Deep Learning Homework 3

Björn Pettersson, bjpe900 2025-81022

01-12-2025

Problem 1 : Continual Learning

How can we learn a sequence of tasks or data streams incrementally over time, without forgetting what they have learned previously?

1.1.a)

Class incremental learning The model receives new classes over time. Each training phase introduces additional labels that were not present before. The model must learn to classify both old and new classes using a single shared classifier, without forgetting the earlier ones.

Domain incremental learning

The set of classes stays the same, but the input distribution changes over time (e.g., new environments, styles, sensors). The model must adapt to the new domain while keeping performance on the old domains, even though the labels themselves do not change.

1.1.b)

The 3 big categories of conntinual learning is:

1. **-Replay Methods:** They store and replay samples from previous tasks (or generate pseudo-samples) alongside new data to mitigate forgetting.
2. **-Regularization based methods:** They add regularization terms to the loss function to preserve important parameters learned from previous tasks. These constraints prevent large updates that could destroy old knowledge.
3. **-Paramter isolation methods:** They isolate model parameters for each task, either by using a fixed network with masks or dynamically growing the architecture, thereby avoiding interference between tasks.

1.2. Joint Training vs Offline CL vs Online CL (10 pts)

1.2.a)

Joint Training

We combine the data for learning all tasks into one dataset and train on this, Catastrophic forgetting is inherently avoided because the model repeatedly sees data from all tasks. This serves as an upper-bound on performance for the other two paradigms.

Offline continual learning (taks incremental)

We sequentially train the model on tasks, ie first task A on dataset A, then task B on dataset B, etc. The model only sees the dataset for the current task during trainig but we train for multiple epochs ie the model sees a given example multiple times. When learning new things the model overwrites the weight values it had from learning previous tasks, leading to catastrophic forgetting. Problem is stability vs plasticity, how much we want to retain ability to perform old tasks, while keeping ability to learn new.

Online continual learning ()

The examples from the tasks are presented like a stream, each example is only shown once. The model trains on a single pass, like a,a,a,a,a,b,b,b,b,b,c,c,c,d,d,d,d,d,d for task from dataset A,B,C,D. This limits the time and data avialable to learn a task since we only see it once, but its closer to how humans learn.

1.2.b)

Most difficult is online continual learning

We still have the challenge of catastrophic forgetting of the previous tasks but we also have limited data for our current task risking underfitting. So in the stability plasticity tradeoff, we have problems with both, where as for offline CL we could at leaset get optimal prameter updates for each task since we had muliple passes to learn them.

Problem 2 : Chain-of-Thought (CoT) Reasoning (20 points)

2.1

One shot prompt:

“

Q: A banker has 20 coins. He puts 5 coins into each of his 3 accounts. How many cointe does he have left

A: We think about the problem step by step, i) the banker puts 5*3=15 coints into accounts. ii) he has 20-15=5 coins left. iii) Answer, the banker has 5 coins left.

Q: A baker has 20 cookies. He packs 4 cookies into each of his 3 boxes. How many cookies does he have left?

A: Lets think about this step by step

“

My one shot prompt, uses the step by step prompt reasoning helping the model see an example of how we want the prolbme broken down and reasoned about, ie how we want to arrive at a given answer. With zero shot, the model may attempt to just output the most likely number, while the reaoning steps here makes the answer more liely to be aligned with the type of reasoning we want.

2.2

Zero shot with CoT:

“

Q:. A baker has 20 cookies. He packs 4 cookies into each of his 3 boxes. How many cookies does he have left ? A. Lets think about this step by step:

“

I added “think step by step” to encurage CoT behaviour, since its zero shot we dont show examples. But thsi line has been proven to make the model reaon and effective for improiving performance. There is no evidence of how or why exactly this works.

2.3.a)

One shot CoT triggers reasoning by providing the model with an example of how to solve a task, which improves generally performance. Another advantage is that we can choose the type of reasoning here after our own example.

2.3.b)

I expect one shot to outperform zero shot, because there are less things that can go wrong thorugh halluciantion since i am providing the model with the reasoning steps it needs to solve this type of taks, while for zero shot it has to make them up even though we can trigger CoT behaviour by the “lets think step by step” prompt.

Problem 3 : Diffusion Models (DDPM vs. DDIM) (20 Points)

3.1.a)

Markov definiton of th eforward process

Independence of guassian noice

Show distribution of the xt conditioned on x0 can e writeen in clsoed form as:

$$q(x_t+ \mid x_0) = N(x_t; \sqrt{\bar{a}_t}, x_0, (1 - \bar{a}_t)I),$$

$$\bar{a}_t = \prod_{s=1}^{t}(1 - \beta_s)$$

We want to derive: $x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1 - \bar{a}_t}\epsilon$, with $\epsilon \sim N(0, I)$.
From: $x_t = \sqrt{1 - \beta_t x_{t-1}} + \sqrt{\beta_t}\epsilon_t$, and $a_t = 1 - \beta_t$:
We derive it as:
With definiton of at:

$$x_t = \sqrt{1 - \beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t$$

$$x_t = \sqrt{a_t}x_{t-1} + \sqrt{1 - a_t}\epsilon_t$$

we apply the formula recursively for x_{t-1}:

$$x_{t-1} = \sqrt{a_{t-1}}x_{t-2} + \sqrt{1 - a_{t-1}}\epsilon_{t-1}$$

gives:

$$x_t = \sqrt{a_t}x_{t-1} + \sqrt{1 - a_t}\epsilon_t$$

$$x_t = \sqrt{a_t}(\sqrt{a_{t-1}}x_{t-2} + \sqrt{1 - a_{t-1}}\epsilon_{t-1}) + \sqrt{1 - a_t}\epsilon_t$$

$$x_t = \sqrt{a_t a_{t-1}}x_{t-2} + \sqrt{a_t(1 - a_{t-1})}\epsilon_{t-1} + \sqrt{1 - a_t}\epsilon_t$$

We know that for independent Normal fucnitons $\sqrt{a}\epsilon_1 + \sqrt{b}\epsilon_2 = \sqrt{a + b}\bar{\epsilon}$:
Hence the two last terms becomes:

$$\begin{aligned}\sqrt{a_t(1 - a_{t-1})}\epsilon_{t-1} + \sqrt{1 - a_t}\epsilon_t = \\ \sqrt{a_t(1 - a_{t-1}) + 1 - a_t}\bar{\epsilon}_{t-1} = \\ \sqrt{a_t - a_t a_{t-1} + 1 - a_t}\bar{\epsilon}_{t-1} = \\ \sqrt{1 - a_t a_{t-1}}\bar{\epsilon}_{t-1} =\end{aligned}$$

hence:

$$x_t = \sqrt{a_t a_{t-1}} x_{t-2} + \sqrt{a_t(1 - a_{t-1})}\epsilon_{t-1} + \sqrt{1 - a_t}\epsilon_t$$

becomes

$$x_t = \sqrt{a_t a_{t-1}} x_{t-2} + \sqrt{1 - a_t a_{t-1}}\bar{\epsilon}_{t-1}$$

If we continue to recusively apply until we get back to x_0, we apply this process t times:

$$x_t = \sqrt{a_t a_{t-1}..a_1} x_0 + \sqrt{1 - a_t a_{t-1}...a_t}\epsilon$$

With our definition of $\bar{a}_t = \prod_{s=1}^{t}(1 - \beta_s) = \prod_{s=1}^{t} a_s = a_1 a_2 ... a_t$:

$$x_t = \sqrt{\bar{a}_t} x_0 + \sqrt{1 - \bar{a}_t}\epsilon$$

So per our process of continously noising our sample from a clean x_0 to a completely random x_t we should now have arrived at a gaussian random noise distirbition after t applications of the forward process, hence:

$$x_t = \sqrt{\bar{a}_t} x_0 + \sqrt{1 - \bar{a}_t}\epsilon$$

$$\epsilon \sim N(0, 1)$$

3.1.b.1)

For T=1000, generating a sinlge image requires 1000 forward (denoising) passes through a large NN. This is computationally expensive becuase 1. 1000 is a large nr of forward passees, 2. the NN is large, 3. We have to pass it thorugh sequentially since it is a markov process each step depends on the previous.

3.1.b.2)

What DDPM does: It generates images by gradually removing noise through many small sequential denoising steps. Each step is learning the reverse process of adding a slight bit of Gaussian noise to a given input image. Repeating this enough times takes us from complete noise to clear image.

Why we cant jump through steps: 1 The denoising process is a markov chain which means taht each step depends on the previous step. 2 the model is only trained to partially denoise a given input. 3 we choose more steps in training this partial denoising over fewer for stability as very few steps i a much more complex task and may lead to worse results.

3.2. DDIM : Deterministic Reverse Process and Comparison (8 pts)

3.2.a)

Solve: $\hat{x}_0(x_t, t) = \frac{x_t - \sqrt{1-\bar{a}_t}\epsilon_\theta(x_t,t)}{\sqrt{\bar{a}_t}}$ in terms of x_t and $\epsilon_\theta(x_t, t)$, and write down estimator:

We ahve the forward process from 1a:

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon$$

We want to solve for x_0 in terms of x_t and predict the noise $\epsilon_\theta(x_t, t)$:

In DDIM we get a networks noise prediction $\epsilon \approx \epsilon_\theta(x_t, t)$, hence we have:

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon_\theta(x_t, t)$$

we isolate $x_0$term:

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon_\theta(x_t, t)$$

$$x_t - \sqrt{1-\bar{a}_t}\epsilon_\theta(x_t, t) = \sqrt{\bar{a}_t}x_0$$

$$x_t - \sqrt{1-\bar{a}_t}\epsilon_\theta(x_t, t) = \sqrt{\bar{a}_t}x_0$$

$$\frac{x_t - \sqrt{1-\bar{a}_t}\epsilon_\theta(x_t, t)}{\sqrt{\bar{a}_t}} = x_0$$

3.2.b)

Show that we can choose a determinist update of the form $x_{t-1} = \sqrt{\bar{a}_{t-1}}\hat{x}_0(x_t, t) + \sqrt{1-\bar{a}_{t-1}}\epsilon_\theta(x_t, t)$:

from 1a the forward process to create noisy image at time t from clean image x0 and noise ϵ:

$$x_t = \sqrt{\bar{a}_t}x_0 + \sqrt{1-\bar{a}_t}\epsilon$$

creating x_{t-1} from x_0:

$$x_{t-1} = \sqrt{\bar{a}_{t-1}}x_0 + \sqrt{1-\bar{a}_{t-1}}\epsilon$$

From 3.2a we have estimate of clean image $\hat{x}_0(x_t, t)$, which and predicted noise $\epsilon \approx \epsilon_\theta(x_t, t)$:

$$x_{t-1} = \sqrt{\bar{a}_{t-1}}x_0 + \sqrt{1-\bar{a}_{t-1}}\epsilon$$

becomes:

$$x_{t-1} = \sqrt{\bar{a}_{t-1}}\hat{x}_0(x_t, t) + \sqrt{1-\bar{a}_{t-1}}\epsilon_\theta(x_t, t)$$

From x_t (noisy) we estiamte the clean image $\hat{x_0}$. We then recreate x_{t-1} as a less noisy version of the smae clean image. Ie we assume the image is the same thorughout the process, the noise stays the same and produces the same output. We dont have any random sampling and use the same noise direction $\epsilon_\theta(x_t, t)$ for both timesteps jsut scaled differently. This is why the process i deterministic.

3.2.c)

How DDIM can use only a small usbset of teimeteps

DDIM is deterministic, ie there is not randomness introduced in between steps unlike in DDPM where the noise introduce is new random sampling each step. DDIM instead uses the same noise scaled differently, and movees towards the same goal end impage.

Each timestep in DDIM, we estimate the image that the noice is added to as well x0. Early in teh process this is a bad estimation but this doesnt matter so much because we add a lot of noise to it anyway. We cant really see what the noise is added to anyways. But we can use this "behind the scenes" peak at the final image from any process step, to jump in the process. In essesnce we always get an estimated final picture each step we just decide how many steps we want to run the denoising process to make it good.

Why this typically makes samilng much faster than DDPM

Sampling in the context of DDIM is the process of generating a new image (sample) from the original distribuiton of clean images from the random noise vector.

It is faster in DDIM than DDPM because we can skip steps because of the deterministic anture of DDIM, see above explaniation.

Problem 4 : Basic Methods of Reinforcement Learning (20 points)

4.1.a)

A process sastisfies the markov property if the future state of the process depnds only on the poresent state and not on the sequence of events preceeding it.

Why does it hold in the students MDP?

Arrows are possible acitons, states are circles with rewards R. So at a given state we ahve the option to move along the connected arrows (actions). Ie the actions we can take and hence where we can end up at state x_{t+1} is a function of the current state x_t. The aciotns in x_t does not depend on how one arrived there ie x_{t-1}. Once we are at x_t the history before that does not matter. Hence the markov propertyy is fulfilled.

4.1.b)

i. Cumulative reward of 1,2,3,4,5

$$R_1 + R_2 + R_3 + R_4 + R_5 = -0.5 + -1.2 + 0 + 2 + 50 = 50.3$$

ii. Probability of sequence

$P(sequence) = P_{relax} * P_{youtube} * P_{study} * P_{submit} = 0.6 * 0.5 * 0.6 * 0.4 = 0.072$

Probability of the whole seequence occurring is $P(sequence) = 7.20000 * 10^{-2}$

4.1.c)

Policy

The agents behaviour, ie a mapping from all states to what actions to take for those states. It can be deterministic or stchastic probability of taking soem action given a state. For the student a policy could be to always study for state 1,2,3 and to submit if studying state 4. :

Policy: Learning agent's way of behaving at a given time • A mapping from perceived states of the environment to actions to be taken when in those states

Reward

The reward is a scalar feedback signal recieved for moving form one state to another. This can be used to train a RL agent. In the student MDP its is the R signal. :

Reward (signal): Agent's benefit at each time step • On each time step, a single number that the environment sends to the reinforcement learning agent

Value Function / Q funciton

ESEtiamtes ling term expectred cumuative reward ffrom a gvein state or state action pair. Value funciton is a funciton only of state, and Q a funciton of state action pairs. For the student MDP it will let the student move to study even though immediate reward is negative, in order to get the higher cumujlative reward including the submissing at +50.:

Value function: Agent's benefit at the end of the episode • The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that stat

Model

(optional) Model (of an environment): Mimicking the behavior of the environment

4.2 Policy Gradient Methods PGM

4.2.a)

Given policy gradient theorem:

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a \mid s)$$

Role of $\mu(s)$ on pollicy state distribution in the polcy gradient theorem

A policy is deciding on the action to take from some set of states visited. If we want to evaluate how "good" a given policy is we need to look at the expected return from visiting the states for the policy and taking the poclicys actions form those states. Hence states that are not visited given our policy has no impact on how good the policy is and by the same logic states visited often has a larger impact. This is why we must weight the states by $\mu(s)$ when calcuating the gradient of the objective.

Why can we not simply weight all states equally

Irrelevant states problem: Some states may be completely unreachable under the current policy $\pi\vartheta$. For example, in the student MDP, if the policy never chooses "Youtube" from state 2, then certain states might never be visited. Optimizing the policy for unreachable states wastes computational resources and provides no benefit to actual performance. Misaligned optimization objective: The goal $J(\vartheta) = v\_\pi\vartheta(s_0)$ measures expected return from the start state s_0 under policy $\pi\vartheta$. States that are never or rarely visited contribute negligibly to this objective, so equal weighting would optimize for the wrong goal. Incorrect gradient direction: Equal weighting could lead to gradient updates that improve performance in rarely-visited states at the expense of frequently-visited states, potentially decreasing overall performance $J(\vartheta)$. Violates the causal structure: The value of improving the policy at state s depends on how often s is reached. A state visited 100 times per episode should naturally have $100\times$ more influence than a state visited once.

4.2.b)

PGM are stocastic gradient ascent realization of genrealised policy iteration GPI

Why REINFORCE can be viewed as a polciy improvement step:

1. REINFORCE udpateds paramters. Moving θ in the direction that increses $J(\theta)$ teh expected reutrn from the start state. 2. The udpate pushes probability mass towrards actions taht yeild higher return and away from actions that yuilded low return. This "improves" the policy. 3. Instead of computeing the full gradient REINFORCE uses sampled trajectories to get an unbiased stochastic estaimte of this gradient. Each sample provides a noicy but correct

in exectation dreicotn for imprvement. 4. Unlike traditional plolicy itteration which makes the plucy greedy wrt q_π REINFICE prefomr ssoft imprvement by gradually shifting aciton rpobabilites the stchocastic poliy naturally explore while still moving towards bettera citons.

Why using sampled return G_t corresponds to imporving an approximation of the action value funciton

REINFORCE is a policy improvement algorithm that determines the gradient of a policy's performance with respect to its parameters θ and uses gradient ascent to update them. It acts as an update step because it maximizes the expected return. The update rule is based on the likelihood ratio method and involves the use of the sampled accumulated return G_t as an unbiased estimate of the true action-value function. Since G_t is directly observed from a trajectory, it is used instead of approximating Q, which would require a separate learning process. The core identity leveraged is that the expected return is equal to Q. Therefore, using Gt in the gradient calculation provides a known, true value estimate, which improves the stability and practicality of the policy gradient estimate.

Problem 5 : PPO vs GRPO for Training Language Models (20 points)

5.a.i

Excessive large policy updates can harm traning due to several fundamental RL related issues. In PPO advantage estimates are computes unser the old policy $\pi_{\theta_{old}}$ hence when policy changes dramatically these advantage estiamtes becomes invalid guised for policy optimization. Making gradient estamtes unreliable. Further more in RL the training data is itself dependent on the current polcy, ie take action get feedback depends on what action. Hence data distirubiton over observations and rewads are constantly chaning, and large policy updates makes this worse by causing dramatic shifts in the data distribution. Lastly RL is very senstivie to hyper paramter turning, if updates are too large teh policy network can be pushe dinto a region of parameter space wehre it collects the next batch of data unde rvery bad policy cuaing performacne collapse from which it may never recover. In essence large updates break the assumption that collected experience remain relevant for optimiziation leading to poor data efficeny and traning instability. This is because PPO is on policy assuming data sued fro trning was collected by or is close to the current policy beeing optimized.

5.a.ii

Describe PPOs clipping idea for preventing such instability

One idea to combat this type of instability is to set a bound for how far away from the current policy we can move when updating our policy. Idea introduced in Trust Regioun Policy Optimisation TRPO which is the basis on which PPO was built.

We move from objective in GPM:

$$\hat{g} = \hat{E}_t[\nabla_\theta log \pi_\theta(a_t \mid s_t)\hat{A}_t]$$

In Trust region methods:

$$max_\theta \hat{E}_t[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t]$$

To make sure updated policy does not move too far away from current policy PPO adds a KL divergence constraint. The problem is that this contraint adds additional overhead and sometimes leads to bad trnaing behaviour. PPO tries to mitigate this by incorporating it directly into the optimisatio objective.

PPO Clipped surrogate Objective:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$$

$$L^{CPI}(\theta) = \hat{E}_t[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t] = \hat{E}_t[r_t(\theta)\hat{A}_t]$$

Main objective function in PPO

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

This is normal policy gradient objective $min(r_t(\theta)\hat{A}_t$

Clipped version of normal polcy gradients objective: $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$

How does this mechanism limit the size of the policy udpate?

How clip works:

Advantage can be possitive or negative affecting the value of the min operator:

- Possitive advantage: Selected action had better than expected effect on outcome. Loss function flattens out when r gets to high. This happens when action is a lot more likely under current policy than under old polcy, in thsi case we dont want to overdo the polciy update so the objective function gets clipped here to limit the effect of the gradient update.

- Negative advantage: Action had an estiamted negative value, objective flattens when r goes near zero. This correpsons to actions that are much less likely now than in the old policy, here we dont want to overdo the update and set the probability of tehse action to 0. Advantage is noisy so we dont want to destry policy based on a single estiamte. If on the other hand the action is very probable r is big and at the same time the advantage is negative the objective tells us to make this action less probably by an amount prorporational to how wrong we were. This is the only region where the unclipped version of the function has a lower value than the clipped one and gets returned by the min operator.

In essence PPO does the same as TRPO objective which is to force the policy updates to be conservative if they move very far away from the current policy. The differnce is that TRPO puses complicated KL divergence to achieve this while PPO uses a simpler min funciton. PPO often outperforms the more complex function anyway.

5.b.i

The per-token KL penalty serves as a regularizer that prevents the fine-tuned policy $\pi\vartheta$ from straying too far from the reference model πref. This prevents overfitting, RLHF is prone to overfitting to the specific reward model which is traned on limted huan feedback data. KL term penallises large devaitons from the reference model. RLHF want to tune answers to human prefernce not fundamentally change the model.

5.b.ii

Reward Hacking Prevention: The reward model is an imperfect approximation of human prefernces learnt from lited data, without the KL penalty, teh polciy can discover inputs or generation patterns that give high rewards but dont represent desirable behaviour. The gradients for these kinds of hacaks could be steep and start to drastically change the model. KL anchors policy to a reference making sure it does not go to far away, into some bisssare policy region.

Distributional shift: Policy may collapse into a narrow mode repeating a smaller "safer" hgih reward policy rather than letting us acceses coerage of the whola language space or abandon natural language complately when token probabilities beocme extreme, the model may only favour some tokens and not others.

In esssence the reward model is imperfect based on human feedback. It is possible that humans or the model itself could take advantage of this and find bissare behaviour that still give high rewards. Hence without tying udpates to a somewhat "sane" reference model with a KL penalty the model could go off the rails.

5.c

How GRPO provides basedline withotu learnt critic:

We group outputs and comptue each outputs rewad relative position in the group of sampled outputs. For each prompt GRPO samples G outputs from the old polciy and comptues the group relative advantage as $\hat{A}_t^{GRPO} = \frac{R-\mu_R}{\sigma_R+\delta}$. Reaward R,average reward in group μ_R, std of rewards in the group σ_R. This is as oppsed to in PPO (agent critic) where advantage estimation is $A_t^{PPO} = R_t - V(s_t)$ where $V(s)$ is achieved by traning a separte critic network. For $\hat{A}_t^{GRPO}$advatnege is derived purely statistically. It meassures the values of outcome R relatie to the average otput ofthe current baatch.

Why this baseline mehcansim is especially suitable for LLMs:

Having a cirtic network as in PPO has some advantages sucha s lower variance and stable learning due to more precise vlaue estimation. However GRPO has other advantages such as memory and compuation efficeiny and significant memory reduction by cutting out the cirtic network and deriving advantage statistically and more inpreceisely. While PPO is best suited for genrela RL tasks, especially those requireing precise value estimation and low variance GRPO is best stuited for large scale models, particularly LMs in RLHF where memory is a bottleneck and rewards ore often sparse/binary.

LLM are as teh name suggests large, PPO requries a ciritc network often similar size of policy network, which requires storage of critic gradients and optimiser states as well as fowrad and backward propagation. Beeing able to essentially cut the memory footproint in half by cutting out the ciritic is great for maing RLHF feasible for very large models.

RLHF often has binary rewards, ie was this response helpful or not, correct/incorrect etc. Hence learning V(s) for all intermediate states overkill and unessisary. Group based output is more natural simply asking what outputs are better. Each output is "descrete" one output / word prediction not continous. Further more LLMs naturally generates G response per rpmpt during infrerence (context window) it is natural to use these seamples to construct advantage dierctly. GRPO mirrors how humans evaluate LLM outputs, comparing multiple responsese to the same query and selecting the better ones which is often how RLHF is carried out in practise.