Ch.6 - Deep Neural Networks

Created	@January 17, 2024 10:40 AM
Book	Deep Learning: Foundations and Concepts

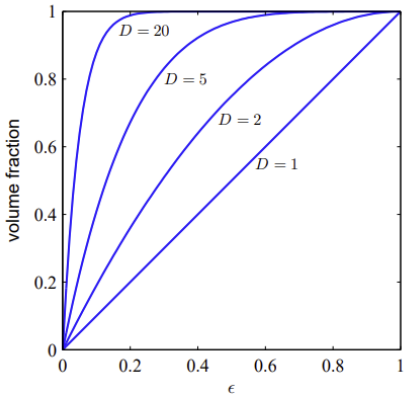
High Dimensional Space

High dimensional feature space embeds a severe limitation where its change rate of volume is huge in respect to $\bar{x}$. The effect of it can be observed in performance of a model.

- Classification would need much larger training set as the dimension grows
- A subtle change would make much difference in estimation

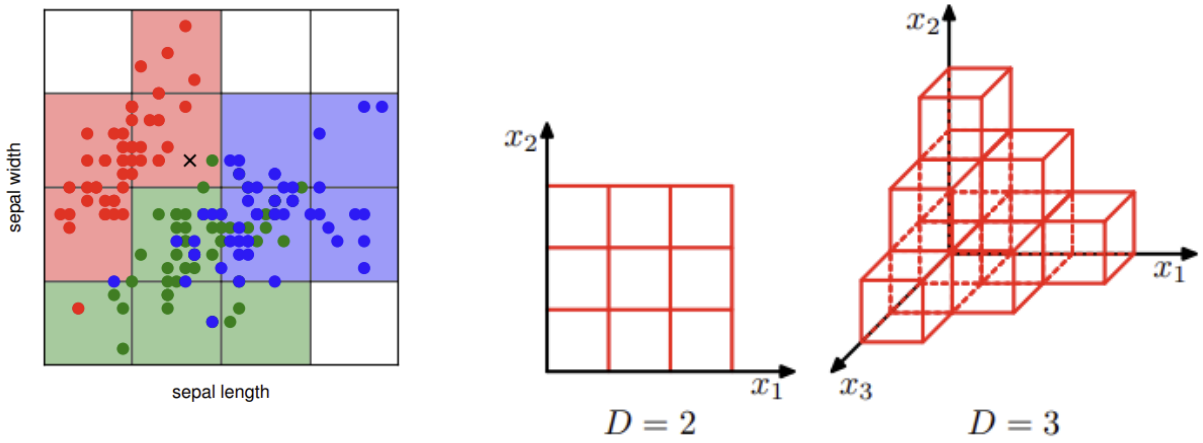
This geometrical feature is plotted as a fraction of the volume of hypersphere by D in the figure left:

$$f_D(r=1,\epsilon) = \frac{V_D(1) - V_D(1-\epsilon)}{V_D(1)}$$



< subtle change will make huge difference if D is large >

Fixed Basis Function



To elaborate more on the curse of dimensionality, a simple classification basis function can be introduced.

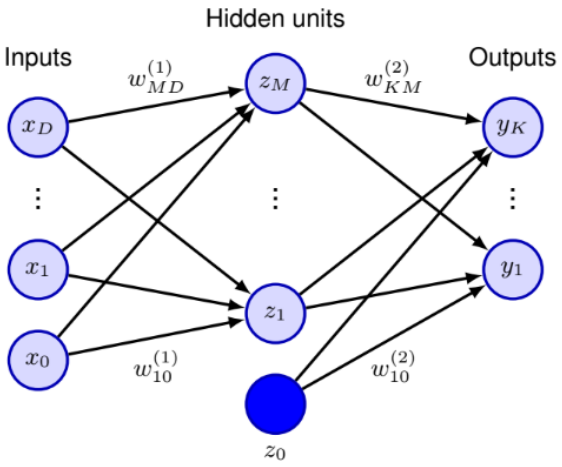
$$\phi(x) = \begin{cases} C_k & \text{the majority class in the cell} \\ 0 & \text{if x lies outside the cell} \end{cases}$$

As the dimension of x grows, the number of grid cell also increases exponentially, requiring much more data to fill the grid cells. Using A data dependent basis function can mitigate this severity such as *Support Vector Machine*, which will be studied later.

ex) $\phi_n(x) = exp(-\frac{||x-x_n||^2}{s^2})$

Multi-Layer Networks

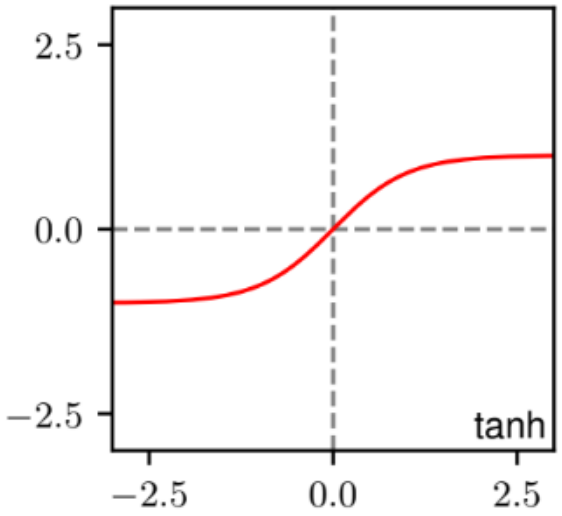
Universal Approximator



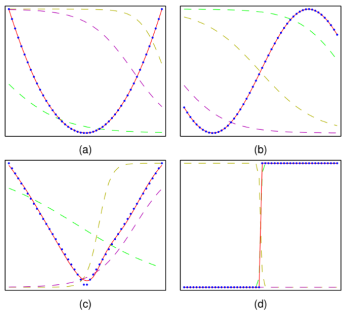
< An example of a simple 2-layer neural network >

A neural network is said to be **universal approximator** in that it has capability of approximating to model any target function over a continuous subset of R^D space.

This is possible by introducing **non-linearity** between layers that transforms layer's outputs as independent inputs to the next layer.



Activation Functions



< Approximation of four different observations: (1) x^2, (2) sin, (3) |x|, (4) Heaviside Step, using tanh activation function >

▼ Should be **Non-linear** to stack layers

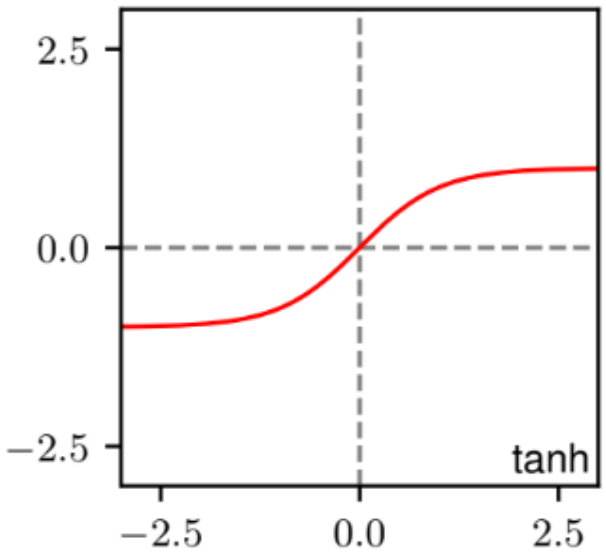
If there's no activation functions but only linear transformation, meaning that increment on input will result in increment on output since:

$$\begin{cases} f(a\mathbf{x}) = af(\mathbf{x}) \\ f(a\mathbf{x} + b\mathbf{y}) = af(\mathbf{x}) + bf(\mathbf{y}) \end{cases}$$

Accordingly, multi-layers with linear activation functions will eventually be one hidden layer: -

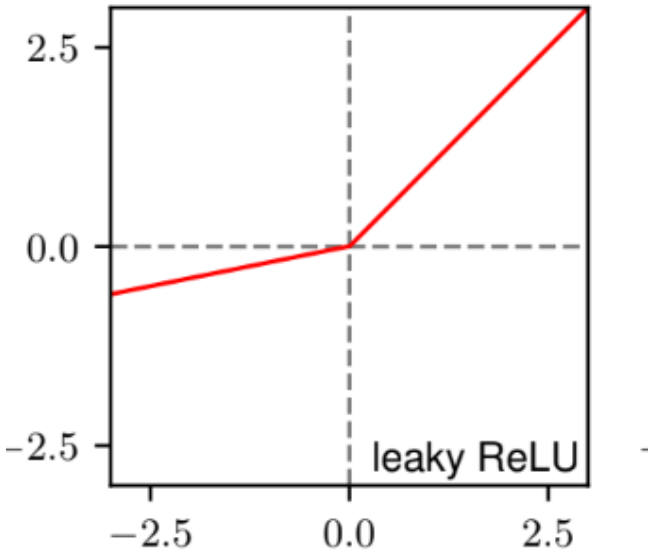
$$f_n(x) = W_n x$$
$$f_0(f_1(f_2(\cdots))) = W'x$$

▼ Sigmoid, tanh functions are more easily affected by **vanishing gradients** problem



For a large input, the gradient becomes almost zero → can't take 'error signals' to do gradient descent

▼ ReLU or Leaky ReLU is widely used today



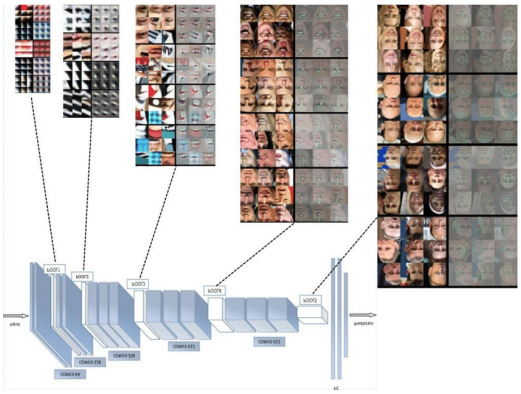
$$Leaky\ ReLU(a) = max(0, a) + \alpha min(0, a)$$

Deep Networks

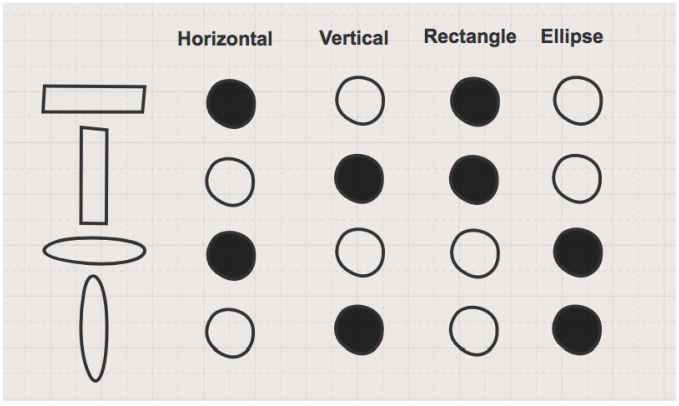
We can easily extend the two-layer network to any finite number L of layers, in which layer l=1, 2, … L computes the following function, where h denotes the activation function, W weights and z the input vector./

$$z^{(l)} = h^{(l)}(W^{(l)} z^{(l-1)})$$

Interpretation



< Hierarchical view of deep neural network >



< Distributed view of deep neural network >

The major advantage of stacking up layers more than two is its capability to learn and transform low-features effectively into more useful features.

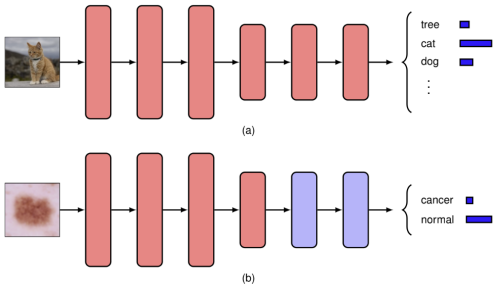
Hierarchical (Layers)

For a example of recognizing objects in image, say a cat, it's highly complex problem for a two-layer network that takes input as raw pixels. The non-linearity of activation functions transforms low-level features to high-level such as pixels to curves then parts. That said, layers can be interpreted as hierarchy of gradually evolving information.

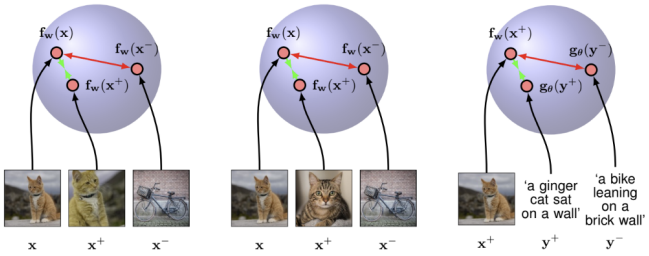
Distributed (Units)

With M units in a given layer, such a layer can represent M different features. However, the features can be further multi-folded by 2^M if we pass them to the next layer to select or not to select each feature. Deep Neural Networks have ability to discover for themselves the useful combinations among those distributed features.

Learning



< Transfer learning that freezes low level layers >



< Contrastive learning with error function based on distance between inputs >

Representation (Default)

> The ability to discover a non-linear transformation of the data that makes subsequent tasks easier to solve.
>
> *ex) pixel → line →curve → part → shape …*

- ***embedding space***: The outputs of one of the hidden layers that transforms any input vector to another space.
- ***auto-encoder***: unsupervised learning algorithm that learns representative features from an input and returns an output quite

similar to the input.

Transfer

> A way to exploit low-level layers of an existing network to better train a model for which training data is in short supply.

- ***pre-training*:** the process of learning parameters using a general task with abundant data set
- ***fine-tuning*:** the process of learning additional parameters on top of a pre-trained model to fit more specific task

Contrastive

> A way to lean a representation such that positive pairs are close to each other in embedding space, and negative inputs are far apart.

A training batch for a contrastive learning consists of three parts

- **anchor**: constant $\mathbf{x}$ that makes positive pairs with a $\mathbf{x}^+$
- **positive pair**: a pair of the anchor and an input that should be close

  $\{x_1^+, x_2^+, x_3^+ \cdots\}$
- **negative set**: a number of inputs that should be far from positive inputs

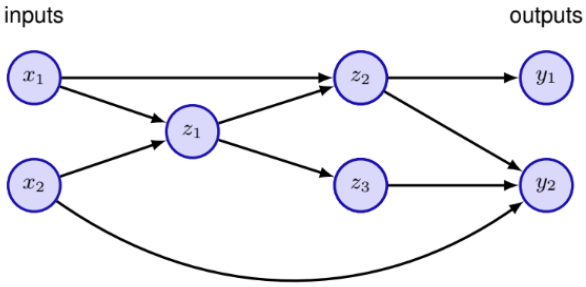
  $\{x_1^-, x_2^-, x_3^- \cdots\}$

Alternatively, a positive pair can be generated by applying corruption to an anchor, which examples include rotation, translation and color shifts. the most commonly used error function is given as follows, which is called ***InfoNCE*** loss.

$$E(w) = -\ln \frac{exp\{f_w(x)^T f_w(x^+)\}}{exp\{f_w(x)^T f_w(x^+)\} + \sum_n exp\{f_w(x)^T f(x_n^-)\}}$$

General Network Architecture

The general restriction to build a neural network is that it should be ***feed-forward***, in other words no closed directed cycles, to ensure outputs are deterministic functions of inputs.



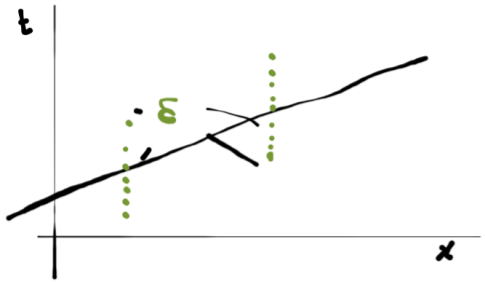
$$z_k = h(\sum_{j \in A(k)} w_{kj} z_j + b_k)$$

where z_k is output, $h()$ denotes activation function, $A(k)$ ancestors of unit k. ($A(y_2) = \{x_2, z_2, z_3\}$)

Error Functions

Error functions can take various forms by types of multi-layer neural networks.

Regression



< Assuming the observations follow Gaussian >

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

For regression, we can assume the output target t follows Gaussian with a deviation ϵ from a regression function $y(x, w)$, when x is given. Our goal is to find the optimal w, σ^2 from the maximum likelihood function:

$$p(t|X, w, \sigma^2) = \prod^{N} p(t_n|y(x_n, w), \sigma^2)$$

To follow the convention of error function, which should be minimized, we can take negative log from the likelihood function as below. We then find the optimal w^* and use it to find the optimal σ^{2*}.

$$E(w) = -\ln p(t|X, w, \sigma^2) = \frac{1}{2\sigma^2}\sum^{N}\{y(x_n, w) - t_n\}^2 + \frac{N}{2}\ln\sigma^2 + \frac{N}{2}\ln(2\pi)$$

$$E(w) \approx \frac{1}{2}\sum^{N}\{y(x_n, w) - t_n\}^2 \text{ for gradient-descent}$$

$$\nabla_{\sigma^2}E(w) = \frac{1}{N}\sum^{N}\{y(x_n, w^*) - t_n\}^2$$

Note that activation function of a regression is the identity so that $a_k = y_k$ and if we take partial derivative of the output function for back propagation:

$$\frac{\partial E(w)}{\partial a_k} = y_k - t_k$$

Binary Classification

In case of a binary classification where a single target t is such that $t = 1 \rightarrow C_1, t = 0 \rightarrow C_2$, whose activation function is a **logistic sigmoid** $y_k = \sigma(a_k)$. The conditional probability $p(C_1|x)$ and $p(C_2|x)$ is then a Bernoulli distribution of the form:

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

The error function for gradient descent is then given by the negative log likelihood as below. Using the derivative of the sigmoid function, the error function yields $\frac{\partial E(w)}{\partial a_k}$ as the same form as the one above.

$$E(w) = -\sum^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$$

$$\text{from } \frac{\partial y_k}{\partial a_k} = y_k(1 - y_k),$$

$$\frac{\partial E(w)}{\partial a_k} = \frac{\partial E(w)}{\partial y_k}\frac{\partial y_k}{\partial a_k} = \underline{y_k - t_k}$$