Deep Learning - Foundations and Concepts

Chapter 6. Deep Neural Networks

nonlineark@github

February 21, 2025

Outline

The curse of dimensionality

In spaces of higher dimensionality, the number of combinations of values must be considered could be huge. This effect is known as combinatorial explosion:

- A polynomial regression of order M for a single input variable needs $M+1$ parameters. If there are D input variables, the number of parameters needed will be $\binom{M+D}{M}$.
- The histogram based classification for 1-dimensional input needs N buckets. If the input is D-dimensional, the number of buckets needed will be N^D.

For a machine learning model, this usually means that the amount of data needed to generlize accurately grows exponentially.

High-dimensional spaces

High-dimensional spaces can defeat one's geometrical intuitions:

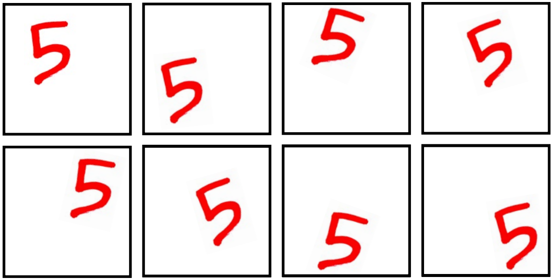
- In spaces of high dimensionality, most of the volume of a hypersphere is concentrated in a thin shell near the surface.
- In spaces of high dimensionality, the probability mass of the Gaussian is concentrated in a thin shell at a specific radius (a soap bubble).

Data manifolds

Although data may be in high-dimensional spaces, real data will generally be confined to a region of the data space having lower effective dimensionality. Effectively, neural networks learn a set of basis functions that are adpated to data manifolds.

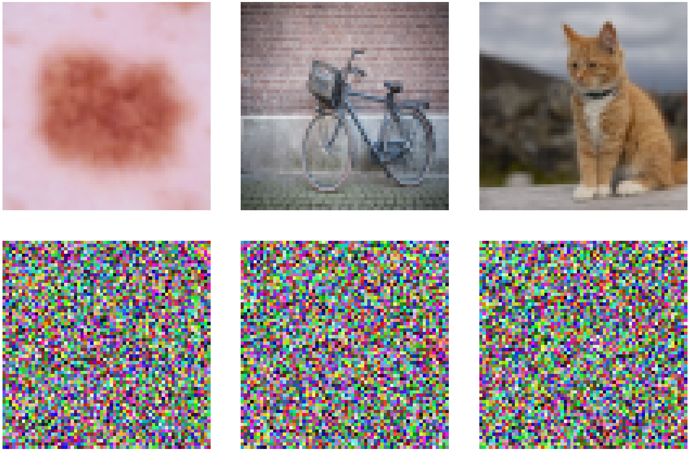
Data manifolds

Figure: Images of a handwritten digit that lives on a nonlinear three-dimensional manifold



Data manifolds

Figure: Natural images vs. randomly generated images



Data-dependent basis functions

- Simple basis functions that are chosen independently of the problem being solved can run into significant limitations.
- Using expert knowledge to hand-craft the basis functions was superseded by data-driven approaches in which basis functions are learned from the training data.
- Methods such as radial basis functions and support vector machines have been superseded by deep neural networks, which are much better at exploiting very large data sets efficiently.

Parameter matrices

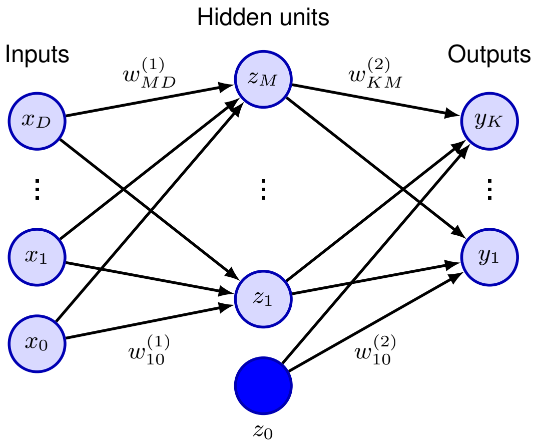
Consider a basic neural network model having two layers of learnable parameters:

$$
\begin{aligned}
a_m^{(1)} &= \sum_{d=1}^{D} w_{md}^{(1)} x_d + w_{m0}^{(1)} \\
z_m^{(1)} &= h(a_m^{(1)}) \\
a_k^{(2)} &= \sum_{m=1}^{M} w_{km}^{(2)} z_m^{(1)} + w_{k0}^{(2)}
\end{aligned}
$$

where h is a differentiable, nonlinear activation function.

Parameter matrices

Figure: Network diagram for a two-layer neural network



Parameter matrices

The bias parameters can be absorbed into the set of weight parameters, so the two-layer neural network can be represented as:

$$y_k(x;w) = f(\sum_{m=0}^{M} w_{km}^{(2)} h(\sum_{d=0}^{D} w_{md}^{(1)} x_d))$$
$$y(x;w) = f(W^{(2)} h(W^{(1)} x))$$

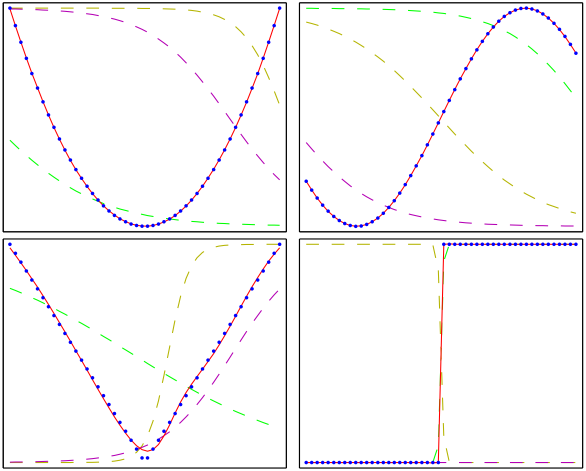
where f and h are activation functions evaluated on each vector element separately.

Universal approximation

- For a wide range of activation functions, two-layer feed-forward networks can approximate any function defined over a continuous subset of $\mathbb{R}^D$ to arbitrary accuracy.
- However, in a practical application, there can be huge benefits in considering networks having many more than two layers that can learn hierarchical internal representations.

Universal approximation

Figure: Two-layer neural networks are universal approximators

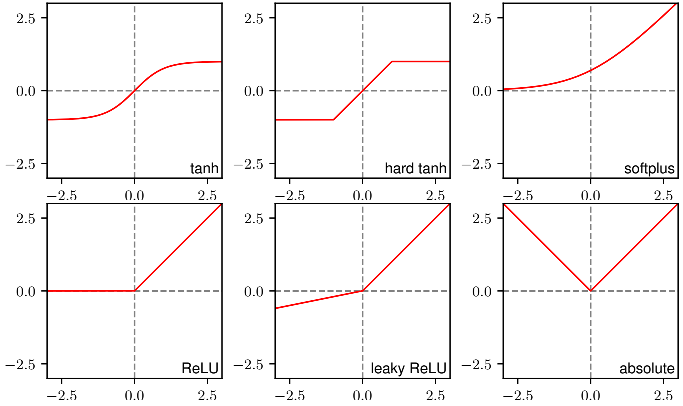


Hidden unit activation functions

- Activation functions for the output units are determined by the kind of distribution being modelled.
- For the hidden units, the only requirement is that they need to be differentiable.
- Obviously, the identity function, sometimes used as the activation function for output units, is not a good option for hidden units.

Hidden unit activation functions

Figure: A variety of nonlinear activation functions



Weight-space symmetries

Consider a two-layer network with M hidden units having $\tanh$ activation functions and full connectivity in both layers:

- Changing the sign of all the weights and the bias feeding into a particular hidden unit can be compensated by changing the sign of all the weights leading out of that hidden unit:
    - 2^M equivalent weight vectors.
- Interchange a particular hidden unit with a different hidden unit:
    - $M!$ equivalent weight vectors.

Deep networks

We can easily extend the two-layer network architecture to any finite number L of layers:

$$z^{(l)} = h^{(l)}(W^{(l)} z^{(l-1)}) \qquad l = 1, \dots, L$$

where $h^{(l)}$ denotes the activation function associated with layer l, and $W^{(l)}$ denotes the corresponding matrix of weight and bias parameters.

Hierarchical representations

The deep neural network architecture encodes a particular form of inductive bias, namely the outputs are related to the input space through a hierarchical representation:

- Low-level features in early layers, e.g., edges.
- Higher-level features in subsequent layers, e.g., eyes.
- Combined in later layers to detect high-level concept, e.g., cats.

Distributed representations

Consider a hidden layer with M hidden units:

- Conceptually, each hidden unit can be thought of as representing a feature, so this hidden layer can represent M different features.
- However, the network can learn a different representation, in which combinations of hidden units represent features, so this hidden layer can represent 2^M different features.

Representation learning

Consider a neural network for a two-class classification problem:

- The final layer can be viewed as a simple linear classifier.
- In the representation of the last hidden layer, the two classes must be well separated by a linear surface.
- This neural network transforms the input data into a representation that's easy for classification purposes.

This ability to discover a nonlinear transformation of the data that makes subsequent tasks easier to solve is called representatioin learning. The learned representation is sometimes called the embedding space.

Transfer learning

- The process of learning parameters using one task that are then applied to one or more other tasks is called pre-training:
  - Send the new training data once through the fixed pre-trained network to obtain the training inputs in the new representation.
  - Iterative gradient-based optimization can then be applied just to the smaller network consisting of the final layers.
- Instead of using a pre-trained network as a fixed pre-processor, it is also possible to apply fine-tuning in which the whole network is adapted to the data for the new task.
- In multitask learning, a network jointly learns more than one related task at the same time. For example, spam email filter for different users.
- Meta-learning: Learn the learning algorithm itself. For example, few-shot learning, one-shot learning.

Contrastive learning

Contrastive learning focuses on extracting meaningful representations by contrasting positive and negative pairs of instances. Given:

- x: The anchor.
- x^+: Positive.
- $\{x_1^-, \ldots, x_N^-\}$: Negative.
- f_w: The neural network function that maps points from input space to a representation space, governed by learnable parameters w. Futher, we require $||f_w(x)|| = 1$.

The loss function (called InfoNCE loss) is defined by:

$$E(w) = -\log \frac{\exp(f_w(x)^T f_w(x^+))}{\exp(f_w(x)^T f_w(x^+)) + \sum_{n=1}^{N} \exp(f_w(x)^T f_w(x_n^-))}$$

General network architectures

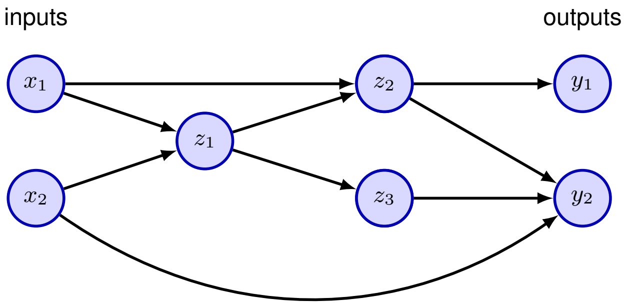
The neural network architecture does not have to be organized into a sequence of fully-connected layers. The network diagram only needs to be restricted to a feed-forward architecture. Each unit in such a network computes a function given by:

$$z_k = h(\sum_{j \in \mathcal{A}(k)} w_{kj} z_j + b_k)$$

where $\mathcal{A}(k)$ denotes the set of ancestors of node k, and b_k denotes the associated bias parameter.

General network architectures

Figure: A neural network having a general feed-forward topology



Regression

For single target variable:

$$p(t|x;w) = \mathcal{N}(t; y(x;w), \sigma^2)$$

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

$$\frac{\partial E}{\partial a_n} = y(x_n;w) - t_n$$

For multiple target variables:

$$p(t|x;w) = \mathcal{N}(t; y(x;w), \sigma^2 I)$$

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

$$\frac{\partial E}{\partial a^n} = (y(x^n;w) - t^n)^T$$

Binary classification

For single target variable:

$$p(t|x;w) = y(x;w)^t (1 - y(x;w))^{1-t}$$

$$E(w) = -\sum_{n=1}^{N} (t_n \log y_n + (1 - t_n) \log(1 - y_n))$$

$$\frac{\partial E}{\partial a_n} = y_n - t_n$$

Binary classification

For multiple target variables:

$$p(t|x; w_1, \dots, w_K) = \prod_{k=1}^{K} y(x; w_k)^{t_k} (1 - y(x; w_k))^{1-t_k}$$

$$E(w_1, \dots, w_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} (t_k^n \log y_k^n + (1 - t_k^n) \log(1 - y_k^n))$$

$$\frac{\partial E}{\partial a^n} = (y^n - t^n)^T$$

Multiclass classification

$$p(t|x; w_1, \dots, w_K) = \prod_{k=1}^{K} y_k(x; w_1, \dots, w_K)^{t_k}$$

$$E(w_1, \dots, w_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_k^n \log y_k^n$$

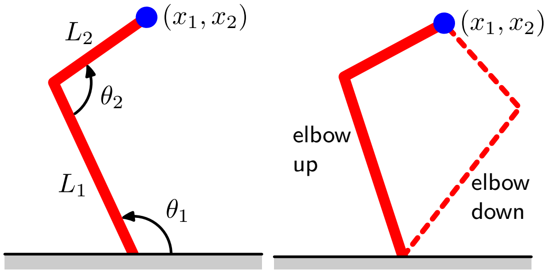
$$\frac{\partial E}{\partial a^n} = (y^n - t^n)^T$$

Robot kinematics example

- For many simple regression problems, the conditional distribution $p(t|x)$ we choose to model is Gaussian.
- Practical machine learning problems can often have significantly non-Gaussian distributions.
- Forward problem (many-to-one) vs. inverse problem (multimodal):
  - Kinematics of a robot arm:
    - Forward problem: Given the joint angles, find the end effector position.
    - Inverse problem: Given the end effector position, find the joint angles.
  - Symptoms in the human body:
    - Forward problem: Given the presence of a particular disease, find the pattern of symptoms.
    - Inverse problem: Given the pattern of symptoms, predict the presence of a disease.

Robot kinematics example

Figure: A robot kinematics example



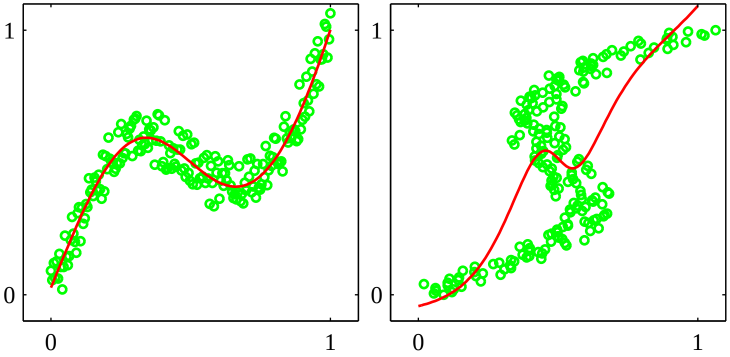
Conditional mixture distributions

The inverse problem for tutorial purposes:

- $\{x_1, \ldots, x_N\}$ is uniformly sampled from $(0, 1)$.
- $t_n = x_n + 0.3 \sin(2\pi x_n) + \epsilon$, where ϵ is some uniform noise over the interval $(-0.1, 0.1)$.
- The inverse problem is obtained by exchanging the roles of x and t.

Conditional mixture distributions

Figure: The inverse problem for tutorial purposes



Conditional mixture distributions

We will model $p(t|x)$ using a mixture model in which both the mixing coefficients as well as the component densities are flexible functions of x:

$$p(t|x) = \sum_{k=1}^{K} \pi_k(x)\mathcal{N}(t; \mu_k(x), \sigma_k^2(x)I)$$

where there are K components in total, and $t \in \mathbb{R}^L$.

Conditional mixture distributions

We now take the various parameters of the mixture model to be governed by the outputs of a neural network that takes x as its input:

- $\pi_k(x)$:
  - K output units, with pre-activations denoted by a_k^π.
  - Activation function: $\pi_k(x) = \frac{\exp(a_k^\pi)}{\sum_{j=1}^K \exp(a_j^\pi)}$.
- $\mu_k(x)$:
  - KL output units, with pre-activations denoted by a_{kl}^μ.
  - Activation function: The identity function, $\mu_{kl}(x) = a_{kl}^\mu$.
- $\sigma_k(x)$:
  - K output units, with pre-activations denoted by a_k^σ.
  - Activation function: $\sigma_k(x) = \exp(a_k^\sigma)$.

Gradient optimization

The error function (the negative log of the maximum likelihood) takes the form:

$$E(w) = -\sum_{n=1}^{N} \log(\sum_{k=1}^{K} \pi_k(x_n)\mathcal{N}(t_n; \mu_k(x_n), \sigma_k^2(x_n)I))$$

Gradient optimization

Let's compute the derivatives of the error of the nth term with respect to the output-unit pre-activations:

$$\frac{\partial E_n}{\partial a_k^\pi} = -\frac{1}{\sum_{j=1}^K \pi_j \mathcal{N}_{nj}} \sum_{j=1}^K \mathcal{N}_{nj} \frac{\partial \pi_k}{\partial a_k^\pi} = \pi_k - \frac{\pi_k \mathcal{N}_{nk}}{\sum_{j=1}^K \pi_j \mathcal{N}_{nj}}$$

$$\frac{\partial E_n}{\partial a_k^\mu} = -\frac{1}{\sum_{j=1}^K \pi_j \mathcal{N}_{nj}} \pi_k \mathcal{N}_{nk} (-\frac{1}{\sigma_k^2})(\mu_k - t_n)^T$$

$$= \frac{\pi_k \mathcal{N}_{nk}}{\sum_{j=1}^K \pi_j \mathcal{N}_{nj}} \frac{(\mu_k - t_n)^T}{\sigma_k^2}$$

$$\frac{\partial E_n}{\partial a_k^\sigma} = -\frac{1}{\sum_{j=1}^K \pi_j \mathcal{N}_{nj}} \pi_k (-\frac{L}{\sigma_k} \mathcal{N}_{nk} + \mathcal{N}_{nk} \frac{||\mu_k - t_n||^2}{\sigma_k^3}) \frac{\mathrm{d}\sigma_k}{\mathrm{d}a_k^\sigma}$$

$$= \frac{\pi_k \mathcal{N}_{nk}}{\sum_{j=1}^K \pi_j \mathcal{N}_{nj}} (L - \frac{||\mu_k - t_n||^2}{\sigma_k^2})$$

Predictive distribution

Compute the conditional mean and variance, we see that:

$$E(t|x) = \int tp(t|x)\mathrm{d}t = \sum_{k=1}^{K} \pi_k \int t\mathcal{N}(t; \mu_k, \sigma_k^2 I)\mathrm{d}t = \sum_{k=1}^{K} \pi_k \mu_k$$

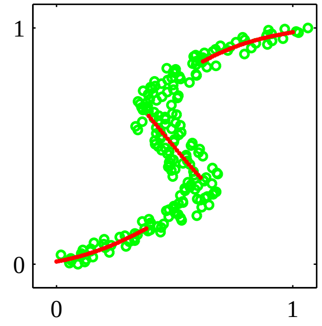
$$\begin{aligned} E(||t-x||^2|x) &= E(||t||^2|x) - ||E(t|x)||^2 \\ &= \sum_{k=1}^{K} \pi_k(||\mu_k||^2 + \sigma_k^2) - ||\sum_{k=1}^{K} \pi_k \mu_k||^2 \end{aligned}$$

Compared with least-squares result, the mixture density network is more general:

- It can reproduce the least-squares result as a special case.
- The variance is more general because it is a function of x.

Predictive distribution

Figure: Plot of the approximate conditional mode



(d)