Symmetric neural networks

YKY

June 21, 2019

1 General case for $y = Ax$

$$\boxed{\text{original}} \quad y_j = \sum_i a_{ij} x_i. \tag{1}$$

Equivariance implies:

$$\begin{aligned} \boxed{\text{swapped}} \quad y_j(\sigma(x_j\ x_k)x) &= \sigma \cdot y_j = y_k \quad \boxed{\text{original}} \\ \sum_{i \neq j,k} a_{ij} x_i + a_{kj} x_j + a_{jj} x_k &= \sum_{i \neq j,k} a_{ik} x_i + a_{jk} x_j + a_{kk} x_k. \end{aligned} \tag{2}$$

Comparing coefficients yields:

$$\begin{aligned} a_{ij} &= a_{ik} \qquad \forall\ j,k,(i \neq j,k) \\ a_{kj} &= a_{jk} \qquad \forall\ j,k \\ a_{jj} &= a_{kk} \qquad \forall\ j,k. \end{aligned} \tag{3}$$

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

$$A = \alpha I + \beta 1 1^T. \tag{4}$$

2 Case for $y_k = A_k x \cdot x$

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

$$y = (Ax) \cdot x + Bx + C. \tag{5}$$

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

$$\boxed{\text{original}} \quad y_k = \sum_j \left[\sum_i a^k_{ij} x_i \right] x_j. \tag{6}$$

Note that the matrix A is "3D" and has $N \times N \times N$ entries.

Equivariance implies:

$$\boxed{\text{swapped}} \quad y_k(\sigma(x_k\ x_h)\cdot x) = \sigma\cdot y_k = y_h \quad \boxed{\text{original}} \tag{7}$$

$$\begin{aligned}
LHS &= \sum_j \left[\sum_{i\neq h,k} a^k_{ij}x_i + a^k_{hj}x_k + a^k_{kj}x_h\right]\sigma\cdot x_j \\
&= \sum_{j\neq h,k}\left[\sum_{i\neq h,k} a^k_{ij}x_i + a^k_{hj}x_k + a^k_{kj}x_h\right]x_j + \left[\sum_{i\neq h,k} a^k_{ih}x_i + a^k_{hh}x_k + a^k_{kh}x_h\right]x_k + \left[\sum_{i\neq h,k} a^k_{ik}x_i + a^k_{hk}x_k + a^k_{kk}x_h\right]x_h \\
&= \sum_{j\neq h,k}\sum_{i\neq h,k} a^k_{ij}x_ix_j + \sum_{j\neq h,k} a^k_{hj}x_kx_j + \sum_{j\neq h,k} a^k_{kj}x_hx_j \\
&\quad + \sum_{i\neq h,k} a^k_{ih}x_ix_k + a^k_{hh}x_k^2 + a^k_{kh}x_hx_k \\
&\quad + \sum_{i\neq h,k} a^k_{ik}x_ix_h + a^k_{hk}x_kx_h + a^k_{kk}x_h^2 \\
RHS &= \sum_j \left[\sum_i a^h_{ij}x_i\right]x_j \\
&= \sum_{j\neq h,k}\sum_{i\neq h,k} a^h_{ij}x_ix_j + \sum_{j\neq h,k} a^h_{kj}x_kx_j + \sum_{j\neq h,k} a^h_{hj}x_hx_j \\
&\quad + \sum_{i\neq h,k} a^h_{ik}x_ix_k + a^h_{kk}x_k^2 + a^h_{hk}x_hx_k \\
&\quad + \sum_{i\neq h,k} a^h_{ih}x_ix_h + a^h_{kh}x_kx_h + a^h_{hh}x_h^2
\end{aligned} \tag{8}$$

Comparing coefficients yields:

$$\begin{aligned}
a^h_{ij} &= a^k_{ij} && \forall\ h,k,(i\neq h,k, j\neq h,k) \\
a^h_{kj} &= a^k_{hj} && \forall\ h,k,(j\neq h,k) \\
a^h_{hj} &= a^k_{kj} && \forall\ h,k,(j\neq h,k) \\
a^h_{ik} &= a^k_{ih} && \forall\ h,k,(i\neq h,k) \\
a^h_{ih} &= a^k_{ik} && \forall\ h,k,(i\neq h,k) \\
a^h_{kk} &= a^k_{hh} && \forall\ h,k \\
a^h_{hh} &= a^k_{kk} && \forall\ h,k \\
a^h_{hk} + a^h_{kh} &= a^k_{hk} + a^k_{kh} && \forall\ h,k.
\end{aligned} \tag{9}$$

How many different colors?

$$\begin{array}{lccc}
N=2\ .... & 6 & /\ 8 & =75\% \\
N=3\ .... & 9 & /\ 27 & =33.3\% \\
N=4\ .... & 11 & /\ 64 & =17/2\% \\
N=5\ .... & 13 & /\ 125 & =10.4\% \\
N=6\ .... & 15 & /\ 216 & =6.9\%
\end{array} \tag{10}$$

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.

3 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:

$$\boxed{\text{swapped}} \quad y_k \cdot \sigma(x_k\ x_h) = y_h \text{ or } y_{h-N/2} \quad \boxed{\text{original}} \tag{11}$$

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

3.1 Linear case

Just notice that the dimension of y is halved:

$$\boxed{\text{original}} \quad y_j = \sum_i a_{ij} x_i. \tag{12}$$

"Folded" equivariance implies:

$$\begin{array}{rcl} \boxed{\text{swapped}} \quad y_j(\sigma(x_j\ x_k)x) & = & \sigma \cdot y_j = y_k \quad \boxed{\text{original}} \\ \sum_{i\neq j,k} a_{ij}x_i + a_{kj}x_j + a_{jj}x_k & = & \sum_{i\neq j,k} a_{ik}x_i + a_{jk}x_j + a_{kk}x_k \end{array} \tag{13}$$

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:

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

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

$$\begin{array}{cccccc} 5 & 1 & 1 & 2 & 3 & 4 \\ 1 & 5 & 1 & 2 & 3 & 4 \\ 1 & 1 & 5 & 2 & 3 & 4 \end{array} . \tag{14}$$

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.

3.2 Quadratic case

4 Training of the NN

作者：zighouse

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

来源：知乎

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。

Traditional neural network:

$$\begin{aligned} \boxed{neuron} \quad & y = \sigma \boldsymbol{w} \cdot \boldsymbol{x} \\ \boxed{layer} \quad & y = \sigma W \boldsymbol{x} \\ \boxed{network} \quad & y = \sigma W \circ \sigma W \ .... \ \boldsymbol{x} \end{aligned} \tag{15}$$

Quadratic neural network:

$$\begin{aligned} \boxed{neuron} \quad & y = \sigma W \boldsymbol{x} \cdot \boldsymbol{x} \\ \boxed{layer} \quad & y = \sigma W \boldsymbol{x} \cdot \boldsymbol{x} \\ \boxed{network} \quad & y = \sigma W \circ \sigma W \ .... \ \boldsymbol{x} \end{aligned} \tag{16}$$

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

$$o_k = \sigma(\text{net}_k) = \sigma\left(\sum_{j=1}^{n} w_{jk} o_j\right). \tag{17}$$

This is replaced by our new neuron:

$$o_k = \text{net}_k = \sum_j \sum_i W_{ij}^k o_i o_j \tag{18}$$

where the 2 summations can be interchanged.

Using the chain rule:

$$\frac{\partial E}{\partial W_{ij}^k} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial W_{ij}^k} \tag{19}$$

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

$$\frac{\partial o_k}{\partial W_{ij}^k} = \frac{\partial}{\partial W_{ij}^k} \left(\sum_{i'} \sum_{j'} W_{i'j'}^k o_{i'} o_{j'} \right) = \frac{\partial}{\partial W_{ij}^k} W_{ij}^k o_i o_j = o_i o_j. \tag{20}$$

If k is an inner neuron, consider E as a function with the inputs being all neurons $N = \{u, v, \ldots, w\}$ receiving input from neuron k,

$$\frac{\partial E(o_k)}{\partial o_k} = \frac{\partial E(\text{net}_u, \text{net}_v, \ldots, \text{net}_w)}{\partial o_k} \tag{21}$$

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

$$\frac{\partial E}{\partial o_k} = \sum_{n \in N} \left(\frac{\partial E}{\partial o_n} \frac{\partial o_n}{\partial o_k} \right) = \sum_{n \in N} \left(\frac{\partial E}{\partial o_n} W_n^k \right) \tag{22}$$

Substituting, we obtain:

$$\begin{aligned} \frac{\partial E}{\partial W_{ij}^k} &= \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial W_{ij}^k} = \frac{\partial E}{\partial o_k} o_i o_j \\ \frac{\partial E}{\partial W_{ij}^k} &= o_i o_j \delta_k \end{aligned} \tag{23}$$

Recall the classic back-prop algorithm:

$$\frac{\partial E}{\partial W_{ij}} = o_i \delta_j \tag{24}$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} \dfrac{\partial L}{\partial \mathcal{O}(o_j)} \dfrac{\partial \mathcal{O}(o_j)}{\partial o_j} & \text{if } j = \text{output neuron} \\ \displaystyle\sum_l w_{jl} \delta_l \frac{\partial \mathcal{O}(o_j)}{\partial o_j} & \text{if } j = \text{inner neuron} \end{cases} \tag{25}$$

Calculate $\frac{\partial U}{\partial W}$. If they are linked then update together.

The co-efficient of $x_i x_j$ is W_{ij}, for the y_k component. We need to calculate the gradient for each W_{ijk}.

The most tricky part is the "additive" constraint:

$$W_{hk}^h + W_{kh}^h = W_{hk}^k + W_{kh}^k \quad \forall\, h, k. \tag{26}$$

How are the 4 derivatives related by the additive constraint? How would this "derived" additive constraint affect weight updating?