one of these groups:

$$g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j \tag{6.37}$$

where $\mathbb{G}^{(i)}$ is the set of indices into the inputs for group i, $\{(i-1)k+1, \ldots, ik\}$. This provides a way of learning a piecewise linear function that responds to multiple directions in the input $\boldsymbol{x}$ space.

A maxout unit can learn a piecewise linear, convex function with up to k pieces. Maxout units can thus be seen as *learning the activation function* itself rather than just the relationship between units. With large enough k, a maxout unit can learn to approximate any convex function with arbitrary fidelity. In particular, a maxout layer with two pieces can learn to implement the same function of the input $\boldsymbol{x}$ as a traditional layer using the rectified linear activation function, absolute value rectification function, or the leaky or parametric ReLU, or can learn to implement a totally different function altogether. The maxout layer will of course be parametrized differently from any of these other layer types, so the learning dynamics will be different even in the cases where maxout learns to implement the same function of $\boldsymbol{x}$ as one of the other layer types.

Each maxout unit is now parametrized by k weight vectors instead of just one, so maxout units typically need more regularization than rectified linear units. They can work well without regularization if the training set is large and the number of pieces per unit is kept low (Cai *et al.*, 2013).

Maxout units have a few other benefits. In some cases, one can gain some statistical and computational advantages by requiring fewer parameters. Specifically, if the features captured by n different linear filters can be summarized without losing information by taking the max over each group of k features, then the next layer can get by with k times fewer weights.

Because each unit is driven by multiple filters, maxout units have some redundancy that helps them to resist a phenomenon called **catastrophic forgetting** in which neural networks forget how to perform tasks that they were trained on in the past (Goodfellow *et al.*, 2014a).

Rectified linear units and all of these generalizations of them are based on the principle that models are easier to optimize if their behavior is closer to linear. This same general principle of using linear behavior to obtain easier optimization also applies in other contexts besides deep linear networks. Recurrent networks can learn from sequences and produce a sequence of states and outputs. When training them, one needs to propagate information through several time steps, which is much easier when some linear computations (with some directional derivatives being of magnitude near 1) are involved. One of the best-performing recurrent network