Deep Learning Refresher

Week 4

Tirtharaj Dash

Linear Algebra Recap I

- Consider a hidden layer $\mathbf{h}$ of a neural network:

$$\mathbf{h} = f(\mathbf{z})$$

  Here, f is a non-linear function applied to a vector $\mathbf{z}$.

- $\mathbf{z}$ is the output of an affine transformation $\mathbf{A} \in \mathbb{R}^{m \times n}$ applied on some input vector $\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{z} = \mathbf{A}\mathbf{x}$$

- For simplicity, assume that biases (a_0) are ignored or included in the input itself ($x_0 = 1$)

Linear Algebra Recap II

- So in matrix form:

$$\boldsymbol{A}\boldsymbol{x} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} — \boldsymbol{a}^{(1)} — \\ — \boldsymbol{a}^{(2)} — \\ \vdots \\ — \boldsymbol{a}^{(m)} — \end{pmatrix} \begin{vmatrix} \\ \boldsymbol{x} \\ \\ \end{vmatrix} = \begin{pmatrix} \boldsymbol{a}^{(1)}\boldsymbol{x} \\ \boldsymbol{a}^{(2)}\boldsymbol{x} \\ \vdots \\ \boldsymbol{a}^{(m)}\boldsymbol{x} \end{pmatrix}_{m\times 1}$$

- Here, $\mathbf{a}^{(i)}$ is the ith row of the matrix $\mathbf{A}$.

- Let's now look closely at: $\mathbf{a}^{(1)}\mathbf{x}$. This is a dot product.
- For sake of visualisation, let's assume: $n = 2$, that is $\mathbf{a}^{(1)} = (a_1, a_2)$ and $\mathbf{x} = (x_1, x_2)$

Linear Algebra Recap III

- We can now expand the dot product:

$$\mathbf{a} \cdot \mathbf{x} = \mathbf{a}^{\mathsf{T}}\mathbf{x} = (a_1, a_2) \cdot (x_1, x_2)$$
$$= a_1x_1 + a_2x_2$$

We know that there are two components of **a**: a_1 and a_2. Let's say the vector **a** makes α angle with $\hat{i}$-axis. Similarly, the vector **x** makes ξ angle with $\hat{i}$-axis.

- Therefore, $a_1 = ||a|| \cos(\alpha)$ and $x_1 = ||x|| \cos(\xi)$. Similarly, $a_2 = ||a|| \sin(\alpha)$ and $x_2 = ||x|| \sin(\xi)$

Linear Algebra Recap IV

- Now, simplifying our dot product:

$$\mathbf{a} \cdot \mathbf{x} = ||\mathbf{a}||||\mathbf{x}||(\cos(\alpha)\cos(\xi) + \sin(\alpha)\sin(\xi))$$
$$= ||\mathbf{a}||||\mathbf{x}||\cos(\xi - \alpha)$$

- The dot product output measures the alignment of the input to a specific row of $\mathbf{A}$. That is:
  - $\xi = \alpha$: **a** and **x** are perfectly aligned.
  - $\xi - \alpha = \pi$: **a** and **x** directioned opposite to each other.
- Linear transformation allows one to see the projection of an input to various orientations as defined by $\mathbf{A}$.

Linear Algebra Recap V

- Another way of understanding:

$$\boldsymbol{A}\boldsymbol{x} = \begin{pmatrix} | & | & & | \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_n \\ | & | & & | \end{pmatrix} \begin{matrix} | \\ \boldsymbol{x} \\ | \end{matrix} = x_1 \begin{matrix} | \\ \boldsymbol{a}_1 \\ | \end{matrix} + x_2 \begin{matrix} | \\ \boldsymbol{a}_2 \\ | \end{matrix} + \cdots + x_n \begin{matrix} | \\ \boldsymbol{a}_n \\ | \end{matrix}$$

- The output is the weighted sum of the columns of matrix **A**. Therefore, the signal is nothing but a composition of the input.

Extending LinAlg to Convolution I

- The numeric computation in a fully connected layer can be written as:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

- Weight matrix: $[4 \times 3]$, input: $[3 \times 1]$; output: $[4 \times 1]$

Extending LinAlg to Convolution II

- Let's now imagine problems where **x** is not just $[3 \times 1]$, but very large (e.g. audio signal). So, correspondingly the weight matrix will also become very big.

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & \cdots & w_{1k} & \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} & w_{24} & \cdots & w_{2k} & \cdots & w_{2n} \\ w_{31} & w_{32} & w_{33} & w_{34} & \cdots & w_{3k} & \cdots & w_{3n} \\ w_{41} & w_{42} & w_{43} & w_{44} & \cdots & w_{4k} & \cdots & w_{4n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Depending on n, this could lead to computationally infeasible learning of the paramter matrix $\mathbf{W}$.

Extending LinAlg to Convolution III

Property of data: Locality

- We do not care for data points that are far away.
- So, we can make the some distant portion of the weight vector w_{1k} to be 0s.
- Consider size of 3 is local for us; therefore, the first row of the matrix becomes a kernel of size 3. Let's denote this size-3 kernel as $\mathbf{a}^{(1)} = \begin{bmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{bmatrix}$.

$$\begin{bmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} & 0 & \cdots & 0 & \cdots & 0 \\ w_{21} & w_{22} & w_{23} & w_{24} & \cdots & w_{2k} & \cdots & w_{2n} \\ w_{31} & w_{32} & w_{33} & w_{34} & \cdots & w_{3k} & \cdots & w_{3n} \\ w_{41} & w_{42} & w_{43} & w_{44} & \cdots & w_{4k} & \cdots & w_{4n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Extending LinAlg to Convolution IV

Property of data: Stationarity

- Natural data have the property of stationarity (i.e. certain patterns/motifs will repeat).
- This helps us reuse kernel $\mathbf{a}^{(1)}$:weight sharing
- We use this kernel by placing it one step further each time (i.e. stride is 1), resulting in the following:

$$\begin{bmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & a_1^{(1)} & a_2^{(1)} & a_3^{(1)} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & a_1^{(1)} & a_2^{(1)} & a_3^{(1)} & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & a_1^{(1)} & a_2^{(1)} & a_3^{(1)} & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & a_1^{(1)} & a_2^{(1)} & a_3^{(1)} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix}$$

(So many 0s in the matrix: sparsity)

Extending LinAlg to Convolution V

- Number of paramters to be learned: 3
- What we saw earlier is a single layer of convolution. Like MLPs, we can also use multiple layers of convolutions with different kernels.
- The kernel matrix **A** that we saw is called a Toeplitz matrix. So, adding multiple convolution layer will lead to many such Toeplitz matrices.
- Next, we will look at a backprop derivation through a simple convolution operation.