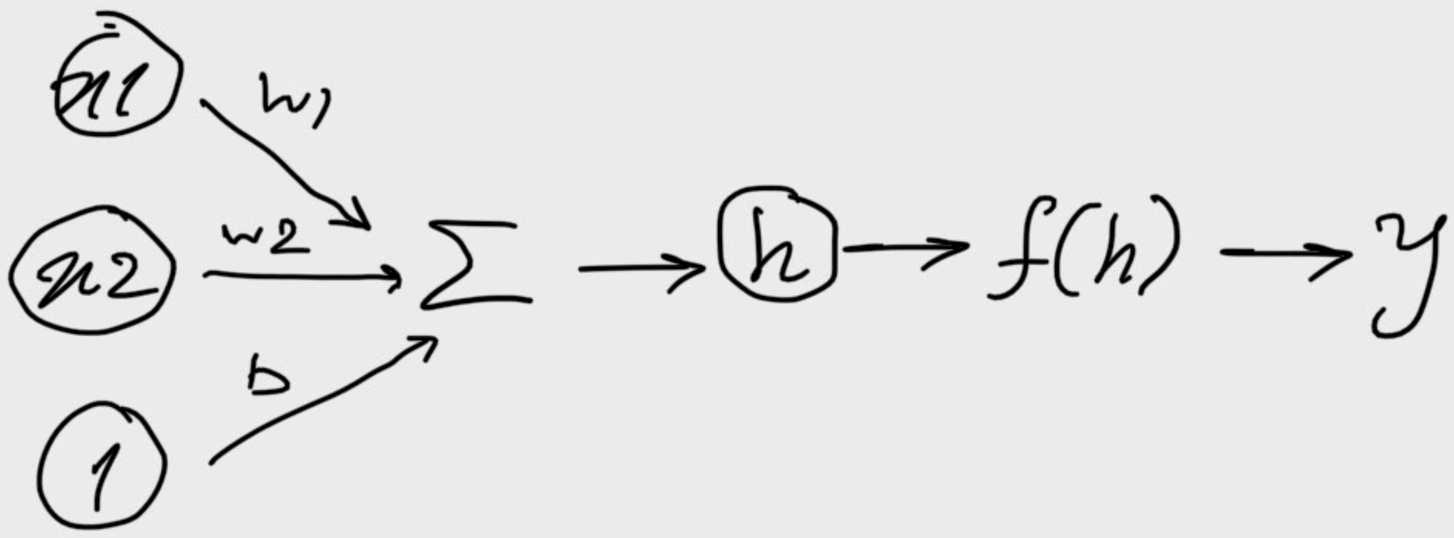
Tensors



$$y = f(w_1 x_1 + w_2 x_2 + b)$$

$$h = [x_1 \cdots x_n] \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

```
def sigmoid(x): return 1/(1+torch.exp(-x))
torch.manual_seed(7)
features = torch.randn((1,5)) ← five random normal vars
weights = torch.randn_like(features) ← a vector with
                                       the shape of features, random.
bias = torch.randn((1,1)) ← a 1x1 bias term

                  elem-by-elem mult
y = sigmoid(torch.sum(features * weights) + bias)
                                  output of a single layer
```

torch.mm and torch.matmul → mat multiplication

torch.mm and torch.matmul → mat multiplication

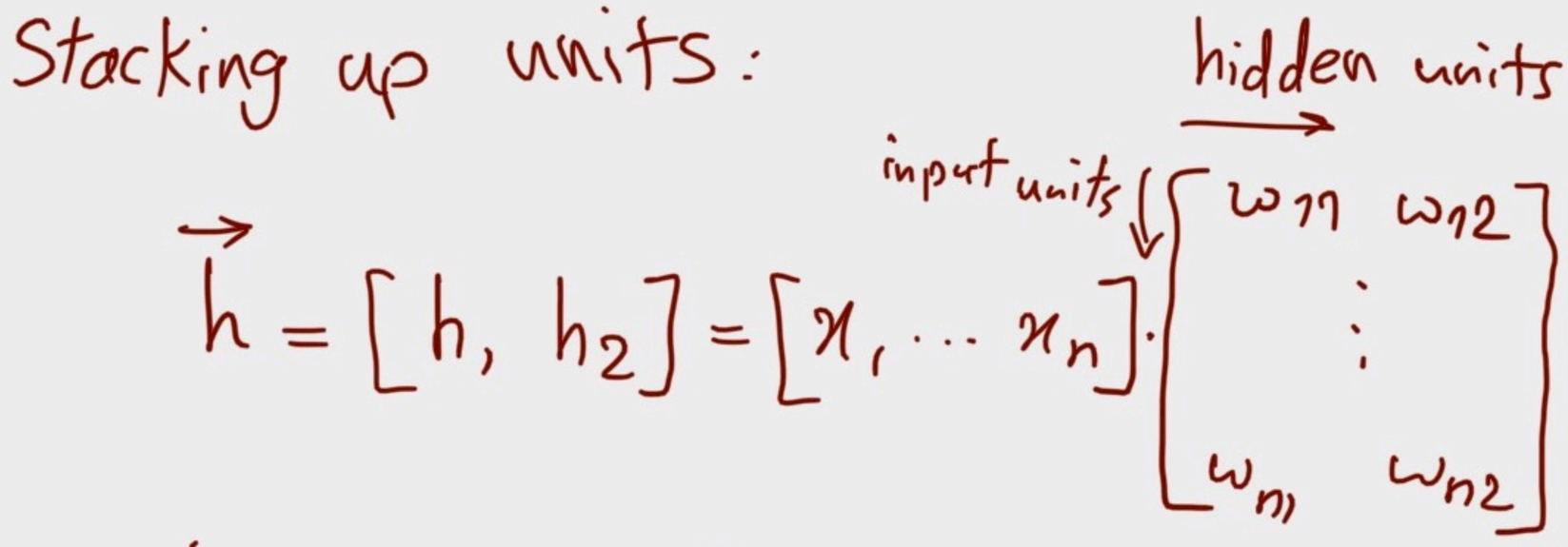
for changing the shape of tensors:

1. weights.reshape(a,b) → returns new tensor
2. weights.resize_(a,b) → Inplace operation

3. weights.view(a,b) → return new data with same data

alt → y = sigmoid(torch.mm(features, weights.view(5,1)+bias)

Stacking up units:



$h = \text{sigmoid}(\text{torch.mm}(\text{features}, w1) + B1)$

$y = \text{output} = \text{sigmoid}(\text{torch.mm}(h, w2) + B2)$

Going between Torch and Numpy.

memory is shared between them

```
a = np.random.rand(4,3)
b = torch.from_numpy(a)
b.numpy()
```