A Primer on ML and AI for Population Research

Kentaro Hoffman
Neural Networks

Definition and Context

Artificial Intelligence (AI): Effort to automate intellectual tasks normally performed by humans.

Machine Learning (ML): Subfield of AI focused on learning from data.

Deep Learning (DL): Subfield of ML emphasizing learning successive layers of representations.

Historical Context

Symbolic AI: Dominant from the 1950s to the 1980s, focused on handcrafted rules.

Early Neural Networks: Emerged in the 1980s with backpropagation.

Kernel Methods: Popular in the 1990s, e.g., Support Vector Machines (SVMs).

Decision Trees and Gradient-Boosting Machines: Gained prominence in the 2000s.

Deep Neural Networks: The current paradigm. Gained prominence in >2010s.

Large Language Models: Gained massive prominence in 2023. Examples include GPT-4, Claude, Copilot

Components of Neural Networks

Layers: The building blocks of neural networks.

Weights: Parameters within the network that are adjusted during training.

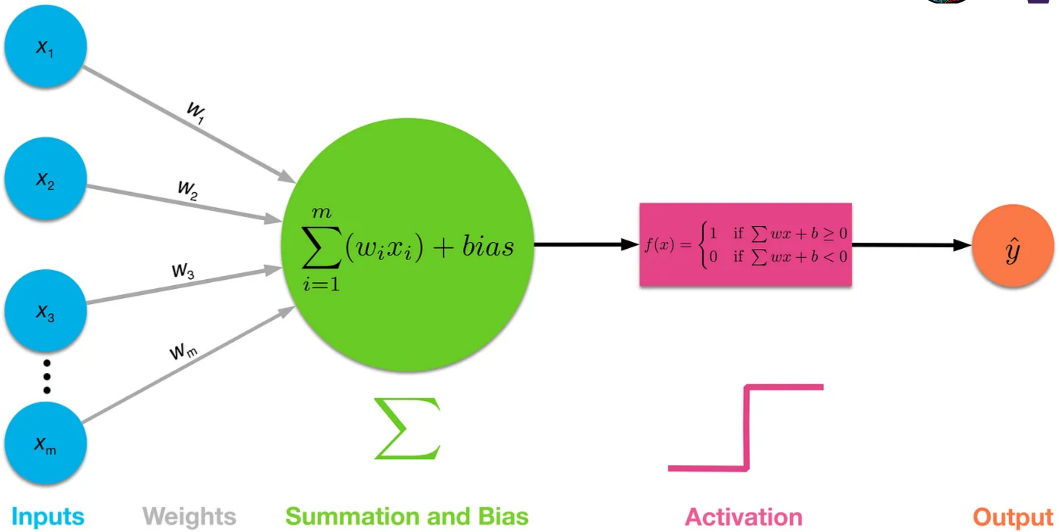
Activation Functions: Functions applied to the output of each layer.

Loss Function: Measures the difference between predicted and actual outputs.

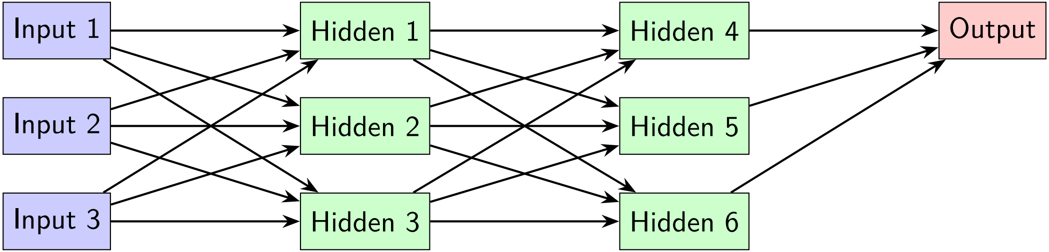
Optimizer: Algorithm that adjusts weights to minimize the loss function.

Backpropagation: Method for updating weights using the gradient of the loss function.

Neural Network Visualization



Deep Neural Network Visualization



Activation Functions

Sigmoid Function: $\sigma(x) = \frac{1}{1+e^{-x}}$

Hyperbolic Tangent (tanh): $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Rectified Linear Unit (ReLU): $\text{ReLU}(x) = \max(0, x)$

Leaky ReLU: $\text{Leaky ReLU}(x) = \max(0.01x, x)$

Softmax Function: $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

Strengths and Weaknesses of Activation Functions

Function	Strengths	Weaknesses	Examples of Use
Sigmoid	Smooth gradient, outputs probabilities	Vanishing gradient problem, slow convergence	Output layer for binary classification
Tanh	Zero-centered output, stronger gradients than sigmoid	Vanishing gradient problem	Hidden layers in RNNs

Strengths and Weaknesses of Activation Functions

Function	Strengths	Weaknesses	Examples of Use
ReLU	Simple, computationally efficient, mitigates vanishing gradient problem	Can cause dead neurons during training	Hidden layers in CNNs and deep networks
Leaky ReLU	Prevents dead neurons, allows small negative values to pass through	Introduces slight computational overhead	Deep networks prone to dying ReLU problem, generative models

Strengths and Weaknesses of Activation Functions

Function	Strengths	Weaknesses	Examples of Use
Softmax	Converts logits to probability distribution, sum equals 1	Computationally expensive, sensitive to large input values	Output layer for multi-class classification, final layer of image classifiers

Logistic Regression as a Simple Neural Network

Logistic regression is a linear model for binary classification that can be viewed as a simple neural network with no hidden layers.

Mathematical Formulation of Logistic Regression

Given input features x, weights W, and "bias" b, the logistic regression model predicts the probability of class 1 as:

$$P(y = 1|x) = \sigma(Wx + b)$$

where σ is the sigmoid activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

NOTE: *The "bias" term b plays the role of the intercept term*

Training Process

Forward Pass:

Compute predictions by passing input through the network.

Example for a single layer:

$$y = f(Wx + b)$$

Loss Function (L):

Measures prediction error.

Example (Mean Squared Error):

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Backpropagation Algorithm

With the Loss from the Forward pass, we now want to figure out how to change our weights to improve our loss

Initialization:

Initialize weights W and biases b with small random values.

Forward Pass + Loss Calculation:

Compute predictions y by passing input x through the network layers and calculating the Loss

Backward Pass (Backpropagation):

Compute gradients of the loss with respect to weights and biases using the chain rule.

Backpropagation (continued)

Example for weight update in a single layer:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial W}$$

Update Weights and Biases:

Adjust weights and biases using gradient descent:

$$W := W - \eta \nabla_W L$$

where η is the learning rate.

Repeat:

Iterate over multiple epochs until convergence.

Tensors

A tensor is a container for data, usually numerical data. It is a generalization of matrices to an arbitrary number of dimensions.

Scalars (Rank 0 Tensors): Single number.

Vectors (Rank 1 Tensors): Array of numbers.

Matrices (Rank 2 Tensors): Array of vectors.

Rank 3 Tensors: Array of matrices.

Rank 4 Tensors: Array of rank 3 tensors.

Example: Gradient Descent in R

Single Variable:

```
gradient_descent <- function(f, df, x0, learning_rate, n_iter) {
  x <- x0
  for (i in 1:n_iter) {
    x <- x - learning_rate * df(x)
  }
  return(x)
}
f <- function(x) { x^2 }
df <- function(x) { 2 * x }
x_min <- gradient_descent(f, df, x0 = 10, learning_rate = 0.1, n_iter =
```

Example: Gradient Descent in R

Multiple Variables:

```
gradient_descent <- function(f, grad_f, x0, learning_rate, n_iter) {
  x <- x0
  for (i in 1:n_iter) {
    x <- x - learning_rate * grad_f(x[1], x[2])
  }
  return(x)
}
f <- function(x, y) { x^2 + y^2 }
grad_f <- function(x, y) { c(2 * x, 2 * y) }
x_min <- gradient_descent(f, grad_f, x0 = c(10, 10), learning_rate = 0.
```

Forward Mode Auto-differentiation

```r
library(torch)

# Define the variable
x <- torch_tensor(3, requires_grad = TRUE)

# Define the function
y <- x^2 + 2 * x + 1

# Compute the gradient
y$backward()

# Print the gradient
print(x$grad)  # Output: tensor(8.)
```

What's going on with Auto-differentiation?

Define the Variable: We create a tensor `x` with the value 3 and set `requires_grad = TRUE` to track all operations on this tensor.

Define the Function: We define the function $f(x) = x^2 + 2x + 1$ using operations provided by the `torch` package.

Compute the Gradient: We call `y$backward()` to compute the gradient of `y` with respect to `x`.

Print the Gradient: We print the gradient stored in `x$grad`, which gives us the value 8.

This example demonstrates how auto-differentiation in the `torch` package leverages tensors to efficiently compute derivatives, making it a powerful tool for optimization tasks such as training neural networks.

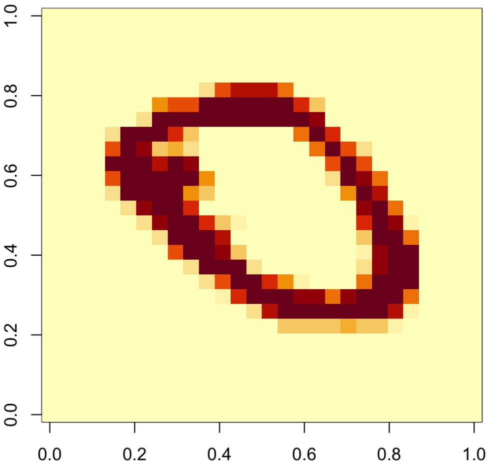
Example: Handwritten Digit Classification

Problem: Classify grayscale images of handwritten digits (28×28 pixels) into 10 categories (0 through 9).

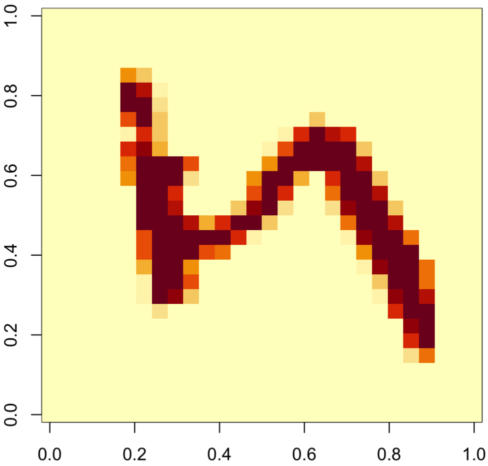
Dataset: MNIST dataset with 60,000 training images and 10,000 test images.

Model: Sequential model with two Dense layers.

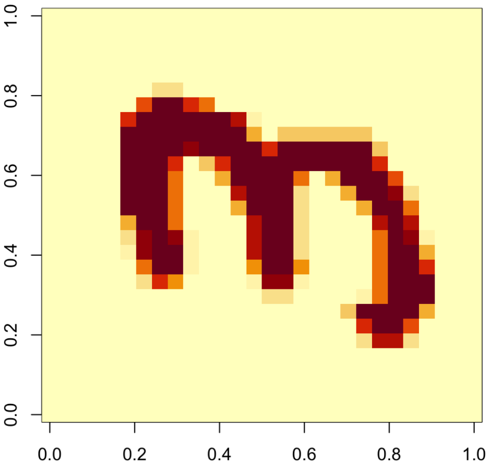
Example: Handwritten Digit Classification



Example: Handwritten Digit Classification



Example: Handwritten Digit Classification



Example: Handwritten Digit Classification (Code)

```
library(keras)
mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y

model <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = 'relu', input_shape = c(28 * 28)
  layer_dense(units = 10, activation = 'softmax')

model %>% compile(
  optimizer = 'rmsprop',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
```

Example: Handwritten Digit Classification (Training)

```r
train_images <- array_reshape(train_images, c(60000, 28 * 28))
train_images <- train_images / 255
test_images <- array_reshape(test_images, c(10000, 28 * 28))
test_images <- test_images / 255

model %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
```

Convolutional Neural Networks (CNNs): Structure

Convolutional Layers:

Apply convolution operations to extract features from input images.

Convolution: A mathematical operation on two functions producing a third function that expresses how the shape of one is modified by the other.

Example:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

where I is the input image and K is the kernel or filter.

Pooling Layers:

Reduce dimensionality by down-sampling.

Example (Max Pooling):

$$P(i,j) = \max_{m,n} I(i+m, j+n)$$

Promising Research Directions

Small Data Approaches:
- Transfer learning from larger populations to smaller ones
- Few-shot learning techniques for sparse demographic contexts

Hybrid Models:
- Combining theory-driven demographic models with neural components
- Example: Integrating Lee-Carter with neural components for better forecasting

Explainable AI (XAI):
- Methods to interpret neural network decisions (crucial for policy)
- SHAP values and LIME explain variable importance in predictions

Practical Tips

Start Simple:

- Begin with shallow networks (1-2 hidden layers)
- Compare with traditional methods as baseline

R or Python?

- R: Keras and torch packages available, familiar for demographers
- Python: More extensive ecosystem, but steeper learning curve

Cloud Computing:

- Google Colab: Free GPU access for model training
- RStudio Cloud: Familiar interface with scalable resources

Development Path:

- Start with pre-built models in R
- Graduate to customized architectures as needed

Exercise: Practice Time with scorchr!