Many Pathways to Deep Learning

• Keras
  Quickest way to implement deep learning

• Andrew Ng (Coursera)
  Foundations for practitioners

• Sebastian Raschka (STAT479, UW-Madison)
  More in-depth training for both research & practice
SUMMARY

• Review of neural network
• Representation learning
• Architectural nature
• Applications
• Demos
Something happened in the mid 1980s that shook up the applied statistics community. Neural networks (NNs) were introduced, and they marked a shift of predictive modeling towards computer science and machine learning. A neural network is a highly parametrized model, inspired by the architecture of the human brain, that was widely promoted as a universal approximator—a machine that with enough data could learn any smooth predictive relationship.

Figure 18.1 shows a simple example of a feed-forward neural network diagram. There are four predictors or inputs $x_j$, five hidden units $a_i$, and a single output unit $o$. The language associated with NNs is colorful: memory units or neurons automatically learn new features from the data through a process called
Chapter 18. Learning from Examples

Output

Input

Links

Activation Function

Input Function

Output

Links

\[ a_j = g(in_j) = g \left( \sum_i w_{i,j} a_i \right) \]

The most common \( g() \) is ReLU — \( g(in_j) = \max(0, in_j) \)

source: Russell & Norvig (2010)
Deeper Network & Multiple Outputs

Figure 18.3: Neural network diagram with three hidden layers and multiple outputs, suitable for the MNIST handwritten-digit problem. The input layer has \( p = 784 \) units. Such a network with hidden layer sizes \( 1024; 1024; 2048 \), and particular choices of tuning parameters, achieves the state-of-the-art error rate of \( 0.93 \% \) on the "official" test data set. This network has close to four million weights, and hence needs to be heavily regularized.

The network in Figure 18.3 is complex, so it is essential to establish a convenient notation for referencing the different sets of parameters. We continue with the notation established for the single-layer network, but with some additional annotations to distinguish aspects of different layers.

From the first to the second layer we have:

\[
\begin{align*}
\mathbf{z}_2 &= \mathbf{w}_1 \mathbf{x}, \\
\mathbf{a}_2 &= \mathbf{g}_2(\mathbf{z}_2).
\end{align*}
\]

(source: Efron & Hastie (2016))
Autoencoder

Figure 18.7 Left: Network representation of an autoencoder used for unsupervised learning of nonlinear principal components. The middle layer of hidden units creates a bottleneck, and learns nonlinear representations of the inputs. The output layer is the transpose of the input layer, so the network tries to reproduce the input data using this restrictive representation. Right: Images representing the estimated rows of \( W \) using the MNIST database; the images can be seen as filters that detect local gradients in the image pixels. In each image, most of the weights are zero, and the nonzero weights are localized in the two-dimensional image space.

1. `1 regularization applied to the rows of \( W \) lead to sparse weight vectors, and hence local features, as was the case in our example.
2. Denoising is a process where noise is added to the input layer (but not the output), resulting in features that do not focus on isolated values, such as pixels, but instead have some volume. We discuss denoising further in Section 18.5.
3. With regularization, the bottleneck is not necessary, as in the figure or in principal components. In fact we can learn many more than \( p \) components.
4. Autoencoders can also have multiple layers, which are typically learned sequentially. The activations learned in the first layer are treated as the input (and output) features, and a model like (18.22) is fit to them.

18.4 Deep Learning

Neural networks were reincarnated around 2010 with “deep learning” as a flashier name, largely a result of much faster and larger computing systems, plus a few new ideas. They have been shown to be particularly successful source: Efron & Hastie (2016)
• A key perspective to understand DL

• Some representations of data are more useful than others

• Learning such representations also from data, rather than handcrafting
Representation Matters

Manifold learning using kernel PCA

source: Sebastian Raschka
**Key to Understand Causality**

- **Cartesian coordinates**
  - $\mathbf{x}$
- **Polar coordinates**
  - $\theta$
  - $r$

Source: Goodfellow, Bengio, & Courville (2016)
Handcraft Is Hard

• Polynomial regression, but in what degree? with which interaction terms?

• Support vector machine, but what kernel to use?

• For machines to recognize faces, which parts are important? Eyes and mouth? Why not nose? How should we express them in numbers?
**Approximation or Representation?**

- All we want is to predict $y$ from $x$
- "Representation" to emphasize the key roles of transformed $x$

![Neural network diagram with three hidden layers and multiple outputs, suitable for the MNIST handwritten-digit problem.](image)

The network in Figure 18.3 is complex, so it is essential to establish a convenient notation for referencing the different sets of parameters. We continue with the notation established for the single-layer network, but with some additional annotations to distinguish aspects of different layers.

From the first to the second layer we have:

$$z^{(2)} = w^{(1)} x_j^{(1)}; \quad (18.1)$$

And:

$$a^{(2)} = g^{(2)}(z^{(2)}); \quad (18.2)$$

Where $w$ are the weights, $x$ are the inputs, $a$ are the activations, and $g$ is the activation function.
Why “DEEP”? 

• “many natural signals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones” —LuCun, Bengio, & Hinton (2015)

• shallow networks can “require many more computational elements, potentially exponentially more” —Bengio (2009)

• “the phrase is just a great brand, it’s just so deep” — Andrew Ng
so many “architectures”

There is a basic recipe but no hard-and-fast rule!
Basic Recipe for Deep Learning Projects

- **High Bias?**
  - No
    - **High Variance?**
      - No: Done!
      - Yes: Overfitting
        - More Data Regularization
  - Yes: Underfitting
    - Bigger Network
    - Train longer

No such thing as the bias-variance tradeoff
Trial and Error

- Which regularization techniques to use?
- How many hidden layers and hidden units in each layer?
- What are good learning rates in gradient descent?
- What are good mini-batch sizes for SGD?
- Does transfer learning work?
- Either end-to-end DL or a series of smaller models?
Scale Drives Deep Learning

source: Tang et al. (2018)
Transfer Learning

- Some intermediate representations are useful for several related but different tasks
  - e.g. economic data shared for predicting apple price and orange price

- A representation learned from one dataset could be used with another dataset
  - e.g. learn from abundant data in one country and transfer it to another country with much less data

Figure 18.3 Neural network diagram with three hidden layers and multiple outputs, suitable for the MNIST handwritten-digit problem. The input layer has $p_D = 784$ units. Such a network with hidden layer sizes $1024; 1024; 2048$, and particular choices of tuning parameters, achieves the state-of-the-art error rate of $0.93\%$ on the "official" test data set. This network has close to four million weights, and hence needs to be heavily regularized.

The network in Figure 18.3 is complex, so it is essential to establish a convenient notation for referencing the different sets of parameters. We continue with the notation established for the single-layer network, but with some additional annotations to distinguish aspects of different layers.

From the first to the second layer we have

$$z_2 = w_1 z_0 \mathbf{X} + a_1,$$

and

$$a_2 = g_2(z_2),$$

where $g_2$ is the activation function for the second layer.
Imagine a situation where you want to classify a totally exotic fruit into edible or noxious. It comes in a smooth, round, bumpy or just so irregular shape of different sizes with various colors.

But, from the past experience on different fruits, you know the right representation — what matters is only “irregularity”.

To learn a good classifier, how many samples at best you need?
ENDE-TO-END DEEP LEARNING

• Direct mapping of $x$ to $y$ with a single neural network
• Pros
  • Simplifying the model
  • Letting the data speak without subjective judgments that limit the model's performance
• Cons
  • Large data requirement
  • Ignoring potentially useful domain knowledge
• DL is so powerful that it is tempting to do direct mapping
End-To-End or Not?

- Some systems are too complex even for DL or we do not have enough data:
  - e.g. self-driving car mapping remote-sensing data \((x)\) to \{accelerate, brake, steer\} decisions \((y)\)
  - e.g. predicting poverty \((y)\) from satellite image \((x)\)

- Depends on whether we have:
  - Sufficient data to learn such complex systems
  - Quality of domain knowledge
**Convolutional Neural Network (CNN)**

- One of the most successful DL models
  - especially for image recognition
- Highlighting representation learning and the architectural nature of DL
- Successively learns representations from raw pixels ($x$) to simple features to more abstract ones. Based on the last representation, then, it classifies the image ($y$).

*Source: Sefik Ilkin Serengil*
Deep Learning Libraries

- Programming platforms to implement DL
- Nobody writes codes from scratch for applying DL these days, just like we don’t use Assembly to implement linear regression.
- Remember the trial-and-error model building. Fast implementation is very important.
- TensorFlow, PyTorch, Keras, etc. Ask Siraj Raval and find your best friends.
**Demo / Practice**

- We use [Google Colab](https://colab.research.google.com) + PyTorch

- Go to
  - [https://yujisaikai.com/teaching/](https://yujisaikai.com/teaching/)
  - Open exercise 1, 2, or 3 link
  - Press 🖌️ Open in playground at the top-left corner
  - (You may need Google account or Wisc account)