# SOM-Based Routing

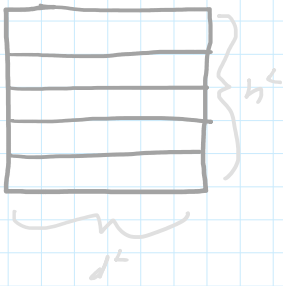
Παρασκευή, 23 Σεπτεμβρίου 2022 5:03 πμ

Let's begin from Primary Capsules

Primary Caps Shape: $[B, n^L, d^L]$, where $n^L = w^L \times h^L \times F^L$

For the schematic representations, let's assume that $n^L = 5$, $h^{L+1} = 2$, $m = 3$ (optional)

Under those assumptions, $C_b^L$:




$S_{simsOut_b}^L$:


**Input: PrimaryCaps $C^L$**

**Output: Similarities $S_{simsOut}^L$**

$S_{simsOut}^L.shape: [B, h^{L+1}]$

Trainable Parameters: W (transformation Matrices)

Non-Trainable Parameters: $C^{L+1}$

$C^{L+1}.shape: [h^{L+1}, d^{L+1}]$

Hyperparameters: reduced_votes, m, normalize_votes, norm_type,

- **reduced_votes**: If True, then each capsule C_i^L produces n^{L+1} votes. If False, each capsule produces m votes, where m is not necessarily equal to n^{L+1}.
- **m**: Transformation Matrices for each capsule. This is used only if reduced_votes ==False. So, it specifies the number of projections that Each capsule C_j^{L+1} has.
- **normalize_votes**: If we set normalize_votes hyperparameter to True then we transform the vote vectors so as every one of them has length no more than one.
- **norm_type**: If we set it to 0 then we apply squashing function as a normalization tecknique. If we set it to 1 we apply tanh normalization to votes and unit normalization to output capsules.

r, radical, Θ, take_into_accont_win_ratio,

- **r**: Number of routing iterations. In each iteration, Capsules C_j^{L+1} are updated once.
- **radical**: If radical is set to True then instead of computing the differences as the votes minus the digit caps, we set the differences to be same as the votes. Radical approach was inspired by the perceptron algorithm (update rule) and differs significantly from the SOM's update rule.
- **Θ**: Specifies the neighborhood function. It is a list of float numbers that determine the gravity of the update for the second, third place. Always, the winner should get the full update so theta at least should contain 1.0 . It is worth mentioning that unlike SOM, our lateral distance is the relative to the winner parent capsule distance.
- **take_into_accont_win_ratio**: If True then the updates for each digit capsule are scaled according to the number of votes they "won" in a batch.

take_into_account_similarity, softmax, tanh_like, normalize_in_loop,

- **take_into_account_similarity**: If it is set to True then the updates are scaled according to the similarity. Also, if set, the individual thetas in the theta list are not taken into consideration. Just the length of the list to define how many inner loop iterations.
- **softmax**: Id set to True then instead of hard winners we get soft winners. So even capsules that do not have any winner may get updated with a small coefficient. Of course then there is no point in using a neighborhood.
- **tanh_like**: Similar to softmax. Just the operation is scaled so that dissimilar parent capsules move further away when updated.
- **normalize_in_loop**: Used to normalize digit capsules in every iteration of the outer loop. This, if many iterations are used improves stability.

lr_SOM

- **lr_SOM**: Learning rate used in SOM-based updates.

# Start of SOM-Based Routing

At first we compute the votes using the transformation matrices. Depending on the value of the reduced_votes hyperparameter, we use different transformation matrices and get different votes.

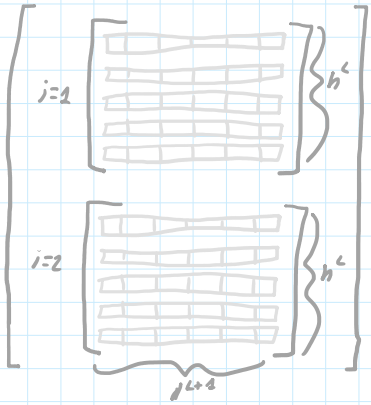
If reduced_votes == True:

$W^L$.shape : $[n^L, d^L, d^{L+1}, n^{L+1}]$

- $\forall b \in [1, B], \forall i \in \Omega_L, \forall j \in \Omega_{L+1} : V^L_{bij:} \leftarrow C^L_{bi:} \times W^L_{i:::j}$
  $[1 \times d^L]$ $[d^L \times d^{L+1}]$

$V^L$.shape : $[B, n^L, n^{L+1}, d^{L+1}]$

$V^L_b$ :




else :

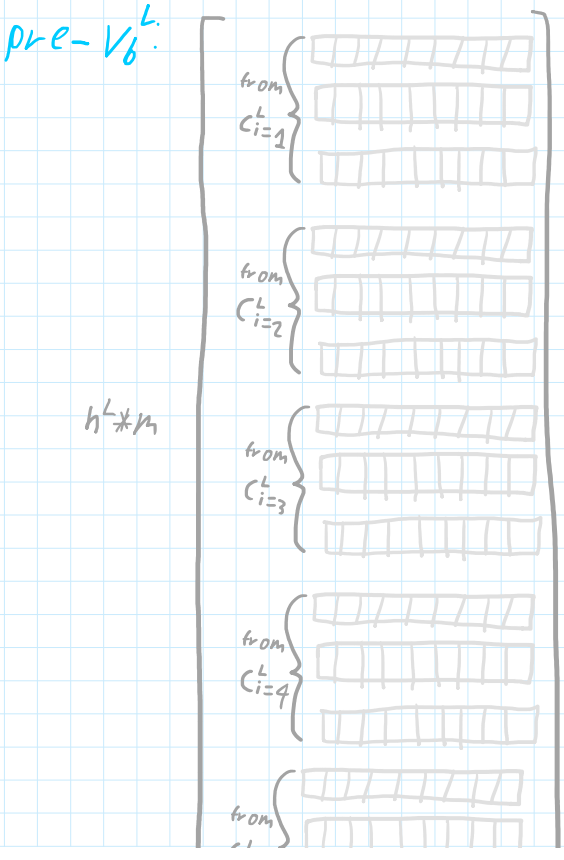
$W^L$.shape $[n^L, d^L, d^{L+1}, m^L]$

- $\forall b \in [1, B], \forall i \in \Omega_L, \forall j_m \in [1, m] :$ $pre\text{-}V^L_{b(i+n^L*(j_m-1))d^{L+1}} \leftarrow C^L_{bi:} \times W^L_{i:::j_m}$
  $[1 \times d^L]$ $[d^L \times d^{L+1}]$

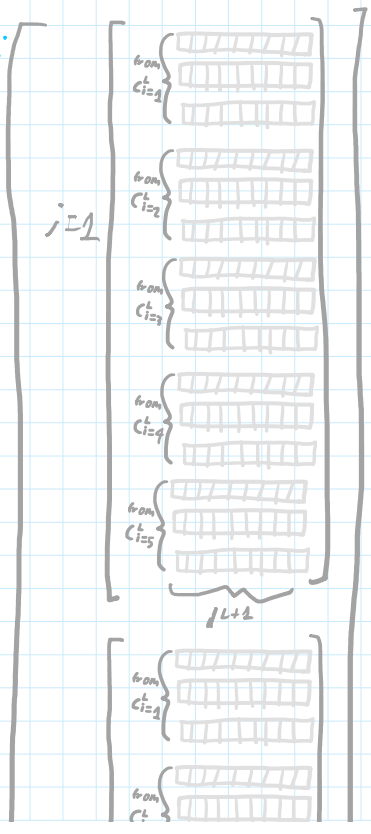
Actually, we compute m projections for each one of the n^L primary capsules and then we concatenate (we get them all together so we lose track of which vote comes from which primary capsule). We will still use the same index "i" but this, no longer means that this particular vote V_{ij}^L comes from capsule C_i^L.

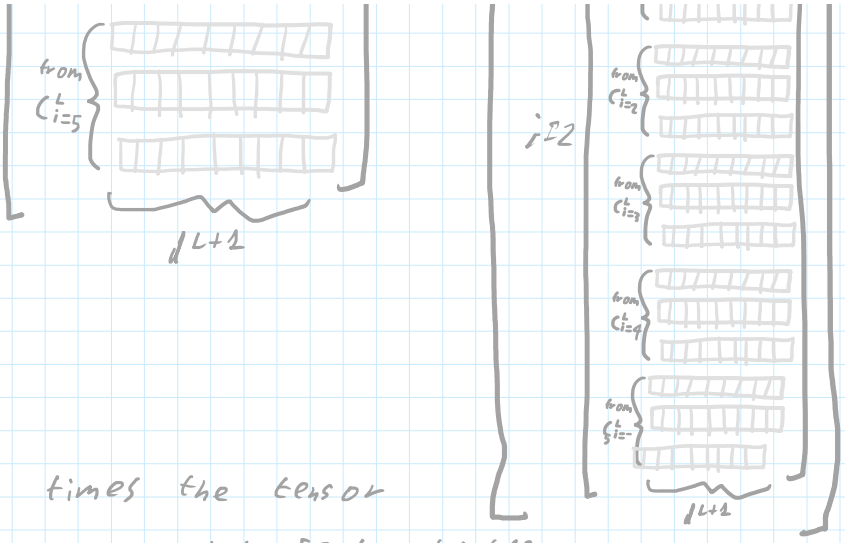
Now $i \in [1, n^L * m]$ so $pre\text{-}V^L_{bi:}$ comes from $C^L_{bi/m}$.

$pre\text{-}V^L_b$ :




$V^L_b$ :






Copy $n^{L+1}$ times the tensor pre-$V^L$ so as to get V.shape: $[B, n^L * m, n^{L+1}, d^{L+1}]$

- $\forall\, j \in \Omega_{L+1}: V^L_{:,:,j,:} \leftarrow$ pre-$V^L$

In other words, stack $n^{L+1}$ copies of pre-$V^L$ vertically.

So, V.shape: $[B, \underbrace{n^L * m}_{\underline{n}^L}, n^{L+1}, d^{L+1}]$

$$\underline{n}^L = \begin{cases} n^L & \text{iff reduced\_votes == True} \\ n^L * m & \text{iff reduced\_votes == False} \end{cases}$$

$$\underline{\Omega}_L = \begin{cases} \Omega_L & \text{iff reduced\_votes == True} \\ \text{set of all votes in tensor pre-}V^L_b, & \text{iff reduced\_votes == False} \end{cases}$$

So... in either case:

V.shape: $[B, \underline{n}^L, n^{L+1}, d^{L+1}]$

if normalize_votes:

Make vote vectors to not exceed length of 1 (L2 norm).

if norm_type == 0:

- $\forall b \in [1, B],\ \forall i \in \underline{\Omega}_L,\ \forall j \in \Omega_{L+1}: V^L_{b,i,j} \leftarrow \text{Squash}(V^L_{b,i,j})$

$$\frac{\|V^L_{b,i,j}\|^2}{2+\|V^L_{b,i,j}\|^2} \cdot \frac{V^L_{b,i,j}}{\|V^L_{b,i,j}\|}$$

else:

- $\forall b \in [1, B],\ \forall i \in \underline{\Omega}_L,\ \forall j \in \Omega_{L+1}: V^L_{b,i,j} \leftarrow \underbrace{\tanh(\|V^L_{b,i,j}\|)}_{\text{scaling term}} \underbrace{\frac{V^L_{b,i,j}}{\|V^L_{b,i,j}\|}}_{\text{unit norm}}$

Result has same shape as before.

If take_into_account_win_ratio:

Initialize WinCount with zeros, WinCount.Shape: $[n^{L+1}]$

for r iterations do:

Initialize SV with zeros, SV.shape: $[B, \underline{n}^L, n^{L+1}, d^{L+1}]$
↳ sparse updates

Time to compute the differences between the $V^L$ and $C^{L+1}$. These differences "D" will be used in the update rule to update the parent capsules $C^{L+1}$.

If radical:

- $D^L \leftarrow V^L$

Radical updates is inspired by Perceptron rule. This is significantly different to the updates computed in the original SOM.

original SOM update rule:
$C_j^{L+1}(s+1) \leftarrow C_j^{L+1}(s) + \alpha(s) * \Theta(c_{(s)}, j)_{BMU} * (\overbrace{V_{bi}^L - C_j^{L+1}(s)}^{D^L})$
↓ SOM learning rate, ↓ neighborhood function

else:

- $\forall b \in [1, B],\ \forall i \in \underline{\Omega}_L:\ D^L_{bi::} \leftarrow V^L_{bi} - C^L$
  $[n^{L+1}, d^{L+1}]$ $[n^{L+1}, d^{L+1}]$

Even if reduced_votes == False, if radical is not True, $D^L$ tensor has different entries for different j's (unlike $V^L$ which contains $n^{L+1}$ copies of pre-$V^L$ iff reduced_votes == False). That is because pre-$V^L$ votes (in $V^L$) are subtracted by different parent capsules.

It is clear that the above update rule can not be parallelizable. Finding the "winner" (Best Matching Unit - BMU) parent capsule for each datapoint - vote $V^L_{bij}$ and then updating $C^{L+1}$ in a serial manner would have been very slow.

We need a new update rule that can be parallel, across parent capsules and across batch items.

$D^L$.Shape: $[B, \underline{n}^L, n^{L+1}, d^{L+1}]$

Initialize mask $M^L$ with ones, $M^L$.Shape: $[B, \underline{n}^L, n^{L+1}, d^{L+1}]$

for each $\theta$ in $\Theta$:

Our metric for similarity is the inner product. By this criterion we pick the BMUs (winners).

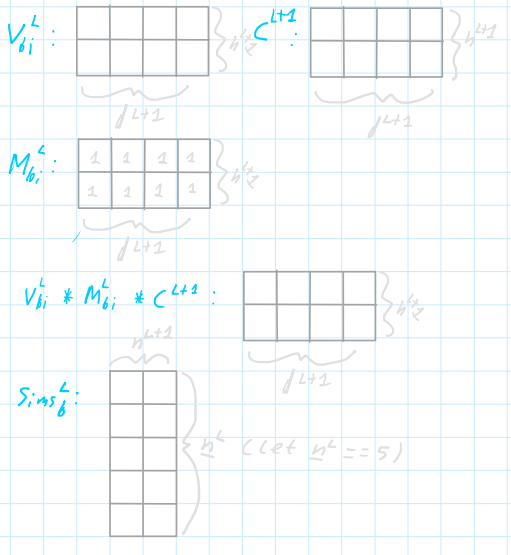
This is another difference to the original SOM, the criterion used there is the Euclidian Distance.

- $\forall b \in [1,B], \forall i \in \underline{\Omega}_L:\ Sims^L_{bi} \leftarrow \sum_k^{d^{L+1}} \left[ \left( V^L_{bi} * M^L_{bi} \right) * C^{L+1} \right]_{bik}$
  $[n^{L+1}, d^{L+1}]$ $[n^{L+1}, d^{L+1}]$

$Sims^L$.Shape: $[B, \underline{n}^L, n^{L+1}]$

$Sims^L$ contains all the similarities between the parent capsules $C^{L+1}$ and their corresponding pose-predictions (votes $V^L$), for each batch.

Take note that iff reduced_votes == False, then all the votes (pre-$V^L$) are compared to all the parent capsules. So no vote is tied to a particular parent capsule. We leave it to the SOM-based routing to make the descrimination. In other words, iff reduced_votes == False, $C^{L+1}$ have same "view" of data.



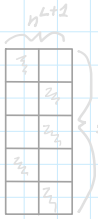

If not softmax nor tauh:

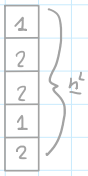
We proceed by finding the Best Matching Units (winners). Competition is, as it should be, between the $C^{L+1}$ as they are trying to explain the datapoints.

In our SOM-analogy, the "datapoints" are in our case the votes while the "nodes" are the parent capsules $C^{L+1}$.

Matching Units (winners). Competition is, as it should be, between the $C^{L+1}$ as they are trying to explain the datapoints (votes). The parent capsule that explains best a child capsule $C^L_{bi}$ (i.e. has the largest similarity with the corresponding vote) wins.

} the votes while the "nodes" are the parent capsules $C^{L+1}$.

$Sims^L_b$: 


$J^L_{winners_b}$: argmax(.) 


- $\forall b \in [1, B], \forall i \in \Omega_L : J^L_{winners_{bi}} \leftarrow \underset{j}{argmax}(Sims^L_{bi:})$

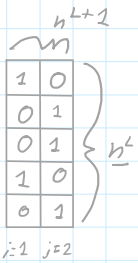
  $J^L_{winners}.shape : [B, n^L]$

  $J^L_{winners_b}$ contains one winner index per child capsule.

- $\forall b \in [1, B] : M^L_{winners_b} \leftarrow OneHot(J^L_{winners_b}, n^{L+1})$

  $M^L_{winners}$ is a mask that has 1's on the positions of sims where the maximum value (across parent capsules) is located.

  $M^L_{winners}.Shape : [B, n^L, n^{L+1}]$

$M^L_{winners_b}$: 


**else if softmax:**

we produce "soft" winners.

- $\forall b \in [1, B] : M^L_{winners_b} \leftarrow \underset{across\ j}{Softmax}(Sims^L_b)$

  Where $M^L_{winners}.Shape : [B, n^L, n^{L+1}]$

  and $\underset{across\ j}{Softmax}(Sims^L_b) = \begin{bmatrix} softmax(Sims^L_{bi=1}) \\ softmax(Sims^L_{bi=2}) \\ \vdots \\ softmax(Sims^L_{bi=n^L}) \end{bmatrix}$

**else if tanh_like**

With tanh_like dissimilar parent capsules will point further away.

- $\forall b \in [1, B] : M^L_{winners_b} \leftarrow \left( \underset{across\ j}{Softmax}(Sims^L_b) - 0.5 \right) * 2$

  where "−" and "*" are pointwise operations.

  Again, $M^L_{winners}.shape : [B, n^L, n^{L+1}]$

**If not take_into_account_similarity and not take_into_account_winner_ratios:**

Original, simple case: Compute the updates of the winning nodes (capsules). For the first iteration of thetas, we will update the winners. On the next iterations we will update the winners 1st degree neighbours. On the third iteration of thetas (if, of course the hyperparameter $\Theta$ contains 3 items) we will update the 2nd degree neighbours of the winning nodes.

- $\forall b \in [1, B], \forall i \in \Omega_L : Usparce^L_{bi} \leftarrow \left( M^L_{winners_{bi}} \times P^L_{bi} \right) * lr\_SOM * \theta$

* $\forall\, b \in [1,B],\ \forall\, i \in \Omega_L$ : $Usparce^L_{bi} \leftarrow \left( M^L_{winners_{bi}} \times P^L_{bi} \right) * lr\_SOM * \theta$

$M^L_{winners_{bi}}$ : $[h^{L+1}]$ ; $P^L_{bi}$ : $[h^{L+1}, d^{L+1}]$ ; $*$ → pointwise

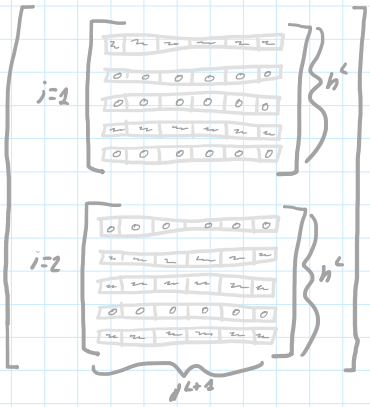
Masked Differences

$Usparce^L$ contains the updates of selected capsules. If on the first iteration, then $Usparce^L$ contains the updates of the winner capsules.

$Usparce^L.shape : [B, n^L, h^{L+1}, d^{L+1}]$

$U^L_{sparce}$ :

similar shapes for $P^L_b$ and $M^L_{winners\,b}$.




As you can see (I suppose), each parent capsule $C^{L+1}_j$ will be updated according to the capsules $C^L_{bi}$ that it attracted.

Child capsules that a parent capsule $C^{L+1}_j$ did not won, will not contribute to it's update (thus, their row will be zero).

That is the case for the 1st iteration. On the next iteration, the capsule $C^{L+1}_j$ that won children capsule $C^L_{bi}$ can not "claim" that capsule again. On the second iteration, the winner capsule for datapoint $C^L_{bi}$ will now be the parent capsule $C^{L+1}_{j'}$ that has the second best similarity.

One significant difference between our SOM and the original is this: Instead of iteratively finding the one parent capsule that best explains a datapoint (a vote $V^L_{bi}$) and updating that parent capsule with that vote, we find many winners (BMU's) and update them in parallel.

The above note is another major difference to the original SOM algorithm. Lateral distance, in our case, is defined as the rank distance on the similarity "leaderboard".

else if take_into_account_similarity:

We will perform similar operations as before. The difference now is that we will scale the Mask of the winners according to the similarities. In that way, the gravity of the update will be proportional to the agreement.

- $SimsSparce^L \leftarrow Sims^L * M^L_{winners}$

  $[B, n^L, n^{L+1}]$ → pointwise $[B, n^L, n^{L+1}]$

  $SimsSparce^L.shape : [B, n^L, n^{L+1}]$

- $b \in [1,B],\ \forall\, i \in \Omega_L$ : $Usparce^L_{bi} \leftarrow \left( SimsSparce^L_{bi} \times P^L_{bi} \right) * lr\_SOM$

  $SimsSparce^L_{bi}$ : $[h^{L+1}]$ ; $P^L_{bi}$ : $[h^{L+1}, d^{L+1}]$ ; $*$ → pointwise

  Scaled, Masked Differences

$Usparce^L.shape : [B, n^L, h^{L+1}, d^{L+1}]$

No need to scale more using $\theta$. Similarity-scaling does this for us. (as we go from iteration to iteration, similarities will decrease)

If take_into_account_winner_ratios:

This is another update-scaling term that will be used later.

- $WinCountPartial^L \leftarrow \sum_i^{n^L} \sum_b^B M^L_{winners_{bi}}$ $[n^{L+1}]$

Contains the number of wins that each

* $WinCountPartial^L \leftarrow \sum_i^{h^L} \sum_b^B M^L_{winners\,bi}$ $[h^{L+1}]$

$WinCountPartial.Shape: [h^{L+1}]$

Contains the number of wins that each parent capsule made in one iteration of $\theta$.

* $WinCount \leftarrow WinCount + WinCountPartial$

↳ contains the number of $C_i^L$'s that each $C_j^{L+1}$ "won" (in all iterations).

If not softmax and not tanh:

Update mask $M^L$ so that in the next iteration of $\theta$ (neighboors with distance 1), we will not have the same winners. In other words, we want to find the neighboors of the winner capsules, not the winners themselves.

* $\forall b \in [1, B], \forall i \in \Omega_L, \forall j \in \Omega_{L+1}: M^L_{bij:} \leftarrow M^L_{bij:} - M^L_{winners\,bij}$ pointwise $[1]$

$M.Shape: [B, h^L, h^{L+1}, d^{L+1}]$ $[B, h^L, h^{L+1}, d^{L+1}]$

M has now zeros at the places where $M^L_{winners}$ had ones.

Aggregate the updates across thetas.

* $SU^L \leftarrow SU^L + U^L_{sparce}$

↳ total updates ↳ partial updates

$SU^L.shape: [B, h^L, h^{L+1}, d^{L+1}]$

End of neighboor loop.

Make updates dense by finding, for each parent capsule $C_j^{L+1}$ the mean of all the votes from which capsules he won (across batch size). This is done to make less updates but more "educated".

* $U^L \leftarrow \frac{\sum_b^B \sum_i^{h^L} SU^L_{bi}}{B * h^L}$ $U^L.shape: [h^{L+1}, d^{L+1}]$

Time to change the DigitCaps (perform update step).

If not take_into_account_winners:

If not radical:

Basic, "simple" case:

* $C^{L+1} \leftarrow C^{L+1} + U^L$

$C^{L+1}.Shape: [h^{L+1}, d^{L+1}]$

As you can see, $C^{L+1}$ is not dependent on a single input. Rather, it changes given a batch at training. So it is not a prediction. The capsule vectors $C^{L+1}$ are learned to represent the invariant properties of the image features. With these instantiation

else if radical:

$C$ . Shape: $[h^{L+1}, d^{L+1}]$

The capsule vectors $C^{L+1}$ are learned to represent the invariant properties of the image features. With these instantiation invariant vectors the votes are compared and the similarities are produced. As we will see in a moment, these aggregated similarities are also the output of our model. Please note that during testing, preferably $r == 0$ so as to not update $C^{L+1}$ anymore.

**else if radical:**

Average over r iterations. (momentum)

* $C^{L+1} \leftarrow \frac{C^{L+1} * (r-1) + U^L}{r}$

**else:** scale updates relative to the number of wins.

* $WinRatio^L \leftarrow \frac{WinCount^L}{n^L * B}$ $\quad WinRatio.Shape: [h^{L+1}]$

* $SoftWinRatio^L \leftarrow Softmax(WinRatio^L)$ (Shape $[h^{L+1}]$) — SoftWinRatio is like assignment probabilities.

* $\forall j \in \Omega_{L+1}$: $U_j^L \leftarrow SoftWinRatio_j^L * U_j^L$ (Shape $[h^{L+1}, d^{L+1}]$)

**If not radical:**

Basic, "simple" case:

* $C^{L+1} \leftarrow C^{L+1} + U^L$

$C^{L+1}$. Shape: $[h^{L+1}, d^{L+1}]$

**else if radical:**

Average over r iterations. (momentum)

* $C^{L+1} \leftarrow \frac{C^{L+1} * (r-1) + U^L}{r}$

**If normalize_l_in_loop:**

Normalize digit capsules before next iteration.

**if norm_type == 0:**

* $\forall j \in \Omega_{L+1}$: $C_j^{L+1} \leftarrow Squash(C_j^{L+1})$

**else:**

* $\forall j \in \Omega_{L+1}$: $C_j^{L+1} \leftarrow \underbrace{\tanh(\|C_j^{L+1}\|)}_{\text{scaling term}} \underbrace{\frac{C_j^{L+1}}{\|C_j^{L+1}\|}}_{\text{unit norm}}$

Repeat if $r > 1$

End of $r$ iterations

Compute the similarities now, using the updated $C^{L+1}$.

* $\forall b \in [1, B]$, $\forall i \in \Omega_L$ : $FinalSims_{bi}^{L+1} \leftarrow \sum_{k}^{I^{L+1}} \left[ V_{bi}^{L} * C^{L+1} \right]_{bik}$

  shape: $[B, n^L, n^{L+1}]$ ; $[B, n^L, n^{L+1}, pl^{L+1}]$

* $\forall b \in [1, B]$ : $SimsOut_b^{L+1} \leftarrow \frac{\sum_{i}^{n^L} FinalSims_{bi}^{L+1}}{n^L}$

  shape: $[B, n^{L+1}]$

Find mean similarity across child capsules.

You can then argmax them and get the prediction for each instance in batch.

In test mode, these two lines below may be the only steps of the algorithm after computing the votes.

* **Return** $SimsOut^{L+1}$, $C^{L+1}$

For a behavior that closely resembles the one of capsules, we can set batch size = 1 and use a bigger kr_som. In this way, capsules $C^{L+1}$ will capture the equivariant properties of the data.