Entropy maximization of neural network ouputs and entropy regularization of reinforcement learning

Tomas Ukkonen, *Novel Insight*, `tomas.ukkonen@novelinsight.fi`, 2021

1. Entropy maximization

Reinforcement learning requires maximization of entropy $H(\boldsymbol{Y})$ of outputs so there is enough exploration. In practise MSE minimized outputs can be tranformed to entropy by using formula:

$$H(\boldsymbol{Y}|\boldsymbol{w}) = -\sum_i \frac{\exp(\alpha\,\boldsymbol{y}(i|\boldsymbol{w}))}{\sum_j \exp(\alpha\,\boldsymbol{y}(j|\boldsymbol{w}))} \log\left(\frac{\exp(\alpha\,\boldsymbol{y}(i|\boldsymbol{w}))}{\sum_j \exp(\alpha\,\boldsymbol{y}(j|\boldsymbol{w}))}\right).$$

In practice we want to maximize $H(\boldsymbol{Y}|\boldsymbol{w})$ and calculate its gradient. First we calculate gradient of entropy function $H(\boldsymbol{Y}) = -\sum_i p_i \log(p_i)$ and we get $\frac{\partial H(\boldsymbol{Y})}{\partial \boldsymbol{w}} = -\sum_i \frac{\partial p_i}{\partial \boldsymbol{w}}(1+\log(p_i))$. Next, we must calculate the $\frac{\partial p_i}{\partial \boldsymbol{w}}$ term:

$$\frac{\partial p_i}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \frac{\exp(\alpha\,\boldsymbol{y}(i|\boldsymbol{w}))}{\sum_j \exp(\alpha\,\boldsymbol{y}(j|\boldsymbol{w}))}$$

$$=$$

$$\left(\alpha\exp(\alpha\,y(i))\,\frac{\partial y(i)}{\partial \boldsymbol{w}}\left(\sum_j \exp(\alpha\,\boldsymbol{y}(j|\boldsymbol{w}))\right) - \alpha\exp(\alpha\,y(i))\sum_j \exp(\alpha\,y(j))\frac{\partial y(j)}{\partial \boldsymbol{w}}\right) / \left(\sum_j \exp(\alpha\,\boldsymbol{y}(j|\boldsymbol{w}))\right)^2$$

This calculates entropy gradient of the neural network output. It can be calculated from the jacobian matrix $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$ given scaling factor alpha α or simply decide α to be one. This assumes outputs from the neural network are logits of probability values.

2. Entropy regularization of reinforcement learning

In practice, we assume error function is now softmax function of Q-values (linear output layer) and the target is also probability distribution of actions and we calculate Kullback-Leibler divergence between those values.

$$D_{KL}(\boldsymbol{Q}_{\text{target}}||\boldsymbol{P})[\boldsymbol{w}] = -\sum_i q(i)\log\left(\frac{p(i|\boldsymbol{w})}{q(i)}\right) = \sum_i q(i)(\log(q(i) - \log(p(i|\boldsymbol{w})))$$

We take the gradient of D_{KL} and get $\nabla_{\boldsymbol{w}} D_{KL} = -\sum_i [q(i)/p(i|\boldsymbol{w})]\nabla_{\boldsymbol{w}} p(i|\boldsymbol{w})$. For zero $p(i)$ values we add small positive epsilon value to probabilities to avoid division by zero. Next we must calculate gradient of softmax output of the neural network which we computed in the previous section. The result is:

$$\frac{\partial p_i}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \frac{\exp(\boldsymbol{y}(i|\boldsymbol{w}))}{\sum_j \exp(\boldsymbol{y}(j|\boldsymbol{w}))}$$

$$=$$

$$\left(\exp(y(i))\,\frac{\partial y(i)}{\partial \boldsymbol{w}}\left(\sum_j \exp(\boldsymbol{y}(j|\boldsymbol{w}))\right) - \exp(y(i))\sum_j \exp(y(j))\frac{\partial y(j)}{\partial \boldsymbol{w}}\right) / \left(\sum_j \exp(\boldsymbol{y}(j|\boldsymbol{w}))\right)^2$$

$$= \exp(y_i)\left(T(\boldsymbol{w})\,\frac{\partial y_i}{\partial \boldsymbol{w}} - \sum_j \exp(y_j)\frac{\partial y_j}{\partial \boldsymbol{w}}\right) / T(\boldsymbol{w})^2$$

$$= (\exp(y_i)/T(\boldsymbol{w})^2)\left(\sum_j (T(\boldsymbol{w})\delta(i-j) - \exp(y_j))\frac{\partial y_j}{\partial \boldsymbol{w}}\right)$$

$$= s(\boldsymbol{w}, i, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$$

$$s(\boldsymbol{w}, i, \boldsymbol{y})[j] = (\exp(y_i)/T(\boldsymbol{w})^2)\,(T(\boldsymbol{w})\delta(i-j) - \exp(y_j))$$

$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{w}} = S(\boldsymbol{w}, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$$

This last formula can be used as a basis for backpropagation. However, this is not fully usable yet because we must also use Kullback-Leible divergence formula which is

$$\nabla_{\boldsymbol{w}} D_{KL} = -\sum_i [q_i / p_i(\boldsymbol{w})] \nabla_{\boldsymbol{w}} p_i(\boldsymbol{w}) = \boldsymbol{e}(\boldsymbol{w})^T \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{w}} = \boldsymbol{e}(\boldsymbol{w})^T S(\boldsymbol{w}, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}} = \boldsymbol{d}(\boldsymbol{w}, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$$

This result gives the last layer error vector for backpropagation which is $\boldsymbol{d}(\boldsymbol{w}, \boldsymbol{y}) = \boldsymbol{S}(\boldsymbol{w}, \boldsymbol{y})\, \boldsymbol{e}(\boldsymbol{w})$ meaning we don't have to calculate whole jacobian matrix for calculating gradient.

This similar method can be also used to calculate gradient of entropy $H(\boldsymbol{Y}|\boldsymbol{w})$ of the output which is used to increase randomness of the selected function. In practise it is:

$$\nabla_{\boldsymbol{w}} H(\boldsymbol{Y}) = -\sum_i \frac{\partial p_i}{\partial \boldsymbol{w}} (1 + \log(p_i)) = \boldsymbol{h}(\boldsymbol{w})^T \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{w}} = \boldsymbol{h}(\boldsymbol{w})^T S(\boldsymbol{w}, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$$

In the discrete reinforcement learning step we use $\boldsymbol{Q}_{\text{new}}(s, \boldsymbol{a})$ as normally and calculate $\boldsymbol{q} = \text{softmax}(\boldsymbol{Q}_{\text{new}})$ to create a target values for Kullback-Leibler divergence based distance optimization to change action selection probabilities. Additionally we add the entropy term so we are minimizing recurrent neural network with the following error function:

$$e(\boldsymbol{w}) = \frac{1}{N} \sum_i D_{KL}(\text{softmax}(\boldsymbol{Q}_{\text{new},i}) || P(\boldsymbol{w})) - \alpha\, H(\boldsymbol{Y}|\boldsymbol{w})$$

And we select α term to be quite small, maybe $\alpha = 0.01$.

Reverse Kullback-Leibler divergence (Better for reinforcement learning)

Reverse KL-divergence should work better with reinforcement learning(???) so we derive reverse KL divergence

$$D_{KL}(\boldsymbol{P}_{\boldsymbol{w}} || \boldsymbol{Q}_{\text{target}})[\boldsymbol{w}] = -\sum_i p(i|\boldsymbol{w}) \log\left(\frac{q(i)}{p(i|\boldsymbol{w})}\right) = \sum_i p(i|\boldsymbol{w}) (\log(p(i|\boldsymbol{w})) - \log(q(i))$$

And the gradient is:

$$\nabla_{\boldsymbol{w}} D_{KL} = -\sum_i [1 + \log(p_i(\boldsymbol{w})) - \log(q_i)] \nabla_{\boldsymbol{w}} p_i(\boldsymbol{w}) = \boldsymbol{e}(\boldsymbol{w})^T \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{w}} = \boldsymbol{f}(\boldsymbol{w})^T S(\boldsymbol{w}, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}} = \boldsymbol{d}(\boldsymbol{w}, \boldsymbol{y})^T \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}}$$

So the implementation of the reverse gradient is easy.