Permutation invariance for neural networks

YKY

August 26, 2019

There are several possible ways to achieve permutation invariance in neural networks:

A. By constraining the weights

B. By converting the input vector

1 Invariant theory

Traditional neural network:

$$
\begin{array}{rl}
\boxed{neuron} & y = \; \oslash \boldsymbol{w} \cdot \boldsymbol{x} \\[4pt]
\boxed{layer} & y = \; \oslash W \boldsymbol{x} \\[4pt]
\boxed{network} & y = \; \oslash W \circ \oslash W \; .... \; \boldsymbol{x}
\end{array}
\tag{1}
$$

Quadratic neural network:

$$
\begin{array}{rl}
\boxed{neuron} & y = \; W \boldsymbol{x} \cdot \boldsymbol{x} \\[4pt]
\boxed{layer} & y = \; W \boldsymbol{x} \cdot \boldsymbol{x} \\[4pt]
\boxed{network} & y = \; W \boldsymbol{x} \circ W \boldsymbol{x} \; .... \; \boldsymbol{x}
\end{array}
\tag{2}
$$

Traditionally, each neuron k with output o_k is defined as:

$$
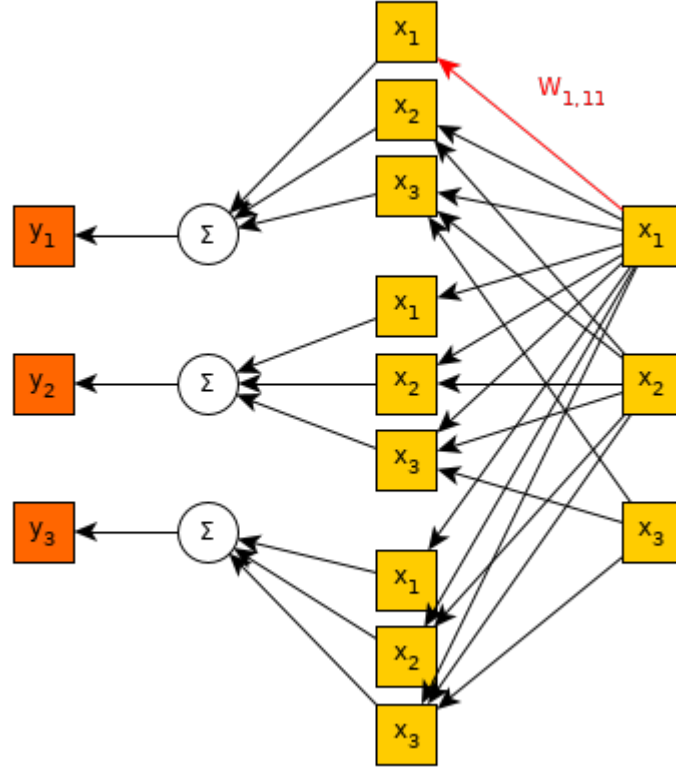o_k = \oslash(\text{net}_k) = \oslash \left(\sum_{j=1}^{n} w_{jk} o_j \right).
\tag{3}
$$

This is replaced by our new neuron:

$$
\boxed{\text{next layer}} \quad o_k = \text{net}_k = \sum_{j} \sum_{i} W_{ij}^{k} o_i o_j \quad \boxed{\text{current layer}}
\tag{4}
$$

where the 2 summations can be **interchanged**.

Let $y = F(x)$ denote the (overall) neural network function. In our case $F(x)$ is a polynomial $\in \mathbb{R}[x]^{\mathfrak{S}_n}$ where $\mathfrak{S}_n$ is the symmetric group of n elements.
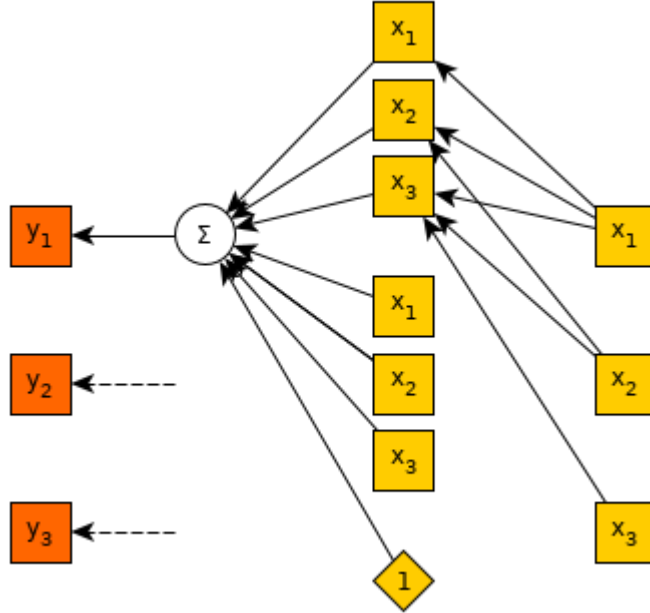
This is one quadratic layer with $i > j$ edges deleted (because $x_i x_j = x_j x_i$):

$$\tag{5}$$

The link in red is W_{11}^1.

This is one network layer complete with quadratic, linear, and constant terms:

$$\tag{6}$$

Notice that this block should be repeated for each output $y_1, ..., y_3$.

2 General case for $y = Wx$

$$\boxed{\text{original}} \quad y_j = \sum_i W_{ij} x_i. \tag{7}$$

Equivariance implies:

$$\boxed{\text{swapped}} \quad y_j(\sigma(x_j \; x_k)x) \;=\; \sigma \cdot y_j = y_k \quad \boxed{\text{original}} \tag{8}$$

$$\sum_{i \neq j,k} W_{ij}x_i + W_{kj}x_j + W_{jj}x_k \;=\; \sum_{i \neq j,k} W_{ik}x_i + W_{jk}x_j + W_{kk}x_k.$$

Comparing coefficients yields:

$$
\begin{aligned}
W_{ij} &= W_{ik} & \forall \, j,k, (i \neq j,k) \\
W_{kj} &= W_{jk} & \forall \, j,k \\
W_{jj} &= W_{kk} & \forall \, j,k.
\end{aligned}
\tag{9}
$$

In other words, the matrix W is of the form:

$$W = \alpha I + \beta 11^T. \tag{10}$$

3 Case for $y_k = W_k x \cdot x$

The general form of a "quadratic" vector function is:

$$y = (Ax) \cdot x + Bx + C. \tag{11}$$

We just focus on the quadratic term $(Wx) \cdot x$:

$$\boxed{\text{original}} \quad y_k = \sum_j \left[\sum_{i \leq j} W_{ij}^k x_i \right] x_j. \tag{12}$$

Note that the matrix W is "3-dimensional" and has $N \times N \times N$ entries.

Let $\sigma := (x_k \rightleftharpoons x_h)$, meaning **transposition** of the two elements. Equivariance implies:

$$\boxed{\text{LHS}} \quad y_k(\sigma \cdot x) = \sigma \cdot y_k = y_h \quad \boxed{\text{RHS}} \tag{13}$$

$$\boxed{\text{LHS}} = y_k(\sigma \cdot x) = \sum_j \left[\sum_{i \le j} W_{ij}^k \, \sigma \cdot x_i \right] \sigma \cdot x_j$$

$$= \sum_j \left[\sum_{\substack{i \le j \\ i \ne h,k}} W_{ij}^k x_i + \underbrace{W_{hj}^k x_k}_{\text{if } h \le j} + \underbrace{W_{kj}^k x_h}_{\text{if } k \le j} \right] \sigma \cdot x_j \qquad \boxed{\text{apply } \sigma \cdot x_i}$$

$$= \sum_{j \ne h,k} \left[\sum_{\substack{i \le j \\ i \ne h,k}} W_{ij}^k x_i + \underbrace{W_{hj}^k x_k}_{\text{if } h \le j} + \underbrace{W_{kj}^k x_h}_{\text{if } k \le j} \right] x_j \qquad \boxed{\text{apply } \sigma \cdot x_j}$$

$$+ \left[\sum_{\substack{i < h \\ i \ne k}} W_{ih}^k x_i + W_{hh}^k x_k + \underbrace{W_{kh}^k x_h}_{\text{if } k \le h} \right] x_k + \left[\sum_{\substack{i < k \\ i \ne h}} W_{ik}^k x_i + \underbrace{W_{hk}^k x_k}_{\text{if } h \le k} + W_{kk}^k x_h \right] x_h$$

$$= \sum_{j \ne h,k} \sum_{\substack{i \le j \\ i \ne h,k}} W_{ij}^k x_i x_j + \sum_{j > h} W_{hj}^k x_k x_j + \sum_{j > k} W_{kj}^k x_h x_j$$

$$+ \sum_{\substack{i < h \\ i \ne k}} W_{ih}^k x_i x_k + W_{hh}^k x_k^2 + \underbrace{W_{kh}^k x_h x_k}_{\text{if } k \le h}$$

$$+ \sum_{\substack{i < k \\ i \ne h}} W_{ik}^k x_i x_h + \underbrace{W_{hk}^k x_k x_h}_{\text{if } h \le k} + W_{kk}^k x_h^2$$

$$\boxed{\text{RHS}} = \sum_j \left[\sum_{i \le j} W_{ij}^h x_i \right] x_j$$

$$= \sum_j \left[\sum_{\substack{i \le j \\ i \ne h,k}} W_{ij}^h x_i + \underbrace{W_{hj}^h x_h}_{\text{if } h \le j} + \underbrace{W_{kj}^h x_k}_{\text{if } k \le j} \right] x_j$$

$$= \sum_{j \ne h,k} \left[\sum_{\substack{i \le j \\ i \ne h,k}} W_{ij}^h x_i + \underbrace{W_{hj}^h x_h}_{\text{if } h \le j} + \underbrace{W_{kj}^h x_k}_{\text{if } k \le j} \right] x_j$$

$$+ \left[\sum_{\substack{i < h \\ i \ne k}} W_{ih}^h x_i + W_{hh}^h x_h + \underbrace{W_{kh}^h x_k}_{\text{if } k \le h} \right] x_h + \left[\sum_{\substack{i < k \\ i \ne h}} W_{ik}^h x_i + \underbrace{W_{hk}^h x_h}_{\text{if } h \le k} + W_{kk}^h x_k \right] x_k$$

$$= \sum_{j \ne h,k} \sum_{\substack{i \le j \\ i \ne h,k}} W_{ij}^h x_i x_j + \sum_{j > h} W_{hj}^h x_h x_j + \sum_{j > k} W_{kj}^h x_k x_j$$

$$+ \sum_{\substack{i < h \\ i \ne k}} W_{ih}^h x_i x_h + W_{hh}^h x_h^2 + \underbrace{W_{kh}^h x_k x_h}_{\text{if } k \le h}$$

$$+ \sum_{\substack{i < k \\ i \ne h}} W_{ik}^h x_i x_k + \underbrace{W_{hk}^h x_h x_k}_{\text{if } h \le k} + W_{kk}^h x_k^2 \tag{14}$$

Comparing coefficients yields the following equations (except for the "colorful" ones):

$$\begin{aligned} W_{ij}^h &= W_{ij}^k & \forall\, h,k.\ j \ne h,k;\ i \le j;\ i \ne h,k \\ W_{hh}^k &= W_{kk}^h & \forall\, h,k. \\ W_{kk}^k &= W_{hh}^h & \forall\, h,k. \end{aligned} \tag{15}$$
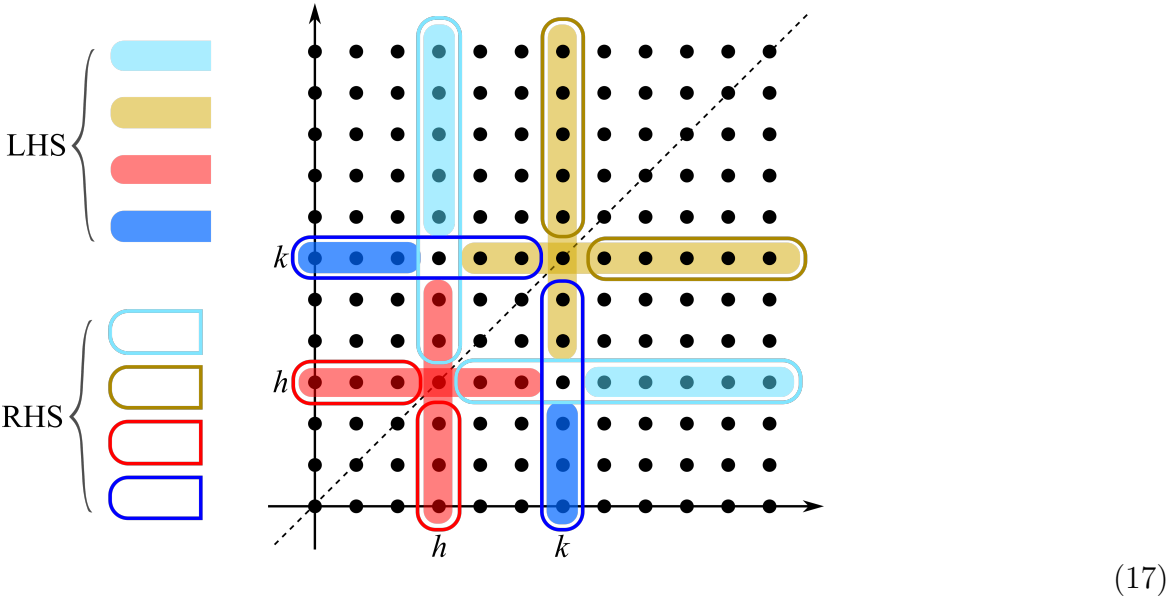
The following diagram shows the situation of the index j in the cyan and olive cases:

(16)

$$j > \max(h, k) \qquad k \geq j > h$$

(17)

How many different colors?

$$
\begin{array}{llll}
N = 1 \ .... & 1 & / \ 1 & = 100\% \\
N = 2 \ .... & 4 & / \ 8 & = 50\% \\
N = 3 \ .... & 6 & / \ 27 & = 22.2\% \\
N = 4 \ .... & 7 & / \ 64 & = 10.9\% \\
N = 5 \ .... & 8 & / \ 125 & = 6.4\% \\
N = 6 \ .... & 9 & / \ 216 & = 4.2\% \\
N = 7 \ .... & 10 & / \ 343 & = 2.9\% \\
N = 8 \ .... & 11 & / \ 512 & = 2.1\% \\
N = 9 \ .... & 12 & / \ 729 & = 1.6\% \\
N = 10 \ .... & 13 & / \ 1000 & = 1.3\% \\
\end{array}
$$

(18)

There would be N **blocks** of $N \times N$ matrices.

All diagonals consists of 2 colors, regardless of N (from 2nd and 3rd equations). This leaves $N(N-1)$ non-diagonal entries per block.

Non-diagonal entries of different blocks are equal, if the block indices are different from the row and column indices. Out of N blocks there would be 2 different sets of non-diagonal weights. (This comes from the 1st equation.)

The last equation causes non-diagonal weights to have a certain symmetry about the diagonal.

For example, the number of monomials in 3 variables and of degree ≤ 8 is:

$$\binom{10}{8} + \binom{9}{7} + \binom{8}{6} + \dots + \binom{4}{2} + \binom{3}{1} + \binom{3}{0}$$
$$= 45 + 36 + 28 + 21 + 15 + 10 + 6 + 3 + 1 = 165 \tag{19}$$

4 Programmatic way to find invariant constraints on weights

Define a scheme to index all the weights in the multi-layer NN: $W_{ij}^{k\ell}$. Then the entire NN function can be expanded as a polynomial with coefficients from W.

For one quadratic layer, there would be a total of $n\left(n^2 - \frac{n(n-1)}{2}\right) = \frac{n^2(n+1)}{2}$ terms. The coefficients for each term would be composed out of $W_{ij}^{k\ell}$. Permuting the input would require coefficients of **like** terms to be equal. We should try all pairwise permutations of n inputs, of which there are $n(n-1)/2$.

On the second layer, the output would be composed of sums and products of polynomials with second-layer weights. Thus the new coefficients would be **polynomials** in multi-layer weights. Invariance or equivariance requires that we compare coefficients of like terms, thus yielding **equalities with polynomials** on both sides. Such conditions seems much more complex than the single-layer conditions for equivariance.

This means that the weights would be in an **algebraic variety** of reduced dimension. Our objective is to update / learn the weights **within** this variety.

5 Quadratic, multi-layer, unconstrained case

In the last section we have **equivariant** layers composed together to form a neural network. Now we relax the constraints so that the multi-layer network is free to have any weights except that the output must be **invariant**.

6 With output space "folded in half"

Now suppose the output is only $1/2$ the dimension of the input. Define a new form of equivariance such that the input permutation would act on the output as "folded in half".

In other words, equivariance is changed to:

$$\boxed{\text{swapped}} \quad y_k \cdot \sigma(x_k\, x_h) = y_h \text{ or } y_{h-N/2} \quad \boxed{\text{original}} \tag{20}$$

where τ is σ acting on y as double its length and identifying $y_i = y_{i+N/2}$.

6.1 Linear case

Just notice that the dimension of y is halved:

$$\boxed{\text{original}} \quad y_j = \sum_i W_{ij} x_i. \tag{21}$$

"Folded" equivariance implies:

$$\boxed{\text{swapped}} \quad y_j(\sigma(x_j \; x_k)x) = \sigma \cdot y_j = y_k \quad \boxed{\text{original}} \tag{22}$$
$$\sum_{i \neq j,k} W_{ij} x_i + W_{kj} x_j + W_{jj} x_k = \sum_{i \neq j,k} W_{ik} x_i + W_{jk} x_j + W_{kk} x_k$$

with the restriction $j \in \{1, ..., N/2\}$, and $k \in \{1, ..., N\}$.

The constraints obtained are same as before, except that index ranges are different:

$$
\begin{array}{rcll}
W_{ij} & = & W_{ik} & \forall\, j, k, (i \neq j, k) \\
W_{kj} & = & W_{jk} & \forall\, j, k \\
W_{jj} & = & W_{kk} & \forall\, j, k
\end{array}
$$

These constraints give rise to a matrix of this form (for the 6×3 case, numbers represent different colors):

$$
\begin{array}{cccccc}
5 & 1 & 1 & 2 & 3 & 4 \\
1 & 5 & 1 & 2 & 3 & 4 \\
1 & 1 & 5 & 2 & 3 & 4 \quad .
\end{array} \tag{23}
$$

This pattern is obtained from my Python code.

NOTE: The above pattern is verified to be NOT equivariant, there is a bug in the equivariant condition.

6.2 Quadratic case

7 Learning algorithm

作者：zighouse

链接：`https://www.zhihu.com/question/327765164/answer/704606353`

来源：知乎

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

神经网络是对一类内部结构固定的非线性函数的俗称，这类函数是输出关于输入以及隐含内部状态的函数，输出与输入呈现非线性特性。当一份输出只与一份输入有关时，常用卷积神经网络来实现。当一份输出与一个相继表达的输入序列相关时，可以用回归神经网络来实现。一般地，神经网络可以技术性地分解成神经元的复合，这里的神经元是在这个神经网络中的一种最基本的非线性函数的俗称，管理着属于它的内部状态，

并基于这些内部状态在神经网络中负责着分配到它的非线性处理。每多一重基本非线性函数的复合，则多一层神经元。如果在某一重复合中出现了两类或者更多类基本非线性函数项的合并，则出现了分支。

神经网络的权值是分解到具体神经元管理的一种内部状态。用反向传播方法来更新神经网络的权值是基于这样一个基本的假设：在一个确定的输入（或者输入序列）并产生当前输出的这个点（权值构成的线性空间中的点）上，输出在这个点上是连续的。即权值点的连续微小变化会导致输出点相应的连续微小变化。这样，当我们希望调节当前权值以使此输出向特定点靠拢时，就得出了基于权值空间中错误/误差/惩罚的梯度的反向传播算法。

如果想在某个神经网络中的两个权值间建立一种约束关系，这两个权值自然就不再相互独立，可以通过考查整个权值构成的线性空间，秩会变小。约束条件只要不改变连续假设，仍然可以求出带约束条件下的梯度。如果改变了连续假设，则意味着非线性分解不恰当，需要重新分解神经网络的基本结构。

7.1 Classic back-prop with quadratic neurons

Traditional neural network:

$$
\boxed{neuron} \quad y = \oslash \boldsymbol{w} \cdot \boldsymbol{x}
$$
$$
\boxed{layer} \quad y = \oslash W \boldsymbol{x}
$$
$$
\boxed{network} \quad y = \oslash W \circ \oslash W \dots \boldsymbol{x} \tag{24}
$$

Quadratic neural network:

$$
\boxed{neuron} \quad y = W \boldsymbol{x} \cdot \boldsymbol{x}
$$
$$
\boxed{layer} \quad y = W \boldsymbol{x} \cdot \boldsymbol{x}
$$
$$
\boxed{network} \quad y = W \boldsymbol{x} \circ W \boldsymbol{x} \dots \boldsymbol{x} \tag{25}
$$

Traditionally, each neuron k with output o_k is defined as:

$$
o_k = \oslash(\mathrm{net}_k) = \oslash \left(\sum_{j=1}^{n} w_{jk} o_j \right). \tag{26}
$$

This is replaced by our new neuron:

$$
\boxed{\text{next layer}} \quad o_k = \mathrm{net}_k = \sum_j \sum_i W_{ij}^k o_i o_j \quad \boxed{\text{current layer}} \tag{27}
$$

where the 2 summations can be **interchanged**.

What follows is just a re-working of traditional back-propagation.

Using the chain rule:

$$
\frac{\partial E}{\partial W_{ij}^k} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial W_{ij}^k} \tag{28}
$$

For the RHS's second factor, only one term in the sum depends on W_{ij}^k, so:

$$
\frac{\partial o_k}{\partial W_{ij}^k} = \frac{\partial}{\partial W_{ij}^k} \left(\sum_{i'} \sum_{j'} W_{i'j'}^k o_{i'} o_{j'} \right) = \frac{\partial}{\partial W_{ij}^k} W_{ij}^k o_i o_j = o_i o_j. \tag{29}
$$

If k is an inner neuron, let $L = \{u, v, \ldots, w\}$ be the **next layer** of neurons receiving input from neuron k. Consider E as a function with the inputs being all neurons in L:

$$\frac{\partial E(o_k)}{\partial o_k} = \frac{\partial E(o_u, o_v, \ldots, o_w)}{\partial o_k}$$

$$\boxed{\text{next layer}} \quad o_\ell = \sum_j \sum_i W_{ij}^\ell o_i o_j \quad \boxed{\text{current layer}} \tag{30}$$

and take the total derivative with respect to o_k. A **recursive** expression for the derivative is obtained:

$$\frac{\partial E}{\partial o_k} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial o_k} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \sum_j W_{kj}^\ell o_j \right) \tag{31}$$

Substituting, we obtain:

$$\frac{\partial E}{\partial W_{ij}^k} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial W_{ij}^k} = \frac{\partial E}{\partial o_k} o_i o_j := \delta_k o_i o_j$$

$$\delta_k = \sum_{\ell \in L} \left(\delta_\ell \sum_j W_{kj}^\ell o_j \right) \tag{32}$$

Our algorithm can be compared the classic back-prop algorithm:

$$\frac{\partial E}{\partial W_{ij}} = \delta_j o_i$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} \dfrac{\partial L}{\partial \mathbb{O}(o_j)} \dfrac{\partial \mathbb{O}(o_j)}{\partial o_j} & \text{if } j = \text{output neuron} \\ \sum_l W_{jl} \delta_l \dfrac{\partial \mathbb{O}(o_j)}{\partial o_j} & \text{if } j = \text{inner neuron} \end{cases} \tag{33}$$

7.2 Shared weights

For our purpose, it is good to know that all the weight-sharing occurs **within** each layer, never across layers.

The coefficient of $x_i x_j$ is W_{ij}, for the y_k component. For each weight W_{ij}^k we need to calculate the gradient $\frac{\partial E}{\partial W_{ij}^k}$, but the weights are in equivalence classes.

Say if the following 2 weights are shared:

$$W_0 := W_{ij} \equiv W_{i'j'} \tag{34}$$

Then the network:

$$y = \sum \sum W_{ij} x_i x_j \tag{35}$$

would contain the shared components:

$$W_0(x_i x_j + x_{i'} x_{j'}) + \text{other terms} \ldots \tag{36}$$

Equality constraints

For simple **equality** of weights, the weights should be collected together. (29) should simply be $\sum o_i o_j$ for the equivated weights.

Additive constraints

The most tricky part is the "additive" constraint:

$$W_{hk}^h + W_{kh}^h = W_{hk}^k + W_{kh}^k \quad \forall\, h, k. \tag{37}$$

This is just like having 4 "not quite independent" variables x, y, u, v satisfying:

$$x + y = u + v \tag{38}$$

and asking what is

$$\frac{\partial(x + y)}{\partial x}\ ? \tag{39}$$

And the solution is to make one of the variables **depend** on the other 3.

For each layer, iterate over every neuron representative, which has a collection of coefficients.

We need to consider equations (29) and (31), but (31) is unaffected by weight-sharing.

In (29), for each equivalence class,

$$W_{hk}^h + W_{kh}^h \qquad W_{hk}^k + W_{kh}^k \quad \forall\, h, k. \tag{37}$$