MPCL Framework v1

2021
January

1 Definitions

C : a model, e.g. a multiclass perceptron.

$e(p,t)$: evaluation function fine-grained enough to distinguish between reliable and unreliable models when averaged over enough (p,t) pairs. p is a prediction value and t is a target value.

m : minimum average evaluation score over which the evaluated model is considered reliable.

D_k : distribution of the feature domain defined over the sample space Ω_k.

$y(x)$: function mapping inputs (from any feature domain) to targets.

P_k : processors mapping features to latent values in $\mathbb{R}^n$.

$P_k(x)_i$: i-th output of $P_k(x)$.

$L_{k,i}$: latent function defined by $L_{k,i} = P_k(x)_i$

1.1 Extrinsic meaning

The functions $L_{1,i}$ and $L_{2,i}$ are said to convey the same extrinsic meaning if the following conditions are met.

(1) C doesn't ignore its i-th input on any interval of $\mathbb{R}$.

(2) One can construct two resampling functions f_1 and f_2 such that $y(X_1) \overset{d}{=} y(X_2)$ (equal in distribution) with the random variables $X_1 \sim f_1(D_1)$ and $X_2 \sim f_2(D_2)$.

(3)

$$\mathop{\mathbb{E}}_{x\sim f_1(D_1)} e(C(P_1(x)), y(x)) > m \text{ and } \mathop{\mathbb{E}}_{x\sim f_2(D_2)} e(C(P_2(x)), y(x)) > m$$

(4) $e(C(l), t)$ has only one global maximum for every $t \in \{y(x) \mid x \in \Omega_1 \cup \Omega_2\}$.

Remarks

- $|\arg\max_l e(C(l), t)| = 1$ rules out softmax as a possible component of C or e.
- Conditions (1) and (4) overlap quite a lot, but keeping them separate makes it easier to relax (4) without weakening (1).
- Condition (2)'s main job is to get input domains out of the picture. Functions such as $y(x) = DomainName(x)$ or $y(x) = x$ should be avoided because they do nothing to abstract out domain-specific information. In fact, they do not satisfy (2) unless D_1 and D_2 are made up of the exact same data points.
- For conventional classification, (2) comes down to adjusting the sample weights such that every class gets the same total weight, a popular strategy for dealing with class imbalance.

1.2 Properties

1.2.1 Transferability

Training P_1 with C locks all $L_{1,i}$ into some distribution Q_1. Training P_2 with C locks all $L_{2,i}$ into a distribution Q_2 nearly identical with Q_1.

Therefore, if one trains a module $M \neq C$ on top of P_1's latent units with data sampled from $f_1(D_1)$, M will not be thrown off balance when presented with data from D_2 via P_2. M will behave in accordance with its training when shown data from a Q_1-alike distribution and will deliver the same performance irrespective of whether the data comes from P_1 or P_2. Put shortly, M is expected to transfer well to $f_2(D_2)$, and to D_2 to a large extent.

Example Let us consider the simple case of a trained classifier C with a single dense layer made up of a full-rank matrix w and a bias vector b, categorizing n latent values into more than n classes. Also, for the sake of simplicity, we will assume that P_1 and P_2 can compute any function and are trained by minimizing the sum of the squared errors $\sum_{x\in\mathcal{X}_1}(wP_1(x)+b-y(x))^2$ and $\sum_{x\in\mathcal{X}_2}(wP_2(x)+b-y(x))^2$ with a training set $\mathcal{X}_1$ sampled from $f_1(D_1)$ and $\mathcal{X}_2$ sampled from $f_2(D_2)$.

The optimal values for the outputs of P_1 and P_2 are $((w^\intercal w)^{-1}w^\intercal(y(x)-b))_{x\in\mathcal{X}_1}$ and $((w^\intercal w)^{-1}w^\intercal(y(x)-b))_{x\in\mathcal{X}_2}$ respectively. Since $(y(x))_{x\in\mathcal{X}_1}$ and $(y(x))_{x\in\mathcal{X}_2}$ are equal in distribution as per (2), it follows that training P_1 and P_2 pushes Q_1 and Q_2 towards the same optimal distribution.

2 Forgetting

Non-catastrophic forgetting is an important feature of continual learning systems. If a processor is no longer active, then it can be safely removed or repurposed. The activity of a processor is measured by how often its output is passed on to downstream modules.

This procedure does not destructively interfere with downstream modules.

3 Work in progress

Clarifying what the condition (1) entails

I may rewrite (1) as:

$$\{l \in \mathbb{R}^n \mid \frac{\partial C(l)}{\partial l_i} = 0\} \text{ must be a finite set}$$

but it might be too strict of a definition, and not easy to work with. A mathematical definition might not be needed anyway. In practice, neural networks can be forced not to ignore latent units, either via an information bottleneck, dropout, or various regularization techniques.

Surrogate for extrinsic meaning

The ideal surrogate for extrinsic meaning verification is a function S such that

$$S(P_1(x)) > S(P_2(x)) \implies e(C(P_1(x), y(x))) > e(C(P_2(x), y(x)))$$

for all x sampled from D_1 or D_2.

You can see from this definition that S helps with choosing the most suitable processor, even in the absence of $y(x)$.

P_1 is not defined on D_2 and P_2 is not defined on D_1. To build surrogates, we need the extra assumption that

(5) Either $D_1 = D_2$, or $(P_1(x))_{x \sim D_1}$ is not similarly distributed with $(P_1(x))_{x \sim D_2}$ and $(P_2(x))_{x \sim D_1}$ is not similarly distributed with $(P_2(x))_{x \sim D_2}$

Intrinsic meaning

On a high level, the purpose of intrinsic meaning is to constrain latent units such that they exhibit the same properties as extrinsic meaning.

While extrinsic meaning is linked to information that is not necessarily available when making predictions, intrinsic meaning should be defined and verifiable from the input alone. It can be interpreted as self-consistency. For example, it can derive from how latent units are supposed to be correlated with one another.