CSCI 5525 Machine Learning Fall 2019

Lecture 9: Neural Networks

Oct 1st, 2019

Lecturer: Steven Wu *Scribe: Steven Wu*

Given a collection of *activation* (or *nonlinearities, transfer, link*) functions $\{\sigma_i\}_{i=1}^L$, weights, and biases, a neural network maps an input vector x to

$$F(x, \theta) = \sigma_L \left(W_L(\dots W_2\sigma_1(W_1 x + b_1) + b_2 \dots) + b_L\right)$$

where θ denotes the set of parameters $W_1, \dots, W_L, b_1, \dots, b_L$.

Choices of activation functions.

- Indicator or threshold: $z \to \mathbf{1}[z \geq 0]$.
- Sigmoid or logistic: $z \to \frac{1}{1+\exp(-z)}$.
- Hyperbolic tangent: $z \to \tanh(z)$.
- Rectificed linear unit (ReLU): $z \to \max\{0, z\}$.
- Identity: $z \to z$. This is often used in the last layer when we evaluate the loss.

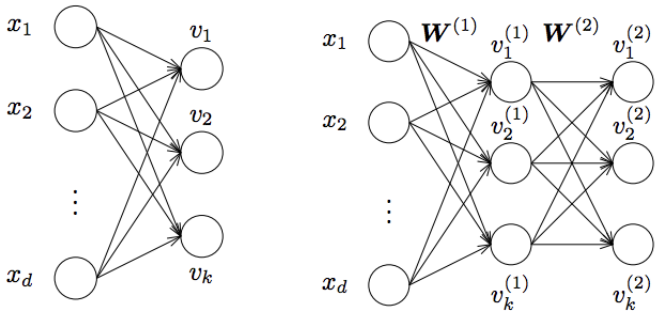

Figure 1: Graphical view of neural network. Image taken from Matus Telgarsky's slides.

DAG view. We can view a neural network as a directed acyclic graph (DAG). The input layer basically have each node corresponding to a single x_i. In the left network of Figure 1,

$$v_j = \sigma(z_j), \qquad z_j = \sum_{i=1}^{d} W_{ij} x_i$$

In some applications, each x_i might be vector-valued. For example, if x_i corresponds to a pixel, it should contain 3 values. In this case, each W_{ij} will also be a vector. Any layer that is not the input layer or the output layer is called a *hidden layer*.

Expressiveness

Recall that the XOR example is not linearly separable. (You are required to show this in your homework.) This also means that without any hidden layer, a neural network (which is just another linear model) will not classify these examples correctly either. What if we introduce a hidden layer with nonlinear activation functions? It turns out this one hidden layer will enable a lot more representation power. In fact, you can use such networks to approximate any "reasonable" functions according to the universal approximation theorem below.

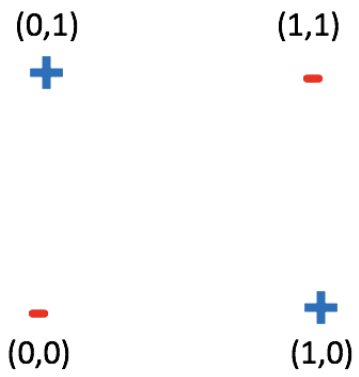

Figure 2: The XOR example.

Theorem 0.1 (Universal approximation theorem)**.** *Let* $f\colon \mathbb{R}^d \to \mathbb{R}$ *be any continuous function. For any approximation error* $\epsilon > 0$*, there exists a set of parameters* $\theta = (W_1, b, W_2)$ *such that for any* $x \in [0,1]^d$

$$|f(x) - W_2\sigma(W_1 x + b)| \leq \epsilon$$

where σ *is a nonconstant, bounded, and continuous function (e.g. ReLU and logistic).*

In other words, a single hidden layer neural network can approximate any continuous function to any degree of precision. However, such a neural network can be very wide, and even though it exists, we may not easily find it. More recently, there has been analogous universal approximation theorem with deep neural network with bounded widths that are essentially the dimension of the data [2].

1 Convolutional Neural Networks

Convolutions of two functions f and g is defined as

$$(f * g)(t) = \int f(a)\, g(t-a) da$$

The first argument f to the convolution is often referred to as the *input*, and the second argument g is called the *kernel, filter, or receptive field*. (Many things in math and engineering are called kernels).

Motivating example from [1]. "Suppose we are tracking the location of a spaceship with a sensor. The sensor provides $x(t)$, the position of the spaceship at each time step t. Both x and t are real valued, that is, we can get a different reading from the lasersensor at any instant in time. To obtain a less noisy estimate of the spaceship's position, we would like to average several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function $w(a)$, where a is the age of a measurement."

$$s(t) = \int x(a)\, w(t-a) da = (x * w)(t)$$

In machine learning, we often use *discrete convolutions*: given two functions (or vectors) $f, g \colon \mathbb{Z} \to \mathbb{R}$

$$(f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)\, g(n-a)$$

The discrete convolution operation can be constructed as a matrix multiplication, where one of the inputs is converted into a *Toeplitz matrix*. In particular, suppose that $f_i = 0$ for any $i \notin [m]$ and $g_i = 0$ for any $i \notin [n]$. We can write:

$$f * g = \begin{bmatrix} f_1 & 0 & \cdots & 0 & 0 \\ f_2 & f_1 & & \vdots & \vdots \\ f_3 & f_2 & \cdots & 0 & 0 \\ \vdots & f_3 & \cdots & f_1 & 0 \\ f_{m-1} & \vdots & \ddots & f_2 & f_1 \\ f_m & f_{m-1} & & \vdots & f_2 \\ 0 & f_m & \ddots & f_{m-2} & \vdots \\ 0 & 0 & \cdots & f_{m-1} & f_{m-2} \\ \vdots & \vdots & & f_m & f_{m-1} \\ 0 & 0 & 0 & \cdots & f_m \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_n \end{bmatrix}$$

Often time, we apply convolution over higher dimensional space. In the case of images, we have two-dimensional convolutions:

$$(f * g)(t, r) = \sum_i \sum_j f(i, j)\, g(t-i, r-j)$$

Convolutions enjoy the commutative property, which means that we can flip the arguments to the two functions:

$$(f * g)(t, r) = \sum_i \sum_j f(t-i, r-j)\, g(i, j)$$

A related concept is the *cross-correlation*:

$$(f * g)(t, r) = \sum_i \sum_j f(t + i, r + j)\, g(i, j)$$

Many machine learning libraries implement cross-correlation and call it convolution [1]. I personally find cross-correlation more intuitive. See Figure 3 for an illustration.

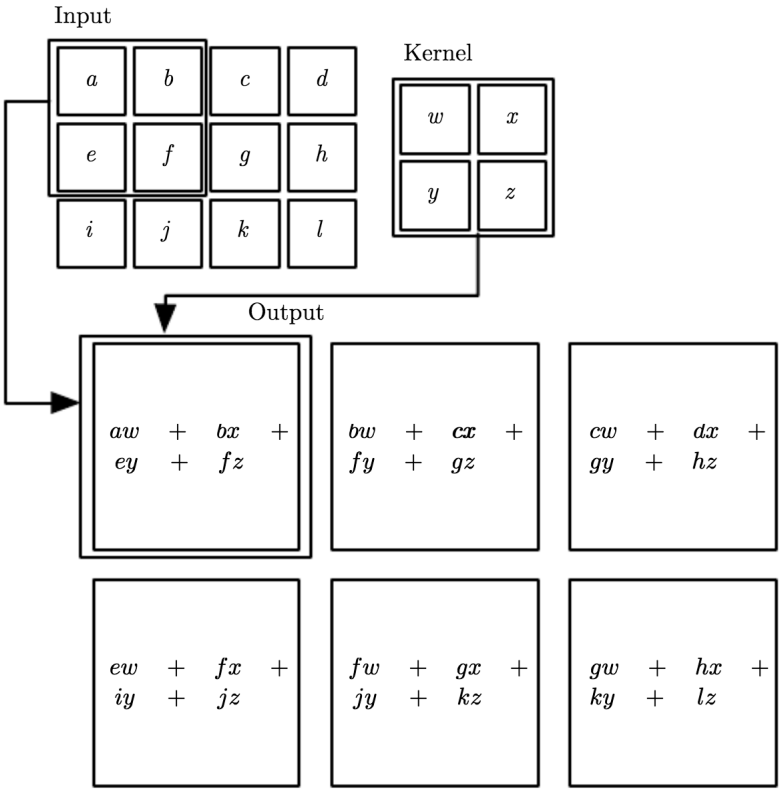

Figure 3: Example of 2D cross-correlation from [1].

There are also variants of convolutional layers:

- **Padding**: surround the input matrix with zeroes (Figure 4).
- **Strides**: shifting the kernel window more than one unit at a time (Figure 5).
- **Dilation**: convolution applied input with defined gaps, using filter of larger size (Figure 6).

Check out more animations here in this github repo.

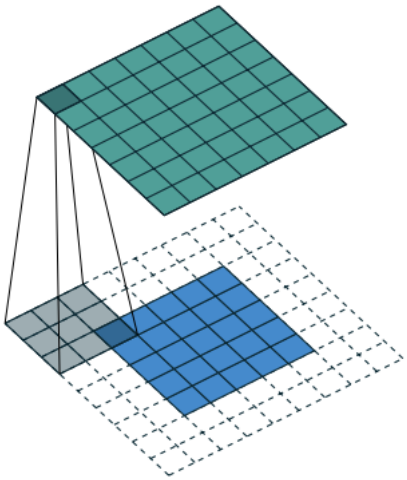

Figure 4: Padding

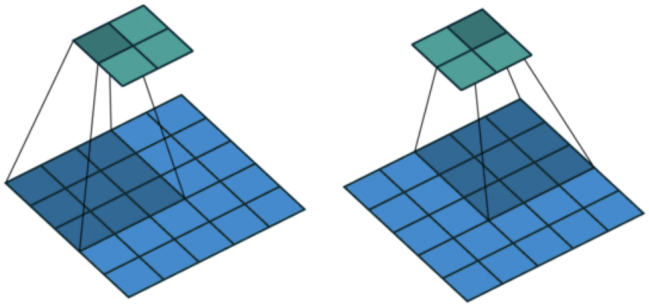

Figure 5: Strided convolution

Sparse representation. A major advantage in convolutional neural networks is that the convolutional layer can be concisely represented due to its sparity structure. See Figure 7. In CNN (top), each node in the input layer only influences a few number nodes in the next layer.

Moreover, the number of parameters we keep around is also much smaller since the kernel is shared across all nodes. In contrast, for fully-connected layers, we need to keep track of a weight coefficient W_{ij} for every pair of nodes i and j across the two layers.

Pooling. Convolutional layers are often composed with max-pooling layers, which provides useful compression. Average pooling also has similar effects. See Figure 8.

A typical CNN has the following architecure:

$$\text{Input } x \to \Big[(\text{Conv} \to \text{ReLU})^N \to (\text{Max}) \text{ Pooling}\Big]^M \to [\text{FC} \to \text{ReLU}]^P \to \text{FC}$$

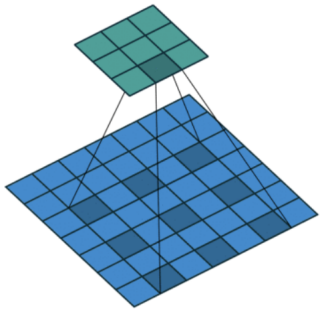

Figure 6: Dilation

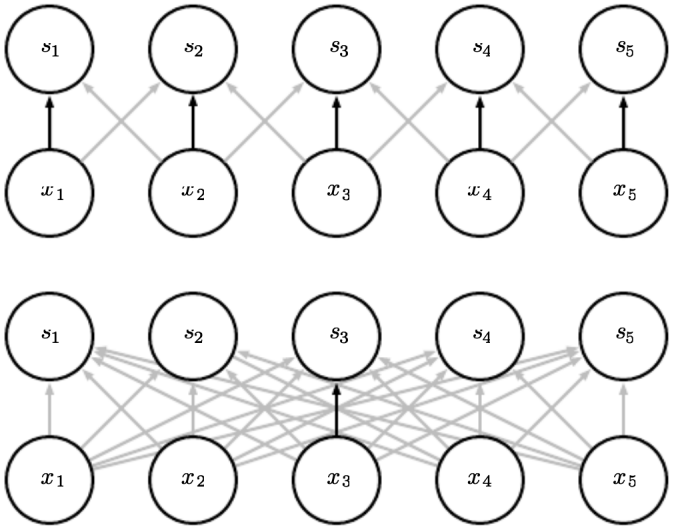

Figure 7: Sparsity of connections with convolutions due to parameter sharing.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6231–6239. Curran Associates, Inc., 2017.

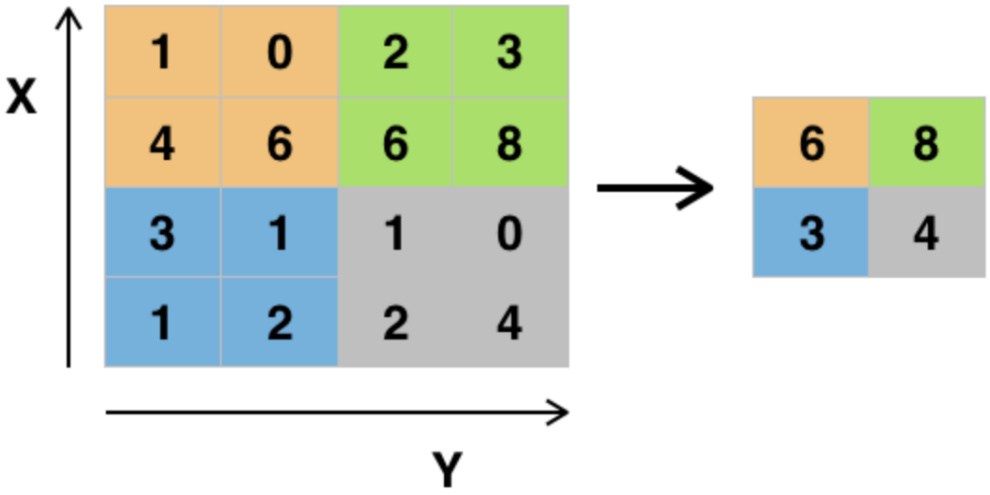

Figure 8: Max pooling layer compresses the repsentation further.