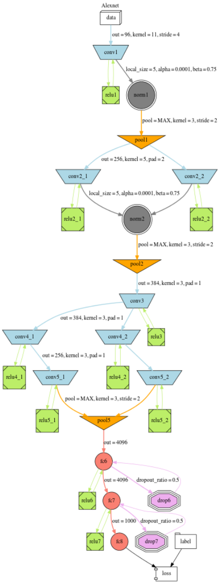
layer 3:
- input features: four images of size 12×12
- $\mathbf{K}_2$: 48 5×5 convolution stencils.
- $\mathbf{b}_2 \in \mathbb{R}^4$ are biases
- $\sigma = \tanh$
- output features: four images of size 12×12

layer 4: average pooling (no trainable weights)

Very effective for MNIST, but today's architectures have become more sophisticated.

Example: The Alexnet [3] for Image Classification

- Complex architectures
- trained on multiple GPUs
- $\approx$ 60 million weights



Key issues: Non-Convexity and Initialization

Optimization problems in learning are generally non-convex.

- training relies on stochastic gradient methods [1]
- initialization is key [2]

Initialization is particularly important. Simple example: Let $\mathbf{y} \in \mathbb{R}^{100} \sim \mathcal{N}(0, \mathbf{I})$ Compare

$$\|\mathbf{K}_3\mathbf{K}_2\mathbf{K}_1\mathbf{y}\|^2 \quad \text{to} \quad \|\mathbf{y}\|$$

for different choices of $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3 \in \mathbb{R}^{100\times 100}$. Sample entries independently, e.g., from standard normal, uniform in $[-0.5, 0.5]$, uniform in $[-0.05, 0.05]$. See more details in [2].

Σ: Deep Neural Networks

Idea: Concatenate many single layers and solve

$$\min_{\mathbf{K}_{0,\ldots,N-1},\mathbf{b}_{0,\ldots,N-1},\mathbf{W}} E\left(\mathbf{W}\mathbf{Y}_N(\mathbf{K}_1,\ldots,\mathbf{K}_{N-1},\mathbf{b}_1,\ldots,\mathbf{b}_{N-1}),\mathbf{C}^{\mathrm{obs}}\right)$$

Discussion:

- empirically shown for some examples to generalize better and be more efficient than wide architectures
- Challenge 1: computational costs (architecture have millions or billions of parameters)
- Challenge 2: design architecture that is easy to train and generalizes well
- Challenge 3: learning leads to very non-convex optimization problems $\rightsquigarrow$ initialization is key. Leads to exploding/vanishing gradient phenomena.

References

[1] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *arXiv preprint [stat.ML] (1606.04838v1)*, 2016.

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *jmlr.org*.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 61:1097–1105, 2012.

[4] Y. LeCun, B. E. Boser, and J. S. Denker. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.