Notes

Imperial College London

Department of Computing

Machine Learning for Imaging

Author:
W (CID: your college-id number)

Date: March 7, 2020

1 Image Classification

The image classification process can be broken down into two steps: Features Extraction and Classification.

Feature Extraction process extracts discriminative features that should be robot to noise and variance. Common feature extraction algorithms include:

- SIFT
- HoG
- SURF
- BRIEF

The extracted features are then used to train a model to perform classification. Examples of model include:

- K Nearest Neighbour (KNN)
- Support Vector Machine (SVM)
- Logistic Regression
- Boosting
- Decision Trees
- Neural Networks

1.1 Model Bias and Variance

An important concept to understand in model prediction is the tradeoff between bias and variance.

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.

Variance is the amount that the estimate of the target function will change if different training data was used.

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. However, there is a tradeoff between bias and variance. As we increase the complexity of the model, bias decrease but variance will increase because the model captures noise and irrelevant features in the training data, leading to overfitting.

1.2 Ensemble Learning

Ensemble Learning is a technique that combines the predictions of several predictors to produce a better predictive model.

Ensemble Learning can be divided into two groups:

- Sequential Ensemble Methods: Generate base learners sequentially. The motivation is to exploit the dependence between the base learners, thus learning a complementary set of predictors that reduce the bias.
- Parallel Ensemble Methods: Generate base learners in parallel. The motivation is to exploit independence between the base learners, thus reduces variance.

In ensemble learning, **weak learners** are models that can be used as building blocks for designing more complex predictors. These basics models perform not so well by themselves either because they have a high bias or because they have too much variance to be robust.

1.2.1 Bagging (Bootstrap Aggregating)

Bagging is a parallel ensemble method. It is used to reduce variance and avoid overfitting. The algorithm is as follow:

1. Sample different training set of the same size **with replacement**
2. Build a model for each of the training set using the same model class
3. Combine the predictions of each model by voting or averaging

Although Bagging can be used with any model class, it is usually used with decision trees (Random Forest Algorithm)

1.2.2 Boosting

Boosting is a sequential ensemble method. Each model in the sequence is fitted giving more importance to observations in the dataset that were badly handled by the previous models in the sequence.

Intuitively, each new model focus its efforts on the most difficult observations to fit up to now, so that we obtain, at the end of the process, a strong learner with lower bias.

2 Neural Network

2.1 Activation Functions

We can apply different activation functions to the neurons in the hidden layers. The activation function for the output layer is dependent on the type of problem we are trying to solve.

2.1.1 Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$\sigma'(z) = \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} = \sigma(z)(1-\sigma(z))$$

Sigmoid is a theoretical interest because it has range $(0,1)$ so it act just like a neuron.

One problem with sigmoid is the **vanishing gradient problem**, most part of the sigmoid is flat and has a gradient of zero making it difficult to update weights in gradient descent.

Another problem with sigmoid is that it **offsets the data**. Even if the data is normally standardized with 0 mean and variance 1, the output of the sigmoid $\sigma(0) = 0.5$

2.1.2 Hyperbolic Tangent

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
$$tanh'(z) = \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2} = 1 - tanh^2(z)$$

$tanh$ is typically better than sigmoid because it does not offset the data, $tanh(0) = 0$. However it still has the **vanishing gradient problem**.

2.1.3 Rectifier Linear Unit (ReLU)

$$relu(z) = max(0,z)$$
$$relu'(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

ReLU is the most commonly used activation function because it trains more easily compare to sigmoid and hyperbolic tangent.

One problem with ReLU is **dead neurons**. Since the gradient is zero for inputs less than equal to zero, the weights won't be update again once z reaches zero. As a result this neuron will always output 0. The solution is to add more hidden units and hope not many of them become dead neurons.

3 Stochastic Gradient Descent

There are different ways to do gradient descent:

- Full gradient descent: Perform gradient descent on full data set

- Batch gradient descent: Perform gradient descent on batch of data
- Stochastic gradient descent: Perform gradient descent using one data at a time

Full gradient descent does not work when the data set becomes too large. On the other hand stochastic gradient descent is takes too long to reduce the cost function. The medium of the two is **batch gradient descent**.

In batch gradient descent, in each epoch we split the data set into N/B batches of size B. Then for each batch, we find the gradient and update the weights and bias.

3.1 Monmentum

In batch gradient descent, we make small steps in the direction of the gradient until it reaches a local minimum. However, gradient descent struggles in **pathological curvatures** (often described as valleys, trenches or ravines).

Momentum takes advantage of the knowledge accumulated from past gradients to determine the direction to go. It computes the **exponential moving average (EMA)** of the gradients, and use it to update the weights.

In normal gradient descent, weights are updated by

$$W_t \leftarrow W_{t-1} - \alpha \frac{\partial J}{\partial W_{t-1}}$$

In momentum, instead of using the derivatives directly, we take the exponentially weighted averages of the derivatives

$$\begin{aligned} V_t &\leftarrow \beta V_{t-1} + (1-\beta)\alpha \frac{\partial J}{\partial W_{t-1}} \\ W_t &\leftarrow W_{t-1} - V_t \end{aligned}$$

V_0 is initialized to 0, in each iteration it accumulates the new calculated gradient. β is the hyperparameter momentum, which is a value between 0 to 1 (usually 0.9).

3.2 Nesterov Momentum

Nesterov Momentum is a slightly different version of the momentum update and can be summarized in two steps

1. Make a jump to the point where the current momentum is pointing to
2. Compute the gradient at that point and make correction

The only difference between Nesterov Momentum and standard momentum is the point where the gradient is estimated.

$$\begin{aligned} V_t &\leftarrow \beta V_{t-1} + \alpha \frac{\partial J}{\partial W_{t-1}} \\ W_t &\leftarrow W_{t-1} - \beta V_t - \alpha \frac{\partial J}{\partial W_{t-1}} \end{aligned}$$

3.3 AdaGrad

AdaGrad introduces a new variable *cache*, and each parameter has its own *cache*. If W is a $M \times N$ matrix, then we create a $M \times N$ *cache* for W.

$$cache \leftarrow cache + (\nabla_W J)^2$$
$$W \leftarrow W - \alpha \frac{\nabla_W J}{\sqrt{cache + \epsilon}}$$

Where $\nabla_W J$ is the derivative of loss w.r.t W and $\epsilon \approx 10^{-9}$ to prevent division by 0. The *cache* is accumulating the square of gradients and so it is always positive.

If a parameter has accumulated large gradients, the cache will be large and will have smaller learning rate.

If a parameter has accumulated small gradients, the cache will be small and will have higher learning rate.

3.4 RMSProps

AdaGrad decreases the learning rate too aggressively. RMSProps introduces a hyperparameter $\beta \approx 0.99$ to slow down the increase of *cache*, hence slow down the decrease of the learning rate.

$$cache \leftarrow \beta \times cache + (1 - \beta)(\nabla_W J)^2$$
$$W \leftarrow W - \alpha \frac{\nabla_W J}{\sqrt{cache + \epsilon}}$$

There is no standard for the initial value of *cache*. Some libraries initialize it to 0, others initialize it to 1.

3.5 Adam Optimization

Adam is a new modern adaptive learning technique that combines the benefit of AdaGrad and RMSProp.

Adam calculates an exponential moving average of the gradient and the squared gradient. $\beta_1 \approx 0.9$ and $\beta_2 \approx 0.99$ controls the decay rate of these moving averages.

$$cache_{mean} \leftarrow \beta_1 \times cache_{mean} + (1 - \beta_1)(\nabla_W J)$$
$$cache_{variance} \leftarrow \beta_2 \times cache_{variance} + (1 - \beta_2)(\nabla_W J)^2$$

Since $cache_{mean}$ and $cache_{variance}$ are initially set to 0, Adam uses bias-correction estimates to overcome the problem with biased towards 0. At iteration t, we update the *cache* as above, and correct these estimates by

$$\hat{cache}_{mean} = \frac{cache_{mean}}{(\beta_1)^t}$$

$$\hat{cache}_{variance} = \frac{cache_{variance}}{(\beta_2)^t}$$

Finally we update the weights using the bias corrected *cache*.

$$W \leftarrow W - \alpha \frac{\hat{cache}_{mean}}{\sqrt{\hat{cache}_{variance} + \epsilon}}$$

4 Regularisation

Regularisation is the process of adding information to solve ill-posed problems and prevent overfitting.

4.1 Early Stopping

Interrupt training when its performance on the validation set starts dropping.

4.2 L1 Regularisation

Sometimes we only want to select a few useful feature to predict the trend. L1 Regularization achieves **sparsity**, meaning the final weight will contains mainly zeros with a few non-zeros.

We simply add a penalty term to the original cost. So L becomes

$$L = L + \lambda|w|$$

4.3 L2 Regularisation

L2 Regularisation encourages small weights and prevents a weight to go extremely large. This is done by adding the squared magnitude of the weights multiplied by a constant to the error function to punish large weights.

$$L = L + \lambda|w|^2$$

4.4 Dropout

Dropout regularization emulates ensemble learning by ignoring random nodes for each iteration during training. Given a neural network with N nodes, and each node can have two states: keep or drop. This is like training 2^N neural network with different configurations.

Dropout prevents over fitting since it forces the neural network to not rely on a single feature.

4.5 Data Augmentation

Generate new training instances from existing ones, artificially boosting the size of the training set. This can be done by rotating, shifting, rescaling, and adding noise to the original image.

5 Weight Initialisation

Given a neural network layer with M inputs and N outputs, we can initialize the weights using the following methods.

- Create random weights with standard normal distribution then scale each weight by 0.01
- Create random weights with standard normal distribution then scale each weight by $\sqrt{2/(M+N)}$. (tanh)
- Create random weights with standard normal distribution then scale each weight by $\sqrt{1/M}$. (tanh)
- Create random weights with standard normal distribution then scale each weight by $\sqrt{2/M}$. (ReLU)
- Glorot Uniform: Select weights from uniform distribution $U\left(-\sqrt{6/(M+N)}, \sqrt{6/(M+N)}\right)$
- LeCun Uniform: Select weights from uniform distribution $U\left(-\sqrt{3/M}, \sqrt{3/M}\right)$

6 Convolution Neural Network

Recall the process of image classification can be broken down into two steps: feature extraction and classification. Using convolution neural network combines the two steps into one because the neural network will automatically extract useful features during training.

6.1 Convolution

In convolution Neural Network, an input has the shape (N, C, W, H) where N is the number of images, C is the number of channels, W and H are the width and height of the images.

In a convolution layer, we have to specify the kernel width and height F and also the number of kernels F. Each kernel is responsible for learning one feature from the image. We can also specify padding p and stride s.

The output of the convolution layer has shape (N, F, W_{out}, H_{out}) where

$$W_{out} = \frac{W_{\text{in}} - F + 2p}{s} + 1 \qquad H_{out} = \frac{H_{\text{in}} - F + 2p}{s} + 1$$

6.2 Pooling

Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. It is also useful for extracting dominant features which are rotational and positional invariant.

In a pooling layer, given input of shape (N, C, W, H), we specify the size of the kernel f and the stride s. The output has shape (N, C, W_{out}, H_{out}) where

$$W_{out} = \frac{W_{\text{in}} - F}{S} + 1 \qquad H_{out} = \frac{H_{\text{in}} - F}{S} + 1$$

Usually we use kernel size 2 with stride 2 and perform max or average operation.

6.3 Fully Connected Layer

Typical CNNs perform alternate Convolution layer and pooling layer on an image. Once the image is small enough, it is connected to a fully-connected layer.

Given the output of a pooling layer has the shape (N, C, W, H), the fully connected layer flattens all the weights into a layer with $C \times W \times H$ neurons.

7 Image Segmentation

Image segmentation helps us to identify the location of a single object in the given image. There are several effects in imaging that may hamper the application of segmentation algorithms. Some of these challenges include

- Noise
- Partial Volume Effect
- Anisotropic Resolution
- Imaging Artifects

7.1 Evaluating Image Segmentation

We can evaluate image segmentation by comparing the result to a **Gold Standard** defined by experts. We can measure the performance using different metrics, these metrics are defined in terms of agreement between prediction and the gold standard:

- True Positive: Areas accepted by prediction and gold standard
- False Positive: Areas accepted by prediction but rejected by gold standard
- True Negative: Areas rejected by prediction and gold standard
- False Negative: Areas rejected by prediction but accepted by gold standard

7.2 Metrics

Accuracy

$$\frac{TP+TN}{TP+TN+FP+FN}$$

Precision

$$\frac{TP}{TP+FP}$$

Recall

$$\frac{TP}{TP+FN}$$

7.3 Overlap

Jaccard Index. The intersection is comprised of the pixels found in both the prediction mask and the ground truth mask. The union is simply comprised of all pixels found in both target and prediction mask.

$$\frac{|Target \cap Prediction|}{|Target \cup Prediction|}$$

Dice Similarity Coefficient. This is most widely used for evaluating segmentation.

$$\frac{2|Target \cap Prediction|}{|Target| \cup |Prediction|}$$

7.4 Segmentation Algorithms

7.4.1 Thresholding

The simplest form of segmentation is just based on a single cutoff in the intensities, with a manually specified threshold. Alternatively, we can specify an upper and a lower threshold to select pixels with specific intensity range.

Image noise causes major problems for thresholding approaches. Typically we remove noise using Gaussian filters before thresholding the image.

7.4.2 Region Growing

Region Growing is a Breadth First Search algorithm. It starts from a user defined seed point and grow a region according to an intensity threshold. In each iteration, it checks the intensity of neighbouring pixels around the border and add the pixel as part of prediction if it is within the threshold.

7.4.3 Markov Random Fields

A Markov Random Field (MRF) is a graphical model of a joint probability distribution. In image segmentation, the edges in the MRF graph will connect geometrically close pixels or edges. Each node x_i is the segment identifier (0 or 1). Additionally, each x_i is connected to the observed pixel value y_i.

The goal is to minimise a energy function

$$E(\boldsymbol{x}) = \sum_i \psi_i(x_i) + \lambda \sum_{ij} \psi_{ij}(x_i, x_j)$$

Where

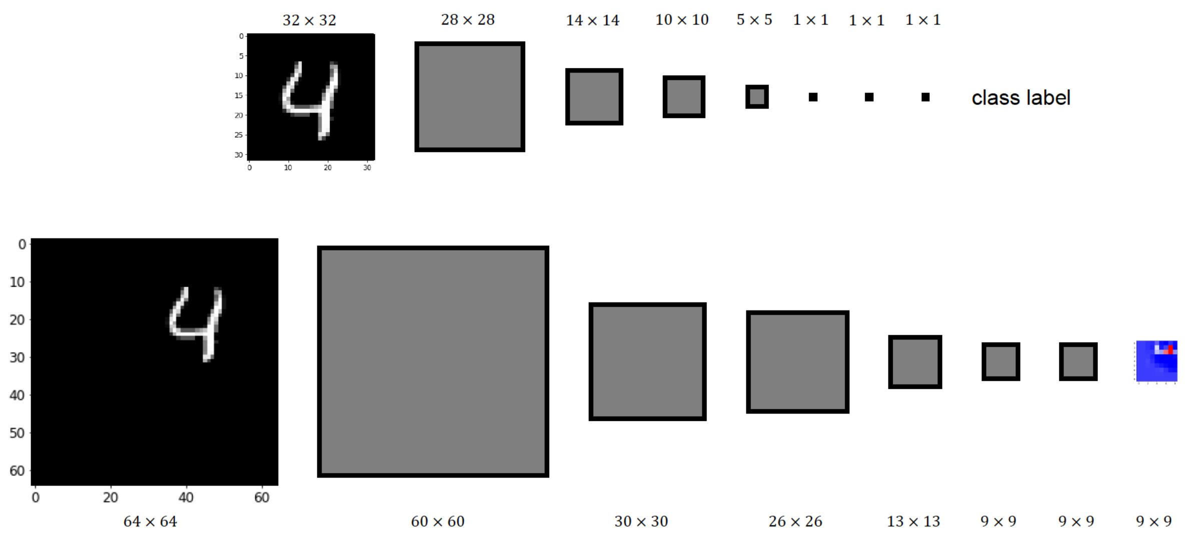
$$\psi_i(0) = -\ln Pr(y_i \mid Background)$$
$$\psi_i(1) = -\ln Pr(y_i \mid Object)$$
$$\psi_{ij}(x_i, x_j) = \begin{cases} 0, & x_i = x_j \\ e^{-\frac{(y_i - y_j)^2}{2\sigma^2}}, & otherwise \end{cases}$$

ψ_{ij} is a penalty for assigning neighbouring pixels with different class. If x_i and x_j are assigned different classes, then their respective observable pixel values y_i and y_j should be significantly different.

7.5 Fully Convolution Network

In classification, conventionally, an input image is downsized and goes through the convolution layers and fully connected (FC) layers, and output one predicted label for the input image.

If we replace all fully connected layers with 1x1 convolution layers, and feed in a larger image as input. The output will no longer be a single label, instead it will be a smaller image where each pixel indicates the presence of the detected object.



If we upsample the output, then we can calculate the pixel-wise output (label map).

8 Object Detection

The difference between object detection algorithms and classification algorithms is that in detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image.

Therefore the neural network has to return two outputs: a label for classification and bounding box coordinates (x, y, w, h) of where it thinks the output is. The loss function is compose of a softmax loss for classification and a L2 loss for the box coordinates.

The major reason why you cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that, the length of the output layer is variable. There can be any number of objects in the image, and the neural network will not know how many bounding box to return.

8.1 Region-based CNN (R-CNN)

R-CNN bypass the problem by using **selective search** to extract just 2000 regions from the image called **region proposals**. Specifically, R-CNN are composed of four main parts:

1. Selective search is performed on the input image to select multiple high-quality proposed regions. These proposed regions are generally selected on multiple scales and have different shapes and sizes. The category and ground-truth bounding box of each proposed region is labeled.

2. Once the proposals are created, R-CNN warps the region to a standard square size and passes it through to a modified version of AlexNet.

3. The AlexNet acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal.

4. The features and labeled bounding box of each proposed region are combined as an example to train a linear regression model for ground-truth bounding box prediction.

8.2 Fast R-CNN

The main performance bottleneck of an R-CNN model is the need to independently extract features for each proposed region. As these regions have a high degree of overlap, independent feature extraction results in a high volume of repetitive computations.

Fast R-CNN uses **Region of Interest Pooling**, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. The CNN features for each region are obtained by selecting a corresponding region from the CNN's feature map. Then, the features in each region are max-pooled.

Whereas R-CNN uses different models to extract image features (CNN), classify (SVM), and tighten bounding boxes (regressor), Fast R-CNN replaced the SVM classifier with a softmax layer on top of the CNN to output a classification. It also added a linear regression layer parallel to the softmax layer to output bounding box coordinates. In this way, all the outputs needed came from one single network.

8.3 Faster R-CNN

Both of the above algorithms(R-CNN and Fast R-CNN) uses selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. In Faster R-CNN, instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a RoI pooling layer.

8.4 You Only Look Once (YOLO)

YOLO or You Only Look Once is an object detection algorithm much different from the region based algorithms. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

How YOLO works is that we take an image and split it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network outputs a class probability and offset values for the bounding box. The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.