Learning

Kirsty McNaught

March 2017

1 Synaptic Plasticity Rules

1.1 Basic Hebb Rule

"Neurons that fire together, wire together!"

For a single input:

$$\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = v\mathbf{u}$$

We assume that synaptic weights change slowly enough that we can average the weight changes for a larger set of input patterns:

$$\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = \langle v\mathbf{u} \rangle$$

1.1.1 Comments:

- ✓ Simultaneous pre- and postsynaptic firing increases synaptic strength.
- × Assuming that inputs are positive (i.e. spike count), the weights only ever increase.
- × The weights are unstable and grow without bound (see later proof)
- × There is no competition between different synapses

1.2 Correlation rules

In supervised learning, $v = \mathbf{w}^T\mathbf{u}$ and we can express the averaged Hebb rule using the input correlations:

$$\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = \langle v\mathbf{u} \rangle = \left\langle (\mathbf{w}^T\mathbf{u})\mathbf{u} \right\rangle = \left\langle \mathbf{u}(\mathbf{u}^T\mathbf{w}) \right\rangle = \left\langle \mathbf{u}\mathbf{u}^T \right\rangle \mathbf{w} = Q\mathbf{w}$$

1.2.1 Comments:

- Same remarks as for Hebb rule; just reformulation for supervised learning.

1.3 Covariance rule

The basic Hebb rule and the correlation rule only allow weight increases (LTP) and not weight decreases (LTD). To allow for negative weight updates, we can subtract a threshold from either the inputs or the outputs.

$$\begin{aligned} \tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} &= (v - v^*)\mathbf{u}; \text{ or} \\ &= v(\mathbf{u} - \mathbf{u}^*); \text{ or} \\ &= (v - v^*)(\mathbf{u} - \mathbf{u}^*) \end{aligned}$$

The last option is a terrible idea, since it implies a weight increase when both inputs and outputs are negative. The other two give the same averaged rule (though they have subtle differences for a single input), if the thresholds are taken as the mean values of v or $\mathbf{u}$. For example,

$$\begin{aligned}\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} &= \langle v(\mathbf{u} - \bar{\mathbf{u}}) \rangle \\ &= \left\langle (\mathbf{w}^T \mathbf{u})(\mathbf{u} - \bar{\mathbf{u}}) \right\rangle \\ &= \left\langle \mathbf{u}(\mathbf{u} - \bar{\mathbf{u}})^T \right\rangle \mathbf{w} \\ &= \left\langle (\mathbf{u} - \bar{\mathbf{u}})(\mathbf{u} - \bar{\mathbf{u}})^T \right\rangle \mathbf{w} \\ &= C\mathbf{w}\end{aligned}$$

1.3.1 Comments:

- ✓ Weights can increase and decrease
- × Weights are still unstable (see later)
- × Both forms have undesirable behaviour. If the right-hand side is $(v - v^*)\mathbf{u}$, then weights will decrease when the output v is zero. If the right-hand side is $v(\mathbf{u} - \mathbf{u}^*)$, then weights will be decreased when inputs are zero. In reality, weights should only change when there is (some) pre- and post-synaptic activity
- × For the version with $(v - \bar{v})\mathbf{u}$, the threshold $\bar{v}$ depends on weights and therefore needs updating during training.
- × Still non-competitive, but can be made competitive with sliding thresholds (see BCM).

1.4 The BCM rule

To prevent changes to weights when either v or u are zero, the BCM rule adds an initial multiplicative factor:

$$\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = v(v - v^*)\mathbf{u}$$

This is still unstable if v^* is fixed, but can be made stable by allowing the threshold to adapt. So long as the threshold grows more rapidly than the output, the updates will be stable. One possible rule allows the threshold to evolve according to v^2 :

$$\tau_v \frac{\mathrm{d}v^*}{\mathrm{d}t} = v^2 - v^*$$

Having a sliding threshold also enables competition - if some synapses are strengthened, the post-synaptic firing rate will increase, which will increase the threshold and push down less active synapses.

- ✓ Weights can increase and decrease
- ✓ Weights are stable
- ✓ Competition occurs between synapses

1.5 Stability of different rules

To analyse the stability of a weight update rule, we consider the time-evolution of the weights. This can usually be done by pre-multiplying both sides of the differential equation by $\mathbf{w}^T$ and noting that $\frac{\mathrm{d}|\mathbf{w}|}{\mathrm{d}t} = 2\mathbf{w}^T \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t}$.

For example, in the case of the basic Hebb rule:

$$\begin{aligned}\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} &= v\mathbf{u} \\ \tau_w \mathbf{w}^T \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} &= v\mathbf{w}^T\mathbf{u} = v^2 \\ \frac{\mathrm{d} \mid \mathbf{w} \mid}{\mathrm{d}t} &= 2v^2 \text{ (always positive unless } v = 0)\end{aligned}$$

2 Normalisation rules

The BCM rule stabilises Hebbian learning using a sliding threshold based on post-synaptic activity. This is equivalent to indirectly using the post-synaptic activity as a proxy for the size of the weights. We can instead apply normalisation directly to the weights themselves.

We will discuss two widely used schemes:

- Subtractive normalisation
- Multiplicative normalisation (Oja rule)

2.1 Subtractive normalisation

2.2 Multiplicative normalisation (Oja rule)

2.3

3 Reinforcement learning

3.1 Rescorla-Wagner rule

The R-W rule combines a linear predictor of reward $v = wu$ with a least-squares delta update. There is no time component; it just models Pavlovian conditioning.

Prediction:

$$v = \mathbf{w}^T \mathbf{u} \tag{1}$$

Square error:

$$\begin{aligned} E &= \frac{1}{2}(r - v)^2 \\ &= \frac{1}{2}\left\|r - \mathbf{w}^T \mathbf{u}\right\|^2 \\ \frac{\mathrm{d}E}{\mathrm{d}w} &= \frac{1}{2} \cdot 2(r - \mathbf{w}^T \mathbf{u}) \cdot (-u) \\ &= -(r - v)\mathbf{u} \end{aligned}$$

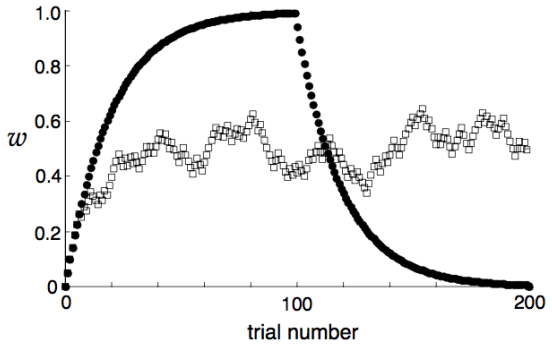
Update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \epsilon\delta\mathbf{u}; \qquad \delta = (r - v)$$

Comments: The R-W rule only allows updates to weights associated with stimuli which are present. This does not, for example, explain backward blocking. It also cannot explain secondary conditioning, wherein an unconditioned stimulus is presented just before a conditioned stimulus (but no reward), leading to an expectation of a reward (since a predictor of a predictor of a reward is itself a predictor of a reward).

The R-W rule uses the *prediction error* $\delta = (r - v)$ which has biological analogues in the activity of dopaminergic cells in the VTA.

The following figure from Dayan and Abbott shows (black circle) acquisition and extinction in pavlovian conditioning. A reward of 1 was given for the first 100 trials; zero reward for trials 100-200. In open circles, partial reinforcement is demonstrated - a reward of 1 was given 50% of the time, and a reward of 0 otherwise.



3.1.1 RW as a filter

We can use recursion to analyse the behaviour of the R-W rule over many trials. Let the initial weight be w_0 for a single-stimulus trial, i.e. $w \leftarrow w + \epsilon(r - w)$. The weights evolve as follows:

$$\begin{aligned} w_1 &= w_0 + \epsilon(r_1 - w_0) = \epsilon r_1 + (1 - \epsilon)w_0 \\ w_2 &= w_1 + \epsilon(r_2 - w_1) \\ &= \epsilon r_1 + (1 - \epsilon)w_0 + \epsilon\left[r_2 - (\epsilon r_1 + (1 - \epsilon)w_0)\right] \\ &= (1 - \epsilon)^2 w_0 + \epsilon r_2 + \epsilon(1 - \epsilon)r_1 \\ w_n &= (1 - \epsilon)^n w_0 + \sum_i^n \epsilon(1 - \epsilon)^{n-i} r_i \end{aligned}$$

The first term can be seen as an exponential decay of the initial weight (i.e. we slowly forget our starting value); the second term is an exponential filter (convolution) over recent rewards.

3.2 Temporal difference learning

TD learning accounts for time between stimuli and rewards. The goal is to predict the expected value of future rewards from any point in time.

$$v = \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle$$

Our predictor for $v(t)$ takes the form of a linear filter over previous stimuli. For a single time-dependent stimulus u(t), this would be:

$$v(t) = \sum_{\tau=0}^{t} w(\tau) u(t-\tau)$$

We'd like to use a delta rule to update this, of the form:

$$w(\tau) \leftarrow w(\tau) + \epsilon \delta(t) u(t-\tau)$$

The tricky bit here is how to compute $\delta(t)$. In theory this should represent the difference between 'current expected future reward' and 'true future reward', but we do not have knowledge of the latter. Instead we do some sneaky recursion as an approximation.

$$\begin{aligned} v(t) &= \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle = \langle r(t) \rangle + v(t+1) \\ \implies \langle r(t) \rangle &= v(t) - v(t+1) \end{aligned}$$

Our prediction error can now be estimated by comparing the expected reward at time t $\hat{r}(t) = v(t) - v(t+1)$ with the actual reward $r(t)$.

$$\begin{aligned} \delta(t) &= r(t) - \hat{r}(t) \\ &= r(t) - [v(t) - v(t+1)] \\ &= r(t) + v(t+1) - v(t) \end{aligned}$$

3.2.1 Results, comments

The following figure from Dayan and Abbott shows the predictions given by temporal difference learning when a reward is given a fixed period of time after the presentation of a stimulus. In particular, the prediction error moves backwards in time from the presentation of the reward (before conditioning has occurred) to the presentation of the stimulus (after conditioning). This is consistent with the activity of dopaminergic cells in the VTA.

Unlike Rescorla-Wagner, the TD rule provides an account of secondary conditioning.

3.3 Choosing actions

- Static action choice: the reward immediately follows the action (e.g. 2-armed bandit) - Sequential action choice: the reward may be delayed until several sequential actions have been performed (e.g. maze).

3.3.1 Static action choice: bandits

Consider a scenario in which the actor must choose between two (or more) actions at each timestep, and will receive a reward immediately. This is an example of a *static action choice*.

In both the following examples, we use a stochastic softmax policy.

$$P[a_i] = \frac{\exp(\beta m_i)}{\sum_j \exp(\beta m_j)}$$

The m_i values are known as *action values* or *action propensities*. The β parameter allows the policy to be tuned to allow for exploration (small β, only slightly takes into account beliefs of action values) or exploitation (large β, almost always chooses 'best' action).

We now need an algorithm to allow us to update our action values m_i.

Indirect actor: In an indirect scheme, the policy is mediated indirectly by the expected volumes. We aim to make $m_i \to \langle r_i \rangle$ using a delta rule.

$$m_i \leftarrow m_i + \epsilon\delta; \qquad \delta = r_i - m_i$$

Direct actor: Instead of basing choice on average rewards, we might choose our action values to directly maximise the expected average reward. We can do this with stochastic gradient descent.

$$\begin{aligned}
\langle r \rangle &= P[a] \langle r_a \rangle + P[b] \langle r_b \rangle \\
\frac{\mathrm{d}\langle r \rangle}{\mathrm{d}m_a} &= \frac{\mathrm{d}\langle r \rangle}{\mathrm{d}P[a]} \frac{\mathrm{d}P[a]}{\mathrm{d}m_a} + \frac{\mathrm{d}\langle r \rangle}{\mathrm{d}P[b]} \frac{\mathrm{d}P[b]}{\mathrm{d}m_a} \\
\frac{\mathrm{d}\langle r \rangle}{\mathrm{d}m_b} &= \frac{\mathrm{d}\langle r \rangle}{\mathrm{d}P[a]} \frac{\mathrm{d}P[a]}{\mathrm{d}m_b} + \frac{\mathrm{d}\langle r \rangle}{\mathrm{d}P[b]} \frac{\mathrm{d}P[b]}{\mathrm{d}m_b}
\end{aligned}$$

Where our probabilities $P[a], P[b]$ are defined by softmax:

$$\begin{aligned}
P[a] &= \frac{\exp(\beta m_a)}{\exp(\beta m_a) + \exp(\beta m_b)} \\
\frac{\mathrm{d}P[a]}{\mathrm{d}m_a} &= \frac{vdu - udv}{v^2} \\
&= \frac{\beta \exp(\beta m_a) \left[\exp(\beta m_a) + \exp(\beta m_b) - \exp(\beta m_a)\right]}{(\exp(\beta m_a) + \exp(\beta m_b))^2} \\
&= \beta \cdot \frac{\exp(\beta m_a)}{\exp(\beta m_a) + \exp(\beta m_b)} \cdot \frac{\exp(\beta m_b)}{\exp(\beta m_a) + \exp(\beta m_b)} \\
&= \beta P[a]P[b] \quad \text{or} \quad \beta P[a](1 - P[a])
\end{aligned}$$

The same can be derived for $dP[a]/m_b$, $dP[b]/m_a$ and $dP[b]/m_b$, giving the final derivative:

$$\frac{\mathrm{d}\langle r \rangle}{\mathrm{d}m_a} = \beta P[a](1 - P[a]) \langle r_a \rangle - \beta P[b]P[a] r_b$$

We now note that we can split the derivative up into two parts, weighted by the likelihood of the respective actions: the first term associated with trials in which action a is selected; the second term with trials in which action b is selected. We also note that the averages are invariant to an arbitrary offset r^*, giving the final updates:

$$\begin{aligned}
m_a &\leftarrow m_a + \epsilon\beta(1 - P[a])(r_a - r^*) \qquad \text{when a selected;} \\
m_a &\leftarrow m_a - \epsilon\beta P[a](r_b - r^*) \qquad \text{when b selected}
\end{aligned}$$

Or more generally, where a is the action taken and i the value being updated:

$$m_i \leftarrow m_i + \epsilon\beta(\delta_{a=i} - P[i])(r_a - r^*)$$

This learning rule performs stochastic gradient descent regardless of the value of r^*. The only thing that changes with the choice of r^* is the variance of the stochastic updates, and hence the learning rate. Typically we'd choose $r^* = \langle r \rangle$ (or some estimate of it).

3.3.2 Indirect vs direct

In the indirect case, the scale of m_a are fixed (with units of reward), and the exploration parameter β (units 1/reward) can be scaled when the scale of rewards change. Even in an unchanging world it allows more explicit control over exploration vs exploitation, since the m_a values in a direct actor setup are unbounded and can scale arbitrarily to offset β.

3.4 Actor-critic

3.5 Q learning

3.6 SARSA

3.6.1 Q vs SARSA

3.7 Perceptron