Neural Net and Deep Learning

🅭 by Xia, Tingfeng

Tuesday 11th February, 2020

Contents

1 Optimization

1.1 Prerequisites

Clairaut's Theorem states that second order derivatives are such that

$$\frac{\partial^2 \mathcal{J}}{\partial \theta_i \partial \theta_j} = \frac{\partial^2 \mathcal{J}}{\partial \theta_j \partial \theta_i} \tag{1.1}$$

The Hessian Matrix is a symmetric matrix (due to Clairaut's Theorem) defined as

$$\mathbf{H} = \nabla^2 \mathcal{J} = \begin{pmatrix} \frac{\partial^2 \mathcal{J}}{\partial \theta_1^2} & \frac{\partial^2 \mathcal{J}}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 \mathcal{J}}{\partial \theta_1 \partial_D} \\ \frac{\partial^2 \mathcal{J}}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 \mathcal{J}}{\partial \theta_2^2} & \cdots & \frac{\partial^2 \mathcal{J}}{\partial \theta_2 \partial \theta_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathcal{J}}{\partial \theta_D \partial \theta_1} & \frac{\partial^2 \mathcal{J}}{\partial \theta_D \partial \theta_2} & \cdots & \frac{\partial^2 \mathcal{J}}{\partial \theta_D^2} \end{pmatrix} \tag{1.2}$$

Second-Order Taylor Approximation Locally, a function can be approximated by its second order Taylor approximation. We have, for $\boldsymbol{\theta}$ that is sufficiently close to $\boldsymbol{\theta}_0$

$$\mathcal{J}(\boldsymbol{\theta}) \approx \mathcal{J}(\boldsymbol{\theta}_0) + \underbrace{\nabla \mathcal{J}(\boldsymbol{\theta}_0)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0)}_{1st\ order} + \underbrace{\frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{H}(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0)}_{quadratic} \tag{1.3}$$

Notice that if $\boldsymbol{\theta}$ is a critical point, then the gradient would be zero. In such case, the approximation will be

$$\mathcal{J}(\boldsymbol{\theta}) \approx \mathcal{J}(\boldsymbol{\theta}_0) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \mathbf{H}(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0) \tag{1.4}$$

Spectral Decomposition Since the Hessian matrix is symmetric, we know that it is spectral decompose-able, that is

$$\mathbf{H} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1} \tag{1.5}$$

Notice that orthogonal matrices are ***necessarily*** invertible, with transpose and inverse equal to each other.

where $\mathbf{Q}$ is the (orthogonal) matrix where columns are the eigenvectors and $\boldsymbol{\Lambda}$ is the diagonal matrix with eigenvalues on diagonal.

1.2 Determining Curvature

Often, we refer to the hessian matrix $\mathbf{H}$ as the curvature of a function. Suppose that we are moving along a line defined by $\boldsymbol{\theta} + t\mathbf{v}$ for some vector $\mathbf{v}$. According to the second order Taylor Approximation, we have

$$\mathcal{J}(\boldsymbol{\theta} + t\mathbf{v}) \approx \mathcal{J}(\boldsymbol{\theta}) + t\nabla \mathcal{J}(\boldsymbol{\theta})^\top \mathbf{v} + \frac{t^2}{2} \mathbf{v}^\top \mathbf{H}(\boldsymbol{\theta})\mathbf{v} \tag{1.6}$$

Thus, in the case where $\mathbf{v}^\top \mathbf{H}\mathbf{v} > 0$, the cost function curves up, i.e. has ***positive curvature*** where as in the case $\mathbf{v}^\top \mathbf{H}\mathbf{v} < 0$, it has a ***negative curvature***.

1.2.1 Definite-ness

- A matrix $\mathbf{A}$ is positive definite if $\mathbf{v}^\top \mathbf{A}\mathbf{v} > 0, \forall \mathbf{v} \neq \mathbf{0}$.
- We sat it is positive semidefinite if $\mathbf{v}^\top \mathbf{A}\mathbf{v} \geq 0, \forall \mathbf{v} \neq \mathbf{0}$.
- Equivalently, matrix is positive definite *if and only if* all its eigenvalues are positive, and is positive semidefinite *iff* all its eigenvalues are non-negative. I will now show the case for positive definite-ness.
  Proof: Let $\mathbf{v} \neq 0 \in \mathbb{R}^n$, then we know

$$\mathbf{v}^\top A\mathbf{v} = \mathbf{v}^\top Q\Lambda Q^\top \mathbf{v} = (\dagger) \tag{1.7}$$

  Now, if we define $\mathbf{z} = \mathbf{v}^\top Q$, we have

$$(\dagger) = \mathbf{z}\Lambda\mathbf{z}^\top = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} z_1 & \dots & z_n \end{bmatrix} = \sum_i \lambda_i z_i^2 \tag{1.8}$$

  Since $\mathbf{v} \neq \mathbf{0}$, we know $\mathbf{z} = Q^\top \mathbf{v} \neq \mathbf{0}$, we know that $(\dagger)$ must be positive and this concludes the proof. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.\dagger$

2 Convolutional Neural Net and Image Classification

2.1 Motivations for Convolution Layer

- Images are, usually, large and hence using a lot of fully connected layers would result in an insane amount of parameter to learn and makes calculation intractable.
- In images, there are usually *local* patterns or relationships. For example in the task of semantic segmentation, usually the pixel at bottom left has very little to do with the pixel at top right and using a fully connected layer where every pixel contributes to every output would be wasteful. Thus, we want to have an operation that focus on ***local*** patterns of the input image.
- Also, the same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts of the image as well which motivates us in using a "filter" to "slide" across the input.

2.2 The Convolution Operator

2.2.1 1-D Signal Processing

Consider two arrays, a and b. The result of the convolution will be a new array, where

In order to make this indexing notation to work, we have to have the indices start at zero rather at one.

$$(a * b)_t = \sum_{\tau} a_\tau b_{t-\tau} \tag{2.1}$$

where the summation over τ is a lazy notation for saying summing over all combinations that makes sense. [1]

2.2.2 2-D Convolution

Consider two two dimensional arrays, A and B. The result of the convolution will be such that the slot at $(A * B)_{ij}$ is calculated as

$$(A * B)_{ij} = \sum_{s} \sum_{t} A_{st} B_{i-s,j-t} \tag{2.2}$$

Usually, we call the matrix/tensor that we convolve the original input with a "kernel" or "filter".

2.2.3 Properties of Convolution

- Convolution is ***Commutative***, i.e.,

$$a * b = b * a \tag{2.3}$$

- Convolution is ***Linear***, i.e.,

$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c \tag{2.4}$$

2.3 Canonical Kernels

Some of these canonical kernels are not working as expected...

2.3.1 Blurring Kernel

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \tag{2.5}$$

[1]We assume infinite zero padding here, more on this later.

2.3.2 Sharpening Kernel

$$\begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} \tag{2.6}$$

2.3.3 Edge Detector Kernel

$$\begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} \tag{2.7}$$

2.3.4 Vertical Edge Detector Kernel

$$\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \tag{2.8}$$

2.4 Convolutional Networks

In a Convolutional Neural Net, of course there will be convolution layers, however there is another sort of layers that are common, called the pooling layer. Intuitively, pooling layers shrink the dimensions by taking a “local pool”.

2.4.1 Pooling Layer

Most commonly, we use the max-pooling operation in the pooling layer, which computes the maximum of the units in a pooling group

$$y_i = j\overset{\max}{\text{in local pooling group}} z_j \tag{2.9}$$

where z represents the input and y is the output. Typically, we use a 2×2 max pooling unit, which outputs

$$\begin{array}{|c|c|}\hline \alpha & \beta \\ \hline \gamma & \theta \\ \hline \end{array} \quad \longrightarrow \quad \boxed{\max\{\alpha, \beta, \gamma, \theta\}} \tag{2.10}$$

Thanks to pooling layers, deeper layers’ filters will cover a larger region of the input than equal-sized filters in the lower layers. We say that deeper pooling layers have larger receptive fields in terms of the original image.

2.4.2 Non-linearity in Convolutional Layers

After convolution operation, it is common to add a non-linear activation function to introduce non-linearity. We are doing this because convolution is a linear operation and stacking convolution layers together without non-linearity is no more powerful than a single linear layer (possibly a fully connected layer). For example, the order of the layers could be

$$\text{Image} \to \underbrace{\text{Convolution} \to \text{ReLU activation}}_{\text{Convolution Layer}} \to \text{Max Pooling} \to \text{Convolution} \to \cdots \tag{2.11}$$

2.4.3 Equivariance and Invariance

TODO: I have no idea...

We want our network's responses to be robust to translations of the input, which could mean the following two things

- Convolution layers are equivariant[2]: if you translate the inputs, the outputs are translate by the same amount.
- Network's predictions are invariant: if you translate the inputs, the prediction should not change. Pooling layers provide invariance to small translations.

2.4.4 Channels in Convolution Layers

Each layer is consist of several feature maps, or ***channels***, each of which is an array (and of the same size). In the case where input is an image, then usually we would have 3 channels for coloured input (RGB channels) and 1 channel if it is in greyscale. Each unit is connected to each unit within its receptive field in the previous layer. This includes *all* of the previous layer's feature maps.

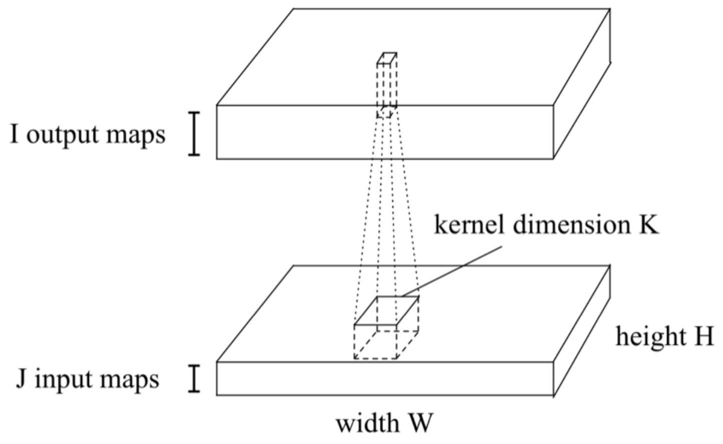
2.5 Size of Convolutional Neural Nets

There are several "measures of sizes" that are interested in, namely

- ***Number of units:*** measures the activations needed to be stored in memory during training for back propagation.
- ***Number of weights:*** since weights need to be stored (and updated at each iteration) in memory.
- ***Number of connections:*** measures the computation costs; approximately 3 add-multiply operations per connection. (1 for the forward pass, and 2 for backward pass.)

[2]Equivariant means roughly"unchanged in terms of distortion"

Below is the number of parameters in a fully connected multilayer perceptron layer along side with those in a convolutional layer. [1]



	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections	W^2H^2IJ	WHK^2IJ

2.5.1 Rule of thumb on size

- Most of the units and connections are in the convolution layers, and
- Most of the weights are in the fully connected layers.

2.6 Supervised Pre-training and Transfer Learning

- In practice, we don't usually train an image classifier from scratch since it is unlikely hat we will have millions of cleanly labeled images for our specific datasets.
- If we want to do a computer vision task, it is common to just "fine-tune" a pre-trained convolutional neural net on ImageNet or OpenImage.
- We will fix most of the weights in the pre-trained network. Only the weights in the last layer will be randomly initialized and learnt on the current dataset/task.

Fine-tune, how?

- This depends on how many training examples do we have in the new dataset. For example, if we have fewer ew examples, then we can fix more weights from the pre-trained networks and only train a small subset of all weights. Of course, vice versa.
- This also depends on how similar is the new dataset to the dataset that our pre-trained model was trained on. That is, we need 'more' fine-tuning if they have dissimilar datasets.
- ***#! Important:*** Learning rate for the fine-tuning stage is often much lower than the learning rate used for training from scratch.

why?

References

[1] Jimmy Ba. Csc421/2516 lecture 5: Convolutional neural network & image classification. 2020.