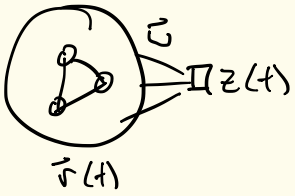
2019.04.11

Derivation of recursive least squares

Consider an RNN with a 1D readout z:



$$z(t') = \sum_{i=1}^{N} w_i(t) r_i(t') \qquad (1)$$

where t is the most recent timestep (so $t' \leq t$).

The goal is to choose weights $\vec{w}(t) = \hat{\vec{w}}(t)$ that minimize the loss:

$$L(t) = \sum_{t'=1}^{t} \lambda^{t-t'} \left[\hat{z}(t') - z(t')\right]^2 + \alpha \lambda^t \sum_{i=1}^{N} \left[w_i(t)\right]^2 \qquad (2)$$

[The second term is necessary so that the problem is well-posed (i.e. Φ^{-1} exists later) even if $t < N$.]

Plan:

1) Minimize $L(t)$ w.r.t. $\vec{w}(t)$ to get an equation for $\hat{\vec{w}}(t)$.

2) Note that the resulting equation can be solved recursively, i.e.

$$\vec{w}(t) = \vec{w}(t-1) + \text{new stuff}$$

Ref: Adaptive Filter Theory, S. Haykin

(2)

First, minimize* (2):

* Note this differs from grad. descent. Here we can set $\partial L/\partial w = 0$ since the problem is linear.

$$0 = \frac{\partial L(t)}{\partial w_i}$$

$$= -2\sum_{t'=1}^{t} \lambda^{t-t'} [\hat{z}(t') - z(t')] \frac{\partial z(t')}{\partial w_i} + 2\alpha\lambda^t w_i(t)$$

$$\overset{(1)}{=} -2\sum_{t'=1}^{t} \lambda^{t-t'} [\hat{z}(t') - z(t')] r_i(t') + 2\alpha\lambda^t w_i(t)$$

$$\overset{(1)}{=} -2\sum_{t'=1}^{t} \lambda^{t-t'} \hat{z}(t') r_i(t') + 2\sum_{t'=1}^{t} \lambda^{t-t'} \sum_j w_j(t') r_j(t') r_i(t')$$

$$+ 2\alpha\lambda^t w_i(t)$$

$$\Rightarrow \sum_j \left[\alpha\lambda^t \delta_{ij} + \sum_{t'=1}^{t} \lambda^{t-t'} r_i(t') r_j(t')\right] \hat{w}_j(t)$$

$$= \sum_{t'=1}^{t} \lambda^{t-t'} \hat{z}(t') r_i(t')$$

$$\Rightarrow \sum_j \Phi_{ij}(t) \hat{w}_j(t) = u_i(t) \qquad (3)$$

where we have defined

$$\Phi_{ij}(t) \equiv \alpha\lambda^t \delta_{ij} + \sum_{t'=1}^{t} \lambda^{t-t'} r_i(t') r_j(t') \qquad (4)$$

$$u_i(t) \equiv \sum_{t'=1}^{t} \lambda^{t-t'} \hat{z}(t') r_i(t') \qquad (5)$$

and used the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

(3)

Eq. (3) can be inverted to get $\hat{\vec{w}}(t)$:

$$\hat{\vec{w}}(t) = \Phi^{-1}(t) \cdot \vec{u}(t) \tag{6}$$

But just inverting $\Phi(t)$ numerically at every t would be very expensive.

We can do better by noting that (4) and (5) have recursive forms:

$$\Phi_{ij}(t) = \lambda \left[\alpha \lambda^{t-1} \delta_{ij} + \sum_{t'=1}^{t-1} \lambda^{t-t'-1} r_i(t') r_j(t') \right] + r_i(t) r_j(t)$$

$$= \lambda \Phi_{ij}(t-1) + r_i(t) r_j(t) \tag{7}$$

$$u_i(t) = \lambda u_i(t-1) + \hat{\xi}(t) r_i(t) \tag{8}$$

What we really want, though, is a recursion relation for $\Phi^{-1}(t)$, not for $\Phi(t)$. It turns out that, since the last term in (7) has rank $1 \ll N$, this can be done easily. In general, if

$$A = B + \vec{r}\vec{r}^{\,T} \tag{9}$$

then

$$A^{-1} = B^{-1} - \frac{B^{-1} \cdot \vec{r}\,\vec{r}^{\,T} \cdot B^{-1}}{1 + \vec{r} \cdot B^{-1} \cdot \vec{r}} \tag{10}$$

④ Proof:

$$A \cdot A^{-1} = \left(B + \vec{r}\vec{r}^T\right) \cdot \left(B^{-1} - \frac{B^{-1} \cdot \vec{r}\vec{r}^T \cdot B^{-1}}{1 + \vec{r} \cdot B^{-1} \cdot \vec{r}}\right)$$

$$= \mathbb{I} + \vec{r}\vec{r}^T \cdot B^{-1} - \frac{\mathbb{I} \cdot \vec{r}\vec{r}^T \cdot B^{-1} - \vec{r}\vec{r}^T \cdot B^{-1} \cdot \vec{r}\vec{r}^T \cdot B^{-1}}{1 + \vec{r} \cdot B^{-1} \cdot \vec{r}}$$

$$= \mathbb{I} + \vec{r}\vec{v}^T - \frac{\vec{r}\vec{v}^T - \vec{r}\left(\vec{v}^T \cdot \vec{r}\right)\vec{v}^T}{1 + \vec{r} \cdot \vec{v}}$$

$$= \mathbb{I} + \frac{\vec{r}\vec{v}^T - \left(\vec{r} \cdot \vec{v}\right)\vec{r}\vec{v}^T - \vec{r}\vec{v}^T + \left(\vec{r} \cdot \vec{v}\right)\vec{r}\vec{v}^T}{1 + \vec{r} \cdot \vec{v}}$$

$$= \mathbb{I}$$

where $\mathbb{I}$ is the identity matrix, and $\vec{v} \equiv \vec{r}^T \cdot B^{-1}$.

So, letting $A = \Phi(t)$ and $B = \lambda\Phi(t-1)$, (7) becomes

$$\Phi^{-1}(t) = \frac{1}{\lambda}\Phi^{-1}(t-1) - \frac{\Phi^{-1}(t-1) \cdot \vec{r}(t)\vec{r}^T(t) \cdot \Phi^{-1}(t-1)}{\lambda^2 + \lambda\vec{r}(t) \cdot \Phi^{-1}(t-1) \cdot \vec{r}(t)}$$

Or, letting $P(t) \equiv \Phi^{-1}(t)$,

$$P(t) = \frac{1}{\lambda}\left[P(t-1) - \frac{P(t-1) \cdot \vec{r}(t)\vec{r}^T(t) \cdot P(t-1)}{\lambda + \vec{r}(t) \cdot P(t-1) \cdot \vec{r}(t)}\right] \quad (11)$$

⑤ For convenience, define the "gain vector":

$$\vec{k}(t) \equiv \frac{P(t-1)\cdot\vec{r}(t)}{\lambda + \vec{r}(t)\cdot P(t-1)\cdot\vec{r}(t)} \tag{12}$$

and note that

$$P(t)\cdot\vec{r}(t) = \frac{1}{\lambda}\left[P(t-1) - \frac{P(t-1)\cdot\vec{r}(t)\vec{r}^{T}(t)\cdot P(t-1)}{\lambda + \vec{r}(t)\cdot P(t-1)\cdot\vec{r}(t)}\right]\cdot\vec{r}(t)$$

$$= \frac{1}{\lambda}P(t-1)\cdot\vec{r}(t)\left[1 - \frac{\vec{r}(t)\cdot P(t-1)\cdot\vec{r}(t)}{\lambda + \vec{r}(t)\cdot P(t-1)\cdot\vec{r}(t)}\right]$$

$$= \frac{P(t-1)\cdot\vec{r}(t)}{\lambda + \vec{r}(t)\cdot P(t-1)\cdot\vec{r}(t)}$$

$$= \vec{k}(t) \tag{13}$$

With (8) and (11)-(13), the weight update equation (6) is

$$\hat{\vec{w}}(t) = P(t)\cdot\vec{u}(t)$$

$$\overset{(8)}{=} P(t)\cdot\left[\lambda\vec{u}(t-1) + \hat{z}(t)\vec{r}(t)\right]$$

$$\overset{(13)}{=} \lambda P(t)\cdot\vec{u}(t-1) + \hat{z}(t)\vec{k}(t)$$

$$\overset{(11)}{=} \left[P(t-1) - \frac{P(t-1)\cdot\vec{r}(t)\vec{r}^{T}(t)\cdot P(t-1)}{\lambda + \vec{r}(t)\cdot P(t-1)\cdot\vec{r}(t)}\right]\cdot\vec{u}(t-1)$$

$$+ \hat{z}(t)\vec{k}(t)$$

(6)

$$= P(t-1)\cdot\vec{u}(t-1) - \vec{k}(t)\vec{r}^{T}(t)\cdot P(t-1)\cdot\vec{u}(t-1) \quad (17)$$

$$+ \hat{z}(t)\vec{k}(t)$$

$$= \hat{\omega}(t-1) - \vec{k}(t)\left[\vec{r}(t)\cdot\hat{\omega}(t-1)\right] + \hat{z}(t)\vec{k}(t) \quad (6)$$

$$= \hat{\omega}(t-1) + \vec{k}(t)\left[\hat{z}(t) - \hat{\omega}(t-1)\cdot\vec{r}(t)\right] \quad (14)$$

Letting $\vec{\xi}(t) \equiv \hat{z}(t) - \hat{\omega}(t-1)\cdot\vec{r}(t)$, this is

$$\boxed{\hat{\omega}(t) = \hat{\omega}(t-1) + \xi(t)\vec{k}(t)} \quad (15)$$

$$= \hat{\omega}(t-1) + \text{new stuff}$$

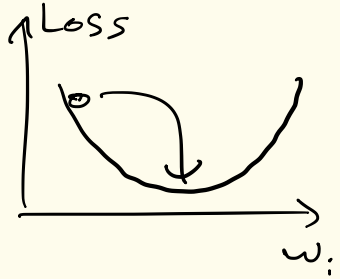
as promised.

(7)

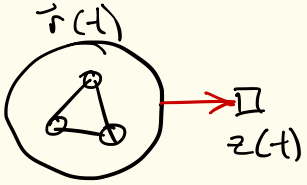
Discussion

- RLS differs from gradient-based methods (e.g. BPTT and RTRL).
  - Grad-based methods solve a nonlinear problem in many small steps.

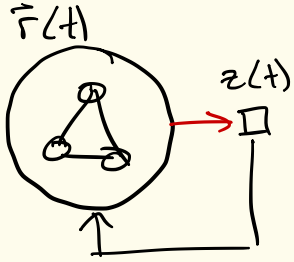
  - RLS solves a linear problem all at once. The error is always small.



- The difficulty with RLS is how to get a basis set $\{r_i(t)\}$ for constructing the output $z(t)$ that is (i) sufficiently rich, and (ii) stable.

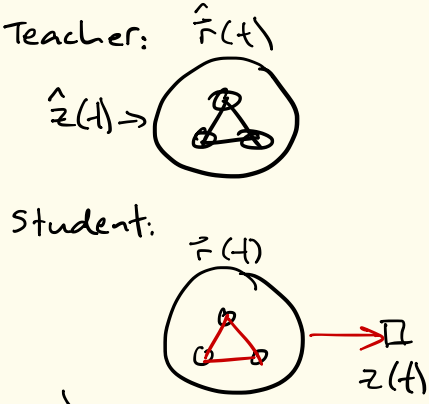


- In the echo state network (Jaeger and Haas, 2004), the RNN is driven with the target $\hat{z}(t)$, and recurrent weights are small.



- In FORCE learning (Sussillo and Abbott, 2009), recurrent weights are large, and the RNN is driven by its own output $z(t)$.
  - Using $z(t)$ rather than $\hat{z}(t)$ as feedback might make the trained network more robust since it introduces noise.
  - Large recurrent weights are good because they lead to a rich basis set $\{r_i(t)\}$. Chaos isn't a problem, though, since feedback suppresses it (Rajan et al, 2010).

- In Full-FORCE, a teacher RNN provides targets $\hat{r}(t)$ for training recurrent weights in a student RNN (DePasquale et al, 2018).



Relation to Kalman filter

Rename variables:

$\hat{\vec{w}} \to \vec{x}(t)$, Target weights → Latent var.

$\vec{w}(t) \to \hat{\vec{x}}(t)$, Readout weights → Est. latent var.

$\vec{r}(t) \to \vec{c}(t)$, RNN activity → Readout weights

$z(t) \to z(t)$, RNN readout → Readout

$\varepsilon(t) \to v(t)$, Readout error → Readout noise

In this language, our problem is, given the update rules

$$\vec{x}(t+1) = \vec{x}(t)$$

$$z(t) = \vec{c}(t) \cdot \vec{x}(t) + v(t),$$

and assuming that $z(t)$ and $c(t)$ are known but $\vec{x}(t)$ is not, to find the

best estimate $\hat{\vec{x}}(t)$ to reproduce $\vec{z}(t)$.

This is a special case of the Kalman filtering problem:

$$\vec{x}(t+1) = F(t+1, t) \cdot \vec{x}(t) + \vec{v}_1(t)$$

$$\vec{z}(t) = C(t) \cdot \vec{x}(t) + \vec{v}_2(t)$$

where $\vec{v}_{1,2}(t)$ are white noise with known variance. The goal is then, given C, F, and $\{\vec{z}(1), \ldots, \vec{z}(t)\}$, to find the best estimate $\hat{\vec{x}}(t')$ of $\vec{x}(t')$. This can have different names:

$$\begin{cases} t' = t \rightarrow \text{"filtering"} \\ t' < t \rightarrow \text{"smoothing"} \\ t' > t \rightarrow \text{"prediction"} \end{cases}$$

The solution looks very similar to the RLS algorithm defined earlier (see Haykin, ch. 10).