Machine Learning, a Review for Tree Segmentations Task

April 11, 2024

Abstract

This is my own notes.

Contents

1 Introduction

Artificial Neural networks represents modes of numerical computations that are differentiable programs. We skips the basics and assume the reader already know something about Deep Neural Network, their components, and the automatic differentiation program on modern deep learning frameworks. For our discussion we introduce some definitions to make for a better presentation of computataional concepts occured in Artificial Neural Networks (ANNs).

1.1 Notations, Common Operations

The index starts with "1" in our writings. But it starts with zero if we are using programming languages such as python. To distinguish these 2 type of indexing on a tensor X, we use $X_{i_1, 1_2, \cdots}$ to denotes indexing using the natural number and we use $X_{[i_1, i_2, \cdots]}$ to denote indexing of the tensor using offset indices that starts with zero.

Definition 1 (Component). A component is a function $f(x; p|w) : \mathbb{R}^m \mapsto \mathbb{R}^n$. x is the inputs and w represent trainable parameters, usually in the form of a multi-dimensional array. And p represents parameters that are not trainable parameters.

Definition 2 (Connection). Let $f : \mathbb{R}^n \mapsto \mathbb{R}^m, g : \mathbb{R}^m \mapsto \mathbb{R}^k$ be two components, then a connection between is a $\mathbb{R}^m \mapsto \mathbb{R}^m$ function $h(x; p|w)$ with trainable parameters w, and parameter p.

Example 1.1.1 (Dense Layer). Let $m, n \in \mathbb{N}$, let $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^n$, then a Dense layer is a $\mathbb{R}^m \mapsto \mathbb{R}^n$ functions with a list of activation functions σ_i for $i = 1, \cdots, n$. Let $x \in \mathbb{R}^m$ be the input, then a dense layer is a component. We define its computation:

$$\text{DnsLyr}(x; m, n|(A, b), \{\sigma_i\}_{i=1}^n) = \left[z \mapsto \bigoplus_{i=1}^n \sigma_i(z_i)\right](Ax + b)\,.$$

Where, inside of $[\cdot]$, we denote the definition of a anonymous function.

Example 1.1.2 (Multi-Layer Perceptron). Let $l_1, l_2, \cdots, l_N$ be integers. We define the Multi-Layer Perceptron to be a composition of dense layer mapping from l_i to l_{i+1} for $i = 1, \cdots, N-1$. Let $\sigma_{i,j}$ represent the activation function for the j th output in the i th layer. Then a Multi-Layer Perceptron (MLP) is a component admit representation

$$\begin{aligned}&\text{MLP}\left(x; l_1, \cdots, l_n|\{(A_i, b_i)\}_{i=1}^N\right) : \mathbb{R}^{l_1} \mapsto \mathbb{R}^{l_N}\\ :=& \left[\bigodot_{i=1}^n \text{DnsLyr}\left((\cdot); l_i, l_{i+1}\left|(A_i, b_i), \{\sigma_{j,i}\}_{j=1}^{l_i}\right.\right)\right](x).\end{aligned}$$

Where $\bigodot$ is functional composition and it represents $\bigodot_{i=1}^n f_i = f_n \circ \cdots \circ f_1(x)$, and $(\cdot)$ represents the input of the anonymous function, in this case it's the dense layer.

2 Basic of Neural Architecture

These concepts and components are relevant to the architecture of Vision Networks.

Definition 3 (Convolution 2D)**.** Let u, v be multi-array of dimension $m \times n$ and $k \times l$. We assume that $m \leq k$ and $n \leq l$. Then the convolution operator $*$ is a mapping from $(\mathbb{C}^M \times \mathbb{C}^n) \times (\mathbb{C}^k \times \mathbb{C}^l) \mapsto \mathbb{C}^{(m-k)\times(n-l)}$. Then the convolution is defined as

$$(u * v)_{t,\tau} = \sum_{i=1}^{k}\sum_{j=1}^{l} u_{i,j}v_{i+t,j+\tau}.$$

2.1 The Input and Output of Multi-Channel Fixed Dimension Signal

In this section, we discuss the shape of the output signal for a 2D/1D signal after

1. Max/average Pooling,
2. Convolution.

Here we note that these above operations places the same type of constraints on the output shape of the signal given the shape of the input. We all these type of operations: "Multi-Channel Fixed Dimenion Transform (MCFDT)". This is not a name from the literature I just made it up. These operations share a common set of parameters that determine the shape of the output tensor given the shape of the output tensor. These are the list of parameters:

1. `in_channel`: Type integers. The number of channels expected for the input signal.
2. `Out_channels`: Type integers. The number of channel expected for the output signal.
3. `kernel_size`: The size of the kernel used for the 1D signal, for certain type of operations. The kernel represent the size of the window where an computations are going to carry out locally on the input signal.
4. `stride`: How many elements does the kernel skips with respect to the input signal for each of the adjacent output elements in the output signal.
5. `padding`: The numbers of zero/null/-inf elements added to the 2 side of the 1D signal.
   (a) `padding_mode`: Determine different modes of padding the boundary of the input signal.
      i. `zero`, (default)
      ii. `reflect`,
      iii. `replicate`,
      iv. `circular`.
6. `dilation`: Spacing between the elements of the kernels.

7. `blocks`: Number of blocked connections from intput channels to output channels, default to 1.

8. `bias`: Add learnable bias to the component, default to 'true'.

Fact 2.1.1 (MCFDT 1D)**.** Let input signal be a tensor of size (N, C, L).

1. N is the number of samples from the batch.

2. C is the number of imput channels for the signal.

3. L is the length of the input signal.

Let p, d, k, s denotes: "padding, dilation, kernel size, stride". Then the output signal is a tensor of size (N, C', L') with

$$L' = \left\lfloor \frac{(L + 2p) - (d(k-1) + 1)}{s} + 1 \right\rfloor . \tag{2.1.1}$$

The parameter C', is user defined.

Remark 2.1.1. The size of the input signal with paddin is $(L + 2p)$. The size of the dilated kernel is $d(k-1)$. The $+1$ on the numerator of the fraction and the $+1$ at the outside of the fraction remains as a mystery to the writer. The process of computing the shape for an input signal of dimension $(N, C, [\cdots])$, is applying the same computations on all the dimensions to compute the output of the signal.

Fact 2.1.2 (MCFDT 2D)**.** Let X be a tensor of shape (C, N, L_1, L_2), then the output signal is of size (C', N, L_1', L_2'), where C' is defined.

1. N is the number of samples from the batch.

2. C is the number of imput channels for the signal.

3. L is the length of the input signal.

Let $p_i, d_i, k_i, s_i,\ i \in \{1, 2\}$ be the "padding, dilation, kernel size, stride" along the L_1 and L_2 dimension of the input signal. Then the shape parameters (L_1', L_2') can be computed by the formula:

$$L_i' = \left\lfloor \frac{(L_i + 2p_i) - (d_i(k_i - 1) + 1)}{s_i} + 1 \right\rfloor \quad \forall i \in \{1, 2\}.$$

Remark 2.1.2. This is analogous to the 1D case and it's just applying the same formula to the corresponding dimension of the tensor to determine the shape of the output tensor.

2.2 Convolutional Layers

In this section we talk about 2D convolution component (pytorch link) inside of an ANNs. The convolution operations module contains more detailed parameters.

Definition 4 (2D Convolution Layers)**.** Assuming that we have a single sample. Let (C, H, W) be the shape of the input tensor. C is the number of channel, and H, W are the height and width. We use this because image tensors are usually in the shape of $(3, H, W)$. Define the component to be a function, mapping from (C', H', W'). Let (C', C, K, L) denotes the dimension of the kernel tensor denoted by: $\mathcal{K}$. Then mathematically, the computation of the output tensor Y given input tensor X can be computed as

$$Y_{c',h',w'} = \text{ReLU}\left(b_{c'} + \sum_{k=1}^{C} (\mathcal{K}_{c',k,:,:} * X)_{h',w'}\right). \tag{2.2.1}$$

Remark 2.2.1. There is a bias term and an activation function for a specific channel. The kernel for an image, which is a 3D tensor, is also a 3D tensor. A single tensor is applied to all channels of the input channel, aggregated by summing up across these different channels.

1. "group", see here for an explanations.

The activaton function above is "ReLU", but it doesn't have to be. See here for the pytorch documentations. The size of the output tensor is determined by fact 2.1.2.

2.3 Pooling Layers

Let's define these quantities

1. (N, C, H, W) is the size of the input signal.
2. (N, C, H', W') is the signal of the output layer.
3. (k_1, k_2) is the size of the kernel.
4. N is usually the size of the batched samples.
5. $[s_1, s_2]$ be the stride parameters for the kernels.

Definition 5 (2D Max Pooling Layers)**.** Let X be the signal of size (N, C, H, W), let the output signal be Y of size (N, C, H', W'), then the output can be precisely described by the following formula:

$$Y_{i,c,h,w} = \max_{\substack{m=1,\cdots,k_1-1 \\ n=1,\cdots,k_2-1}} \{X_{i,c,hs_1+m,s_2w+n}\}.$$

Remark 2.3.1. See max pool 2d for pytorch documentation for this component.

Definition 6 (2D Averge Pooling Layer)**.** Let X be the input signal of size (N, C, H, W), let output signal Y be of size (N, C, H', W'), then the output for an average pooling layer can be computed as

$$Y_{[i,c,h,w]} = \text{mean}\left(X_{[i,c,hs_1+m,hs_2+n]}, m \in \{0, \cdots, k_1 - 1\}, n \in \{0 \cdots, k_2 - 1\}\right).$$

Remark 2.3.2. See averaeg pool 2d for pytorch documentation for this component.

2.4 Convolution 2D Normalization Layer

The convolution 2D Normalized layer sandwitch one batch normalization between layers of 2D convolutions. For more information visit Conv2DNormActivation.

2.5 Tranpose Convolution Layer

A transpose convolution is not a partial reverse of the convolution operations. It simply another convolution layer where parameters such as: "`stride, padding, dilation`" means different things. It increase the dimension of the input signal instead of decreasing it. It's usually used as a learnable upsample component inside of a Neural Network. For more information visit CONV2DTRANSPOSED for pytorch documentations.

2.6 Batch Normalization Layer

3 Graphical Model and Probability Dependence

Graphical models are used to model probability dependence between random variables. In the context of deep learning, it helps with training hierarchical models.

4 Variational Autoencoder, Bayesian Learning

5 Computer Vision Networks

5.1 Residual Net

Residual net is one of the most phenomenal vision network. Back in the early day people use to train deeper and deeper convolutional networks, but the convergence of the error gets worse as the layer deepens. He et al.[1] proposed a remedy. The remedy for low convergence of raining and test error to set up the neural network to learn perturbations on an identity map, they call it the residual neural network.

6 Image Instance Segmentations

We introduce the problem of Image Instance Segmentations. Given an image that contains objects of interests, we want:

1. Classify objects by the categories
2. Segment/identify their locations on the images.
3. Differentiate different instance of the objects that belong to category of interest.

6.1 Faster RCNN

In this section, we take a look at "Faster RCNN" proposed by Ren et al.[2]. Read this blog post for a quick overview of the neural network.

6.1.1 Region Proposal Network

The RPN takes input from *convolutional feature map* (hidden layer of another vision network) propose rectangular region back in the original images that may contain objects of intersts. It consists of a small network (referred as the *mini network*) that slides through the convolutional feature map to produce outputs. In paper, the mini network takes in a matrix from the convolutional feature map as the input. The output at each location is fed into two components, one for box regression, and the other one for box classifications. The feature maps are low dimensional laten features produced by any generic vision networks. We clear up some key words here:

1. *Convolutional Feature Map*, in the paper, it refers to a 2 dimensional tensor that is the hidden layer of any vision network, i.e: VGG, AlexNet, or RestNet.

2. *Reception Field.* Take any outputs (i, j, c) from the convolutional feature map, the ouput can be tranced back to a region in the original image, i.e: pixels (i', j', c') in the original image where perturbing pixels values will change the output at (i, j, c) in the convolutional feature map.

3. *Mini Network.* It's the neural network that we slide through the center of every position in the convolutional feature map. The same mini-network is used for the convolutional feature map of the entire image.

The pytorch implementations of the Region proposal network can be found here. In MATLAB, it's implemented as one single atomic component and it's made by the function `regionProposalLayer`.

Definition 7 (Anchor Boxes)**.** Let a tensor X of shape (H, W, C) be the feature map resulted from a single input sample. The mini network takes in $X_{[i-1:i+1,j-1:j+1,:]}$, from the feature map and it produces *anchor boxes* and classifications scores for each box. These *anchors boxes* represent regions with a fixed set of ratios and sizes, corresponding to a region in the original image (it will be used by other components inside of RCNN neural network) centered at the center of the reception field of $X_{[i,j,:]}$. The center of the reception field of any give output location (i, j) in the convolutional feature map are referred to as *anchor*.

Remark 6.1.1. Anchor boxes are produced by the mini network and it consists of k boxes at each anchor. The mini network only predict predetermined type of anchor boxes. We state the following facts associated with the anchor boxes.

1. The output for the bounding box are 4 of the coordinate related to the center of the anchor.

2. An anchor is in the input image, framing the object of interests and the parameters are the relevant coordinate.

We now discuss the mini network (RPN Head) component. In the paper, It takes in a $n \times n$ region from the feature map, pass it through several layers of convolutional network, branch into two 1×1 convolution layer before. They are then flattened into a vector and feed into two dense stacked linear neural networks. These two dense networks are classifications, and regression network. The source code of this component is line 15 here. We will talk more about it when it comes to training the RPN using a loss function.

7 Case Study | The Deepforest Project

Links:

1. Deepforest repo source code commit `7992b82`, main.py.
2. Deep forest documentations.

7.1 Techinical Details for Programmers

8 Case Study | Direct LiDAR Point Cloud Architecture

9 Case Study | Forest Allometrics Extractions, Predictions

A Appendix Section 1

This is the appendix section.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. arXiv:1512.03385 [cs].

[2] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, January 2016. arXiv:1506.01497 [cs].