50.021 -AI

Alex

Week 11: Generative Adversarial Networks

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Fast Detection - Yolo and Yolo9000

Hi Students, often one wants a detection of an object in the image. That is a set of boundingboxes like:

Image credit: the Yolo paper

The goal of this lecture is not to give you material for memorizing. The goal is:

- get to know one real-time object detector
- understand that detection outputs bounding boxes, and what are important parameters of such an output box - its coordinates x, y, h, w, and the probability $p(c)$ of an object class c
- understand the IoU measure

Yolo `https://arxiv.org/abs/1506.02640` and Yolo9000 `https://arxiv.org/abs/1612.08242`, `https://pjreddie.com/darknet/yolo/` offer 40 fps+ detection results.[1]

[1] I find that more valuable for students soon to be in worklife than successes in Go which require 200+ GPUs.

How does that work roughly? Design how?

First is to think what should be the input and output - at training and at test time!

The desired output should be a set of bounding boxes.

One bounding box is a set of coordinates, can be for examples (x, y, h, w) - center of box in x-dimension (width), center of box in y-dimension (height), height, width.

Additionally one wants a class label, so one can output a vector of probabilities for K classes, as a softmax would output it $(p_{car}, p_{mouse}, p_{flower})$.

Then a prediction for one boundingbox would be (x, y, h, w, p_c) where p_c is the softmax vector of class probabilities.

A prediction for one boundingbox would be (x, y, h, w, p_c) where p_c is the softmax vector of class probabilities. Yolo adds a sixth element: a composite score C composed of:

- a confidence value $p(Object)$ that the box contains a real object from any of the classes.
- multiplied by a prediction of how good the box covers an object, if there is one.

How good a box covers an object is measured in detection research usually by **Intersection-Over-Union** measure (IoU)

$$IoU = \frac{area(\text{predicted box} \cap \text{ ground truth box})}{area(\text{predicted box} \cup \text{ ground truth box})}$$

So composite score C is a product of some object probability estimate and IoU.

Why is that C useful?

- At test time: predicting C gives you a measure of quality - how good this box detects something. Low $C \approx 0$ means: either low probability of object or low IoU measure. Both indicates low usefulness of the outputted box.
- C can be used as a helper to deal with an output which has a fixed number of bounding boxes. What happens if we always output a fixed large number of boxes, say 100 boxes per image, but there are only few objects in the image? Surplus boxes can simply predict $C \approx 0$. When choosing what detection to show to the user, simply throw away boxes with C below some threshold. But: a fixed number of boxes means: can output a vector of fixed dimensionality as the final layer - that can help to design a simple network, we will see that soon.
- at training time you can enforce $C \approx 0$, If you have a box over no object, or with very low IoU.

Why Yolo can be fast?

Why object detection was slow before? State of the art before Yolo was working like that:

- for an image create a large number, say $N = 500$ candidate bounding boxes using some heuristic
- run an image classifier over every of the 500 candidate bounding boxes to detect object classes or background - makes sense, right?
- if prediction is above a threshold, then take it and refine the location of the candidate bounding boxes.

Running an image classifier many times is slow. Refining, too.

Yolo: Use a neural network that outputs a fixed number of F boxes $(x, y, h, w, (\mathbf{p}_c), C)$, no matter how many objects are in the image. So output is simply a layer with $F \cdot (5 + |\#(\mathbf{p}_c)|)$ neurons.

Important here: you run a neural network once to output some vector of fixed dimensionality - compared to running a classifier 500 times, in yolo all the lower layers with their convolutional kernels are run just once - fast!

As said before, the problem that we often predict more boxes than there are objects in the image is solved by the output score C. The algorithm can learn to output low C for boxes that are surplus and not needed for any object. Low C indicates, that one should not use this predicted box (bcs either the probability of object or overlap to a real object is low).

Another trick to get a fixed number of boxes: Yolo divides the image into a $SxS = 7x7$ grid, and predicts $B = 2$ boxes for every grid cell. (x, y) are coordinates of the middle point of the bounding box relative to the grid cell boundaries, and therefore in $(0, 1)$ – bounded values are better regression targets than unbounded values. Same: h, w lies in $(0, 1)$, $h = 1$ means that box height is the height of the whole image.

See page 3 in `https://arxiv.org/abs/1506.02640` for the network architecture.

Correction: Yolo predicts one vector of class probabilities $(\mathbf{p}_c)$ per grid cell, not per box

We need a Loss function for training: They formulate the detection output as one regression problem.

1_{ib}^{obj} is an indicator function which is 1 if box b of grid cell i is

responsible for a prediction, that is: the center of the object is in grid cell i and box b has the best IoU overlap among all boxes of grid cell i.

1_i^{obj} is an indicator function which is 1 if the center of the object is in grid cell i

$$
\begin{aligned}
L = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{b=0}^{B} 1_{ib}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{b=0}^{B} 1_{ib}^{obj} [(\sqrt{h_i} - \sqrt{\hat{h}_i})^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2] \\
+ \sum_{i=0}^{S^2} \sum_{b=0}^{B} 1_{ib}^{obj} (C_i - \hat{C}_i)^2 \\
+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{b=0}^{B} 1_{ib}^{noobj} (C_i - \hat{C}_i)^2 \\
+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

Two more tricks in here:

- square-root on height and width - a heuristic to make height and width errors in smaller boxes get higher weights: an error of 10 pixels is more severe in a box of 10 pixel height, than in a box of 1000 pixel height!
- λ weights for boxes without objects (low) and for box localization coordinates (set to high values) - puts more weight on getting localization errors low, puts less weight on learning low C for boxes without object - the latter improves training stability.

The main trick of Yolo is:

- output a fixed number of boxes.
- Measure box quality by the outputted confidence measure C. Not needed boxes should predict $C \approx 0$
- At test time do not show those boxes with too low C
- prediction is fast, because one uses one forward pass of one network that outputs all the needed parameters for a fixed large number of boxes.
- web has code to run it with your webcam

one drawback of Yolo: it may fail to detect small objects, and dense groups of Objects.

Yolo 9000 – several improvements. A few are given below:

- pretrain a classification network, at first on 224×224, then on 448×448 inputs (higher resolution) on imagenet, to make it fit for higher resolutions
- use a 13×13 grid instead of 7×7
- predict bounding box center positions b_x, b_y relative to grid cell coordinates c_x, c_y as before: $b_x = \sigma(t_x) + c_x$
- for bounding box sizes: learn good starting points: Cluster bounding box sizes to get $k = 5$ centroids p_w, p_h (mean-vectors as output from k-means). Each grid cell predicts now k boxes - one for every size centroid
- each of the k box sizes is predicted as deviation from its centroid of k-means. If the centroid has size parameters p_w, p_h, then learn t_w to predict a size $b_w = p_w e^{t_w}$.