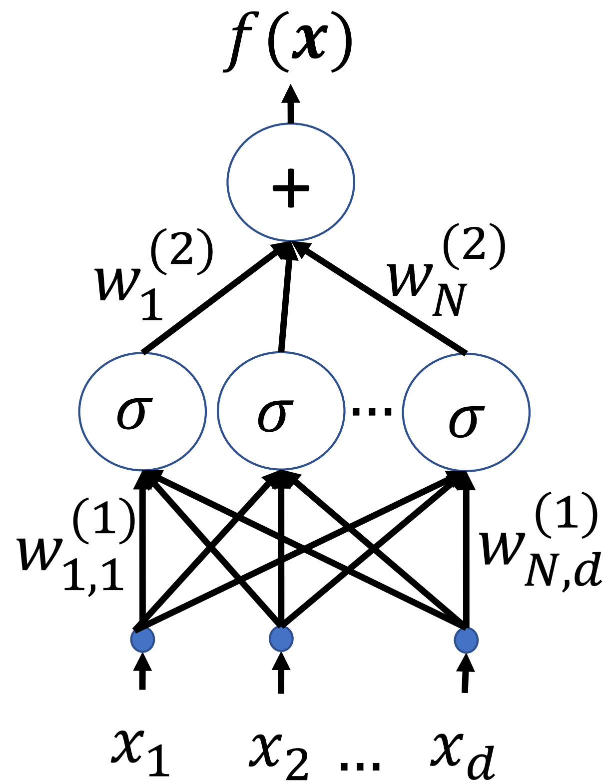
Lecture 12: Nonlinear Regression

Mark Hasegawa-Johnson



Outline

- From linear to nonlinear regression
- Rectified linear units (ReLU)
- Training a two-layer network: Back-propagation

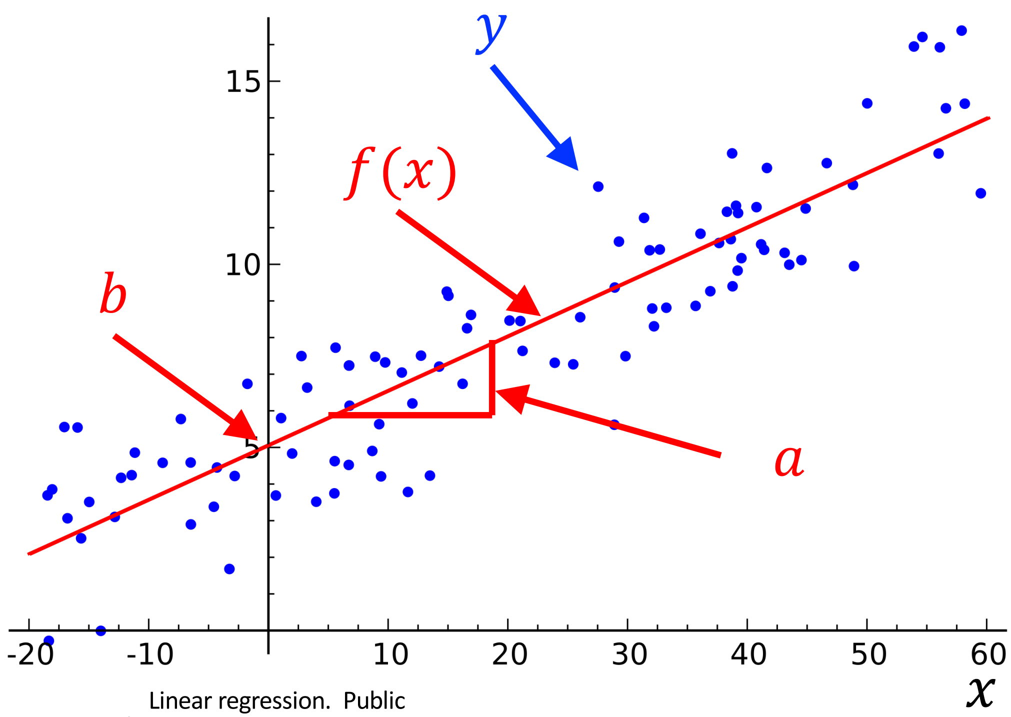
Linear regression

Linear regression is used to estimate a real-valued target variable, y, using a linear function of another variable, x:

$$f(x) = ax + b$$

… or of a vector, $\boldsymbol{x}$, …

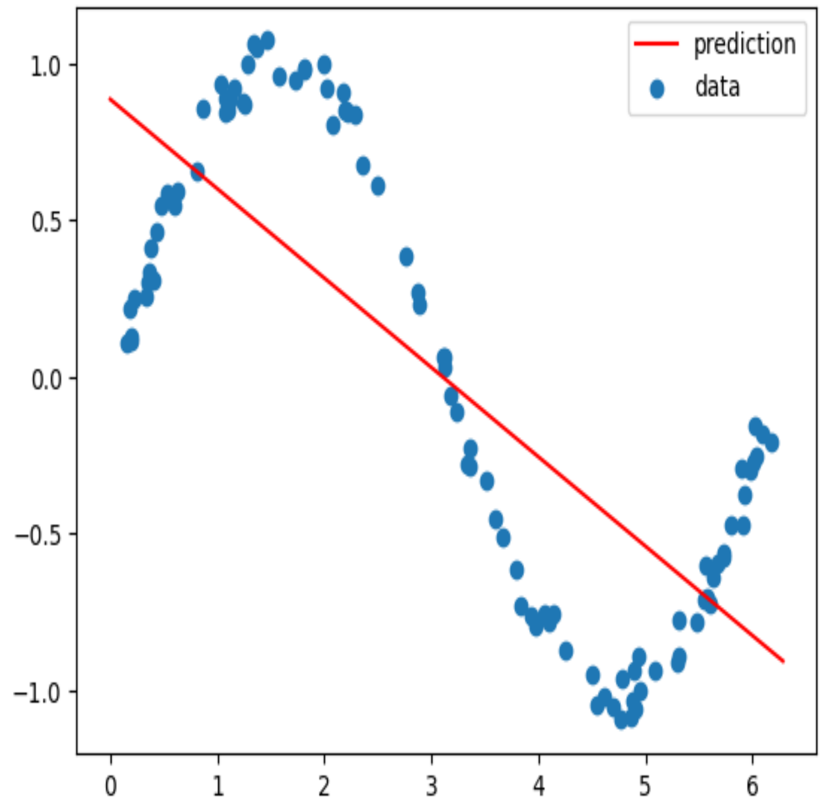
$$f(x) = \boldsymbol{w}^T \boldsymbol{x}$$



Linear regression. Public domain image, Seewaqu, 2010

Linear regression can’t fit nonlinear data without nonlinear features

Here’s an example from. Linear regression can’t fit nonlinear data unless it has nonlinear features.



Today: Two-layer neural nets

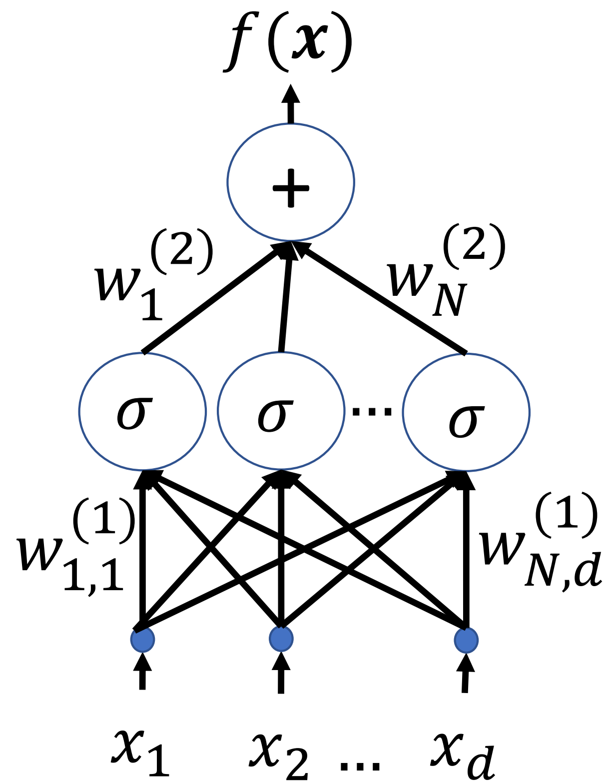
Today we want to consider a general method for nonlinear regression, called a "two-layer neural net," given by:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{(2),T}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)$$

…where the first and second-layer weights and biases are

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & \cdots & w_{1,d}^{(1)} \\ \vdots & \ddots & \vdots \\ w_{N,1}^{(1)} & \cdots & w_{N,d}^{(1)} \end{bmatrix}, \qquad \boldsymbol{w}^{(2)} = \begin{bmatrix} w_1^{(2)} \\ \vdots \\ w_N^{(2)} \end{bmatrix}$$

… and $\sigma(\boldsymbol{z})$ applies the logistic sigmoid to every element of the vector $\boldsymbol{z}$.



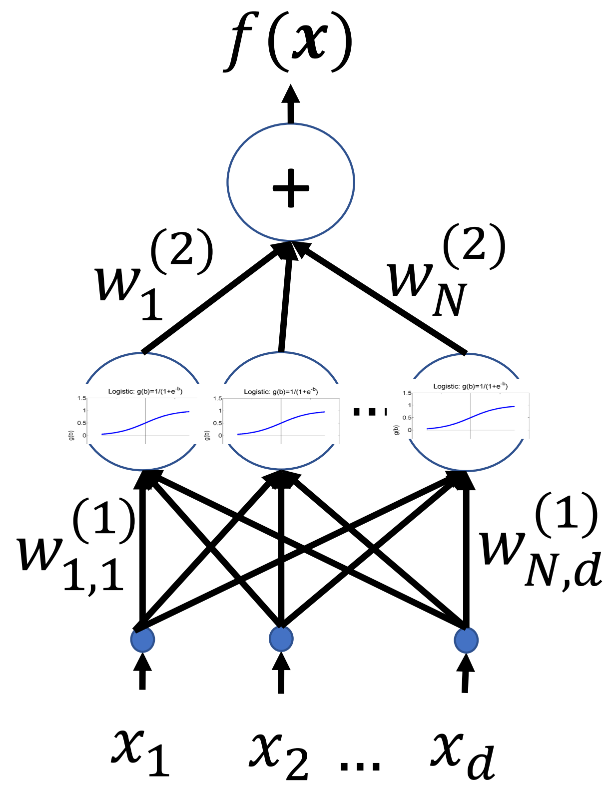
Why does it work?

Suppose we write the first-layer weight matrix as a stack of row vectors, like this: $\boldsymbol{W}^{(1)} = \left[\boldsymbol{w}_1^{(1)} \quad \cdots \quad \boldsymbol{w}_N^{(1)}\right]^T$. Then the hidden node vector is

$$\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right) = \begin{bmatrix} \sigma\left(\boldsymbol{w}_1^{(1),T}\boldsymbol{x}\right) \\ \vdots \\ \sigma\left(\boldsymbol{w}_N^{(1),T}\boldsymbol{x}\right) \end{bmatrix}$$

Each of its elements is

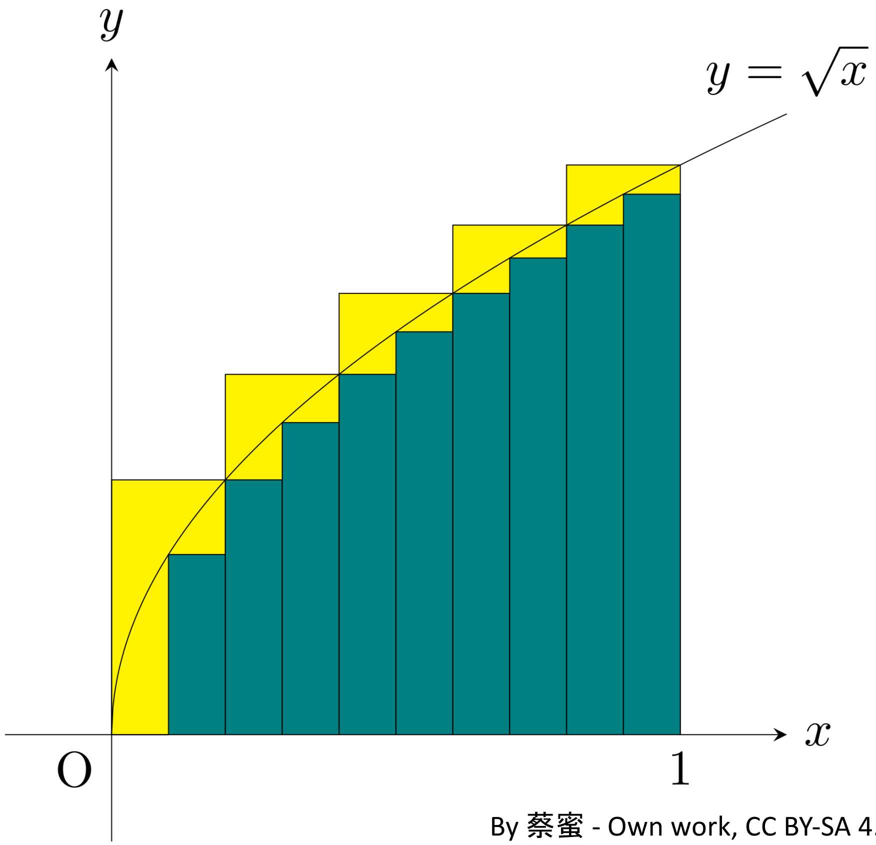
$$\sigma\left(\boldsymbol{w}_j^{(1),T}\boldsymbol{x}\right) \approx \begin{cases} 0 & \boldsymbol{w}_j^{(1),T}\boldsymbol{x} < 0 \\ 1 & \boldsymbol{w}_j^{(1),T}\boldsymbol{x} > 0 \end{cases}$$



Why does it work?

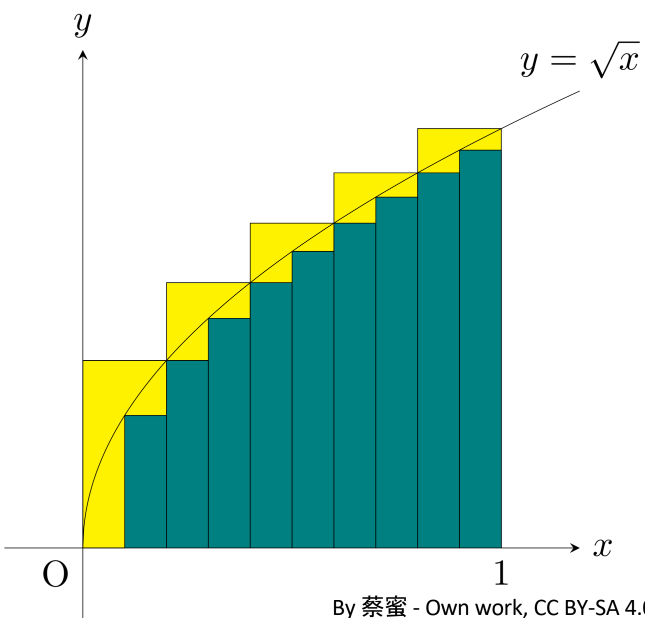
$$f(\boldsymbol{x}) = \boldsymbol{w}^{(2),T}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right) \approx \sum_{j:\boldsymbol{w}_j^{(1),T}\boldsymbol{x}>0} w_j^{(2)}$$

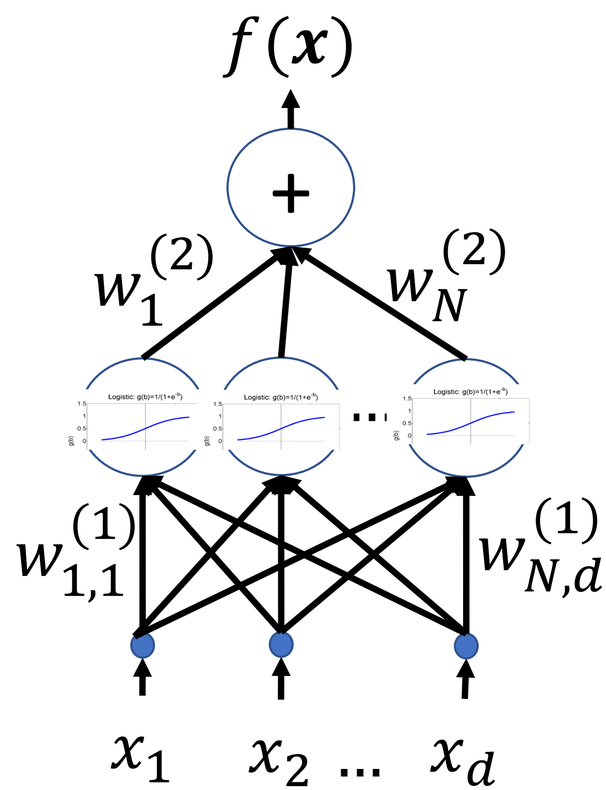
… but here's the cool thing. A long time ago, Isaac Newton proved that any nonlinear function can be approximated by a piece-wise constant function arbitrarily well, if you have enough pieces.



The universal approximation theorem of two-layer neural networks

… and therefore, any nonlinear function can be approximated by a two-layer neural network arbitrarily well, if you have enough hidden nodes.



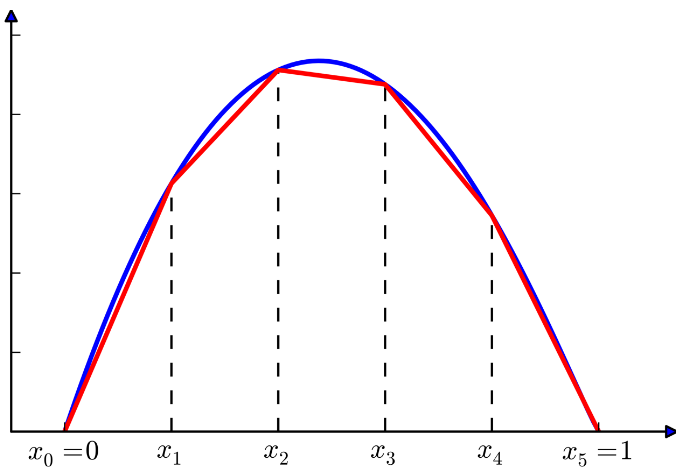


Outline

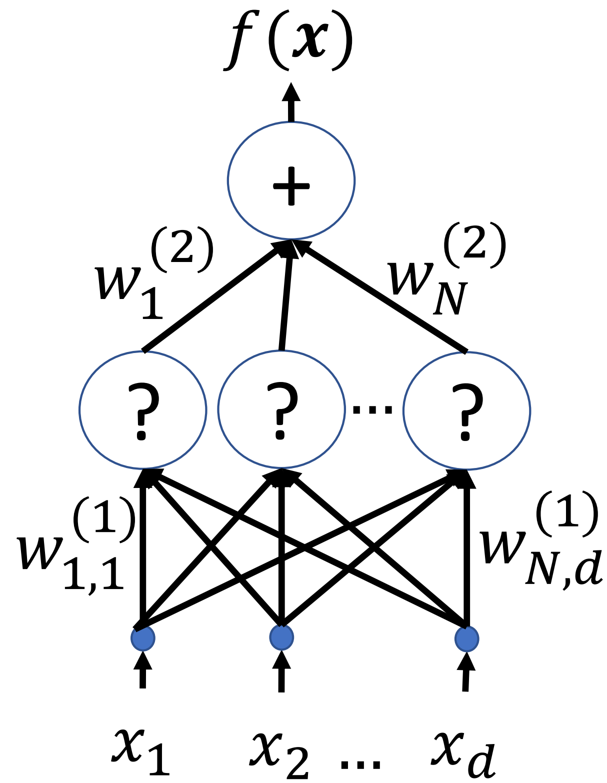
- From linear to nonlinear regression
- Rectified linear units (ReLU)
- Training a two-layer network: Back-propagation

How about piece-wise linear approximations?

What if we want a piece-wise linear output function, instead of piece-wise constant?



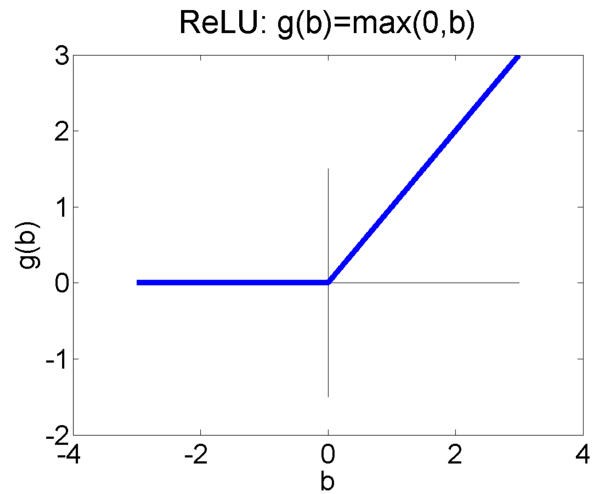
Public domain image, Krishnavedala, 2011



For a PWL neural net, the hidden nodes are ReLU

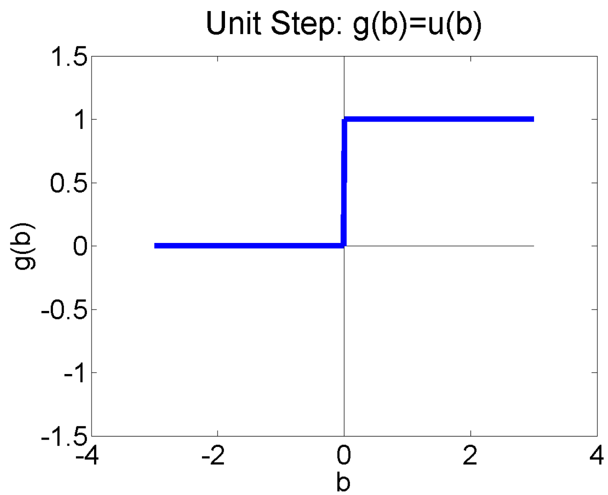
If the goal is PWL classification boundaries, we can achieve that by using hidden nodes that are the simplest possible PWL function: a Rectified Linear Unit, or ReLU:

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



This is differentiable everywhere except z=0; its derivative is the unit step function:

$$\frac{\partial \mathrm{ReLU}(z)}{\partial z} = u(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$



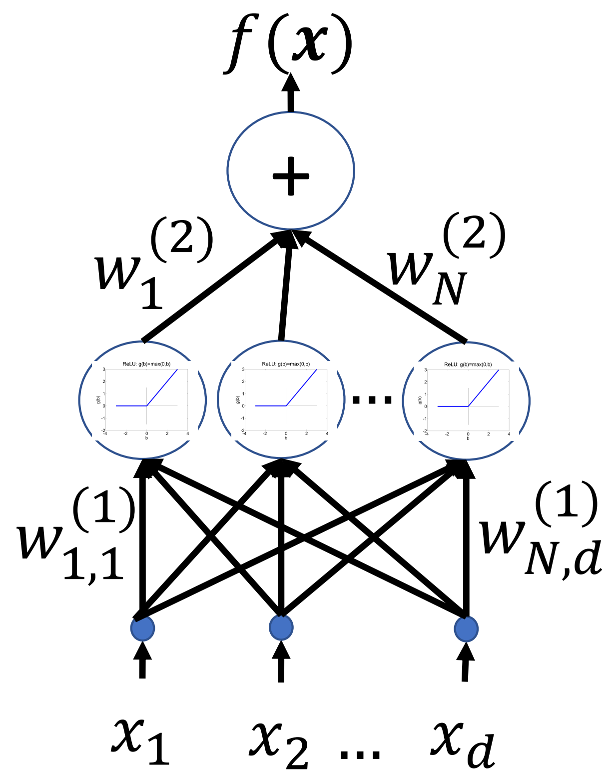
2-layer NN with ReLU hidden nodes

With ReLU hidden nodes, the hidden node vector is

$$\mathrm{ReLU}\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right) = \begin{bmatrix} \mathrm{ReLU}\left(\boldsymbol{w}_1^{(1),T}\boldsymbol{x}\right) \\ \vdots \\ \mathrm{ReLU}\left(\boldsymbol{w}_N^{(1),T}\boldsymbol{x}\right) \end{bmatrix}$$

Each of its elements is

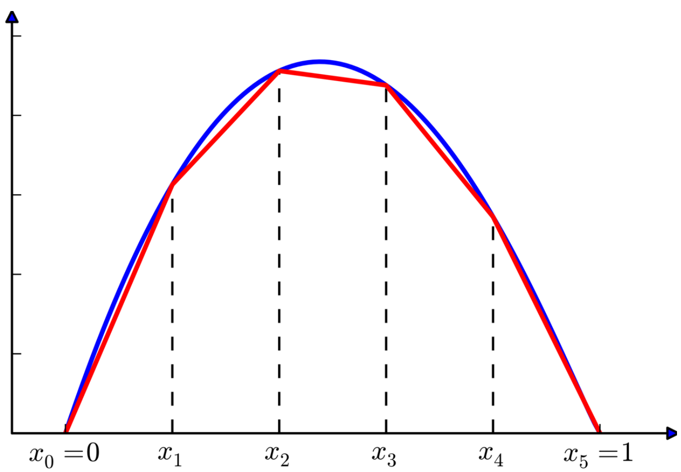
$$\mathrm{ReLU}\left(\boldsymbol{w}_j^{(1),T}\boldsymbol{x}\right) = \begin{cases} 0 & \text{if } \boldsymbol{w}_j^{(1),T}\boldsymbol{x} \le 0 \\ \boldsymbol{w}_j^{(1),T}\boldsymbol{x} & \text{if } \boldsymbol{w}_j^{(1),T}\boldsymbol{x} \ge 0 \end{cases}$$



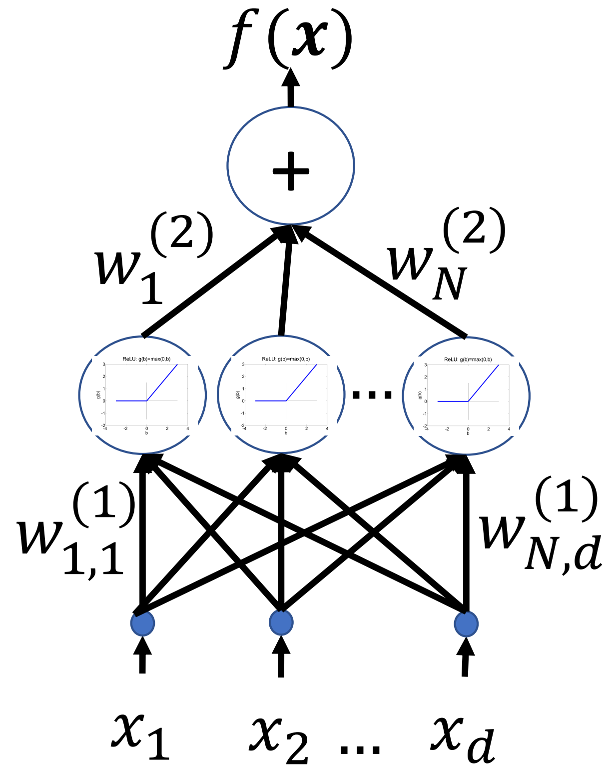
2-layer NN with ReLU hidden nodes

With ReLU hidden nodes, $f(\boldsymbol{x})$ is a perfectly piece-wise linear function of $\boldsymbol{x}$:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{(2),T}\mathrm{ReLU}\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right) = \sum_{j:\boldsymbol{w}_j^{(1),T}\boldsymbol{x}\geq 0} w_j^{(2)}\boldsymbol{w}_j^{(1),T}\boldsymbol{x}$$



Public domain image, Krishnavedala, 2011



Outline

- From linear to nonlinear regression
- Rectified linear units (ReLU)
- Training a two-layer network: Back-propagation

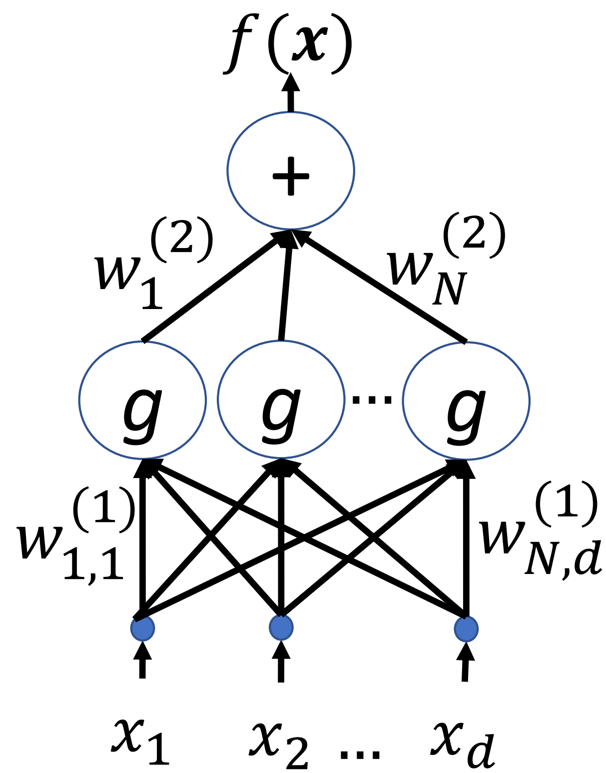
Training a neural net: mean-squared error

Now suppose we have a two-layer NN, with some hidden node nonlinearity $g(\boldsymbol{z})$ that might be either ReLU or sigmoid:

$$f(\boldsymbol{x}_i) = \boldsymbol{w}^{(2),T} g\left(\boldsymbol{W}^{(1)} \boldsymbol{x}_i\right)$$

Suppose we want to train $\boldsymbol{w}^{(2)}$ and $\boldsymbol{W}^{(1)}$ to minimize MSE:

$$\mathcal{L} = \frac{1}{2n} \sum_{i=1}^{n} (f(\boldsymbol{x}_i) - y_i)^2$$



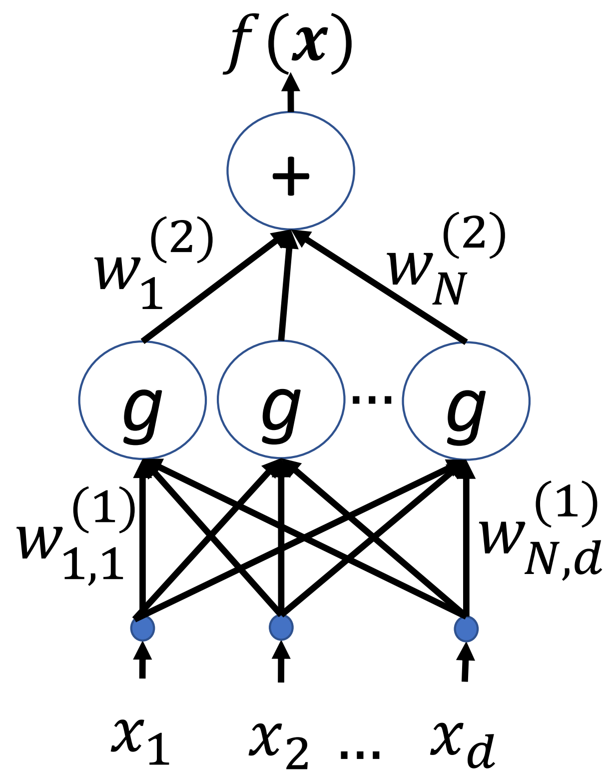
Training a neural net: gradient descent

We can do this using gradient descent:

$$\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(1)}}$$

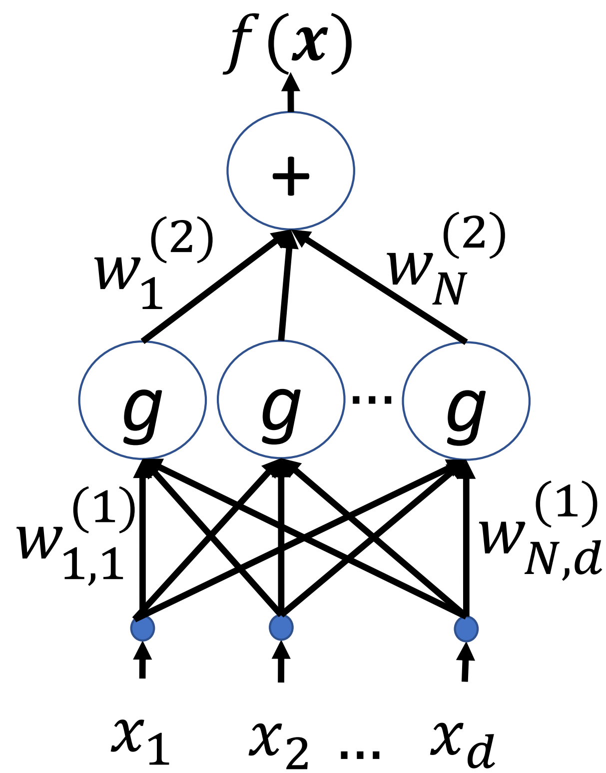
$$\boldsymbol{w}^{(2)} \leftarrow \boldsymbol{w}^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^{(2)}}$$

How do we find those derivatives?



Training a neural net: back-propagation

We can find the derivatives using the chain rule of calculus. Since the chain rule propagates backward through the network, this is called "back-propagation."



Training a neural net: back-propagation

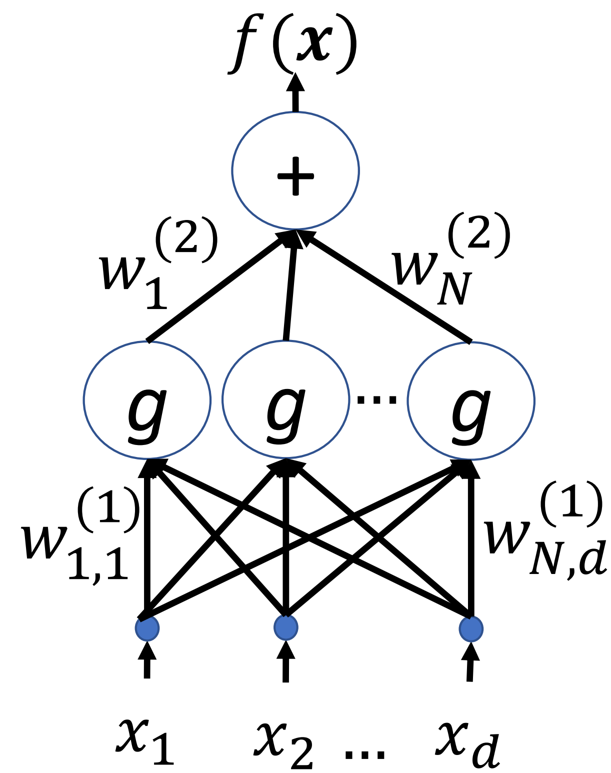
For example:

$$\mathcal{L} = \frac{1}{2n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2$$

$$\frac{\partial \mathcal{L}}{\partial f(\boldsymbol{x}_i)} = \frac{1}{n}(f(\boldsymbol{x}_i) - y_i)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^{(2)}} = \sum_{i=1}^{n}\frac{\partial \mathcal{L}}{\partial f(\boldsymbol{x}_i)}\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{w}^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(1)}} = \sum_{i=1}^{n}\frac{\partial \mathcal{L}}{\partial f(\boldsymbol{x}_i)}\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{W}^{(1)}}$$

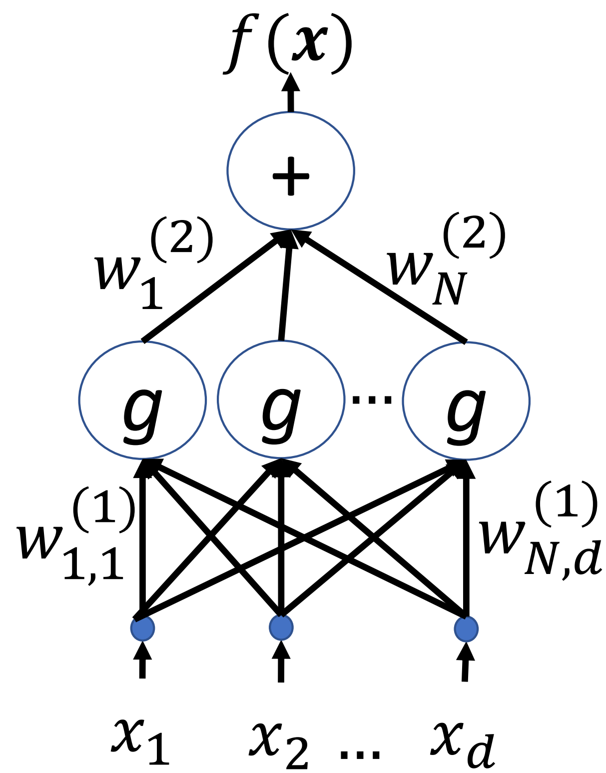


Back-propagation

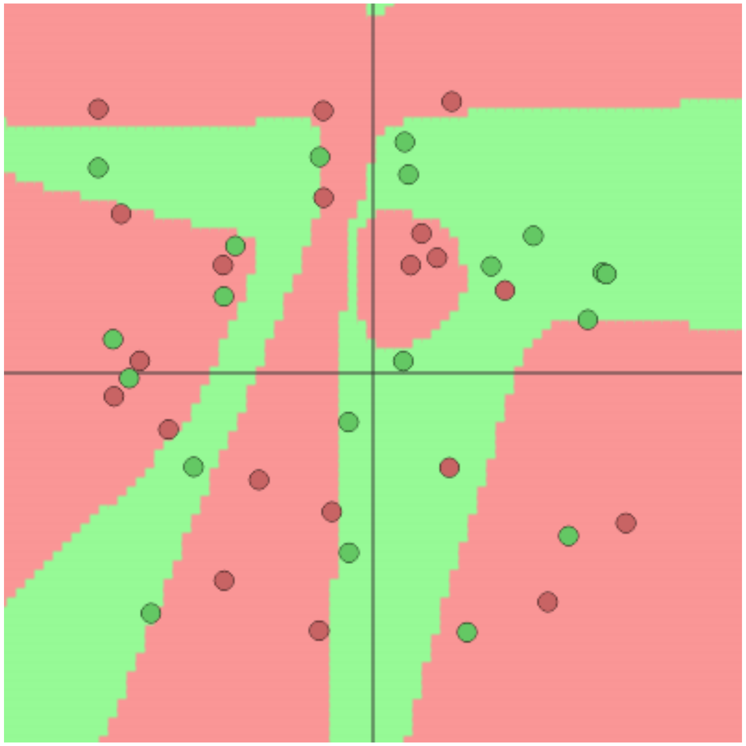
The derivatives are not always easy to find in matrix form. But since both ReLU and sigmoid are differentiable, we can always find the derivatives in scalar form!

$$f(\boldsymbol{x}_i) = \boldsymbol{w}^{(2),T} g\left(\boldsymbol{W}^{(1)} \boldsymbol{x}_i\right)$$

$$\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{W}^{(1)}} = \begin{bmatrix} \frac{\partial f(\boldsymbol{x}_i)}{\partial w_{1,1}^{(1)}} & \cdots & \frac{\partial f(\boldsymbol{x}_i)}{\partial w_{1,d}^{(1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\boldsymbol{x}_i)}{\partial w_{N,1}^{(1)}} & \cdots & \frac{\partial f(\boldsymbol{x}_i)}{\partial w_{N,d}^{(1)}} \end{bmatrix}$$



Approximating an arbitrary nonlinear boundary using a two-layer network



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Stochastic gradient descent

You can also reduce computation per step by using SGD:

- From a very large training dataset, randomly choose a training token $(\boldsymbol{x}_i, y_i)$. Forward-propagate to find the neural net prediction, $f(\boldsymbol{x}_i)$, and the loss

$$\mathcal{L}_i = \frac{1}{2n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2$$

- Back-propagate to find the gradients, then do gradient descent:

$$\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} - \eta\frac{\partial\mathcal{L}_i}{\partial\boldsymbol{W}^{(1)}}$$

$$\boldsymbol{w}^{(2)} \leftarrow \boldsymbol{w}^{(2)} - \eta\frac{\partial\mathcal{L}_i}{\partial\boldsymbol{w}^{(2)}}$$

- Repeat!

Try the quiz!

Go to PrairieLearn, try the quiz!

Summary

- Piece-wise constant nonlinear regression:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{(\mathbf{2}),\boldsymbol{T}} \sigma\left(\boldsymbol{W}^{(\mathbf{1})} \boldsymbol{x}\right)$$

- Piece-wise linear regression:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{(\mathbf{2}),\boldsymbol{T}} \mathrm{ReLU}\left(\boldsymbol{W}^{(\mathbf{1})} \boldsymbol{x}\right)$$

- Back-propagation:

$$\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} - \eta \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{W}^{(1)}}$$

$$\boldsymbol{w}^{(2)} \leftarrow \boldsymbol{w}^{(2)} - \eta \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{w}^{(2)}}$$