Lecture 3: Variational supervised learning

In this lecture we will see that most traditional machine learning techniques can be implemented using the framework on variational inference. After the abstraction of the previous lecture, we will now go in a applicational *tour de force* to see the full potential of what you learned.

1 Supervised learning

We start by framing supervised learning in a variational way. Specifically, we will begin with a simple Bayesian logistic regression model and then move on to Bayesian neural networks and deep learning.
In a supervised learning task, the data consist of a set of pairs $(y_j, \boldsymbol{x}_j)$ consisting of a vector of features $\boldsymbol{x}_j$ and a target y_j. In classification problems, the target y_j is a discrete class label that determines the class membership of the data-point. For example, in a medical classification problem the class label can be a diagnosis while the features can be lab test results. A probabilistic classifier can have very high accuracy while having a poor probabilistic calibration since the latter is affected by the behavior of the classifier in the failure cases.
In this course, we are interested in probabilistic model that do not just predict a single class label but instead output a predictive probability distribution over possible labels. This is important since the information encoded in the feature is often incomplete so that it cannot fully specify a single label.
We can quantify the error calibration of a classifier defined by the predictive distribution $p(y_n \mid x_n)$ as the average log probability of the misclassified test data-points:

$$\frac{1}{N_{\text{mis}}} \sum_{(\boldsymbol{x}_n, y_n) \in \mathcal{M}_{\text{mis}}} \log p(y_n \mid x_n), \tag{1}$$

where $\mathcal{M}_{\text{mis}}$ is the set of all the N_{mis} miss-classified data-points in the test set. We define a test pair as misclassified if the prediction with highest probability did not correspond to the correct label. Note that this quantity does not depend on the accuracy as it ignores the correctly classified data-points.
There are two main forms of classification models. On one hand, the discriminative form uses a model that directly predicts the probability of the label given the features. on the other hand, the generative approach is based on learning a generative model for each class and then invert it using Bayes theorem to get a probability over the labels.

1.1 Binary logistic regression

Logistic regression is the most well-known form of discriminative classification model. In the sake of simplicity, we will start by discussing the binary classification case where y_j is either 0 or 1. The basic assumption behind (binary) logistic regression is that the logarithm of the odd ratio of the label is a linear function of the feature vector:

$$\log\left(\frac{\rho_j}{1-\rho_j}\right) = \boldsymbol{w}\cdot\boldsymbol{x}_j \tag{2}$$

where ρ_j is the probability of $y_j = 1$ and $\boldsymbol{w}$ is a vector of weights. This model can be reformulated using the logistic sigmoid function:

$$\rho_j(\boldsymbol{w}) = \text{Sigmoid}(\boldsymbol{w}\cdot\boldsymbol{x}_j) = \frac{e^{\boldsymbol{w}\cdot\boldsymbol{x}_j}}{1+e^{\boldsymbol{w}\cdot\boldsymbol{x}_j}} \tag{3}$$

where we expressed the probability of a function of the weights. If we assume that the data-points have been sampled independently, we can now express the probability of the data given the logistic regression model as follows

$$p(D \mid \boldsymbol{w}) = \prod_{j}^{J} \rho_j(\boldsymbol{w})^{y_j}\rho_j(-\boldsymbol{w})^{1-y_j} \tag{4}$$

where we used the fact that

$$\text{Sigmoid}(-x) = 1 - \text{Sigmoid}(x)\ .$$

The starting point of most training method is the log-likelihood function. In this case, the log-likelihood is:

$$l_{LR}(\boldsymbol{w}) = \sum_{j}^{J}\left(y_j \log\rho_j(\boldsymbol{w}) + (1-y_j)\log\rho_j(-\boldsymbol{w})\right)\ . \tag{5}$$

which can be expressed explicitly as follows:

$$l_{LR}(\boldsymbol{w}) = \sum_{j}^{J}\left(y_j\ \boldsymbol{w}\cdot\boldsymbol{x}_j - \log\left(e^{\boldsymbol{w}\cdot\boldsymbol{x}_j} + e^{-\boldsymbol{w}\cdot\boldsymbol{x}_j}\right)\right) \tag{6}$$

In classical logistic regression a point-estimate of the weights is obtained by maximizing the log-likelihood function. however, this can lead to poor probabilistic calibration. For example, the maximum likelihood probabilities converge to a binary zero-one assignment when the dataset is linearly separable.

Bonus exercise Show that the norm of the maximum likelihood estimate of the weights tends to infinity when the dataset is linearly separable.

We can improve the probabilistic calibration by adoptive a Bayesian perspective. We start by defining a prior distribution over the weights. The simplest (albeit usually unmotivated) approach is to use a spherical Gaussian prior distribution:

$$p(\boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{w}; \boldsymbol{0}, \sigma^2 \mathrm{I}\right) \ . \tag{7}$$

The posterior over the weights is obtained by normalizing the joint distribution:

$$\log p(D, \boldsymbol{w}) = l_{LR}(\boldsymbol{w}) - \frac{1}{2\sigma^2} \left\|\boldsymbol{w}\right\|^2 - \frac{K}{2} \log 2\pi\sigma^2 \ . \tag{8}$$

However, the normalization integral is intractable since the likelihood is not Gaussian. We can therefore approximate the posterior using stochastic variational inference. The first step is to define a parameterized variational family. The simplest choice is to use a mean-field family, where the approximate posterior over each weight is statistically independent from each other:

$$q(\boldsymbol{w}; \boldsymbol{\mu}, \boldsymbol{s}) = \prod_{k}^{K} \mathcal{N}\left(w_k; \mu_k, s_k^2\right) \ . \tag{9}$$

This distribution is parameterized by a mean μ_k and a standard deviation s_k for each of the weights. We can train these variational parameters by maximizing the ELBO:

$$\mathrm{ELBO}(\boldsymbol{\mu}, \boldsymbol{s}) =_q \mathbb{E}_{\boldsymbol{w} \sim q(\boldsymbol{w})} \left[l_{LR}(\boldsymbol{w}) - \frac{\left\|\boldsymbol{w}\right\|^2}{2\sigma^2} + \frac{\left\|(\boldsymbol{\mu} - \boldsymbol{w}) \oslash \boldsymbol{s}\right\|^2}{2} + \frac{1}{2} \sum_{k}^{K} \log s_k^2 \right] \ , \tag{10}$$

where $\oslash$ denotes entry-wise division and $=_q$ denotes equality up to additive terms that do not depend on the variational parameters. We can get an estimator of the gradient of the ELBO by using the reparameterization trick. We start by wrinting the sampled weights $\boldsymbol{w}$ as a function of a standard Gaussian vector:

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon} \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\boldsymbol{0}, \mathrm{I}) \ , \end{aligned} \tag{11}$$

where $\odot$ denotes the entry-wise product. We can now plug-in the reparameterization formula into the our expression for the ELBO:

$$\mathrm{ELBO}(\boldsymbol{\mu}, \boldsymbol{s}) =_q \mathbb{E}_{\boldsymbol{w} \sim q(\boldsymbol{w})} \left[l_{LR}(\boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon}) - \frac{\left\|\boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon}\right\|^2}{2\sigma^2} + \frac{\left\|\boldsymbol{\epsilon}\right\|^2}{2} \right] + \frac{1}{2} \sum_{k}^{K} \log s_k^2 \ , \tag{12}$$

Note that $\left\|\boldsymbol{\epsilon}\right\|^2$ does not depend on the variational parameters. We can therefore simplify the expression into

$$\mathrm{ELBO}(\boldsymbol{\mu}, \boldsymbol{s}) =_q \mathbb{E}_{\boldsymbol{w} \sim q(\boldsymbol{w})} \left[l_{LR}(\boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon}) - \frac{\left\|\boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon}\right\|^2}{2\sigma^2} \right] + \frac{1}{2} \sum_{k}^{K} \log s_k^2 \ . \tag{13}$$

We can finally obtain an unbiased Monte Carlo estimator by replacing the expectation with an average over N samples:

$$\mathcal{E}(\boldsymbol{\mu}, \boldsymbol{s}, N) = \frac{1}{N} \sum_{n=1}^{N} \left(l_{LR}(\boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon}_n) - \frac{\|\boldsymbol{\mu} + \boldsymbol{s} \odot \boldsymbol{\epsilon}_n\|^2}{2\sigma^2} \right) + \frac{1}{2} \sum_{k}^{K} \log s_k^2 \ . \tag{14}$$

Now that we have a Monte Carlo estimator of the ELBO, we can train the variational parameters by stochastic gradient descent:

$$\boldsymbol{\mu}_{n+1} = \boldsymbol{\mu}_n + \eta_n \nabla_{\boldsymbol{\mu}} \mathcal{E}(\boldsymbol{\mu}_n, \boldsymbol{s}_n, N) \tag{15}$$
$$\boldsymbol{s}_{n+1} = \boldsymbol{s}_n + \eta_n \nabla_{\boldsymbol{s}} \mathcal{E}(\boldsymbol{\mu}_n, \boldsymbol{s}_n, N) \ . \tag{16}$$

where η_n is an adaptive learning rate. Note that the samples $\boldsymbol{\epsilon}_n$ have to be re-sampled at each gradient update.
After training, we obtained an approximate posterior $q(\boldsymbol{w}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{s}})$ distribution over the weights that we can use to quantify the uncertainty over the true classification boundary. We can now predict the label y^* given a new feature vector $\boldsymbol{x}^*$ can then be obtained by averaging out over the weights:

$$p(y^* \mid \boldsymbol{x}^*) = \mathbb{E}_{\boldsymbol{w} \sim q(\boldsymbol{w}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{s}})}[p(y^* \mid \boldsymbol{w}, \boldsymbol{x}^*)] \tag{17}$$
$$\approx \frac{1}{M} \sum_{m=1}^{M} p(y^* \mid \boldsymbol{w}_m, \boldsymbol{x}^*), \qquad \boldsymbol{w}_m \sim q(\boldsymbol{w}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{s}}) \ . \tag{18}$$

Note that this prediction reduces to the maximum likelihood prediction $p(y^* \mid \hat{\boldsymbol{w}}_m^{(ML)}, \boldsymbol{x}^*)$ when the posterior is fully concentrated on the maximum likelihood estimator. This happens under certain condition in the limit of infinitely many data-points since the posterior uncertainty and the effect of the prior gradually reduces as the available information increases.

1.1.1 The multivariate normal approach

So far, we approximated the posterior using a mean field approximation. However, this neglects all correlations between the values of the weights in the posterior. In this section we will therefore discuss how to learn correlated posterior distributions using the multivariate normal approach. The idea is to define the variational posterior as a multivariate normal distribution:

$$q(\boldsymbol{w}; \boldsymbol{\mu}, C) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}, C) \tag{19}$$
$$= \frac{1}{(2\pi|\det(C)|)^{K/2}} e^{-\frac{1}{2}(\boldsymbol{\mu}-\boldsymbol{w})^T C^{-1} (\boldsymbol{\mu}-\boldsymbol{w})} \tag{20}$$

where $\boldsymbol{\mu}$ is a mean vector and C is a covariance matrix. Unfortunantly, performing gradient descent on the space of covariance matrices is problematic since these matrices have to be constrained to be symmetric and positive definite, meaning that:

$$C^T = C$$

$$\boldsymbol{v}^T C \boldsymbol{v} \geq 0$$

for all possible vectors $\boldsymbol{v}$. This preperty is only respected by a subset of all possible matrices and performing naive gradient updates would most certainly lead us outside this set. The simplest solution is to re-parameterize the distribution in terms of an arbitrary matrix S and express C as the inner product of S:

$$C = S^T S \ . \tag{21}$$

For obvious reasons, S is often referred as a matrix square root of C. Note however that there are infinitely many ways to express a matrix C in this way, meaning that we have the freedom to chose a convenient form for S. It easy to check that a matrix of the form $C = S^T S$ is guaranteed to be symmetric. In fact

$$C^T = (S^T S)^T = S^T (S^T)^T = S^T S = C \ .$$

Here we used the fact that the transposition of a product is given by the inverted-order product of the transpositions. It is also straightforward to show that the matrix is positive-definite:

$$\boldsymbol{s}^T (S^T S) \boldsymbol{v} = (S\boldsymbol{v})^T (S\boldsymbol{v}) = \|S\boldsymbol{v}\|^2 \geq 0 \ .$$

As noted before, we have freedom in choosing a convenient square root. It is possible to show that every symmetric and positive definite matrix can be decomposed into the inner product of an upper diagonal matrix:

$$C = U^T U \ .$$

where $U_{jk} = 0$ when $j < k$. This is the famous Cholesky decomposition. We are finally in the position to reparameterize an arbitrary multivariate normal variable in terms of a mean vector and an upper diagonal matrix:

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{\mu} + U^T \boldsymbol{\epsilon} \qquad (22) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \mathrm{I}) \ . \end{aligned}$$

The transformed distribution is still a multivariate Gaussian since Gaussian distributions are stable under affine combinations. Furthermore:

$$\mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{w}] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\mu} + U^T \boldsymbol{\epsilon}\right] = \boldsymbol{\mu} + U^T \mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}] = \boldsymbol{\mu} \tag{23}$$

and

$$\begin{aligned} &\mathbb{E}_{\boldsymbol{\epsilon}}\left[(\boldsymbol{w} - \boldsymbol{\mu})(\boldsymbol{w} - \boldsymbol{\mu})^T\right] = \qquad (24) \\ &\mathbb{E}_{\boldsymbol{\epsilon}}\left[(U^T \boldsymbol{\epsilon})(U^T \boldsymbol{\epsilon})^T\right] = U^T \mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\right] U = U^T U = C \ . \qquad (25) \end{aligned}$$

We therefore succeeded in re-parameterizing an arbitrary multivariate Gaussian in terms of a mean vector $\boldsymbol{\mu}$ and the array of the (non-zero) upper diagonal valued U^+ of the matrix U. We can finally give an expression for the reparameterized Bayesian logistic regression ELBO with multivariate Gaussian variational distribution:

$$\begin{aligned}\text{ELBO}(\boldsymbol{\mu}, U^+) =& _q\mathbb{E}_{\boldsymbol{\epsilon}}\left[l(\boldsymbol{\mu} + U^T\boldsymbol{\epsilon})\right] \\ &- \mathbb{E}_{\boldsymbol{\epsilon}}\left[\frac{1}{2\sigma^2}\left\|\boldsymbol{\mu} + U^T\boldsymbol{\epsilon}\right\|^2\right] + \frac{K}{2}\sum_{k=1}^{K}\log|U_{kk}| \ .\end{aligned} \tag{26}$$

where we use the fact that the determinant of an upper triangular matrix is viven by the product of the diagonal entries:

$$\log|\det(U)| = \sum_{k}^{K}\log|U_{kk}| \ . \tag{27}$$

We can now obtain a Monte Carlo estimator and perform SGD in the usual way.

1.2 Bayesian deep learning

Logistic regressions are a very simple statistical technique. However, everything we learned so far generalize straightforwardly to general deep learning architectures. Consider a D-layered binary classification network parameterized by a set of D wights:

$$\rho_j = \text{Sigmoid}(f(x_j; \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(D)})) \ . \tag{28}$$

The weights can parameterized fully connected layers, convolutional layers, batch norm layers, LSTM gates and so on.
In general, the weights $\boldsymbol{w}^{(d)}$ are in the form of a tensor with p indices. For example, in fully connected layers the weights have the shape of a matrix while in 2D convolutions have the shape of a 3-tensor. However, for our purposes we just care about the number of scalar parameters in each weight array and we will therefore denote them using vector notation. We will also denote the concatenation of all weights as $\boldsymbol{w}^{(1:D)}$.
The log likelihood of the network can be expressed as usual:

$$l_{NN}(\boldsymbol{w}^{(1:D)}) = \sum_{j}^{J}\Big(y_j\log\rho_j(\boldsymbol{w}^{(1:D)}) + (1-y_j)(1-\log\rho_j(\boldsymbol{w}^{(1:D)}))\Big) \ . \tag{29}$$

We can transform this network into a Bayesian classification model by assigning a prior to the weights. Again, the simplest (and jet completely unmotivated!) option is to use spherical Gaussians:

$$p(\boldsymbol{w}^{(1:D)}) = \prod_{k=1}^{T}\mathcal{N}(w_k^{(1:D)}; 0, \sigma^2) \ . \tag{30}$$

where T is the total number of parameters of the deep model. To perform variational inference, we can now define a Gaussian mean-field variational distribution:

$$q(\boldsymbol{w}^{(1:D)}) = \prod_{k=1}^{T} \mathcal{N}(w_k^{(1:D)}; \mu_k^{(1:D)}, s_k^{(1:D)}) \ . \tag{31}$$

Note that this is a very drastic approximation as the weights in the true posterior will likely be coupled in a very complex way. The concatenated wight vector can be reparameterized as usual:

$$\begin{aligned} &\boldsymbol{w}^{(1:D)} = \boldsymbol{\mu}^{(1:D)} + \boldsymbol{s}^{(1:D)} \odot \boldsymbol{\epsilon}^{(1:D)} && (32) \\ &\boldsymbol{\epsilon}^{(1:D)} \sim \mathcal{N}(\mathbf{0}, \mathrm{I}) \ . \end{aligned}$$

We can not obtain the usual Monte Carlo reparameterization gradient estimator:

$$\mathcal{E}(\boldsymbol{\mu}, \boldsymbol{s}, N) = \frac{1}{N} \sum_{n=1}^{N} \left(l_{NN}(\boldsymbol{\mu}^{(1:D)} + \boldsymbol{s}^{(1:D)} \odot \boldsymbol{\epsilon}_n^{(1:D)}) - \frac{\left\| \boldsymbol{\mu}^{(1:D)} + \boldsymbol{s}^{(1:D)} \odot \boldsymbol{\epsilon}_n^{(1:D)} \right\|^2}{2\sigma^2} \right) \tag{33}$$

$$+ \frac{1}{2} \sum_{k}^{K} \log \left(s_k^{(1:D)} \right)^2 \ . \tag{34}$$

Et voilà, we now have a general approach to convert an arbitrary deep learning network into a variational model that can be trained by SGD. Instead of training the weights directly, we now train the mean and standard relation of the approximate posterior of each weight. This gives us a quantification of uncertainty in weight space that can then be used to obtained a (hopefully!) better calibrated predictive distribution:

$$p(y^* \mid \boldsymbol{x}^*) = \mathbb{E}_{\boldsymbol{w}^{(1:D)} \sim q(\boldsymbol{w}^{(1:D)}; \hat{\boldsymbol{\mu}}^{(1:D)}, \hat{\boldsymbol{s}}^{(1:D)})} \left[p(y^* \mid \boldsymbol{w}^{(1:D)}, \boldsymbol{x}^*) \right] \tag{35}$$

$$\tag{36}$$

1.2.1 Problems with Bayesian deep learning

While it is interesting that we can convert any deep learning model into a Bayesian classifier, there are many problems that need to be considered in this form of probabilistic deep learning. First of all, the probabilistic calibration of any Bayesian classifier depends on the choice of a meaningful prior and there is not a valid reason to think that uncoupled Gaussian priors over the weights induce an appropriate prior distribution for the average classification problem. Even more seriously, the Gaussian mean field approximation is likely to be largely inappropriate in this complex setting, likely leading to a severe underestimation of the uncertainty and consequently to a bad error calibration.

1.3 Multi-class classification

All the techniques we developed so far for the binary case can be straightforwardly extended to the multi-class setting by changing the likelihood function. We will denote one-hot encoded class labels as vectors $\boldsymbol{y}_j = ([y_j]_1, \ldots, [y_j]_M$, where M is the number of classes. We can turn the output $\boldsymbol{f}(x_j; \boldsymbol{w}^{(1:D)})$ of a network with M output nodes into class probabilities by using a softmax function:

$$[\boldsymbol{\rho}_j(\boldsymbol{w}^{(1:D)})]_m = \frac{e^{f_m(x_j;\boldsymbol{w}^{(1:D)})}}{\sum_h^M e^{f_h\left(x_j;\boldsymbol{w}^{(1:D)}\right)}} \, . \tag{37}$$

We can now define a multi-class classification likelihood using a categorical distribution:

$$\begin{aligned} \log p(D \mid \boldsymbol{w}^{(1:D)}) &= \sum_m^M [y_j]_m \log\left([\boldsymbol{\rho}_j(\boldsymbol{w}^{(1:D)})]_m\right) && (38) \\ &= \left(\sum_m^M [y_j]_m f_m(x_j; \boldsymbol{w}^{(1:D)})\right) - \log\left(\sum_h^M e^{f_h\left(x_j;\boldsymbol{w}^{(1:D)}\right)}\right) \, . && (39) \end{aligned}$$

Unfortunately, this log likelihood can be numerically very unstable if implemented naively since the sum of exponential functions inside the log tends to easily overflow/underflow. We can solve this problem by noticing that, for any number α we have

$$\begin{aligned} \log\left(\sum_h^M e^{f_h\left(x_j;\boldsymbol{w}^{(1:D)}\right)}\right) &= \log\left(e^{\alpha} e^{-\alpha} \sum_h^M e^{f_h\left(x_j;\boldsymbol{w}^{(1:D)}\right)}\right) && (40) \\ &= -\alpha + \log\left(\sum_h^M e^{f_h\left(x_j;\boldsymbol{w}^{(1:D)}\right)-\alpha}\right) \, . && (41) \end{aligned}$$

We can now choose α to be equal to the $f_h\left(x_j; \boldsymbol{w}^{(1:D)}\right)$ with the highest value. This removes a large number from the exponent and reduces the risk of overflow.

1.4 Regression and beyond

In a regression problem, the target y_j is a real number instead of a discrete label. From a probabilistic point of view, a regression model differs from a classification model only by the use of a different likelihood. Consider again a network $f(x_j; \boldsymbol{w}^{(1:D)})$ with linear activation functions in the output layer. We can reproduce the standard squared loss by using a Gaussian likelihood with unit variance:

$$\begin{aligned} \log p(D \mid \boldsymbol{w}^{(1:D)}) &= \log \prod_j \mathcal{N}(y_j; f(x_j; \boldsymbol{w}^{(1:D)}), 1) && (42) \\ &= -\frac{1}{2} \sum_k \left(y_j - f(x_j; \boldsymbol{w}^{(1:D)})\right)^2 && (43) \end{aligned}$$

where I omitted additive constant terms. All the details of the variational inference is identical to our classification case just with a different log likelihood function.

1.4.1 Robust likelihoods

While there is not much freedom in choosing a probabilistically justified loss function in the classification case, the squared loss is far from being the only choice in regression problems and it is often sub-optimal in practice. A reason for the sub-optimality is that the square function is highly influenced by few outliers data-points since large deviations are magnified while small deviations are suppressed. A possible, more balanced alternative is given by the Laplace distribution:

$$\mathcal{L}(x; \mu, s) = \frac{1}{2s} e^{-|x-\mu|/s} \tag{44}$$

which leads to the absolute deviation loss:

$$\log p(D \mid \boldsymbol{w}^{(1:D)}) = \log \prod_j \mathcal{L}(x; f(x_j; \boldsymbol{w}^{(1:D)}), 1) \tag{45}$$

$$= -\frac{1}{2} \sum_k |y_j - f(x_j; \boldsymbol{w}^{(1:D)})| \tag{46}$$

1.4.2 Predicting counting data: Poisson likelihood

As you can see, classification and regression problems are just special cases of supervise statistical prediction problems corresponding to specific likelihoods. Real-world data can come in many different other forms and a good machine learning engineer should be familiar with a larger family of problems and consequently likelihood functions. A common example are counting data, where the target y_j is a natural number. In these situations, statisticians often use Poisson distributions:

$$\mathcal{P}(y; \lambda) = \lambda^y \frac{e^{-\lambda}}{y!} \ . \tag{47}$$

which leads to the loss function

$$y_j \log f(x_j; \boldsymbol{w}^{(1:D)}) - f(x_j; \boldsymbol{w}^{(1:D)}) \ . \tag{48}$$

Note however than n a Poisson likelihood the variance is always equal to the mean and this can be too restrictive in some situations.

1.5 Example: Variational frequency analysis

Supervised learning goes much beyond the standard kind of problems that you can find in a machine learning blog and encompass all areas of science and engendering. As an example, I will show you how to perform a Fourier analysis of a binary sequence. We start by defining a analog signal as a mixture of a J sine and cosine waves:

$$s(t) = \sum_{j}^{J} \left(\alpha_j \cos(\omega_j t) + \beta_j \sin(\omega_j t)\right) \ . \tag{49}$$

where ω_j are fixed angular frequency values. We can now define a data generation model for a binary sequence using Bernoulli distributions:

$$y_1, \dots, y_T \sim \prod_t \text{Sigmoid}(g s(t))^{y_t} \text{Sigmoid}(-g s(t))^{1-y_t} \tag{50}$$

where g is a gain term that regulates the amount of noise in the sampled signal. We can now obtain a probabilistic model by assigning probability distributions to the Fourier amplitudes α_j and β_j. For simplicity, here we will use normal distributions. However, remember that the right distribution to use depends on your specific problem.

$$p(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_j \mathcal{N}(\alpha_j; 0, \sigma^2)\mathcal{N}(\beta_j; 0, \sigma^2) \tag{51}$$

We can now define a Gaussian mean field model (or better a multivariate normal or something even more expressive) and train the parameters by stochastic gradient descent.

$$q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_j \mathcal{N}(\alpha_j; \mu_j^{(\alpha)}, {s_j^{(\alpha)}}^2)\mathcal{N}(\beta_j; \mu_j^{(\beta)}, {s_j^{(\beta)}}^2) \tag{52}$$

Exercise Find a reparameterized Monte Carlo estimator of the ELBO.