Deep learning

1.1. From neural networks to deep learning

François Fleuret

https://fleuret.org/dlc/
Many applications require the automatic extraction of “refined” information from raw signal (e.g. image recognition, automatic speech processing, natural language processing, robotic control, geometry reconstruction).
Our brain is so good at interpreting visual information that the “semantic gap” is hard to assess intuitively.
Our brain is so good at interpreting visual information that the “semantic gap” is hard to assess intuitively.

This: is a horse
```python
>>> from torchvision.datasets import CIFAR10
>>> cifar = CIFAR10('./data/cifar10/', train=True, download=True)
Files already downloaded and verified
>>> x = torch.from_numpy(cifar.data)[43].permute(2, 0, 1)
>>> x[:, :4, :8]
tensor([[ 99,  98, 100, 103, 105, 107, 108, 110],
        [100, 100, 102, 105, 107, 109, 110, 112],
        [104, 104, 106, 109, 111, 112, 114, 116],
        [109, 109, 111, 113, 116, 117, 118, 120]],
       [[166, 165, 167, 169, 171, 172, 173, 175],
        [166, 164, 167, 169, 171, 172, 174],
        [169, 167, 170, 171, 171, 173, 174, 176],
        [170, 169, 172, 173, 175, 176, 177, 178]],
       [[198, 196, 199, 200, 200, 202, 203, 204],
        [195, 194, 197, 197, 197, 199, 200, 201],
        [197, 195, 198, 198, 198, 199, 201, 202],
        [197, 196, 199, 198, 198, 199, 200, 201]], dtype=torch.uint8)
```
Extracting semantic automatically requires models of extreme complexity, which cannot be designed by hand.

Techniques used in practice consist of

1. defining a parametric model, and
2. optimizing its parameters by “making it work” on training data.
Extracting semantic automatically requires models of extreme complexity, which cannot be designed by hand.

Techniques used in practice consist of

1. defining a parametric model, and
2. optimizing its parameters by “making it work” on training data.

This is similar to biological systems for which the model (e.g. brain structure) is DNA-encoded, and parameters (e.g. synaptic weights) are tuned through experiences.
Extracting semantic automatically requires models of extreme complexity, which cannot be designed by hand.

Techniques used in practice consist of

1. defining a parametric model, and
2. optimizing its parameters by “making it work” on training data.

This is similar to biological systems for which the model (e.g. brain structure) is DNA-encoded, and parameters (e.g. synaptic weights) are tuned through experiences.

Deep learning encompasses software technologies to scale-up to billions of model parameters and as many training examples.
There are strong connections between standard statistical modeling and machine learning.
There are strong connections between standard statistical modeling and machine learning.

Classical ML methods combine a “learnable” model from statistics (e.g. “linear regression”) with prior knowledge in pre-processing.
There are strong connections between standard statistical modeling and machine learning.

Classical ML methods combine a “learnable” model from statistics (e.g. “linear regression”) with prior knowledge in pre-processing.

“Artificial neural networks” pre-dated these approaches, and do not follow this dichotomy. They consist of “deep” stacks of parametrized processing.
From artificial neural networks to “Deep Learning”
Networks of “Threshold Logic Unit”

(McCulloch and Pitts, 1943)
1949 – Donald Hebb proposes the Hebbian Learning principle (Hebb, 1949).
1951 – Marvin Minsky creates the first ANN (Hebbian learning, 40 neurons).
1949 – Donald Hebb proposes the Hebbian Learning principle (Hebb, 1949).
1951 – Marvin Minsky creates the first ANN (Hebbian learning, 40 neurons).
1958 – Frank Rosenblatt creates a perceptron to classify $20 \times 20$ images.
1949 – Donald Hebb proposes the Hebbian Learning principle (Hebb, 1949).
1951 – Marvin Minsky creates the first ANN (Hebbian learning, 40 neurons).
1958 – Frank Rosenblatt creates a perceptron to classify $20 \times 20$ images.
1949 – Donald Hebb proposes the Hebbian Learning principle (Hebb, 1949).

1951 – Marvin Minsky creates the first ANN (Hebbian learning, 40 neurons).

1958 – Frank Rosenblatt creates a perceptron to classify $20 \times 20$ images.


This model follows Hubel and Wiesel’s results.
Network for the T-C problem

Trained with back-prop.

(Rumelhart et al., 1988)
LeNet family

10 output units

layer H3
30 hidden units

layer H2
12 x 16 = 192 hidden units

layer H1
12 x 64 = 768 hidden units

fully connected
~ 300 links

fully connected
~ 6000 links

~ 40,000 links from 12 kernels
5 x 5 x 8

~ 20,000 links from 12 kernels
5 x 5

256 input units

(LeCun et al., 1989)
ImageNet Large Scale Visual Recognition Challenge.

Started 2010, 1 million images, 1000 categories

(http://image-net.org/challenges/LSVRC/2014/browse-synsets)
AlexNet

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, 4This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional. 5(Krizhevsky et al., 2012)
Top-5 error rate on ImageNet

(Gershgorn, 2017)
GoogleNet (Szegedy et al., 2015)  

ResNet (He et al., 2015)
3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position $i$ can depend only on the known outputs at positions less than $i$.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

\[(Vaswani et al., 2017)\]
Deep learning is built on a natural generalization of a neural network: a graph of tensor operators, taking advantage of

- the chain rule (aka “back-propagation”),
- stochastic gradient decent,
- convolutions,
- parallel operations on GPUs.

This does not differ much from networks from the 90s.
This generalization allows to design complex networks of operators dealing with images, sound, text, sequences, etc. and to train them end-to-end.

(Tran et al., 2020)
The End
References


D. Gershgorn. **The data that transformed AI research—and possibly the world,** July 2017.


