4.1. Multilayer Perceptron

Lecture based on "Dive into Deep Learning" **http://D2L.AI** (Zhang et al., 2020)

Prof. Dr. Christoph Lippert

Digital Health & Machine Learning

Recap

Linear regression and softmax regression map inputs directly to the outputs via a single linear transformation:

$$\hat{\mathbf{o}} = \mathrm{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

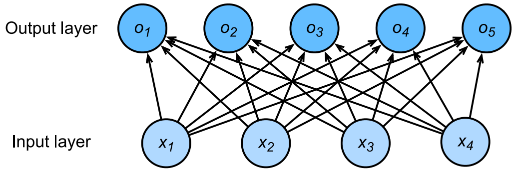


Figure: Single layer perceptron with 5 output units.

But: Here, we assume a linear relationship between input and output. Linearity is a *strong assumption*.

Linear models

- predict probability of repaying a loan.

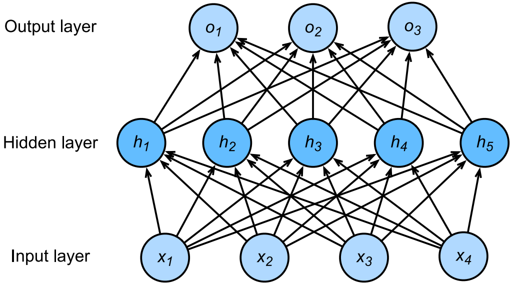
  $\rightarrow$ higher income would be more likely to repay
- ...

Linear models for image classification
Should increasing the intensity of a pixel always increase the likelihood that the image depicts a cat?

- Classify regardless of pixel brightness
- We need to account for interactions among pixels

Multilayer perceptron (MLP)

- Model more complex relationships between inputs and outputs by allowing interactions among the many features.
- Learn more complex classifications by incorporating one or more stacked **hidden layers**.
- Each layer feeds into the layer above it, until we generate an output.



- Multilayer perceptron with hidden layers.
- This example contains a hidden layer with 5 hidden units in it.

From linear to nonlinear

We can write out the calculations that define this one-hidden-layer MLP in mathematical notation as follows:

$$
\begin{aligned}
\mathbf{h} &= \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \\
\mathbf{o} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \\
\hat{\mathbf{y}} &= \mathrm{softmax}(\mathbf{o})
\end{aligned}
$$

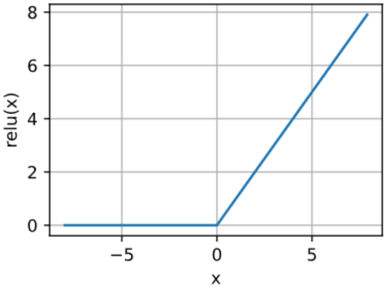
Is this enough?

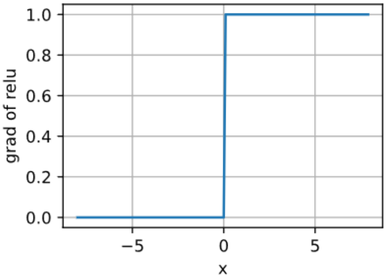
$$
\begin{aligned}
\mathbf{o} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \\
&= \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\
&= (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2) \\
&= \mathbf{W}\mathbf{x} + \mathbf{b}
\end{aligned}
$$

In order to get a benefit from multilayer architectures, we need to add a non-linearity σ to be applied to each of the hidden units after each layer's linear transformation.

$$
\begin{aligned}
\mathbf{h} &= \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\
\mathbf{o} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \\
\hat{\mathbf{y}} &= \mathrm{softmax}(\mathbf{o})
\end{aligned}
$$

Rectified Linear Unit (ReLU)

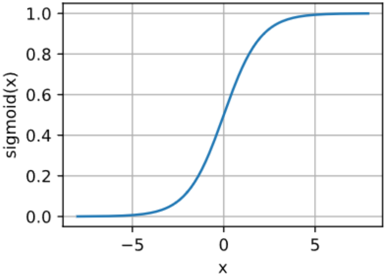


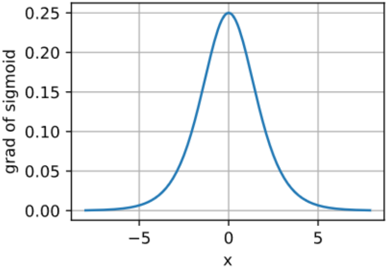


$$\mathrm{ReLU}(z) = \max(z, 0).$$

- Negative input $\rightarrow$ derivative is 0
- Positive input $\rightarrow$ derivative is 1
- input=0 is not defined, but we set the derivative to 0

Logistic Sigmoid function

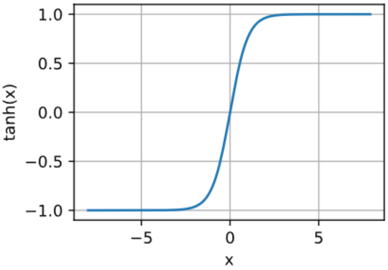


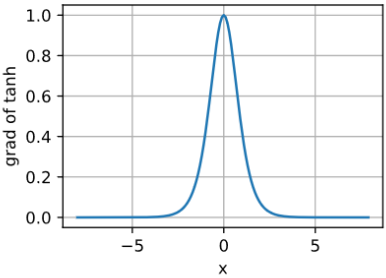


$$\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}.$$

$$\begin{aligned}\frac{d}{dx}\text{sigmoid}(x) &= \frac{\exp(-x)}{(1+\exp(-x))^2} \\ &= \text{sigmoid}(x)\left(1-\text{sigmoid}(x)\right)\end{aligned}$$

Tanh-function





$$\mathsf{tanh}(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x).$$

Deep Architectures

- Stacking such hidden layers, e.g. $\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$ and $\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$ on top of each other
- Account for interactions because the hidden neurons depend on the values of each of the inputs
- With a single-hidden-layer neural network, with enough nodes, and the right set of weights, we can model any function!
- *Learning that function is the hard part.*
- approximate many functions much more compactly if we use deeper (vs wider) neural networks

MLP with 2 hidden layers

The matrix $\mathbf{X}$ denotes a mini-batch of inputs.

The calculations to produce outputs from an MLP with two hidden layers can thus be expressed:

$$
\begin{aligned}
\mathbf{H}_1 &= \sigma(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1) \\
\mathbf{H}_2 &= \sigma(\mathbf{W}_2\mathbf{H}_1 + \mathbf{b}_2) \\
\mathbf{O} &= \mathrm{softmax}(\mathbf{W}_3\mathbf{H}_2 + \mathbf{b}_3)
\end{aligned}
$$

- we define the non-linearity σ to apply to its inputs in a row-wise fashion, i.e. one observation at a time
- *softmax* also denotes a row-wise operation
- the activation functions that we apply to hidden layers are not merely row-wise, but component wise
- after computing the linear portion of the layer, we can calculate each nodes activation without looking at the values taken by the other hidden units

Summary

- The multilayer perceptron
  - adds one or multiple **fully-connected hidden layers** between the output and input layers
  - transforms the output of the hidden layer via an activation function.
- Commonly-used activation functions include
  - the ReLU function
  - the sigmoid function
  - the tanh function