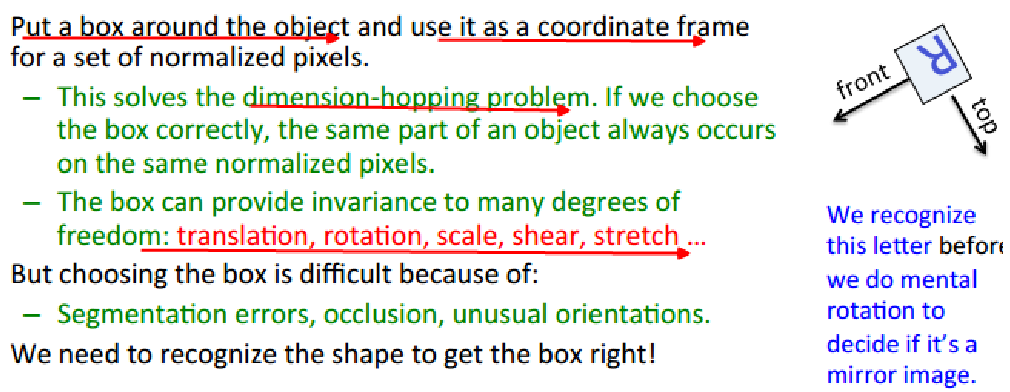
L5 - why object recognition is difficult

2017/11/08 22:27

- **why object recognition is difficult**
  - the machine process
    - to take a bunch of numbers that describe the intensities of pixels and go from there to the label of an object.
  - the difficulties
    - Segmentation: Real scenes are cluttered with other objects
      - which piece from which object?
      - hidden parts
    - Lighting
      - The intensities of the pixels are determined as much by the ligh:ng as by the objects.
      - so same obj at different light often has different intensities
    - Deformation
      - Objects can deform in a variety of non-affine ways
      - e.g. various handwritten 2
    - Affordances
      - Object classes are often defined by how they are used
      - chairs is sth to sit on/iin
    - viewpoint
      - Changes in viewpoint cause changes in images that standard learning methods cannot cope with.
  - the consequence
    - With so much uncertainties, it's gonna be very hard for any hand engineered program to be able to do a good job of those things.

- **Ways to achieve viewpoint invariance**
  - viewpoint variance
    - various viewpoints make the object show on different pixels in an image, which is unlike other ML problems

- Approach 1: The invariant feature approach
    - How
        - Extract a large, redundant set of features that are invariant under transformations
    - Why
        - With enough invariant features, there is only one way to assemble them into an object.
        - The relationships between features are captured by other features. With overlapping and redundant features, one feature will tell you how two other features are related.
    - ATT
        - But for recognition, we must avoid forming features from parts of different objects.
    - then the difficulty goes to how to find this big feature set!
- Approach 2: The judicious normalization approach
    - use a bounding box to set the coordinate of the object to avoid dimension-hopping problem.
        - but if the box is required to be added by the machine, it's a chicken-egg problem. Manual work is fine but expensive



    - variant: The brute force normaliza:on approach
        - When training the recognizer, use well-segmented, upright images to fit the correct box.
        - At test time try all possible boxes in a range of posi:ons and scales.
            - – This approach is widely used for detec:ng upright things like faces and house numbers in unsegmented images.
            - – It is much more efficient if the recognizer can cope with some variation in position and scale

so that we can use a coarse grid when trying all possible boxes.

- **Approach 3:** Convolutional neural networks for hand-written digit recognition
  - The replicated feature approach - CNN
    - apply the same feature detectrors with different position in an image to get one feature map
      - Could also replicate across scale and orientation
        - but it's tricky and expensive
      - Replica:on greatly reduces the number of free parameters to be learned.
        - the weights are shared
    - Use several different feature types to get several feature maps
      - Allows each patch of image to be represented in several ways.
  - Backpropagation with weight constraints to train the nets
    - We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints, wi = wj.

$$\textit{To constrain}: \quad w_1 = w_2$$
$$\textit{we need}: \quad \Delta w_1 = \Delta w_2$$

$$\textit{compute}: \quad \frac{\partial E}{\partial w_1} \quad \textit{and} \quad \frac{\partial E}{\partial w_2}$$

$$\textit{use} \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad \textit{for } w_1 \textit{ and } w_2$$

    - What does replicating the feature detectors achieve
      - it's equivariant translation not invariant translation
        - the position of the obj changes, the neural activities change accordingly.
      - **Invariant knowledge**: If a feature is useful in some locations during training, detectors for that feature will be available in all locations during testing.
    - Pooling the outputs of replicated feature detectors

- Get a small amount of translational invariance at each level to give a single output to the next level.
  - this reduce the #inputs to the next layer, thus allowing us to have many more different feature maps
- types
  - averaging four neighboring replicated detectors
  - max, works slightly better
- Problem
  - after several levels of pooling, we lost information about the precise position of the objects
    - this makes it impossible to use the precise spatial relationships between high-level parts for recognition
    - e.g. to detect the face works well, but to detect whose face may fail

- e.g. LeNet - handwritten digits recognizer
  - Key character
    - \- Many hidden layers
    - \- Many maps of replicated units in each layer.
    - \- Pooling of the outputs of nearby replicated units.

- - A wide net that can cope with several characters at once even if they overlap.
  - what's the theory behind it?
- - A clever way of training a complete system, not just a recognizer.
  - max margin

- Look the impressive demos of LENET at hHp://yann.lecun.com

▪ Priors and Prejudice - how to use the prior knowledge to train the NN

- one approach: use the knowledge to the design of NN, like LeNet
  - factors
    - local connectivity
    - weight sharing
    - activation functions
  - This is less intrusive than handdesigning the features.
    - - But it still prejudices the network towards the particular way of solving the problem that we had in mind.
- Alternatively, we can use our prior knowledge to

create a whole lot more training data

- it's intuitive. but it require a lot of work and long time to train
- and this can make a huge difference. Ciresan et. al. create a many kinds of synthesis digit data, train only a dumb huge net and achieve only with 35 errors, even better than the LeNet at the error aspect.

- How to detect a significant drop in the error rate?
  - errors are a single number, telling not much about the goodness
  - The McNemar test tells

	model 1 wrong	model 1 right
model 2 wrong	29	1
model 2 right	11	9959

	model 1 wrong	model 1 right
model 2 wrong	15	15
model 2 right	25	9945

- on the left, it's a 11:1 ratio while on the right, it's a 5:3 ratio. So clearly, on the left case, model2 is much better while in the left case, model 1 is only slightly better

- Convolu:onal neural networks for object recogni:on
  - can the experience learned from MNIST be transferred to the 3D objects in the real world?
    - Recognizing real objects in color photographs is much more complicated than recognizing hand-wriHen digits
      - \- Hundred times as many classes (1000 vs 10)
      - \- Hundred times as many pixels (256 x 256 color vs 28 x 28 gray)
      - \- Two dimensional image of three-dimensional

scene.

            - a lot of info has lost

        - \- Cluttered scenes requiring segmentation

            - overlapped, hidden...

            - the effect of light luminance

        - \- Multiple objects in each image.

    - Will the same type of convolu:onal neural network work?

        - training a lot of synthesis images is still a problem for the current PC [2013]

        - currently, no clear answer

- the competition

The ILSVRC-2012 competition on ImageNet

- The dataset has 1.2 million high-resolution training images.
- The classification task:
  - Get the "correct" class in your top 5 bets. There are 1000 classes.
- The localization task:
  - For each bet, put a box around the object. Your box must have at least 50% overlap with the correct box.
- Some of the best existing computer vision methods were tried on this dataset by leading computer vision groups from Oxford, INRIA, XRCE, …
  - Computer vision systems use complicated multi-stage systems.
  - The early stages are typically hand-tuned by optimizing a few parameters.

    - University of Toronto (Alex Krizhevsky) with only 16.4% error for the first bet

        - some tricks used

            - architecture

              - 7 hidden layers not counting some max pooling layers.
              - The early layers were convolutional.
              - The last two layers were globally connected.

            - The activation functions:

                - ReLU in every hidden layer.

                    - These train much faster and are

more expressive than logistic units.

- **Competitive normalization**
  - to suppress hidden activities when nearby units have stronger activities. This helps with variations in intensity.

- Data
  - Train on random 224x224 patches from the 256x256 images to get more data. Also use left-rightreflections of the images.
- Regularization
  - dropout

- This stops hidden units from relying too much on other hidden units.

- The hardware
  - this is also a key factor

- He uses a very efficient implementation of convolutional nets on two Nvidia GTX 580 Graphics Processor Units (over 1000 fast little cores)
  – GPUs are very good for matrix-matrix multiplies.
  – GPUs have very high bandwidth to memory.
  – This allows him to train the network in a week.
  – It also makes it quick to combine results from 10 patches at test time.
- We can spread a network over many cores if we can communicate the states fast enough.
- As cores get cheaper and datasets get bigger, big neural nets will improve faster than old-fashioned (*i.e.* pre Oct 2012) computer vision systems.

- A2
  - Key words in the description
    - data set
      - 4-grams
      - The training set consists of 372,550 4-grams.
        - 372550 X 4
        - Each entry is an integer that is the index of a word in the vocabulary.
      - The validation and test sets have 46,568 4-grams each.
    - operation
      - > load data.mat
      - > fieldnames(data)
        - 'data.vocab'
        - 'data.trainData', 'data.validData' and 'data.testData'

- > data.vocab
    - To see the list of words in the vocabulary
- split the data
    - >[train_x, train_t, valid_x, valid_t, test_x, test_t, vocab] = load_data(100);
        - This will load the data, separate it into inputs and target, and make mini-batches of size 100 for the training set.
- train
    - > model = train(1);
        - This will train the model for one epoch
        - The training method will output a 'model' (a struct containing weights, biases and a list of words).

- work
    - You have to fill in parts of the code in fprop.m and train.m.
    - Once the code is correctly filled-in, you will see that the cross entropy starts decreasing.
    - At this point, try changing the hyperparameters (number of epochs, number of hidden units, learning rates, momentum, etc) and see what effect that has

on the training and validation cross entropy.