

Regardless of whether we use standard deviation, variance, or precision, we must ensure that the covariance matrix of the Gaussian is positive definite. Because the eigenvalues of the precision matrix are the reciprocals of the eigenvalues of the covariance matrix, this is equivalent to ensuring that the precision matrix is positive definite. If we use a diagonal matrix, or a scalar times the diagonal matrix, then the only condition we need to enforce on the output of the model is positivity. If we suppose that \mathbf{a} is the raw activation of the model used to determine the diagonal precision, we can use the softplus function to obtain a positive precision vector: $\boldsymbol{\beta} = \zeta(\mathbf{a})$. This same strategy applies equally if using variance or standard deviation rather than precision or if using a scalar times identity rather than diagonal matrix.

It is rare to learn a covariance or precision matrix with richer structure than diagonal. If the covariance is full and conditional, then a parametrization must be chosen that guarantees positive-definiteness of the predicted covariance matrix. This can be achieved by writing $\boldsymbol{\Sigma}(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{B}^\top(\mathbf{x})$, where \mathbf{B} is an unconstrained square matrix. One practical issue if the matrix is full rank is that computing the likelihood is expensive, with a $d \times d$ matrix requiring $O(d^3)$ computation for the determinant and inverse of $\boldsymbol{\Sigma}(\mathbf{x})$ (or equivalently, and more commonly done, its eigendecomposition or that of $\mathbf{B}(\mathbf{x})$).

We often want to perform multimodal regression, that is, to predict real values that come from a conditional distribution $p(\mathbf{y} | \mathbf{x})$ that can have several different peaks in \mathbf{y} space for the same value of \mathbf{x} . In this case, a Gaussian mixture is a natural representation for the output (Jacobs *et al.*, 1991; Bishop, 1994). Neural networks with Gaussian mixtures as their output are often called **mixture density networks**. A Gaussian mixture output with n components is defined by the conditional probability distribution

$$p(\mathbf{y} | \mathbf{x}) = \sum_{i=1}^n p(c = i | \mathbf{x}) \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}^{(i)}(\mathbf{x}), \boldsymbol{\Sigma}^{(i)}(\mathbf{x})). \quad (6.35)$$

The neural network must have three outputs: a vector defining $p(c = i | \mathbf{x})$, a matrix providing $\boldsymbol{\mu}^{(i)}(\mathbf{x})$ for all i , and a tensor providing $\boldsymbol{\Sigma}^{(i)}(\mathbf{x})$ for all i . These outputs must satisfy different constraints:

1. Mixture components $p(c = i | \mathbf{x})$: these form a multinoulli distribution over the n different components associated with latent variable¹ c , and can

¹We consider c to be latent because we do not observe it in the data: given input \mathbf{x} and target \mathbf{y} , it is not possible to know with certainty which Gaussian component was responsible for \mathbf{y} , but we can imagine that \mathbf{y} was generated by picking one of them, and make that unobserved choice a random variable.