Deep Learning - Foundations and Concepts

Chapter 19. Autoencoders

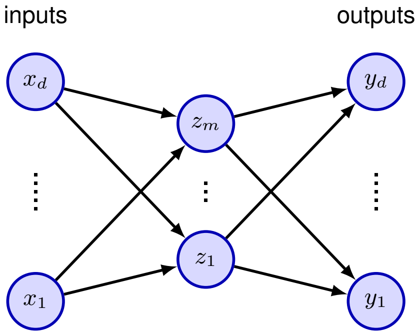
nonlineark@github

April 19, 2025

Outline

Linear autoencoders

Figure: An autoencoder neural network having two layers of weights



Linear autoencoders

Consider a two-layer neural network having D inputs, D output units and M hidden units, with $M < D$. The targets used to train the network are simply the input vectors themselves, so that the network attempts to map each input vector onto itself. We choose a sum-of-squares error of the form:

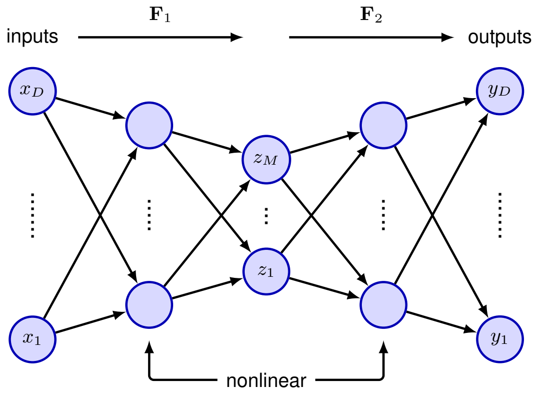
$$E(w) = \frac{1}{2} \sum_{n=1}^{N} ||y(x_n; w) - x_n||^2$$

Linear autoencoders

- If the hidden units have linear activation functions, then it can be shown that when the error function is minimized, the network performs a projection onto the M-dimensional subspace that is spanned by the first M principal components of the data.
- Even with nonlinear hidden units, the minimum error solution is again given by the projection onto the principal component subspace. There is therefore no advantage in using two-layer neural networks to perform dimensionality reduction.

Deep autoencoders

Figure: A four-layer auto-associative network that can perform a nonlinear dimensionality reduction



Deep autoencoders

Consider the four-layer auto-associative network shown on the previous slide:

- The output units are linear, and the M units in the second layer can also be linear.
- However, the first and third layers have sigmoidal nonlinear activation functions.

We can view this network as two successive functional mappings F_1 and F_2:

- F_1 projects the original D-dimensional data onto an M-dimensional subspace defined by the activations of the units in the second layer.
- F_2 maps from the M-dimensional hidden space back into the original D-dimensional input space.

Deep autoencoders

- Such a network effectively performs a nonlinear form of PCA.
- However:
    - Training the network now involves a nonlinear optimization, and computationally intensive nonlinear optimization techniques must be used.
    - There is the risk of finding a sub-optimal local minimum of the error function.
    - The dimensionality of the subspace must be specified before training the network.

Sparse autoencoders

Instead of limiting the number of nodes in one of the hidden layers in the network, an alternative way to constrain the internal representation is to use a regularizer to encourage a sparse representation:

$$\tilde{E}(w) = E(w) + \lambda \sum_{k=1}^{K} |z_k|$$

where $E(w)$ is the unregularized error, and the sum over k is taken over the activation values of all the units in one of the hidden layers.

Denoising autoencoders

The idea of denoising autoencoders is to take each input vector x_n and to corrupt it with noise to give a modified vector $\tilde{x}_n$ which is then input to an autoencoder to give an output $y(\tilde{x}_n; w)$. The network is trained to reconstruct the original noise-free input vector:

$$E(w) = \sum_{n=1}^{N} ||y(\tilde{x}_n; w) - x_n||^2$$

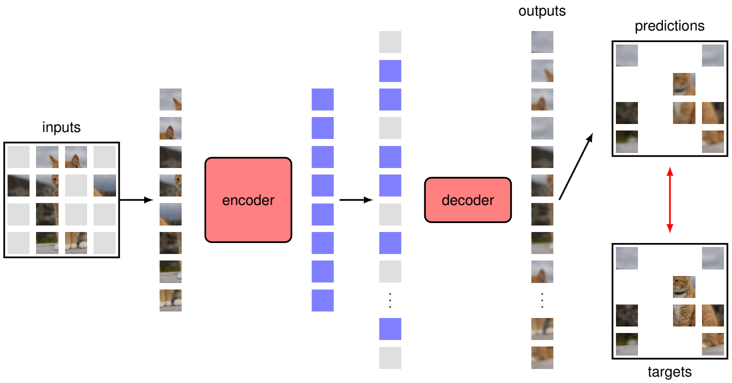
Denoising autoencoders

- One form of noise involves setting a randomly chosen subset of the input variables to zero.
- An alternative approach is to add independent zero-mean Gaussian noise to every input variable, where the scale of the noise is set by the variance of the Gaussian.

Masked autoencoders

- In a masked autoencoder, a deep network is used to reconstruct an image given a corrupted version of that image as input. The form of corruption is masking, or dropping out, part of the input image.
- This technique is generally used in combination with a vision transformer architecture.
- Compared to language, images have much more redundancy along with strong local correlations. The best internal representations are learned when a relatively high proportion of the input image is masked, typically 75%.
- The decoder is discarded and the encoder is applied to the full image with no masking and with a fresh set of output layers that are fine-tuned for the required application.

Masked autoencoders



Variational autoencoders

Because evaluating the likelihood function for a latent-variable model is intractable, the variational autoencoders (VAEs) instead work with an approximation to this likelihood when training the model:

- Use of the evidence lower bound (ELBO) to approximate the likelihood function.
- Amortized inference in which a second model, the encoder network, is used to approximate the posterior distributions over latent variables in the E step.
- Making the training of the encoder model tractable using the reparameterization trick.

Evidence lower bound

Consider a generative model:

- The distribution over the M-dimensional latent variable z is given by a zero-mean unit-variance Gaussian: $p(z) = \mathcal{N}(z; 0, I)$.
- The conditional distribution $p(x|z; w)$ over the D-dimensional data variable x is governed by the output of a deep neural network $g(z; w)$.

Evidence lower bound

We know that for an arbitrary probability distribution $q(z)$ over a space described by the latent variable z, we have:

$$\log p(x;w) = \mathcal{L}(w) + \mathrm{KL}(q(z)||p(z|x;w)) \geq \mathcal{L}(w)$$

$$\mathcal{L}(w) = \int q(z) \log \frac{p(x|z;w)p(z)}{q(z)} \mathrm{d}z$$

$$\mathrm{KL}(q(z)||p(z|x;w)) = -\int q(z) \log \frac{p(z|x;w)}{q(z)} \mathrm{d}z$$

Although the log likelihood $\log p(x;w)$ is intractable, we will seek a way to evaluate $\mathcal{L}(w)$ as an approximation to the true log likelihood.

Evidence lower bound

Now consider a set of training data points $\mathcal{D} = \{x_1, \dots, x_N\}$, which are assumed to be drawn independently from the model distribution $p(x)$. The log likelihood function for this data set is given by:

$$\log p(\mathcal{D}; w) = \sum_{n=1}^{N} \mathcal{L}_n(w) + \sum_{n=1}^{N} \mathrm{KL}(q_n(z_n) || p(z_n | x_n; w))$$

$$\mathcal{L}_n(w) = \int q_n(z_n) \log \frac{p(x_n | z_n; w) p(z_n)}{q_n(z_n)} \mathrm{d}z_n$$

Evidence lower bound

- Theorectically, we can set each $q_n(z_n)$ equal to the corresponding posterior distribution $p(z_n|x_n;w)$, which gives zero Kullback-Leibler divergence, and hence the lower bound is equal to the true log likelihood.
- However, in practice, exact evaluation of $p(z_n|x_n;w)$ is intractable. We therefore need to find an approximation to the posterior distribution.

Amortized inference

- In the variational autoencoder, we train a single neural network, called the encoder network, to approximate all the posterior distributions $p(z_n|x_n; w)$.
- The encoder produces a single distribution $q(z|x; \phi)$ that is conditioned on x, where ϕ represents the parameters of the network.
- The objective function, given by the evidence lower bound, now has a dependence on ϕ as well as w, and we maximize the bound jointly with respect to both sets of parameters.

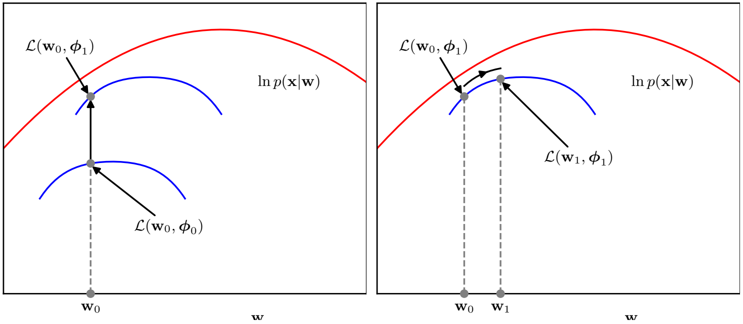
Amortized inference

A typical choice for the encoder is a Gaussian distribution with a diagonal covariance matrix:

$$q(z|x;\phi) = \prod_{m=1}^{M} \mathcal{N}(z_m; \mu_m(x,\phi), \sigma_m^2(x,\phi))$$

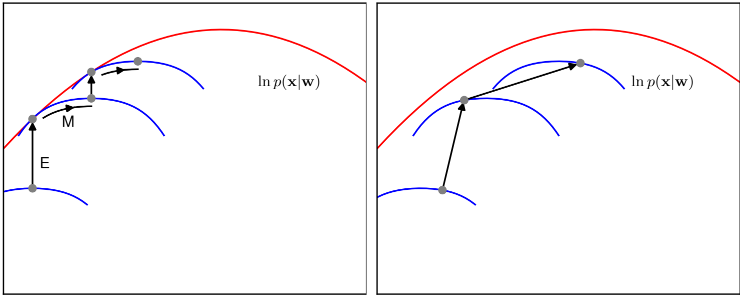
Amortized inference

Figure: Illustration of the optimization of the ELBO


$\mathcal{L}(\mathbf{w}_0, \boldsymbol{\phi}_1)$
$\ln p(\mathbf{x}|\mathbf{w})$
$\mathcal{L}(\mathbf{w}_0, \boldsymbol{\phi}_0)$
$\mathbf{w}_0$
$\mathbf{w}$
$\mathcal{L}(\mathbf{w}_0, \boldsymbol{\phi}_1)$
$\ln p(\mathbf{x}|\mathbf{w})$
$\mathcal{L}(\mathbf{w}_1, \boldsymbol{\phi}_1)$
$\mathbf{w}_0$
$\mathbf{w}_1$
$\mathbf{w}$

Amortized inference

Figure: Comparison of the EM algorithm with ELBO optimization in a VAE


$\ln p(\mathbf{x}|\mathbf{w})$
M
E
$\ln p(\mathbf{x}|\mathbf{w})$

The reparameterization trick

For data point x_n, we can write the contribution to the lower bound in the form:

$$
\begin{aligned}
\mathcal{L}(w,\phi) &= \int q(z_n|x_n;\phi)\log\frac{p(x_n|z_n;w)p(z_n)}{q(z_n|x_n;\phi)}\mathrm{d}z_n \\
&= \int q(z_n|x_n;\phi)\log p(x_n|z_n;w)\mathrm{d}z_n - \mathrm{KL}(q(z_n|x_n;\phi)||p(z_n))
\end{aligned}
$$

The reparameterization trick

For the first term, we could try to approximate the intergral over z_n with a simple Monte Carlo estimator:

$$\int q(z_n|x_n;\phi)\log p(x_n|z_n;w)\mathrm{d}z_n \approx \frac{1}{L}\sum_{l=1}^{L}\log p(x_n|z_n^{(l)};w)$$

where $z_n^{(l)}$ are samples drawn from the encoder distribution $q(z_n|x_n;\phi)$. This is easily differentiated with respect to w, but the gradient with respect to ϕ is problematic.

The reparameterization trick

We can resolve this by making use of the reparameterization trick in which we reformulate the Monte Carlo sampling procedure such that derivatives with respect to ϕ can be calculated explicitly. Instead of drawing samples of z_n directly, we draw samples from $\mathcal{N}(\epsilon; 0, I)$:

$$z_{nm}^{(l)} = \mu_{nm} + \sigma_{nm}\epsilon_{nm}^{(l)}$$

where $\mu_{nm} = \mu_m(x_n, \phi)$ and $\sigma_{nm} = \sigma_m(x_n, \phi)$. This makes the dependence on ϕ explicit and allows gradients with respect to ϕ to be evaluated.

The reparameterization trick

The second term for $\mathcal{L}(w, \phi)$ is a Kullback-Leibler divergence between two Gaussian distributions and can be evaluated analytically:

$$\mathrm{KL}(q(z_n|x_n;\phi)||p(z_n)) = \frac{1}{2}\sum_{m=1}^{M}(-1 - \log\sigma_{nm}^2 + \mu_{nm}^2 + \sigma_{nm}^2)$$

The full error function for the VAE, therefore becomes:

$$\begin{aligned} z_n^{(l)} &= \mu_n + \mathrm{diag}(\sigma_{n1}, \ldots, \sigma_{nM})\epsilon_n^{(l)} \\ \mathcal{L}(w, \phi) &= \sum_{n=1}^{N}(\frac{1}{L}\sum_{l=1}^{L}\log p(x_n|z_n^{(l)}; w) \\ &\quad + \frac{1}{2}\sum_{m=1}^{M}(1 + \log\sigma_{nm}^2 - \mu_{nm}^2 - \sigma_{nm}^2)) \end{aligned}$$

The reparameterization trick

Algorithm 1: Variational autoencoder training

```
repeat
    $\mathcal{L} \leftarrow 0$;
    for $m \leftarrow 1$ to $M$ do
        $\epsilon_{nm} \sim \mathcal{N}(\epsilon; 0, 1)$;
        $z_{nm} \leftarrow \mu_{nm} + \sigma_{nm}\epsilon_{nm}$;
        $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{2}(1 + \log \sigma_{nm}^2 - \mu_{nm}^2 - \sigma_{nm}^2)$;
    end
    $\mathcal{L} \leftarrow \mathcal{L} + \log p(x_n | z_n; w)$;
    $w \leftarrow w + \eta \nabla_w \mathcal{L}$;
    $\phi \leftarrow \phi + \eta \nabla_\phi \mathcal{L}$;
until *converged*;
return $w, \phi$;
```

The reparameterization trick

Problems when training VAEs:

- Posterior collapse: The variational distribution $q(z|x;\phi)$ converges to the prior distribution $p(z)$ and therefore becomes uninformative because it no longer depends on x.
- Latent code is not compressed.

Both problems can be addressed by introducing a coefficient β in front of the second term in $\mathcal{L}(w,\phi)$ to control the regularization effectiveness of the Kullback-Leibler divergence (further references).