CSE 659A: Advances in Computer Vision

Spring 2019: T-R: 2:30-4pm @ Cupples II/230

Instructor: Ayan Chakrabarti (ayan@wustl.edu).

http://www.cse.wustl.edu/~ayan/courses/cse659a/

Feb 26, 2019
• Project 1 Proposal Feedback posted.
• Most proposals were fine.
  ▪ Comments might include some caveats or suggestions in some cases.
  ▪ Look at them ASAP.
• Label each pixel: based on semantic class, but with a separate label for each instance of that class.
• Last time: Instance Segmentation by Detection
  ▪ Take proposal bounding boxes: where each box is expected to contain one instance
  ▪ Do a 'foreground' / background labeling, where foreground is of dominant object in the window.
  ▪ Also produce a class label for this 'foreground'.
Let’s say you had an image region with pixels with one of two labels (instances). Consider each of the two sets (all pixels with the same label).

Distance Transform = Label each pixel based on the distance (in pixels) from the boundary of it’s own set. Compute it as number of horizontal+vertical steps of shortest path to boundary.

If we had this distance transform map, we could consider the set of all pixels with distance transform value > 1, and find connected components as separate instances.
INSTANCE SEGMENTATION

If we had this distance transform map, we could consider the set of all pixels with distance transform value > 1, and find connected components as separate instances.

Let's say you had an image region with pixels with one of two labels (instances). Consider each of the two sets (all pixels with the same label).

Distance Transform = Label each pixel based on the distance (in pixels) from the boundary of it’s own set. Compute it as number of horizontal+vertical steps of shortest path to boundary.
INSTANCE SEGMENTATION

Initial Regular Segmentation Map

Distance Transform Map

Threshold + Connected Components

But even though distance transform values are per-pixel, computing them requires ‘global’ reasoning.

Think of the distance transform as a negative height map.

Now, let's say you can't estimate the distance transform itself directly, but you can estimate gradients of the distance transform—because these will often be based on local edge information.

Watershed Transform: Choose points of local minima (say based on regions with minimum gradient magnitude), and start ‘flooding’ this region from those minima. Boundaries are pixels where you need to build ‘dams’ to separate different flood basins.
Think of the distance transform as a negative height map.

Now, let's say you can't estimate the distance transform itself directly, but you can estimate gradients of the distance transform—because these will often be based on local edge information.

Watershed Transform: Choose points of local minima (say based on regions with minimum gradient magnitude), and start 'flooding' this region from those minima. Boundaries are pixels where you need to build 'dams' to separate different flood basins.
Traditional Methods used boundary detectors as estimates of 'watershed gradient', and then simulated this flooding (through a breadth-first search + priority queue based on gradient magnitude).

Bai and Urtasun: Have a direction network which first predicts "direction" of distance transform gradient. Second network then takes this as input and produces distance transform map. Learn both end-to-end, with supervision for both.
INSTANCE SEGMENTATION
INSTANCE SEGMENTATION

(a) Input Image  
(b) GT angle of $\bar{u}_p$  
(c) GT Watershed Energy  
(d) GT Instances  
(e) Sem. Segmentation of [34]  
(f) Pred. angle of $\bar{u}_p$  
(g) Pred. Watershed Transform  
(h) Pred. Instances
So far, made semantic predictions as one of N pre-determined classes.
Implicit assumption is that every image (pixel) we encounter will be one of these classes, and we have to predict which one.
But that's a limiting view of explaining semantic content in images.
- People take photographs of all kinds of things. (Check your facebook feed!)
- And we want to be able to extract general semantic meaning, not just for a pre-determined set of classes.
A group of people shopping at an outdoor market.

- Will train this on a large dataset of images and associated captions.
- But how do we express the output? What kind of architecture do we use?
A group of people shopping at an outdoor market.

Another Kind of Classification?

- We can't consider the set of all possible sentences and assign a different label to each!

But we can build a vocabulary of words, and assign a different 'id' to each.

0=A, 1=Aardvark, 2=An, 3=And, …

- We will represent a word $S_i$ by a D-dimensional one-hot vector.
  - D is the size of our vocabulary
  - Each entry of $S_i$ 0, except at the index corresponding to the correct word.
    E.g., $S_i = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, …]$ for "And".
Vinyals et al., Show and Tell: A Neural Image Caption Generator, CVPR 2015.

A group of people shopping at an outdoor market.

We want to produce a sentence, which is a sequence of words.

\[ S = \{ S_0, S_1, S_2, S_3, \ldots, S_T \} \]

- We want our neural network to look at an image, and produce a probability distribution \( p(S) \) over all possible sentences.
- But a generic joint distribution is too high-dimensional.

\[
p(S) = p(S_0) \times p(S_1 | S_0) \times p(S_2 | S_0, S_1) \times p(S_3 | S_0, S_1, S_2) \ldots \times p(S_T | S_0, S_1, \ldots, S_{T-1})
\]

Standard factorization in terms of conditionals, starting from the first word, and conditioning each word on all words that came before.

Note that the model is NOT pairwise. \( S_i \) depends on all words \( < i \), not just \( i - 1 \).
We will use a neural network that produces $p(S_i | \{S_0, \ldots S_{i-1}\})$. This is a D-dimensional distribution over the vocabulary.

$$p_i = f(X, S_0, S_1, S_2, \ldots S_{i-1})$$

Where $X$ is the input image, and the other inputs are all the words that came before.

But this function will have different number of inputs for different location $i$ in the sentence.

$$p_i, m_i = f(S_{i-1}, m_{i-1})$$

- The network outputs $p_i$ which is the desired distribution, but also a vector $m_i$ for internal use in subsequent calls to $f$.

- $m_i$ is an auxiliary 'hidden' state or memory vector of some chosen dimensionality, in which the network $f$ learns to summarize information from all previous inputs.
  - There is no "ground-truth" value for $m_i$. It is an internal representation learned based on losses defined on $p_i$'s alone.

- Note that the output $p_i$ is a distribution over words. But you will "choose" a specific word $S_i$ from that distribution. It is this specific chosen word that will be the input to $f$ for the next time step.
We are learning a common function $f$ for all time-steps.

- We introduce two special words, "Start" and "Stop", to our vocabulary to denote the beginning and end of our sentence.
- At time step 0, the input $m_{-1}$ is set to all zeros. The input $S_{-1}$ is a feature vector of the image itself.
- At time step 1, the input $S_0$ is the one-hot embedding of the start word.

For image features, we use features extracted from the final (and possibly other) layer of a regular classification CNN architecture (often pre-trained on ImageNet).

But this feature vector may not be the same size as the vocabulary.

So instead, we have for $x_i \in \mathbb{R}^F$,

$$p_i, m_i = f(x_{i-1}, m_{i-1})$$

- $F$ is the dimensionality of the image feature vector.
  - When the input $x$ corresponds to the image, it is the feature vector itself.
  - When we are providing a word, we set $x_i = W e S_i$, where $W e$ is a learned $F \times D$ matrix.
The LSTM here is our function $f$ (more on that later).

Training

- For each training example, we create as many blocks of $f$ as necessary based on length of sentence.
- The word inputs to the LSTM (from below) are the "true" words of the sentence.
- For each $p_i$ produced for each word, we have a cross-entropy loss on the true word at that location in the sentence.
- This includes a loss for the "stop" word at the end.
Training

So the main difference from "regular" training is:

- We just have a bunch of repeating layers that share weights.
- The total number of layers varies from training example to example.
So repeated application of our function $f(\cdot)$ corresponds to a recurrent neural network.

But RNNs typically have a problem with vanishing / exploding gradients.

**Long Short Term Memory (LSTM)**

- A special kind of recurrent neural network, designed to address this.
- Essentially, it propagates information by multiplication with "gating" values that are also computed based on the input.
- The gating functions are set to saturate between $[0, 1]$ or $[-1, 1]$.
- Prevents intermediate values and gradients from exploding/vanishing.
Long Short Term Memory (LSTM)

\[
\begin{align*}
    i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1}) \\
    f_t &= \sigma(W_{fx}x_t + W_{fm}m_{t-1}) \\
    o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1}) \\
    c_t &= f_t \circ c_{t-1} + i_t \circ h(W_{cx}x_t + W_{cm}m_{t-1}) \\
    m_t &= o_t \circ c_t \\
    p_{t+1} &= \text{Softmax}(m_t)
\end{align*}
\]

- \(i_t, f_t, o_t\) are "gating" values. \(\sigma(\cdot)\) is sigmoid.
- Interpreted as how much to consider the input, whether to forget previous state, and how much to weight the output.
- Two hidden states \(c_t, m_t\) propagated to future time-steps. \(h(\cdot)\) is tanh.
- There's an interpretation of what the network is doing, but the main justification for using LSTMs is empirical: it works well!
- More recent work on other RNN architectures (Gated Recurrent Units) that work just as well.
So this is what we used for training, when we knew what the ground truth sequence.

But what do we do for inference? Fundamentally, we have a way of getting a score for a possible sequence, but how do we score all possible sequences?

Our search space is $D^L$, where $L$ is the maximum possible length. Since our network $f$ takes in a specific word from the previous time-step as input, we would need to run it over all possible sequences to get scores.
Inference

Greedy Approach:

- At each time step $i$, select the best possible word:

$$p_i, m_i = f(S_{i-1}, m_{i-1}), \quad S_i = \arg \max p_i$$

- But this doesn't guarantee that our sequence has the highest score.

$$p_1(A) > p_1(B), \quad \text{but} \quad p_1(A) \times \max p_2(\cdot|A) < p_2(B) \times \max p_2(\cdot|B)$$

Exhaustive Search

- At time step 0, generate all possible words, and record their scores.
- At time step $i$, feed all possible partial sentences to the model (those without a STOP), and construct expanded sentences with $D$ possible added words for each, keeping track of the change in score.
- When all sentences have a "STOP" at the end, look at all scores and select the best.

Number of sentences increases by upto a factor $D$ at each time-step!
Inference

*Beam Search Decoding:*

- At time step 0, keep the $k$ most likely words (and record their scores).
- At time step $i$, run all $k$ sentences through the model. Consider all the $D \times k$ sentence fragments, and keep only the top $k$.

This set might include the best word added to each fraction from step $i - 1$.

But you might also choose to keep multiple possible word additions to the same fragment, and drop another fragment completely (called, "falling off the beam").

- When you reach a time step where all $k$ fragments end in "STOP", pick the best one.
**LANGUAGE + VISION**

- Image Captioning: Just the first task at this intersection!
  
  Input: Image, Output: Free-form Text

- Visual Question Answering (visualqa.org)
  
  Input: Image + Free-form Text, Output: Free-form text (or single-word/yes-no answers)

- Use an LSTM to first process the input question: each cell only outputs $m_t$, not $p_t$ (no generation).
- Take the memory vector $m_t$ at the end, concatenate with image features from a CNN.
- Send this through a "decoding" LSTM to generate natural language output.