Knowledge Injection via Lagrangian Approaches

Let's start with the classics

Lagrangians to the Rescue

A popular way to handle soft constraints in ML is inspired by Lagrangians

Given a learning problem:

$$\operatorname*{argmin}_{\theta}\left\{L(\hat{y}) \mid \hat{y} = f(x;\theta)\right\}$$

And soft constraints modeled as inqualities on a vector function:

$$g(\hat{y}) \leq 0$$

- With $g(\hat{y}) = \{g_k(\hat{y})\}_k^{m_c}$

Lagrangians to the Rescue

A popular way to handle soft constraints in ML is inspired by Lagrangians

Given a learning problem:

$$\operatorname{argmin}_{\theta} \left\{ L(\hat{y}) \mid \hat{y} = f(x; \theta) \right\}$$

And soft constraints modeled as inqualities on a vector function:

$$g(\hat{y}) \leq 0$$

- With $g(\hat{y}) = \{g_k(\hat{y})\}_k^{m_c}$

The idea is to turn the constraints into loss terms

- Doing tha will steer the model towards satisfying the constraints
- ...And can be though of as a form of regularization

In fact, an early example of this approach is called Semantic Based Regularization

Lagrangian-like Loss

In practice, we usually form a modified, Lagrangian-like, loss:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T h(g(\hat{y}))$$

Where h is sometimes referred to as a penalizer

- Intuitively, we don't use the constraint violation as it is
- ...But we build a function based on its value

Lagrangian-like Loss

In practice, we usually form a modified, Lagrangian-like, loss:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T h(g(\hat{y}))$$

Where h is sometimes referred to as a penalizer

- Intuitively, we don't use the constraint violation as it is
- ...But we build a function based on its value

Why the penalizer instead of using a classic Lagrangian?

...Which by the way would be:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T g(\hat{y})$$

Lagrangian-like Loss

In practice, we usually form a modified, Lagrangian-like, loss:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T h(g(\hat{y}))$$

Where h is sometimes referred to as a penalizer

- Intuitively, we don't use the constraint violation as it is
- ...But we build a function based on its value

Why the penalizer instead of using a classic Lagrangian?

...Which by the way would be:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T g(\hat{y})$$

There are two main, non-trivial, reasons

A Stop-gain Mechanism

We are considering inequality constraints

$$g(\hat{y}) \leq 0$$

- Predictions with $g_k(\hat{y}) < 0$ are equivalent to those with $g_k(\hat{y}) = 0$
- ...But in a classical Lagrangian approach a slack translates to a reward

In classical Lagrangian theory, this is avoided by the KKT coditions

...And in particular by complementary slackness:

$$\lambda \odot g(\hat{y}) = 0$$

- This condition can only be achieved by updating λ and $\hat{y}$ together
- ...And typically some specialized optimization algorithm

When $\hat{y}$ comes from a non-linear model, the condition is tricky to achieve

A Stop-gain Mechanism

However, there's a far easier alternative

We can just use non-linearity to remove the reward effect, e.g. by clipping:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

- The maximum operator will neutralized any reward when $g_k(\hat{y}) < 0$
- ...Which is effectively equivalent to forcing λ_k to 0

A Stop-gain Mechanism

However, there's a far easier alternative

We can just use non-linearity to remove the reward effect, e.g. by clipping:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

- The maximum operator will neutralized any reward when $g_k(\hat{y}) < 0$
- ...Which is effectively equivalent to forcing λ_k to 0

With the new penalizer:

- When all constraints are feasible, we preserve the original loss function
- When a constraint is infeasible, we introduce a penalty

And this is true as long as $\lambda \geq 0$

This approach comes from penalty methods

Semantic-based Calibration

Using a clipped penalizer makes it also easier to choose the multipliers

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

- There's no more need to optimize $\hat{\boldsymbol{y}}$ (i.e. $\boldsymbol{\theta}$) and $\boldsymbol{\lambda}$ together
- ...Since any fixed vector $\boldsymbol{\lambda} \geq 0$ will result in meaningful penalties

But how should we choose a λ vector among the valid ones?

Semantic-based Calibration

Using a clipped penalizer makes it also easier to choose the multipliers

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

- There's no more need to optimize $\hat{y}$ (i.e. θ) and λ together
- ...Since any fixed vector $\lambda \geq 0$ will result in meaningful penalties

But how should we choose a λ vector among the valid ones?

This is made trickier by the fact that we have soft constraints

- We exepect our constraints to be useful
- ...But we don't want them satisfied at the expense of accuracy!

Semantic-based Calibration

In theory, there's a simple approach for calibrating λ

- Since our goal is to improve accuracy
- ...We can just assess the quality of a $\boldsymbol{\lambda}$ vector by cross-validation
- Then we can search for an optimal $\boldsymbol{\lambda}$

In practice, however, this approach doest not always work

Semantic-based Calibration

In theory, there's a simple approach for calibrating λ

- Since our goal is to improve accuracy
- ...We can just assess the quality of a λ vector by cross-validation
- Then we can search for an optimal λ

In practice, however, this approach doest not always work

In most cases, knowledge injection is used when supervised data is scarce

...And in this situation cross-validation is not very reliable

- We can optimize the training-set accuracy instead
- ...But that comes at the risk of overfitting

Is there an alternative?

Semantic-based Calibration

In general, calibrating λ is still an open problem

...But we can make it simpler by thinking a bit about the penalizer semantic:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

A key difficulty here is that the two loss terms use different "currencies"

- In most cases, $L(\hat{y})$ will represent a (negative) log likelihood
- While each $\max(0, g_k(\hat{y}))$ represents a violation

Semantic-based Calibration

In general, calibrating λ is still an open problem

...But we can make it simpler by thinking a bit about the penalizer semantic:

$$\mathcal{L}(\theta, \lambda) = L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

A key difficulty here is that the two loss terms use different "currencies"

- In most cases, $L(\hat{y})$ will represent a (negative) log likelihood
- While each $\max(0, g_k(\hat{y}))$ represents a violation

A way around this issue consists in converting the currency

Typically, we link the violation to a probability

- This can be done by assuming a (prior) distribution for constrait violation
- ...And then deriving a penalizer based on that assumption

Semantic-based Calibration

Let's make an example for a (scalar) constraint $g_k(\hat{y}) \leq 0$

The violation is given by:

$$\max(0, g_k(\hat{y}))$$

Semantic-based Calibration

Let's make an example for a (scalar) constraint $g_k(\hat{y}) \leq 0$

The violation is given by:

$$\max(0, g_k(\hat{y}))$$

Let's assume it is associated to an exponential distribution with rate γ:

$$P(\hat{y} \mid x) = \gamma e^{-\gamma \max(0, g_k(\hat{y}))} \qquad \text{conditional, since: } \hat{y} = f(x; \theta)$$

Using an exponential is a reasonable choice for a soft-constraint in regression:

- It means we expect small violations to be quite likely
- ...But large violations to be very rare

Semantic-based Calibration

We can now derive the corresponding (negative) log likelihood:

$$-\log P(\hat{y} \mid x) = -\log \gamma + \gamma \max(0, g_k(\hat{y}))$$

Since we focus on optimization, we don't care about constant terms:

$$\operatorname*{argmin}_{\hat{y}} -\log P(\hat{y} \mid x) = \operatorname*{argmin}_{\hat{y}} \left(\gamma \max(0, g_k(\hat{y}))\right)$$

By plugging the main loss and iterating on all constraints we get:

$$L(\hat{y}) + \gamma^T \max(0, g(\hat{y}))$$

- ...Which is essentially our Lagrangian-like loss

Semantic-based Calibration

In doing this, we've learned something

In our Lagrangian-like loss:

$$L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

- λ represents a vector of rates for exponential distributions
- ...Which enables using domain expertise to calibrate the multipliers

Semantic-based Calibration

In doing this, we've learned something

In our Lagrangian-like loss:

$$L(\hat{y}) + \lambda^T \max(0, g(\hat{y}))$$

- λ represents a vector of rates for exponential distributions
- ...Which enables using domain expertise to calibrate the multipliers

The same approach can be used in other settings

...We just need to make suitable assumptions

- E.g. on classification problems our constraints might be binary predicates
- ...And we might want to use a Bernoulli distribution

Equality Constraints

Equality constraints allow for a simpler formulation

In principle, given an equality constraint:

$$g_k(\hat{y}) = 0$$

We can state it as two inequality constraints:

$$g_k(\hat{y}) \leq 0 \quad \text{and} \quad -g_k(\hat{y}) \leq 0$$

...And build two (weighted) violation terms:

$$\lambda'_k \max\left(0, g_k(\hat{y})\right) \quad \text{and} \quad \lambda''_k \max\left(0, -g_k(\hat{y})\right)$$

- With $\lambda'_k, \lambda''_k \geq 0$

Equality Constraints

Summing the two terms leads to a simplified formula

$$\lambda_k' \max\left(0, g_k(\hat{y})\right) + \lambda_k'' \max\left(0, -g_k(\hat{y})\right) = \lambda_k |g_k(\hat{y})|$$

- Where $\lambda_k = \lambda_k' + \lambda_k''$ and there is no sign restriction

Equality Constraints

Summing the two terms leads to a simplified formula

$$\lambda'_k \max\left(0, g_k(\hat{y})\right) + \lambda''_k \max\left(0, -g_k(\hat{y})\right) = \lambda_k |g_k(\hat{y})|$$

- Where $\lambda_k = \lambda'_k + \lambda''_k$ and there is no sign restriction

In this situation, it also makes sense to assume a Normal distribution

$$P(\hat{y} \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{g(\hat{y})^2}{2\sigma^2}}$$

From which we can derive the loss:

$$L(\hat{y}) + \lambda^T g(\hat{y})^2$$

- Where λ corresponds to: $1/(2\sigma^2)$

Differentiability

It's worth talking about differentiability

- Lagangian approaches for knowledge injection
- ...Are most common with differentiable constraints

...Even if differentiability is not strictly needed

Differentiability

It's worth talking about differentiability

- Lagangian approaches for knowledge injection
- ...Are most common with differentiable constraints

...Even if differentiability is not strictly needed

Differentiability might be needed

...Depending on which training algorithms is used, e.g.:

- Gradient descent
- Gradient boosting
- ...

...Which means that we need differentiability when using Neural Networks