DATA 586: Advanced Machine Learning

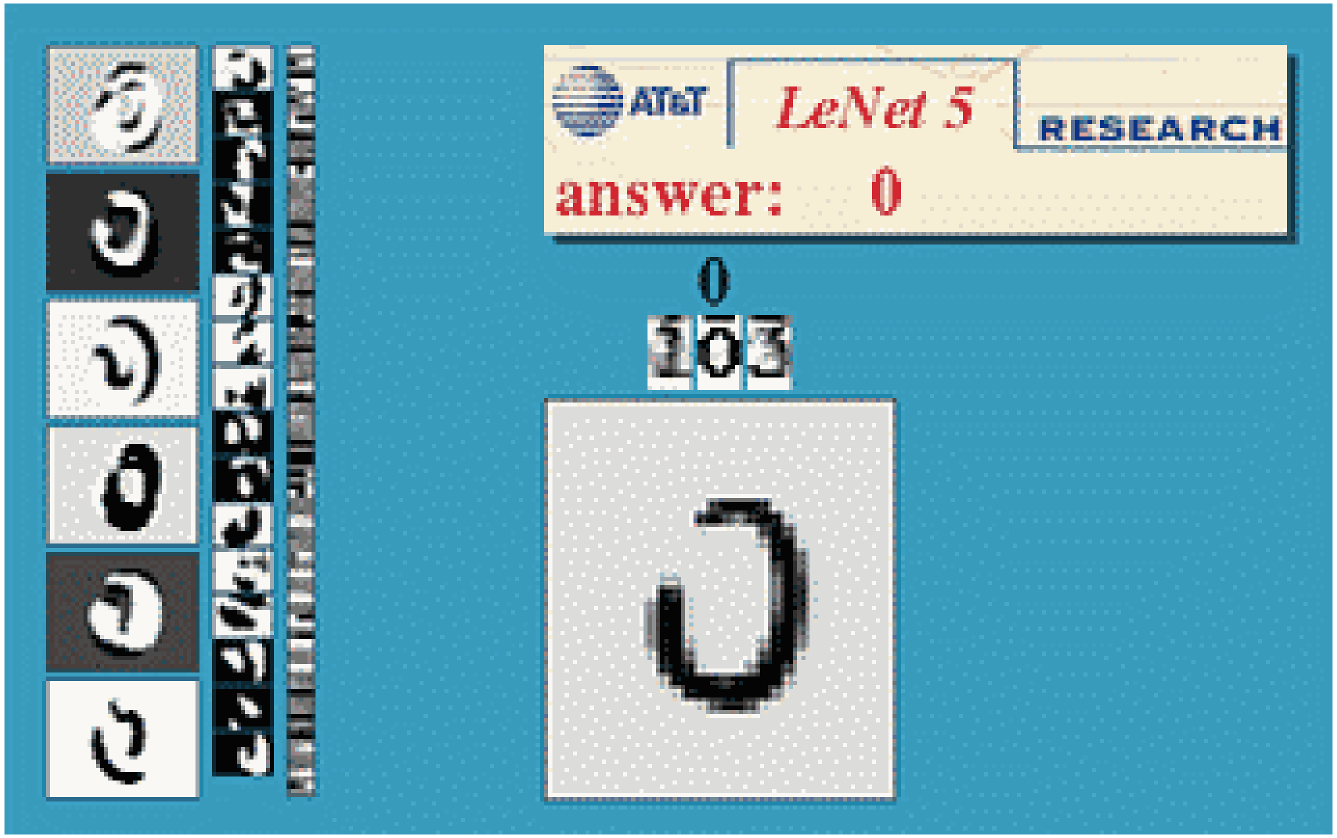
2023W2

Shan Du

Multilayer Neural Networks

- Modern neural networks typically have more than one hidden layer, and often many units per layer.
- In theory a single hidden layer with a large number of units has the ability to approximate most functions. However, the learning task of discovering a good solution is made much easier with multiple layers each of modest size.

Multilayer Neural Networks



Digit Recognition

https://en.wikipedia.org/wiki/Handwriting_recognition

Multilayer Neural Networks

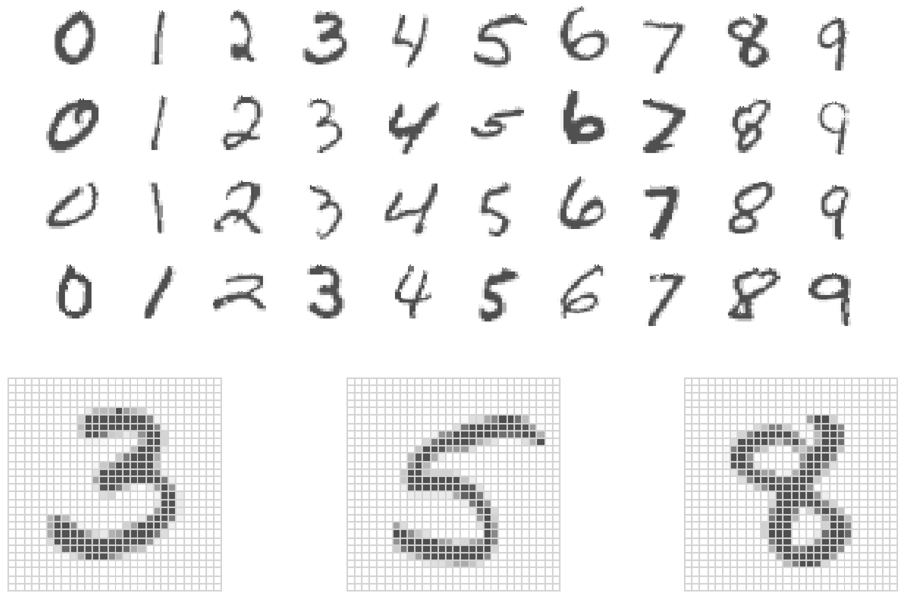


FIGURE 10.3. *Examples of handwritten digits from the* MNIST *corpus. Each grayscale image has* 28×28 *pixels, each of which is an eight-bit number (0–255) which represents how dark that pixel is. The first 3, 5, and 8 are enlarged to show their 784 individual pixel values.*

Multilayer Neural Networks

- The pixels are stored in the input vector X (in, say, column order). The output is the class label, represented by a vector $Y = (Y_0, Y_1, \ldots, Y_9)$ of 10 dummy variables, with a one in the position corresponding to the label, and zeros elsewhere.
- In the machine learning community, this is known as *one-hot encoding*. There are 60,000 training images, and 10,000 test images.

Multilayer Neural Networks

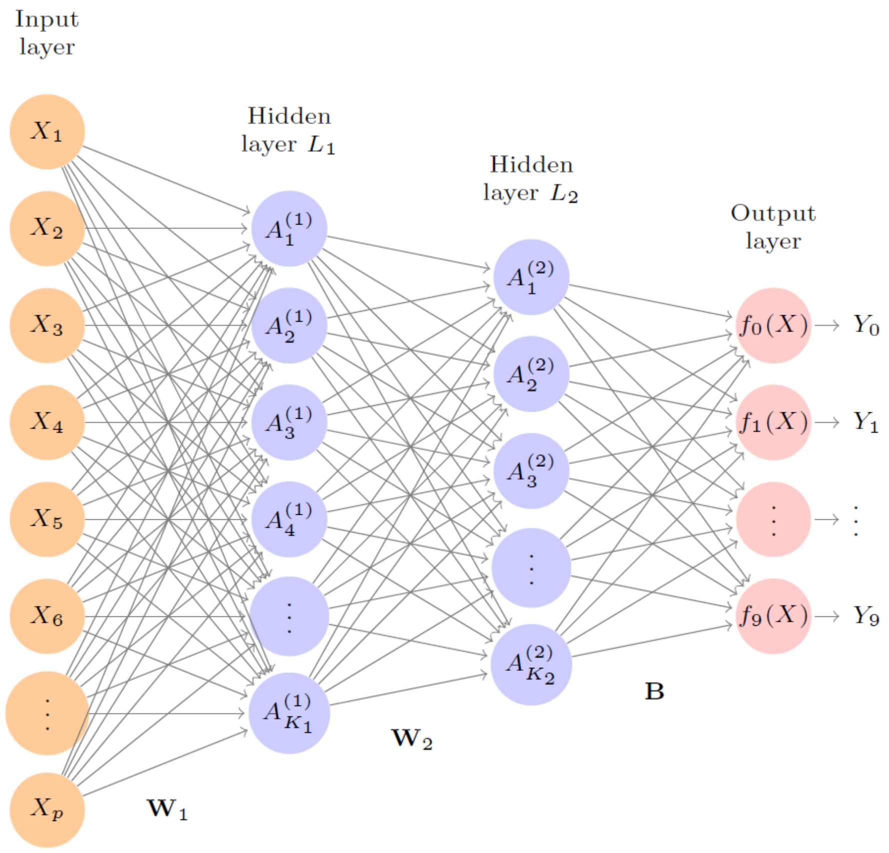


FIGURE 10.4. *Neural network diagram with two hidden layers and multiple outputs, suitable for the* MNIST *handwritten-digit problem. The input layer has $p = 784$ units, the two hidden layers $K_1 = 256$ and $K_2 = 128$ units respectively, and the output layer* 10 *units. Along with intercepts (referred to as* biases *in the deep-learning community) this network has 235,146 parameters (referred to as weights).*

Multilayer Neural Networks

- It differs from single layer neural networks in several ways:
  - It has two hidden layers L_1 (256 units) and L_2 (128 units) rather than one.
  - It has ten output variables, rather than one. In this case the ten variables really represent a single qualitative variable and so are quite dependent.
  - The loss function used for training the network is tailored for the *multiclass classification* task.

Multilayer Neural Networks

- The first hidden layer is similar to that in the single layer neural network with

$$A_k^{(1)} = h_k^{(1)}(X) = g(w_{k0}^{(1)} + \sum\nolimits_{j=1}^{p} w_{kj}^{(1)} X_j)$$

for $k = 1, \ldots, K_1$.

- The second hidden layer treats the activations $A_k^{(1)}$ of the first hidden layer as inputs and computes new activations

$$A_l^{(2)} = h_l^{(2)}(X) = g(w_{l0}^{(1)} + \sum\nolimits_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)})$$

for $l = 1, \ldots, K_2$.

Multilayer Neural Networks

- Notice that each of the activations in the second layer $A_l^{(2)} = h_l^{(2)}(X)$ is a function of the input vector X.
- This would also be the case with more hidden layers. Thus, through a chain of transformations, the network is able to build up fairly complex transformations of X that ultimately feed into the output layer as features.

Multilayer Neural Networks

- The notation $\mathbf{W}_1$ in Figure 10.4 represents the entire matrix of weights that feed from the input layer to the first hidden layer L_1. The dimension is $(784 + 1) \times 256 = 200{,}960$.
- Each element $A_k^{(1)}$ feeds to the second hidden layer L_2 via the matrix of weights $\mathbf{W}_2$. The dimension is $(256 + 1) \times 128 = 32{,}896$.

Multilayer Neural Networks

- We now get to the output layer, where we now have ten responses rather than one. The first step is to compute ten different linear models similar to our single layer model

$$
\begin{aligned}
Z_m &= \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} h_l^{(1)}(X) \\
&= \beta_{m0} + \sum_{l=1}^{K_2} \beta_{ml} A_l^{(2)}
\end{aligned}
$$

for $m = 0, 1, \dots, 9$. The matrix $\mathbf{B}$ stores $(128 + 1) \times 10 = 1{,}290$ weights.

Multilayer Neural Networks

- If these were all separate quantitative responses, we would simply set each $f_m(X) = Z_m$ and be done. However, we would like our estimates to represent class probabilities $f_m(X) = \Pr(Y = m|X)$, just like in multinomial logistic regression.
- So we use the special *softmax* activation function

$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{l=0}^{9} e^{Z_l}}$$

for $m = 0, 1, \ldots, 9$.

Multilayer Neural Networks

- This ensures that the 10 numbers behave like probabilities (non-negative and sum to one).
- Even though the goal is to build a classifier, our model actually estimates a probability for each of the 10 classes. The classifier then assigns the image to the class with the highest probability.

Multilayer Neural Networks

- To train this network, since the response is qualitative, we look for coefficient estimates that minimize the negative multinomial log-likelihood

$$-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log\left(f_m(x_i)\right),$$

also known as the *cross-entropy*.

Multilayer Neural Networks

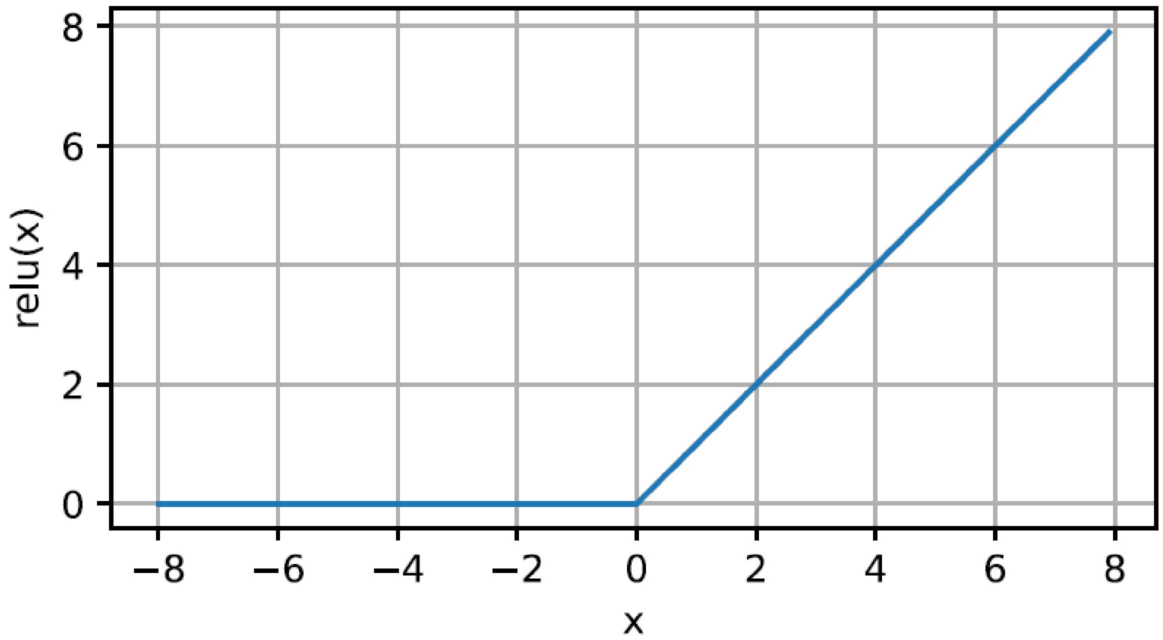
Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

TABLE 10.1. *Test error rate on the* MNIST *data, for neural networks with two forms of regularization, as well as multinomial logistic regression and linear discriminant analysis. In this example, the extra complexity of the neural network leads to a marked improvement in test error.*

More About Activation Functions

- Activation functions decide whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it. They are differentiable operators to transform input signals to outputs, while most of them add non-linearity.

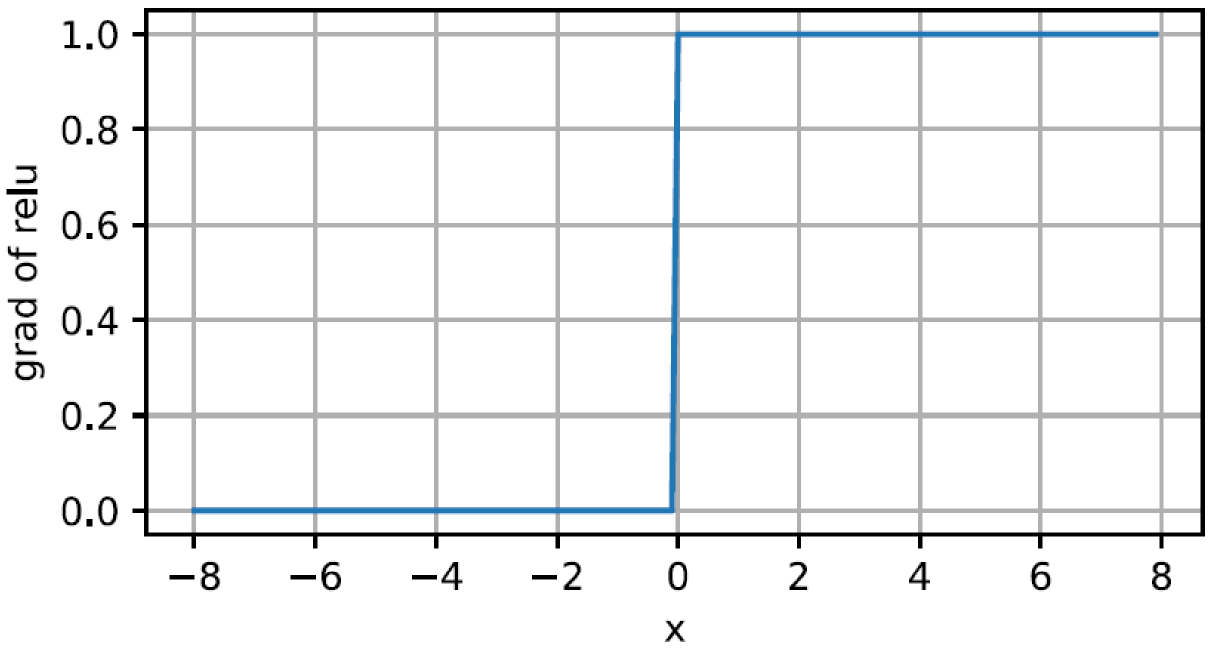
More About Activation Functions



$$ReLU(x) = \max(x;\ 0)$$

- Note that the ReLU function is not differentiable when the input takes value precisely equal to 0. In these cases, we default to the left-hand-side derivative and say that the derivative is 0 when the input is 0.

More About Activation Functions



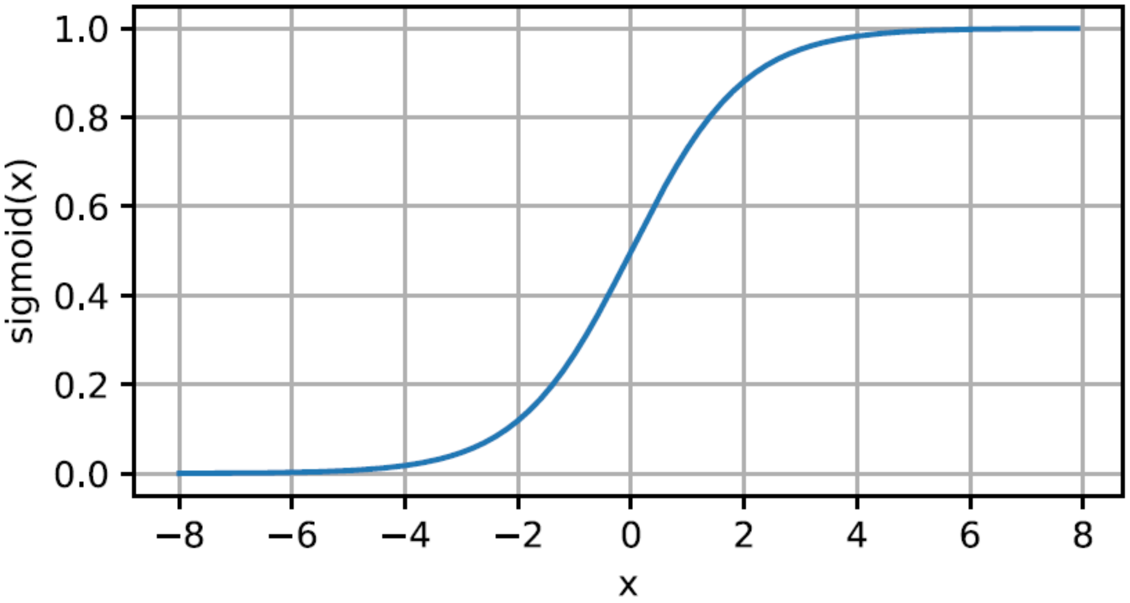
- The reason for using ReLU is that its derivatives are particularly well behaved: either they vanish or they just let the argument through. This makes optimization better behaved and it mitigated the well-documented problem of vanishing gradients that plagued previous versions of neural networks (more on this later).

More About Activation Functions

- There are many variants to the ReLU function, including the parameterized ReLU (pReLU) function. This variation adds a linear term to ReLU, so some information still gets through, even when the argument is negative:

$$pReLU(x) = \max(0;\ x) + \min(0;\ x)$$

More About Activation Functions

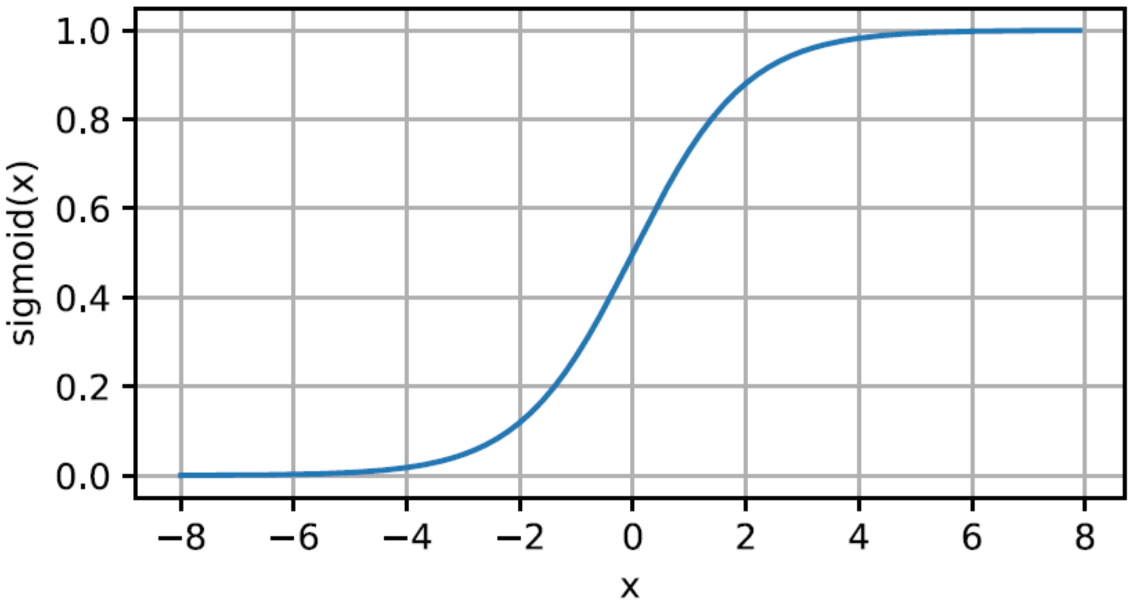


$$sigmoid(x) = \frac{1}{1 + \exp(x)}$$

the squashing function

- In the earliest neural networks, scientists were interested in modeling biological neurons which either *fire* or *do not fire*. Thus, the pioneers of this field focused on thresholding units. A thresholding activation takes value 0 when its input is below some threshold and value 1 when the input exceeds the threshold.

More About Activation Functions

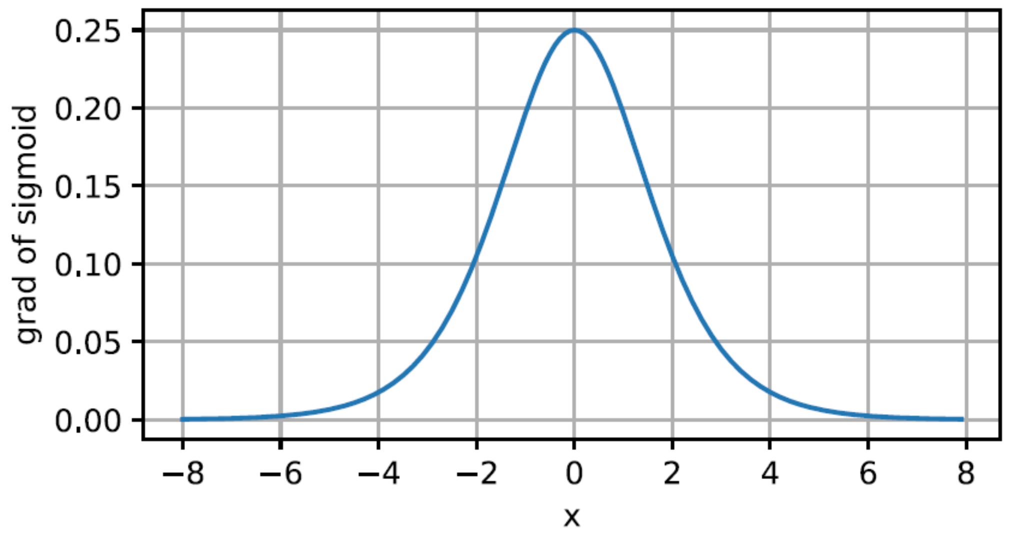


$$sigmoid(x) = \frac{1}{1 + \exp(x)}$$

the squashing function

- When attention shifted to gradient based learning, the sigmoid function was a natural choice because it is a smooth, differentiable approximation to a thresholding unit. Sigmoids are still widely used as activation functions on the output units, when we want to interpret the outputs as probabilities for binary classification problems: you can think of the sigmoid as a special case of the softmax.

More About Activation Functions



The derivative of the sigmoid function is:

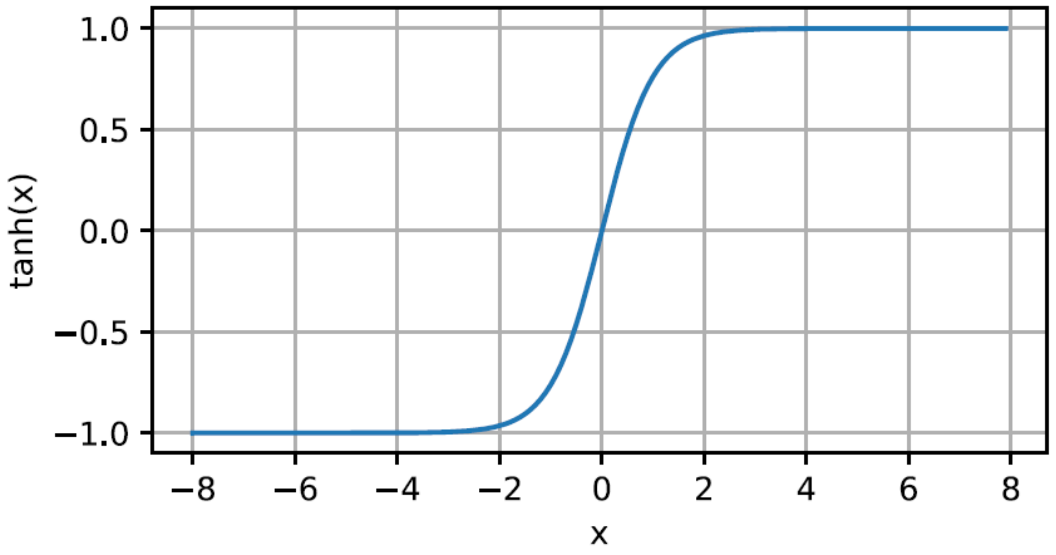
$$\frac{d}{dx}\operatorname{sigmoid}(x) = \frac{\exp(-x)}{(1+\exp(-x))^2} = \operatorname{sigmoid}(x)\left(1-\operatorname{sigmoid}(x)\right)$$

- However, the sigmoid has mostly been replaced by the simpler and more easily trainable ReLU for most use in hidden layers. Much of this has to do with the fact that the sigmoid poses challenges for optimization since its gradient vanishes for large positive and negative arguments.

More About Activation Functions

- Tanh Function
  - Like the sigmoid function, the tanh (hyperbolic tangent) function also squashes its inputs, transforming them into elements on the interval between -1 and 1.

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



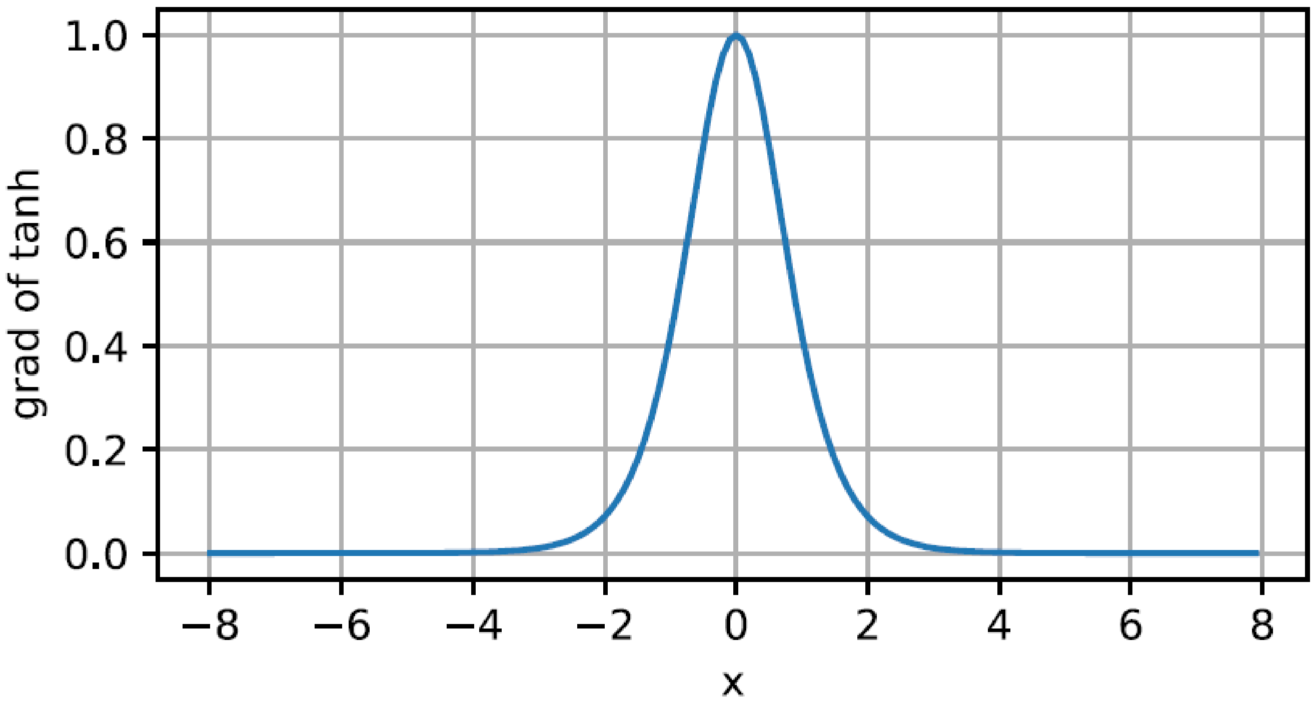
More About Activation Functions

- Note that as input nears 0, the tanh function approaches a linear transformation. Although the shape of the function is similar to that of the sigmoid function, the tanh function exhibits point symmetry about the origin of the coordinate system.

More About Activation Functions

The derivative of the tanh function is:

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


grad of tanh
x