UNIVERSITY OF OSLO

Candidate no

Faculty of mathematics and natural sciences

Deferred constituent exam in: TEK5040 — Deep Learning for Autonomous Systems

Day of examination: MOCK EXAM

This problem set consists of 6 pages.

Appendices: None.

Permitted aids: None.

Please make sure that your copy of the problem set is complete before you attempt to answer anything.

Contents

Problem 1 Tensorflow (weight 5%)

```
12
20
```

Remember that the feeded values holds for that exact run.

(Continued on page 2.)

Problem 2 Data efficiency (weight 5%)

The most common method is transfer-learning/fine-tuning, where you first train on a larger dataset, for a similar task. Then you reuse most of the weights for you target task/data. Other methods could be to use, multi-task learning where you add extra labels that guide your training or surrogate losses without extra labels that helps the training process.

Problem 3 Self-supervised learning (weight 8%)

Building assumptions into your loss-function is a way to incorporate knowledge into your model. You limit the configurations of your network, into one that follow those assumptions. If your assumptions are strict enough, you may only be left with configurations that makes your model work. In the vid2depth network, they built into the loss the assumption that edges should occur at the same positions for the input image and the output depth estimation. By it self, this assumption is not enough strong enough to make depth estimation work, but it does rule out a lot of bad networks. They also assumed the image was taken of an overlapping region of a 3D-world, where everything was static, and there were no color or light changes. With these assumptions incorporated in the loss function, the only way to get a low loss was to explain the variance in terms of depth/geometry of the world and self-movement. Even though these assumptions was not absolutely true, they worked often enough for the model to learn something about the world.

Similarly it has been shown that a model can learn object tracking, simply by building in the assumption that objects don't change color.

To get full score:

- You can build assumptions into loss functions
- examples of assumptions: color constancy, fixed 3d world, aligned edges

Problem 4 Tracking (weight 8%)

4a

If you base your network on an existing object detection network, the best solution might be to use an approach similar to the MDNet. In you training process you have make a separate last layer for each training video and object, and share the rest of the for all examples. You sample many random crops around your expected target location, run them through the network that classifies them as target or not target. A crop is counted as target if it overlaps above a certain threshold with the true target. You train this network based on a training-set of videos, where a

(Continued on page 3.)

specific object is marked with a bounding box in each video. In the training process you train the whole network, both the specific last layer and the shared base network.

When running the tracking network, you keep the shared part of the network fixed, but create a new last layer that you train based on the initial frame and random crops, similar to the training stage.

To get full score: Should give a rough overview of a tracking network, based on one of the three methods from the course.

4b

Perhaps the most typical problem with a tracking network is to confuse similar objects, say mixing one person with another. This could either be caused by the objects being the same object in the pretrained network, if transfer learning was used, or that you have few objects of the same type in each video frame. This could be solved by specifically selecting similar objects as object/non-object when training, or using distraction-aware inference.

Other typical problems:

- Tracker only using a few features of an object
  - Alleviated by e.g. regularized attention
- Tracker diverging over longer time periods
  - Stopping learning of object features early
  - Using memory network architectures

Problem 5 3D segmentation (weight 5%)

- Dense data like MRI, CT etc.
- Complex shapes where it is hard to get good view to capture all the data e.g. large 3D maps of building
- Spare or noisy data where it's hard to find views that groups related data in a systematic way.

Problem 6 Optimization (weight 5%)

If N is the number of weights, the number of second derivatives are N^2 and may become prohibitively expensive to compute and store for a large number of parameters. (There exists approximate second order methods though that have a somewhat wider applicability).

(Continued on page 4.)

Problem 7 Batch normalization (weight 5%)

We usually use stored running averages created during training, or use statistics based on the whole training set.

Problem 8 Bidirectional RNNs (weight 4%)

Be able to take into account context from the "future".

Problem 9 External memory (weight 8%)

1. Normalize the vector using e.g. softmax function to get probabilities p_1, p_2 and p_3.
2. - In the case of *hard* addressing, draw one of the three memory with probabilities p_1, p_2 and p_3 for M_1, M_2 and M_3 respectively.
   - In the case of *soft* addressing, take a weighted mean of the memory cells, i.e. $r = p_1M_1 + p_2M_2 + p_3M_3$.

Problem 10 RL value functions (weight 5%)

We have

$$\begin{aligned} v_\pi(s) &= \pi(a_1|s)q_\pi(s,a_1) + \pi(a_2|s)q_\pi(s,a_2) \\ &= 0.2 * (-10) + 0.8 * 10 \\ &= (-2) + 8 \\ &= 6 \end{aligned}$$

Problem 11 RL policy gradient (weight 7%)

$\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the direction for which the probability, or probability density value, of taking a_t from state s_t increases the most. Thus if the return G_t is positive, this term gives a contribution in a direction that increases the probability of the chosen action. If we get a negative return, we try to decrease the probability of action a_t by going in the opposite direction. Intuitively, if we see a good outcome we increase the probability of the action, if see a bad outcome, we decrease it.

Problem 12 Word embedding (weight 5%)

Let the one-hot encoding vector of the considered input word be $\boldsymbol{x} \in \mathbb{R}^{V\times 1}$ and the word vector $\boldsymbol{h} \in \mathbb{R}^{d\times 1}$. Then $\boldsymbol{h} = \boldsymbol{W}^T\boldsymbol{x}$ and $\boldsymbol{z} = \boldsymbol{U}\boldsymbol{h}$. If the

(Continued on page 5.)

y^{th} element of $\boldsymbol{z}$ is $z(y)$, then $P(w_o = y) = \frac{\exp(z(y))}{\sum_{y=1}^{V} \exp(z(y))}$, where w_o is the considered output word.
The i^{th} row of the matrix $\boldsymbol{W}$ is the word vector of the i^{th} word in the dictionary.

Problem 13 Loss functions (weight 5%)

Negative sample loss (NSL) is equivalent to Noice contrastive estimation (NCE), when the distribution the noise samples are drawn from is uniform such that $kP_n(y) = 1$.

Problem 14 Bayesian deep learning (weight 5%)

- Maximum Likelihood (ML)

$$\widehat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} p\left(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}\right)$$

- Maximum A-Posteriori (MAP)

$$\widehat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} p\left(\boldsymbol{Y}, \boldsymbol{w}|\boldsymbol{X}\right) = \arg\max_{\boldsymbol{w}} p\left(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}\right) p(\boldsymbol{w})$$

- Bayesian

$$p\left(\boldsymbol{w}|\boldsymbol{Y}, \boldsymbol{X}\right) = \frac{p\left(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}\right) p\left(\boldsymbol{w}\right)}{P\left(\boldsymbol{Y}|\boldsymbol{X}\right)} = \frac{p\left(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}\right) p\left(\boldsymbol{w}\right)}{\int P\left(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}\right) p\left(\boldsymbol{w}\right) d\boldsymbol{w}}$$

Problem 15 Evidence lower bound (weight 5%)

The first term encourages data to be explained correctly with model parameters drawn from the auxiliary distribution. The second term requires that the distance between the auxiliary distribution and the posterior distribution would be minimized.

Problem 16 Guided cost learning (weight 5%)

With the demonstration trajectories, we back-propagate $\frac{1}{N}$ and with system generated trajectories we back-propagate $-\frac{w_j}{Z}$.

Basic steps in GCL:

1. Initialize cost function c_θ (with random or otherwise selected values)
2. Initialize policy (q_0)

(Continued on page 6.)

3. Gather expert demonstrations ($\mathcal{D}_{\text{demo}}$)
4. Run policy and gather trajectories ($\mathcal{D}_{\text{samp}}$)
5. Feed expert demonstration through the cost function and back-propagate $\frac{1}{N}$
6. Feed system trajectories through the cost function and back-propagate $-\frac{w_j}{Z}$
7. Update cost function
8. Optimize policy (Reinforcement learning) with the updated cost function
9. repeat from step 4.

Problem 17 Dialog systems (weight 5%)

Issues addressed with reinforcement learning:

- Maximum Likelihood criterion for predicting the next dialog turn will lead to poor dialogs (dull responses, repetition etc.)
- Non differentiable evaluation metrics relevant to longer term goals of the dialog.

Reinforcement learning applied to dialog system training:

- Agent: The Recurrent Net
- State: Previous dialog turns
- Action: Next dialog utterance
- Policy: Generate the next dialog utterance (action) given the previous dialog turns (state)
- Reward: Score computed based on relevant factors such as ease of answering, information flow, semantic coherence etc.

Variance can be reduced by introducing a baseline, which must be independent of w^s. Modified equation is

$$\nabla_\theta L(\theta) = (r(\boldsymbol{w}^s) - b)\nabla_\theta \log p_\theta(\boldsymbol{w}^s)$$

Problem 18 Sequence modeling (weight 5%)

Recurrent network: $O(n)$
Convolution network with contiguous kernels $O(n/k)$
Self attention network $O(1)$