Parameter Tuning for Maximum Entropy

Vanilla Sampling

Recall that for a *fixed length uniform sampler* for a TRE φ we need to do two steps:

1. Sample a duration T
2. Sample a word in the slice $L_n^\varphi(T) := \{w \in L_n(\varphi) \mid \theta(w) = T\}$

Last time I showed how my tool can

- compute the slice volume $V_n^\varphi(T)$ as a piecewise polynomial
- sample T according to the pdf $p(T) = \frac{V_n^\varphi(T)}{\int_0^\infty V_n^\varphi(T')dT'}$
- sample $w \in V_n^\varphi(T)$ uniformly

This way of sampling T is what I called the "vanilla sampling" last time. It is a way of actually sampling uniformly, but can only be used for bounded expressions.

Alternatively, we can sample T according to the maximum entropy pdf.

Maximum Entropy Sampling

Now we use the pdf

$$p_\lambda(T) = \frac{e^{\lambda_1 T + \cdots + \lambda_m T^m} V_n^\varphi(T)}{\int_0^\infty e^{\lambda_1 T' + \cdots + \lambda_m T'^m} V_n^\varphi(T')dT'}$$

for sampling. This is the general form of a maximum entropy solution.

The interesting new thing which this pdf allows us to do is controlling the moments of our samples. Intuitively, we have a sort of correspondence

$$\vec{\lambda} \leftrightarrow \vec{\mu}$$

between the moments μ_i and the parameters λ_j.

One direction is clear using the definition of moments:

$$\mu_i(\vec{\lambda}) = \int_0^\infty T^i p_\lambda(T)dT$$

However, we want to do the inverse: Fix a target $\vec{\mu}^*$ and find parameters $\vec{\lambda}$, such that $\vec{\mu}(\vec{\lambda}) = \vec{\mu}^*$. For this, we will probably not find a way to do it analytically, so we instead solve the optimization problem

$$\min_{\vec{\mu}} MSE(\vec{\mu}, \vec{\mu}^*)$$

using either gradient descent, Euler-Raphson or similar.

Note that there is a nice representation of the partial derivatives of μ_i with regards to λ_j, so it should be straightforward to use gradient based methods for optimization.

$$\frac{d\mu_i}{d\lambda_j}(\vec{\lambda}) = \mu_{i+j}(\vec{\lambda}) - \mu_i(\vec{\lambda}) \cdot \mu_j(\vec{\lambda})$$

This implementation is what I am currently working on!
This general idea can also be used in a more practical setting to fix (for example) mean μ and standard deviation σ^2 of the sampler.

Some Experiments

Statistical Test of the Moment Theory

To validate the theory above, I made a simple example: Sample many times and see whether the experimental moments and the theoretical moments match up. The results are as expected and we see the former converge towards the latter:

```
Sampled 3000 words in 209.1936013698578s.
Average word duration: 1.51818014851118
statistical 2nd moment: 2.39111546778172
statistical 3rd moment: 3.88701192686922
[[1.51993797]
 [2.39551502]
 [3.8949701 ]]
```

Comparison with TA/WordGen

We are also currently looking into examples, where the WordGen tool explodes and mine does not (to show a benefit to my technique). While experimenting, we found potential bugs on both sides. **Investigating...**

However, the example managed to slow down WordGen while remaining normally fast for mine.

```
TAKiller.tre
< a.<b.<c.<d*>_[0,1]>_[0,2]>_[0,3]>_[0,4]
\`\`\`
```