Neural Networks are Universal Approximators

Joshua Benjamin III

February 23, 2024

The Concept

- The idea of defining hypothesis classes for function approximation predates neural networks.
- Historical context provided by Weierstrass's theorem.
- How is this carried out in practice when implementing and training a neural network?

Weierstrass's Theorem

Theorem (Weierstrass, 1865.) Let $g : [0,1] \to \mathbb{R}$ be any continuous function. Then, g can be ε-approximated in the sup-norm by some polynomial of sufficiently high degree.

- Weierstrass's approach involved convolving with a Gaussian and using Taylor series.
- Bernstein provided a more direct constructive proof.
- The proof is based on constructing a set of interpolating basis functions.
- Bernstein's polynomials densely span the space of continuous functions.

Taylor's Formula and Theorem with Remainder

Taylor's formula for a function $f(x)$ that is infinitely differentiable at a point a is given by:

$$f(x) = f(a)+f'(a)(x-a)+\frac{f''(a)}{2!}(x-a)^2+\ldots+\frac{f^{(n)}(a)}{n!}(x-a)^n+R_n(x)$$

where $R_n(x)$ is the remainder (or error) term after n terms, which can be expressed in the Lagrange form as:

$$R_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x-a)^{n+1}$$

for some value c between a and x.
Taylor's theorem provides a precise bound on how good the approximation is by giving the remainder term $R_n(x)$.

Universal Approximation

Definition (Universal Approximator)

Let $\mathcal{F}$ be a given hypothesis class. Then, $\mathcal{F}$ is a universal approximator over some domain S if for every continuous function $g : S \to \mathbb{R}$ and approximation parameter $\varepsilon > 0$, there exists $f \in \mathcal{F}$ such that:

$$\sup_{x \in S} |f(x) - g(x)| \leq \varepsilon.$$

The Weierstrass theorem showed that the set of all polynomials is a universal approximator.

Stone-Weierstrass Theorem

A generalization of this theorem shows that other families of functions that behave like polynomials are also universal approximators. This is called the Stone-Weierstrass theorem, stated as follows.

Theorem (Stone-Weierstrass, 1948)

If the following hold:

1. *Every $f \in \mathcal{F}$ is continuous.*
2. *For every x, there exists $f \in \mathcal{F}$ such that $f(x) \neq 0$.*
3. *For every x, x', $x \neq x'$, there exists $f \in \mathcal{F}$ such that $f(x) \neq f(x')$.*
4. *$\mathcal{F}$ is closed under additions and multiplications.*

then $\mathcal{F}$ is a universal approximator.

Neural Network Architecture

- A neural network consists of layers of neurons connected by weights.
- Neurons within a layer are not connected.
- Each neuron in a layer receives input from all neurons in the previous layer.

Neural Network Formula

Given a layer l, the output $a^{(l)}$ of the neurons in this layer can be calculated as:

$$a^{(l)} = \sigma(W^{(l)}a^{(l-1)} + b^{(l)})$$

Where:

- $a^{(l-1)}$ is the output from the previous layer.
- $W^{(l)}$ is the weight matrix for layer l.
- $b^{(l)}$ is the bias vector for layer l.
- σ is the activation function (e.g., sigmoid, ReLU).

Neural Networks as Universal Approximators

We will use this property to show that in very general situations, several families of neural networks are universal approximators. To be precise, let $f(x)$ be a single neuron:

$$f_{c,w,b} : x \mapsto c\sigma(\langle w, x\rangle + b)$$

and define

$$\mathcal{F} = \text{span}\{f_{c,w,b}\}$$

as the space of all possible single-hidden-layer networks with activation σ.

Theorem (Cosine Activation)

If we use the cosine activation $\sigma(\cdot) = \cos(\cdot)$, then $\mathcal{F}$ is a universal approximator.

Cosine case proof

This result is the original "universal approximation theorem" and can be attributed to Hornik, Stinchcombe, and White. Other similar results are due to Cybenko and Funahashi but using techniques from functional analysis rather than Stone-Weierstrass. These all were published in 1989.

Completing the Proof

All we need to do is to show that the space of (possibly unbounded width) single-hidden-layer networks satisfies the four conditions of Stone-Weierstrass.

- **Continuity**: Composition of continuous function.
- **Identity**: For every x, $\cos(\langle 0, x\rangle) = \cos(0) = 1 \neq 0$.
- **Separation**: For every $x \neq x'$, $f(z) = \cos\left(\frac{1}{\|x-x'\|_2^2}\langle z - x', z - x\rangle\right)$ separates x, x'.
- **Closure**: Closure under additions is trivial by adding more hidden units. Closure under multiplications can be seen from the trig identity $\cos(\langle u, x\rangle)\cos(\langle v, x\rangle) = \frac{1}{2}(\cos(\langle u + v, x\rangle) + \cos(\langle u - v, x\rangle))$. This means that products of two cosine neurons can be expressed by the *sum* of two (other) cosine neurons. This completes the proof.

Sigmoidal Activation Functions

Hornik et al showed a more general result for sigmoidal activations which just means any function that satisfies σ such that:

$$\lim_{z \to -\infty} \sigma(z) = 0 \quad \text{and} \quad \lim_{z \to \infty} \sigma(z) = 1.$$

This result covers "threshold" activations, like tanh, other regular sigmoids, etc.

Theorem

If we use any sigmoidal activation $\sigma(\cdot)$ that is continuous, then $\mathcal{F}$ is a universal approximator.

Activation Functions

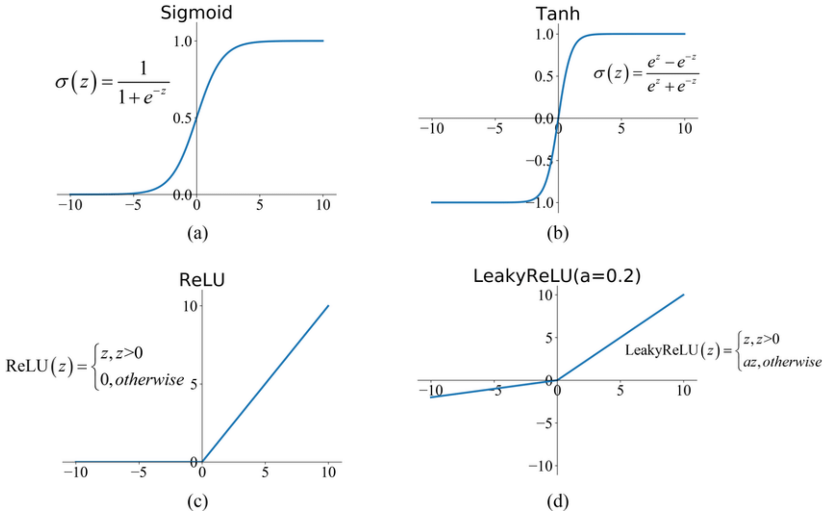

Figure: Various Activation Functions

Universal Approximation Theorem

Let $C(X, \mathbb{R}^m)$ denote the set of continuous functions from a subset X of a Euclidean $\mathbb{R}^n$ space to a Euclidean space $\mathbb{R}^m$. Let $\sigma \in C(\mathbb{R}, \mathbb{R})$.
Then σ is not polynomial if and only if for every $n \in \mathbb{N}$, $m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m)$, $\varepsilon > 0$ there exist $k \in \mathbb{N}$, $W \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$, $C \in \mathbb{R}^{m \times k}$ such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon$$

where

$$g(x) = C \cdot (\sigma \circ (W \cdot x + b))$$

The Multivariable Chain Rule

If a variable z depends on two variables y and x, which in turn depend on a third variable t, the chain rule can be expressed as follows:

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}$$

In the case of more variables, for a function $u(x_1, x_2, \ldots, x_n)$ where each x_i is a function of t, the chain rule generalizes to:

$$\frac{du}{dt} = \sum_{i=1}^{n} \frac{\partial u}{\partial x_i}\frac{dx_i}{dt}$$

Backpropagation

Backpropagation is used to calculate the gradient of the loss function with respect to the weights and biases of the network. For a network with a loss function L, the gradient with respect to the weights in layer l is calculated as:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}$$

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}}$$

Where:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

is the weighted input to layer l,
$\frac{\partial L}{\partial a^{(l)}}$ is the gradient of the loss with respect to the output of layer l, which is calculated during backpropagation from the final layer back to the input layer.

Gradient Descent

Gradient Descent updates the weights and biases of the network to minimize the loss function:

$$W^{(l)} = W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}$$

$$b^{(l)} = b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}}$$

where η is the learning rate.

Stochastic Gradient Descent (SGD)

SGD updates parameters using a small subset of the training data, called a mini-batch, which allows for faster convergence:

$$W^{(l)} = W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}} \text{ (mini-batch)}$$

$$b^{(l)} = b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}} \text{ (mini-batch)}$$

RMSProp Optimizer

RMSProp adjusts the learning rate for each parameter based on the moving average of squared gradients:

$$r_t = \rho r_{t-1} + (1-\rho)\left(\frac{\partial L}{\partial \theta_t}\right)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{r_t + \epsilon}} \cdot \frac{\partial L}{\partial \theta_t}$$

Where:

- θ_t represents the parameters (weights and biases) at time t.
- $\frac{\partial L}{\partial \theta_t}$ is the gradient of the loss function L with respect to θ_t.
- r_t is the moving average of the squared gradients.
- ρ is the decay rate.
- η is the learning rate.
- ϵ is a small constant for numerical stability.

Adam Optimizer

Adam combines the advantages of two SGD extensions, AdaGrad and RMSProp, and computes individual adaptive learning rates for different parameters:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \theta_t} \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial \theta_t}\right)^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

Where:

- θ_t represents parameters (weights and biases) at time t.
- m_t and v_t are estimates of the first and second moments of the gradients, respectively.

PyTorch Example