**PyTorch Active Learning Library**

Robert Munro, Human-in-the-Loop Machine Learning

An Active Learning library in PyTorch, implementing major techniques to sample items for annotation based on: model's confusion; gaps in the model's knowledge; and distributional properties of the data and target domain.

### Machine Learning Knowledge Quadrant:

<table>
<thead>
<tr>
<th>Knowns</th>
<th>Unknowns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confident Model Predictions  (Known Knowns)</td>
<td>Non-Confident Model Predictions (Known Unknowns)</td>
</tr>
<tr>
<td>Current Model State</td>
<td>Uncertainty Sampling</td>
</tr>
<tr>
<td>Latent Information in Related Models (Unknown Knowns)</td>
<td>Gaps in Model Knowledge (Unknown Unknowns)</td>
</tr>
<tr>
<td>Transfer Learning</td>
<td>Diversity Sampling</td>
</tr>
</tbody>
</table>

- **Least Confidence**: difference between the most confident prediction and 100% confidence
  \[
  n \left( 1 - P_\theta(y^*_1 | x) \right) / (n - 1)
  \]

- **Margin of Confidence**: difference between the top two most confident predictions
  \[
  1 - (P_\theta(y^*_1 | x) - P_\theta(y^*_2 | x))
  \]

- **Ratio of Confidence**: ratio between the top two most confident predictions
  \[
  P_\theta(y^*_2 | x) / P_\theta(y^*_1 | x)
  \]

- **Entropy**: the difference between all predictions, as defined by information theory
  \[
  - \sum_y P_\theta(y | x) \log_2 P_\theta(y | x) / \log_2(n)
  \]

- **Model-based Outliers**: sampling for low activation in logits and hidden layers to identify gaps in the model's knowledge

- **Advanced Active Learning**: combining multiple Active Learning techniques and incorporating Unsupervised Machine Learning, Domain Adaptation, and Transfer Learning

**Examples:**

The heatmaps show the differences between the Uncertainty Sampling methods on a three-label dataset. The methods use a probability distribution like:

\[
\text{prob} = \text{torch.tensor([0.032, 0.643, 0.087, 0.236])}
\]

most_conf = torch.max(prob)
labs = prob.numel()
numerator = (labs * (1 - most_conf))
denominator = (labs - 1)
least_conf = numerator / denominator

\[
\text{prob}, _ = \text{torch.sort(prob, descending=True)}
difference = (\text{prob.data[0]} - \text{prob.data[1]})
margin_conf = 1 - difference
\]

\[
\text{prob}, _ = \text{torch.sort(prob, descending=True)}
\]
ratio_conf = (prob.data[1] / prob.data[0])

prbslogs = prob * torch.log2(prob)
numerator = 0 - np.sum(prbslogs)
denominator = math.log2(prob.numel())
entropy = numerator / denominator

Code is open source & written to accompany: