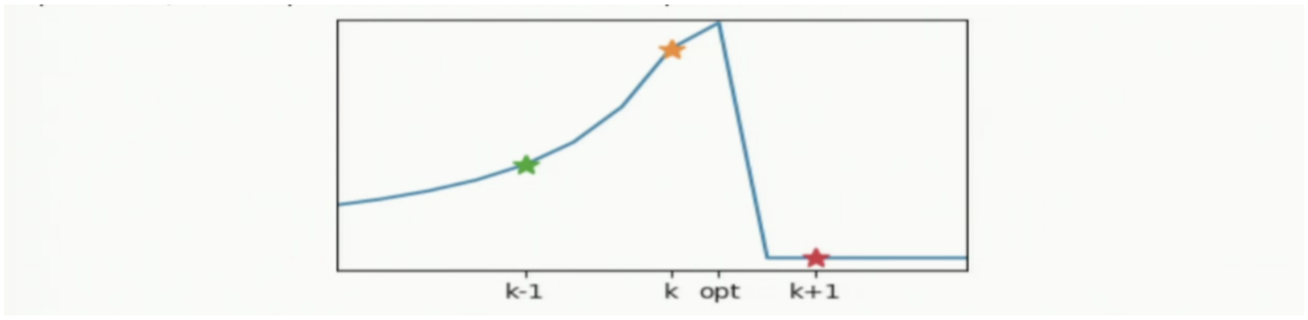
advanced policy gradient methods & TRPO

- downsides of the sampling policy gradient method we've seen so far:
  - we saw that policy gradient is an on policy algorithm - we need to sample from the current policy to get an estimate of the reward gradient w.r.t policy params. the problem with this is that this is not sample efficient. we are not able to re-use past data.
  - we saw that one way to get around that is to do importance sampling. thing is if there is quite a mismatch between p (the target distribution) and q (the distribution we have access to), then the variance of our importance sampling estimate becomes very high, not because of individual importance sampling ratios but because we take the product of many importance sampling ratios in a trajectory, they either vanish or blow up.

$$\begin{aligned}
\operatorname{var}(\hat{\mu}_Q) &= \frac{1}{N}\operatorname{var}\left(\frac{P(x)}{Q(x)}f(x)\right) \\
&= \frac{1}{N}\left(\underset{x\sim Q}{\mathrm{E}}\left[\left(\frac{P(x)}{Q(x)}f(x)\right)^2\right] - \underset{x\sim Q}{\mathrm{E}}\left[\frac{P(x)}{Q(x)}f(x)\right]^2\right) \\
&= \frac{1}{N}\left(\underset{x\sim P}{\mathrm{E}}\left[\frac{P(x)}{Q(x)}f(x)^2\right] - \underset{x\sim P}{\mathrm{E}}[f(x)]^2\right)
\end{aligned}$$

  - so how can we use data we currently have in order to generate a gradient update?
- there is another downside, which is that small steps in parameter space may not end up being small steps in the value or reward. The shape of the expected total reward as a function of policy parameters function is really dependent on the problem domain. for some tasks, when we take a very small step in policy params we can overshoot a peak in a way that we cannot recover easily. this is obviously really bad for us:



When we overshoot, we not only did we jump really low, but also it is really difficult to recover from this because our new local gradients are quite bad (almost flat region)

- so we need to take into consideration the underlying policy funtion's shape.
- Now some theory:
  - We will introduce something called the Relative Policy Performance Identity → $J(\pi') - J(\pi) = E_{\tau\ from\ \pi'}(\sum_{t=0}^{\infty}\gamma^t A^{\pi}(s_t, a_t))$. This is the difference in performance of π' and π.

- the proof of the above identity is quite straightforward. Just apply the definition of the advantage function strictly. There is also an assumption that the starting state distributions are the same regardless of policy.

$$
\begin{aligned}
J(\pi') - J(\pi) &= \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(s_t, a_t)\right] \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t \left(R(s_t, a_t, s_{t+1}) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)\right)\right] \\
&= J(\pi') + \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^{t+1} V^{\pi}(s_{t+1}) - \sum_{t=0}^{\infty}\gamma^t V^{\pi}(s_t)\right] \\
&= J(\pi') + \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=1}^{\infty}\gamma^{t} V^{\pi}(s_{t}) - \sum_{t=0}^{\infty}\gamma^t V^{\pi}(s_t)\right] \\
&= J(\pi') - \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[V^{\pi}(s_0)\right] \\
&= J(\pi') - J(\pi)
\end{aligned}
$$

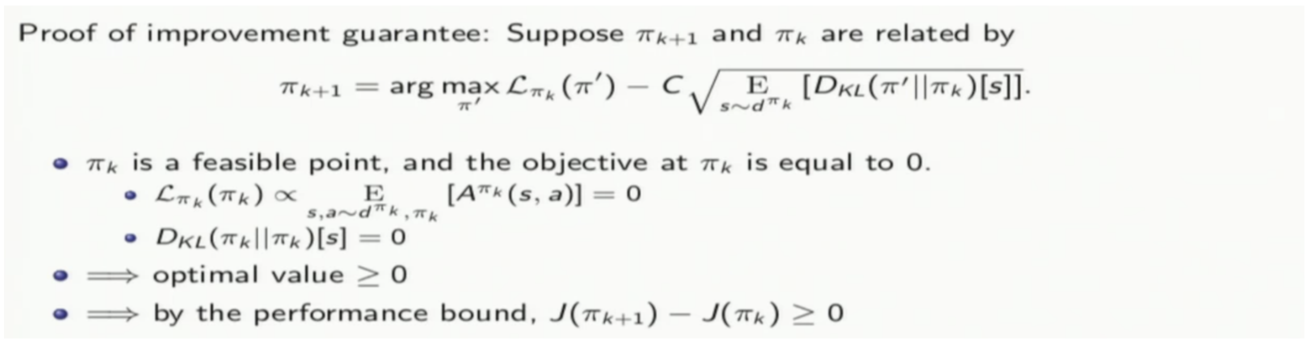
- now can we use the Relative Policy Performance Identity as an objective function for our policy update? The nice feature of this identity is that this defines the performance of π' in terms of the advantages from π.
- But it still requires trajectories sampled from π', which we don't even know (our optimization objective at each iteration is to find that). A new trick to get around this: instead of sampling from the trajectories of π', we will sample from the **discounted future state distribution** of a policy which is how likely we are to be in a particular state in the future, discounted by how far in the future we will be there. But we still don't have this discounted future state distribution for π'. so what do we do? importance sampling to our rescue! Using this discounted future state distribution from π (not π') and using importance sampling and doing some basic math, we get:

$$
\begin{aligned}
J(\pi') - J(\pi) &= \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(s_t, a_t)\right] \\
&= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi'}}\left[A^{\pi}(s,a)\right] \\
&= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right]
\end{aligned}
$$

- Now, we are mostly good, BUT we still have the states above being sampled from π', which we'd like to really avoid. what if we just ignore it? Turns out, as long as π' and π are pretty close, this is a good approximation. Turns out the error here is bounded by the square root of some constant C times the KL divergence between these policies, which in practice turns out to be not too bad. so we will go with

$$
\begin{aligned}
J(\pi') - J(\pi) &\approx \mathcal{L}_{\pi}(\pi') \\
\mathcal{L}_{\pi}(\pi') &= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi} \\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right] \\
&= \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)}A^{\pi}(s_t,a_t)\right]
\end{aligned}
$$

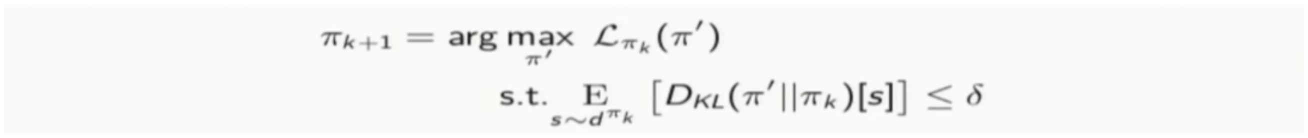
- Now our expectation is entirely from policy π which we have sampled already. The above is called the surrogate advantage function $\mathcal{L}_\pi(\pi')$. Remember this is basically the difference in performance between π and π'. Now we just need to optimize for π'.
- The big win is we have gotten rid of the product of importance sampling weights that we saw in our vanilla policy gradient methods.
- now we can take the gradient of this surrogate objective function and take a step. in fact, we can take multiple steps using the same data → this is the other big win. now we have gotten around the importance sampling blowing up/vanishing problem and we have also made it so it doesn't have to be like full on policy all the time.
- now using the theoretical bounds of error during estimation of the above (remember we ignored the fact that states are sampled from the old policy?), we can derive a result that we will always get better:


Proof of improvement guarantee: Suppose π_{k+1} and π_k are related by

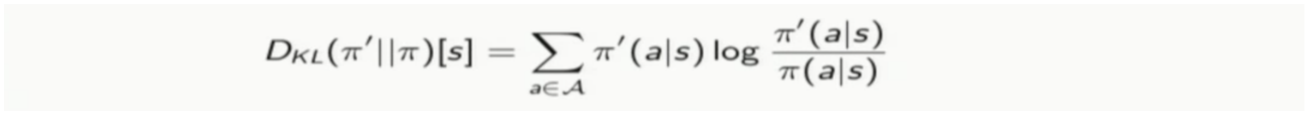
$$\pi_{k+1} = \arg\max_{\pi'} \mathcal{L}_{\pi_k}(\pi') - C\sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi_k}}\left[D_{KL}(\pi'||\pi_k)[s]\right]}.$$

- π_k is a feasible point, and the objective at π_k is equal to 0.
  - $\mathcal{L}_{\pi_k}(\pi_k) \propto \mathop{\mathrm{E}}_{s,a\sim d^{\pi_k},\pi_k}[A^{\pi_k}(s,a)] = 0$
  - $D_{KL}(\pi_k||\pi_k)[s] = 0$
- $\Longrightarrow$ optimal value ≥ 0
- $\Longrightarrow$ by the performance bound, $J(\pi_{k+1}) - J(\pi_k) \geq 0$

The above assumes C is small enough. If γ is too close to 1, C turns out to be pretty big. What we can do for that is to use a constraint that bounds the KL divergence which will be a hyper parameter of our algorithm. This constraint leads to what is called the **trust region.** so now our optimization problem will be:

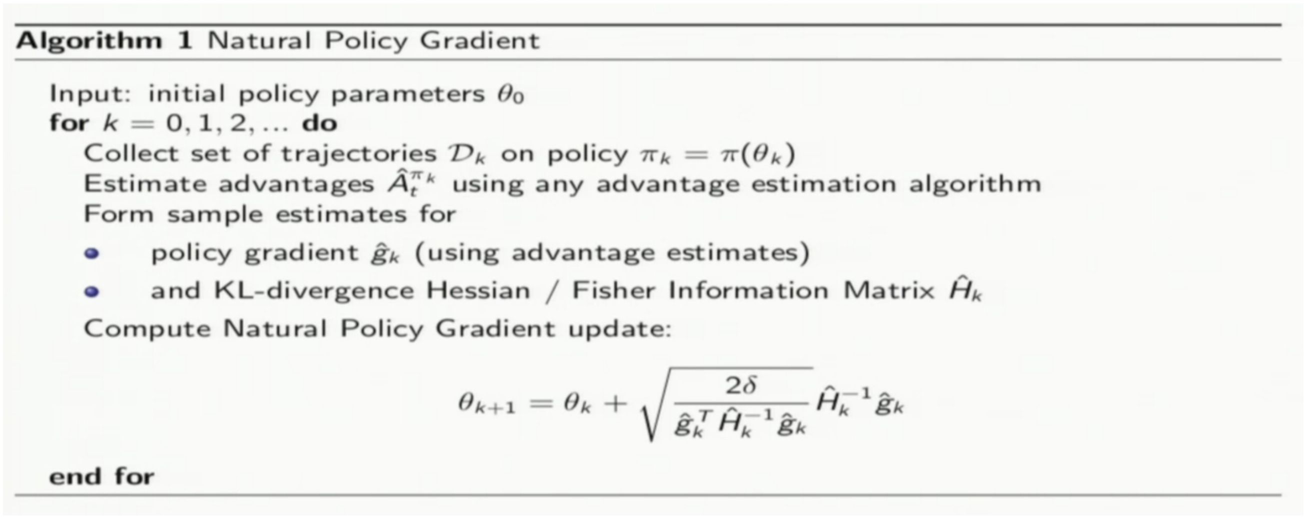

$$\begin{aligned}\pi_{k+1} = \arg\max_{\pi'}\ & \mathcal{L}_{\pi_k}(\pi') \\ \text{s.t.}\ & \mathop{\mathrm{E}}_{s\sim d^{\pi_k}}\left[D_{KL}(\pi'||\pi_k)[s]\right] \leq \delta\end{aligned}$$

- The way we get KL divergence (which is an optimization constraint) between two policies is by using sampling of the following expression:


$$D_{KL}(\pi'||\pi)[s] = \sum_{a\in\mathcal{A}} \pi'(a|s)\log\frac{\pi'(a|s)}{\pi(a|s)}$$

- Because we get this theoretical bound that says our policy is getting no worse as long as the KL divergence is constrained (which is not really proved here, but an intuition is given), we have this notion of distance between policies and we prevent the jumping off the cliff problem we saw earlier. In order to see a full proof, we will need to see how C is derived but we don't go into that here.
- Now how the heck would we solve this constrained optimization problem for a complex model class like a neural net? Well, we solve an approximate version of this problem by linearizing the objective and quadratifying the constraint (which uses a Hessian matrix of the KL divergence, called the Fisher information matrix) using a method called **natural policy gradient,** the details of which we won't go into

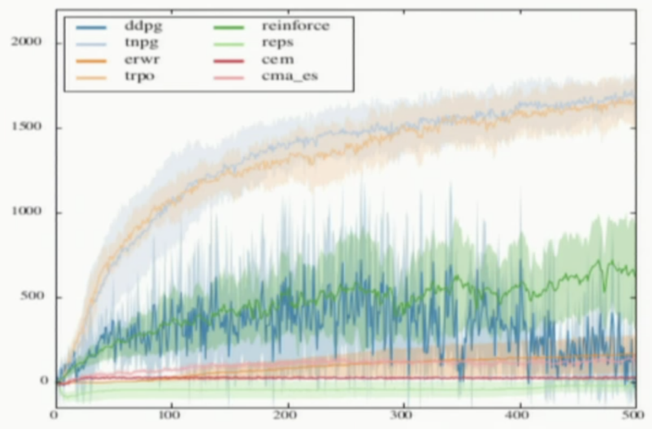
here. the Hessian turns out to be positive semi-definite so the inequality constraint is a convex constraint. (to understand more, take a convex optimization class). The natural policy algorithm:



Algorithm 1 Natural Policy Gradient

```
Input: initial policy parameters $\theta_0$
for $k = 0, 1, 2, \dots$ do
    Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
    Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
    Form sample estimates for
    •   policy gradient $\hat{g}_k$ (using advantage estimates)
    •   and KL-divergence Hessian / Fisher Information Matrix $\hat{H}_k$
    Compute Natural Policy Gradient update:
        $\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$
end for
```

- In order to find $H^{-1}g$ for a complex model like a neural net, we can't invert the Hessian (its crazy to even find it because $O(n^2)$, forget inverting it). so instead we use an algorithm that called conjugate gradient (CG) algorithm that solves $Hx = g$ iteratively within $O(n^2)$.
- Now, natural gradient linearizes $\mathcal{L}$ and quadratifies the constraint. This might not be accurate enough sometimes and if we take a biggish step, (what is big depends on how bad of an approximate it is in that region) we make the same mistake as pointed out at the beginning of these notes. natural policy gradient does not check for this. an algorithm called **trust region policy optimization** provides checks for this: it makes sure that the step actually improves the policy and if it doesn't, it does a backtracking line search shrinking the step size until it finds one that does improve the policy. Because of this backtracking line search, TRPO is able to take larger steps and converge faster!
- Look at how much better natural gradients and TRPO perform compare to vanilla policy gradient methods even including the discounted future state distribution + importance sampling idea.



- Proximal Policy Optimization (PPO) is a family of algorithms that approximately enforce the KL constraint mentioned above without computing natural gradients:
    - How does this do this? By penalizing the KL divergence in an unconstrained optimization problem and then iteratively adapting the penalty co-efficient depending on if the KL divergence is being

violated or not.

- the other approach is to use what is called a clipped objective. This is basically a heuristic method to clip the region we trust to a small enough region where our step probably won't overshoot. The math is in the PPO paper. This method is not so theoretically motivated but tends to work well in practice.