Deep Learning Introduction and Explainable AI

Lecture 4: From Non-Linearity to Interpretability

Prof. Asc. Endri Raco

November 2025

Section 1

Introduction and Motivation

Slide 1: Course Overview

Lecture Progression:

- L1-L2: Logistic Regression (Interpretable, Linear)
- L3: Ensemble Methods (Tree-based, Partially Interpretable)
- **L4: Neural Networks (Powerful, Opaque)** ← We are here

Central Question:

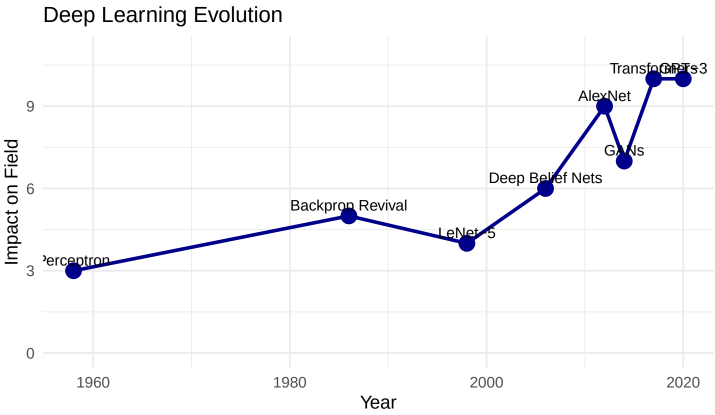
How do we build models that are both **powerful** AND **understandable**?

Slide 2: Why Deep Learning Now?

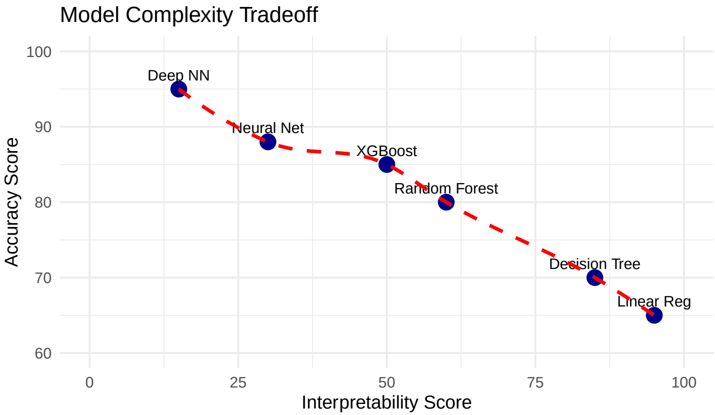
Three Converging Forces:

1. **Data Availability**
   - ImageNet: 14M labeled images
   - Text corpora: Trillions of tokens
2. **Computational Power**
   - GPUs: 100x faster for matrix operations
   - Distributed training infrastructure
3. **Algorithmic Innovation**
   - ReLU activation (2011)
   - Batch Normalization (2015)
   - Adam optimizer (2014)

Slide 3: Historical Milestones



Slide 4: The Interpretability-Accuracy Tradeoff



Slide 5: From Logistic Regression to Neural Networks

Logistic Regression (Review):

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

What if we:

1. Stack multiple logistic units in layers?
2. Allow non-linear transformations between layers?
3. Learn hierarchical features automatically?

⇒ **Neural Networks**

Section 2

The Perceptron: Building Block

Slide 6: Biological Inspiration

Natural Neuron Components:

- **Dendrites:** Input receivers $(x_1, x_2, \ldots, x_n)$
- **Cell Body:** Integration (weighted sum + activation)
- **Axon:** Output transmission
- **Synapses:** Connection weights $(w_1, w_2, \ldots, w_n)$

Key Insight: Neurons fire when cumulative input exceeds threshold

Warning: Useful metaphor, not biological accuracy!

Slide 7: The Mathematical Perceptron

Definition: A computational unit that:

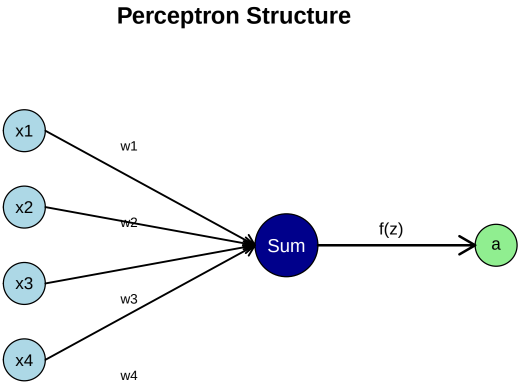
1. Receives inputs $\mathbf{x} = (x_1, x_2, \ldots, x_n)$
2. Computes weighted sum: $z = \sum_{i=1}^{n} w_i x_i + b$
3. Applies activation: $a = f(z)$

Compact Notation:

$$a = f(\mathbf{w}^T \mathbf{x} + b)$$

where $\mathbf{w} = (w_1, \ldots, w_n)^T$

Slide 8: Perceptron Visualization

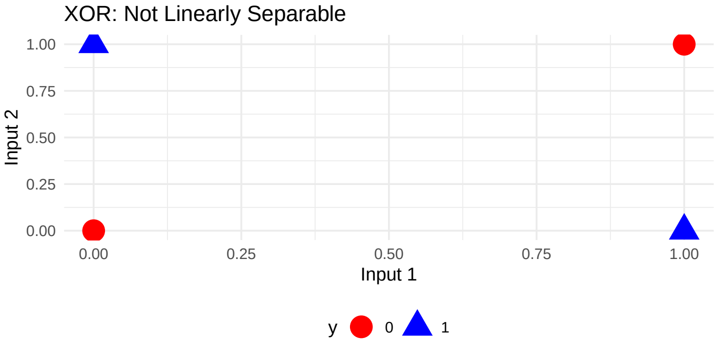


Slide 9: Simple Perceptron in R

```r
# Perceptron function
perceptron <- function(x, w, b, activation = "step") {
  z <- sum(w * x) + b

  if (activation == "step") {
    return(ifelse(z >= 0, 1, 0))
  } else if (activation == "sigmoid") {
    return(1 / (1 + exp(-z)))
  }
}

# Example: AND gate
x1 <- c(0, 0, 1, 1)
x2 <- c(0, 1, 0, 1)
w <- c(1, 1)
b <- -1.5

for (i in 1:4) {
```

Slide 10: Perceptron Limitations

The XOR Problem (Minsky & Papert, 1969):



Solution: Multiple layers (Multi-Layer Perceptron)

Section 3

Activation Functions

Slide 11: Why Non-Linear Activations?

Theorem: Without non-linear activations, a deep network collapses to a single linear transformation.

Proof sketch:

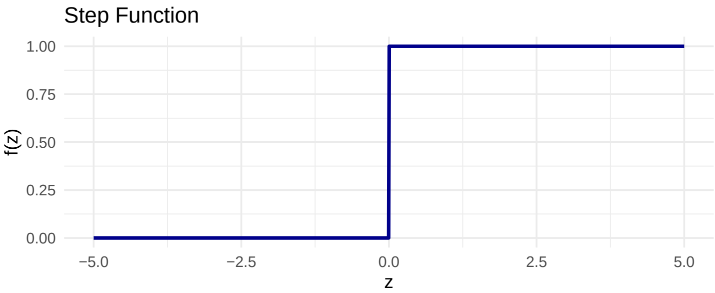
$$h_1 = W_1 x, \quad h_2 = W_2 h_1 = W_2 W_1 x = W_{combined} x$$

Key Point: Non-linearity enables learning complex decision boundaries

Slide 12: Step Function (Historical)

Definition:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



Problem: Derivative is zero almost everywhere (cannot use gradient descent)

Slide 13: Sigmoid Activation

Definition:

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

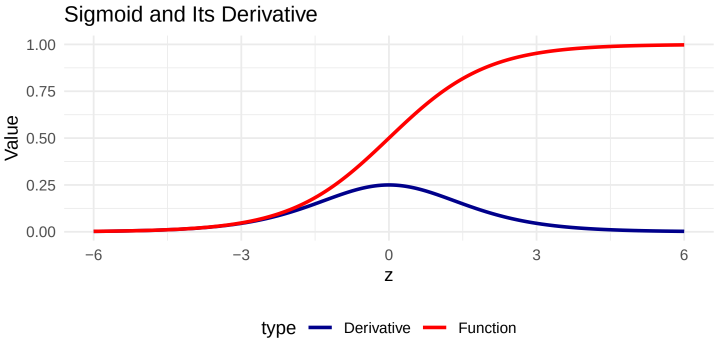
Derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

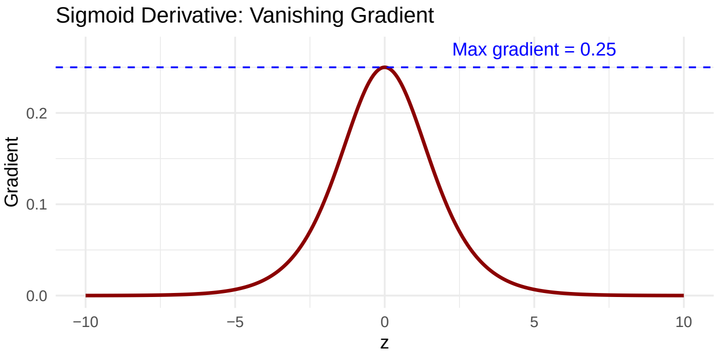
Properties:

- Output range: $(0, 1)$
- Smooth and differentiable
- Interpretable as probability

Slide 14: Sigmoid Visualization



Slide 15: Sigmoid Problems - Vanishing Gradient



Issue: For $|z| > 5$, gradient ≈ 0 (slow learning)

Slide 16: Hyperbolic Tangent (tanh)

Definition:

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

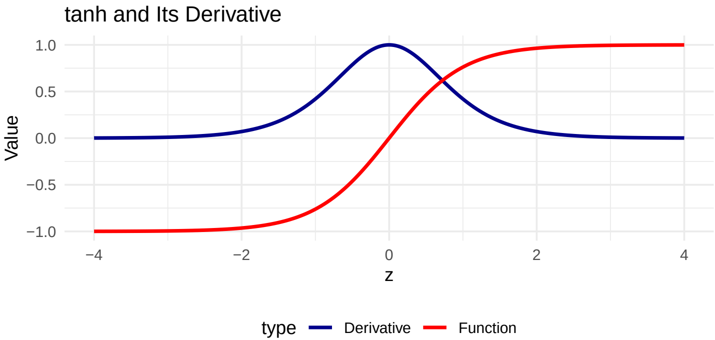
Derivative:

$$\tanh'(z) = 1 - \tanh^2(z)$$

Advantages over Sigmoid:

- Zero-centered: output range $(-1, 1)$
- Stronger gradients (max derivative = 1)

Slide 17: Tanh Visualization



Slide 18: ReLU - The Modern Standard

Rectified Linear Unit (ReLU):

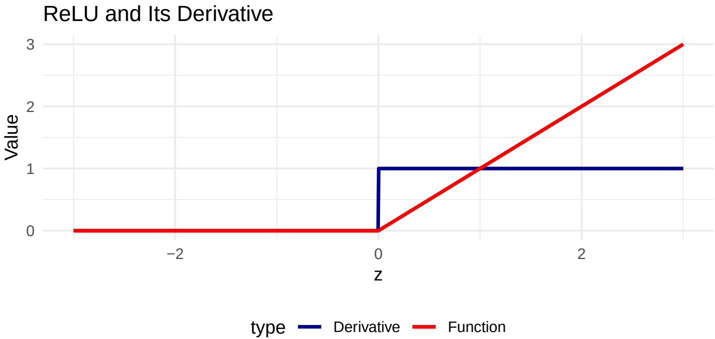
$$\mathsf{ReLU}(z) = \max(0, z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Derivative:

$$\mathsf{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Introduced: Nair & Hinton (2010)

Slide 19: ReLU Advantages



Benefits:

1. No vanishing gradient for $z > 0$
2. Computationally efficient
3. Sparsity (many neurons output zero)

Slide 20: ReLU Variants

Leaky ReLU:

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}, \quad \alpha = 0.01$$

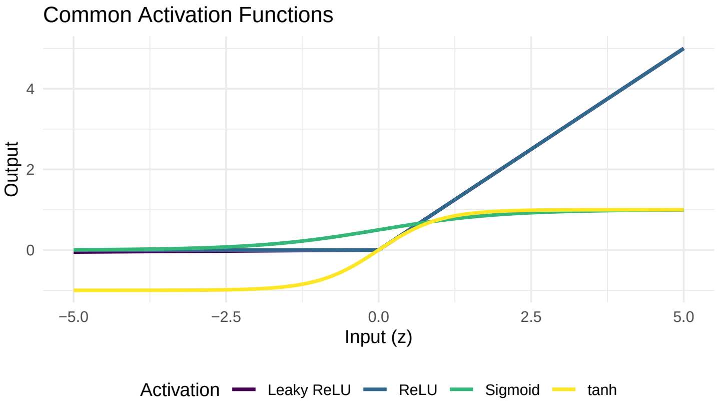
Parametric ReLU (PReLU):

- α is learned during training

ELU (Exponential Linear Unit):

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases}$$

Slide 21: Activation Comparison



Slide 22: Implementing Activations in R

```r
# Define activation functions
sigmoid <- function(z) 1 / (1 + exp(-z))
tanh_act <- function(z) tanh(z)
relu <- function(z) pmax(0, z)
leaky_relu <- function(z, alpha = 0.01) pmax(alpha * z, z)

# Test on sample data
z_test <- c(-2, -1, 0, 1, 2)

cat("Input:", z_test, "\n")
```

```
## Input: -2 -1 0 1 2
```

```r
cat("Sigmoid:", round(sigmoid(z_test), 3), "\n")
```

```
## Sigmoid: 0.119 0.269 0.5 0.731 0.881
```

```r
cat("tanh:", round(tanh_act(z_test), 3), "\n")
```

Section 4

Feedforward Neural Networks

Slide 23: Multi-Layer Perceptron (MLP) Architecture

Architecture Components:

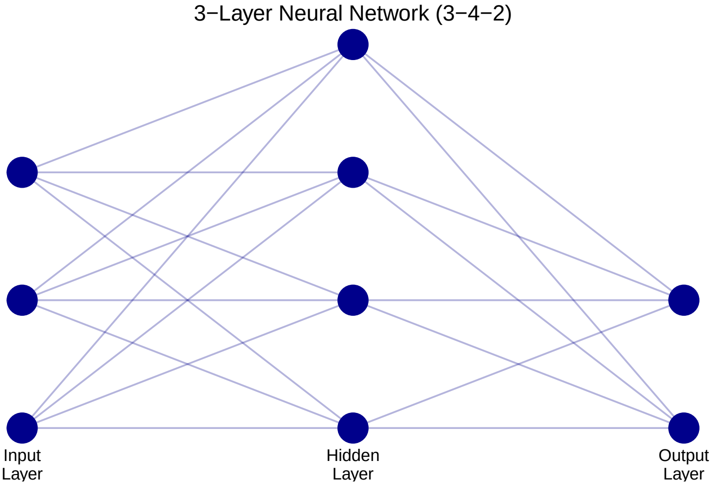
1. **Input Layer:** Raw features $\mathbf{x} \in \mathbb{R}^n$
2. **Hidden Layers:** L layers with neurons
3. **Output Layer:** Predictions $\hat{y}$

Forward Pass (Layer l):

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

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

Slide 24: Network Architecture Visualization



Slide 25: Matrix Notation for Forward Pass

Single Sample ($\mathbf{x}$ is a column vector):

$$\mathbf{a}^{(1)} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{a}^{(2)} = f(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)})$$

$$\hat{y} = \mathbf{a}^{(L)}$$

Batch Processing ($\mathbf{X}$ is a matrix with samples as rows):

$$\mathbf{A}^{(l)} = f(\mathbf{A}^{(l-1)}\mathbf{W}^{(l)T} + \mathbf{b}^{(l)})$$

Slide 26: Building MLP in Keras (R)

```r
library(keras)

# Define network architecture
model <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = "relu",
              input_shape = c(784)) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 10, activation = "softmax")

# Compile model
model %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

Slide 27: Complete MLP Example - MNIST

```
# Load MNIST dataset
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# Preprocess
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255
x_test <- x_test / 255

y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

# Train model
history <- model %>% fit(
```

Slide 28: Universal Approximation Theorem

Theorem (Cybenko, 1989; Hornik et al., 1989):

A feedforward network with: - One hidden layer - Finite number of neurons - Non-polynomial activation function

can approximate **any continuous function** on compact subsets of $\mathbb{R}^n$ to arbitrary accuracy.

Implication: Neural networks are *universal function approximators*

Caveat: Theorem says nothing about: - How many neurons needed - How to find the weights (training)

Slide 29: Depth vs. Width

Empirical Findings:

1. **Shallow & Wide:** Harder to train, requires more parameters
2. **Deep & Narrow:** More efficient representation, better generalization

Intuition: Deep networks learn hierarchical features

- Layer 1: Edges, textures
- Layer 2: Parts, shapes
- Layer 3: Objects, concepts

Example: ImageNet with ResNet-50 (50 layers) vs. single hidden layer (infeasible)

Slide 30: The Interpretability Challenge

From L3 (Random Forest) to L4 (Neural Networks):

Model	Interpretability Method
Random Forest	Feature importance (Gini, permutation)
Neural Network	???

Why NNs are Hard to Interpret:

1. **Non-linear transformations** across multiple layers
2. **Millions of parameters** (distributed representations)
3. **No direct feature-to-output mapping**

⇒ **Need for Explainable AI (XAI)** methods

Section 5

Training Neural Networks

Slide 31: The Learning Problem

Goal: Find weights $\mathbf{W}$ and biases $\mathbf{b}$ that minimize prediction error

The Process:

1. **Forward Pass:** Compute predictions $\hat{y}$
2. **Calculate Loss:** How wrong are we? $L(\hat{y}, y)$
3. **Backward Pass:** Calculate gradients (which direction to adjust weights?)
4. **Update Weights:** Take a small step in the right direction

Analogy: Walking down a foggy mountain—you feel the slope under your feet and take small steps downhill.

Slide 32: Loss Functions - Measuring Error

For Regression (continuous output):

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

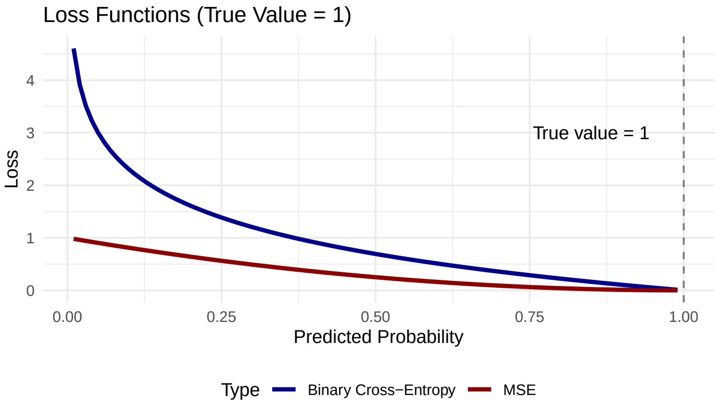
For Binary Classification:

$$\text{Binary Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

For Multi-Class Classification:

$$\text{Categorical Cross-Entropy} = -\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

Slide 33: Visualizing Loss Functions



Slide 34: Loss Functions in R

```r
# Mean Squared Error
mse_loss <- function(y_true, y_pred) {
  mean((y_true - y_pred)^2)
}

# Binary Cross-Entropy (with numerical stability)
bce_loss <- function(y_true, y_pred) {
  epsilon <- 1e-7  # Prevent log(0)
  y_pred <- pmax(pmin(y_pred, 1 - epsilon), epsilon)
  -mean(y_true * log(y_pred) +
        (1 - y_true) * log(1 - y_pred))
}

# Example
y_true <- c(0, 1, 1, 0, 1)
y_pred <- c(0.1, 0.9, 0.7, 0.2, 0.8)

cat("MSE Loss:", mse_loss(y_true, y_pred), "\n")
```

Slide 35: Gradient Descent - The Core Idea

Question: If loss $= f(\mathbf{w})$, how do we find the best $\mathbf{w}$?

Gradient Descent Rule:

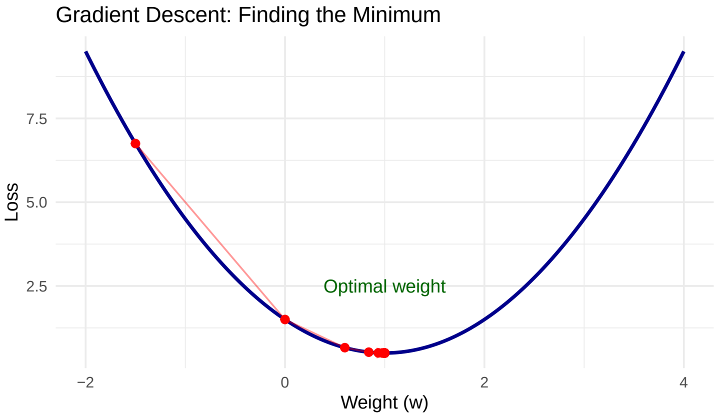
$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \frac{\partial L}{\partial \mathbf{w}}$$

where η is the **learning rate**

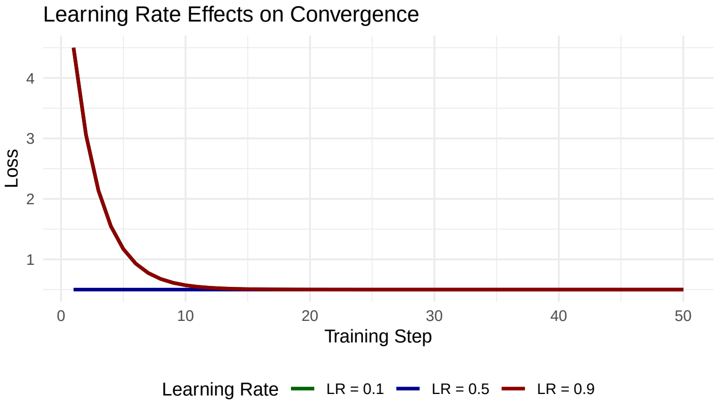
Intuition:

- Gradient $\frac{\partial L}{\partial \mathbf{w}}$ points **uphill** (direction of steepest increase)
- We move in the **opposite direction** (downhill)
- η controls step size

Slide 36: Gradient Descent Visualization



Slide 37: Learning Rate Impact



Key Takeaway: Too small = slow, too large = unstable!

Slide 38: Backpropagation - The Chain Rule

The Challenge: Network has many layers. How do we compute $\frac{\partial L}{\partial \mathbf{w}^{(1)}}$ for the first layer?

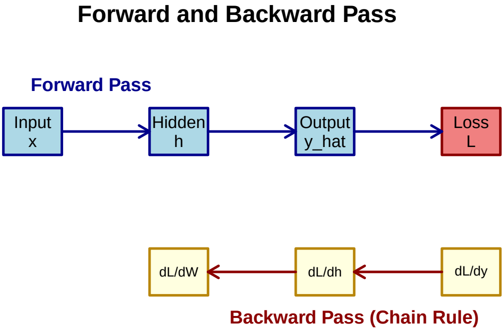
Solution: Use the **chain rule** from calculus!

Example (2-layer network):

$$\frac{\partial L}{\partial w^{(1)}} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial w^{(1)}}$$

Intuition: How does changing $w^{(1)}$ ripple through the network to affect final loss?

Slide 39: Backpropagation Visual Flow



Slide 40: Simple Backpropagation Example

```r
# Tiny network: x -> w1 -> h -> w2 -> y_hat
# Loss: (y - y_hat)^2

# Forward pass
x <- 2.0
y <- 5.0
w1 <- 0.5
w2 <- 1.0

h <- w1 * x                 # h = 1.0
y_hat <- w2 * h             # y_hat = 1.0
loss <- (y - y_hat)^2       # loss = 16.0

cat("Forward Pass:\n")
```

```
## Forward Pass:
```

```r
cat("h =", h, ", y_hat =", y_hat, ", Loss =", loss, "\n\n")
```

Slide 41: Gradient Descent Update

```r
# Update weights using gradients
learning_rate <- 0.1

w1_new <- w1 - learning_rate * d_loss_d_w1
w2_new <- w2 - learning_rate * d_loss_d_w2

cat("Weight Updates:\n")
```

```
## Weight Updates:
```

```r
cat("w1:", w1, "->", w1_new, "\n")
```

```
## w1: 0.5 -> 2.1
```

```r
cat("w2:", w2, "->", w2_new, "\n\n")
```

```
## w2: 1 -> 1.8
```

```r
# Check new loss
```

Slide 42: Batch vs. Stochastic Gradient Descent

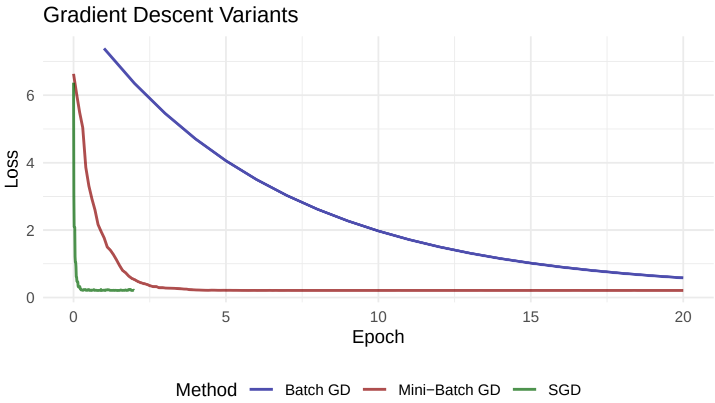
Three Variants:

1. **Batch GD:** Use ALL training data to compute gradient
   - Accurate but slow for large datasets
2. **Stochastic GD (SGD):** Use ONE random sample
   - Fast but noisy updates
3. **Mini-Batch GD:** Use a small batch (e.g., 32 samples)
   - **Best trade-off** (default in practice)

Update Rule (Mini-Batch):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{B} \sum_{i \in \text{Batch}} \nabla_{\mathbf{w}} L_i$$

Slide 43: SGD Variants Comparison



Slide 44: Advanced Optimizers - Momentum

Problem with SGD: Can oscillate in ravines (steep in one direction, shallow in another)

SGD with Momentum:

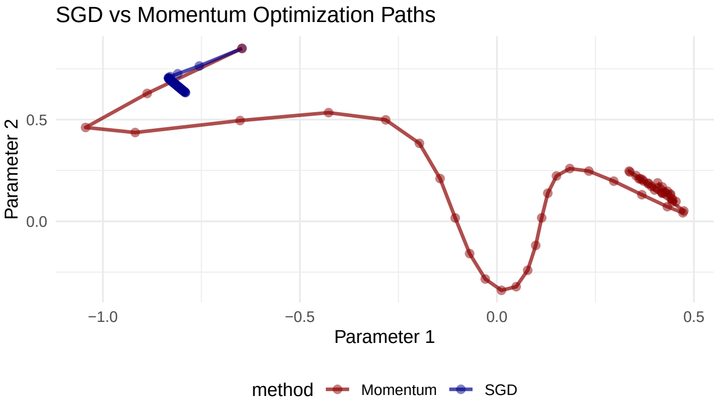
$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + \nabla_{\mathbf{w}} L$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \mathbf{v}_t$$

Intuition: Rolling ball gains momentum—dampens oscillations, accelerates in consistent directions

Typical value: $\beta = 0.9$

Slide 45: Momentum Visualization



Slide 46: Adam Optimizer - The Modern Standard

Adam (Adaptive Moment Estimation):

Combines: 1. **Momentum** (moving average of gradients) 2. **RMSprop** (adaptive learning rates per parameter)

Update Rules:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L)^2$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Default values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.001$

Slide 47: Optimizer Comparison in Practice

```r
library(keras)

# Define simple model
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = c(10)) %>%
  layer_dense(units = 1)

# Try different optimizers
optimizers <- list(
  "SGD" = optimizer_sgd(learning_rate = 0.01),
  "SGD+Momentum" = optimizer_sgd(learning_rate = 0.01,
                                  momentum = 0.9),
  "Adam" = optimizer_adam(learning_rate = 0.001)
)

# Compile with each optimizer
model %>% compile(
```

Slide 48: Weight Initialization - Why It Matters

Problem: Bad initialization can cause:

1. **Vanishing gradients** (weights too small)
2. **Exploding gradients** (weights too large)
3. **Dead neurons** (ReLU outputs always zero)

Example - All Zeros:

If $\mathbf{W} = 0$, all neurons in a layer compute the same function! (Symmetry problem)

Slide 49: Initialization Strategies

Xavier/Glorot Initialization (for sigmoid/tanh):

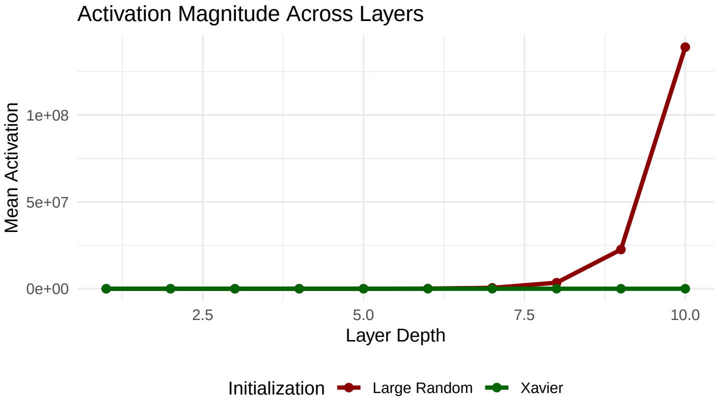
$$W \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right)$$

He Initialization (for ReLU):

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

Intuition: Scale variance based on layer size to maintain signal flow

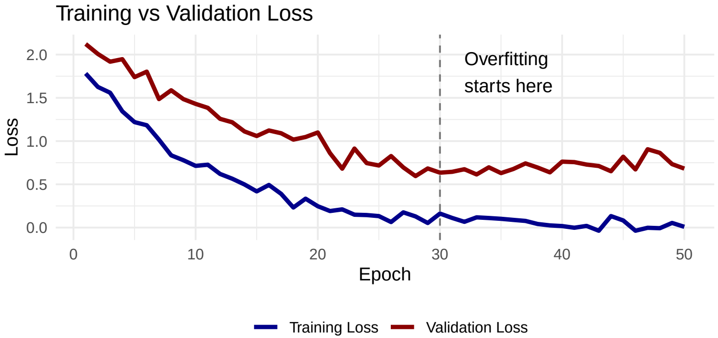
Slide 50: Initialization Impact



Green (Xavier): Stable signal propagation!

Slide 51: Overfitting in Neural Networks

The Problem: Deep networks have millions of parameters—can memorize training data!



Slide 52: Regularization - L2 (Weight Decay)

Add penalty to loss function:

$$L_{total} = L_{data} + \lambda \sum_{l} \|\mathbf{W}^{(l)}\|^2$$

Effect: Encourages smaller weights (simpler models)

In R/Keras:

```r
model <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = "relu",
              kernel_regularizer = regularizer_l2(0.01)) %>%
  layer_dense(units = 10, activation = "softmax")
```

Slide 53: Dropout - Randomly Drop Neurons

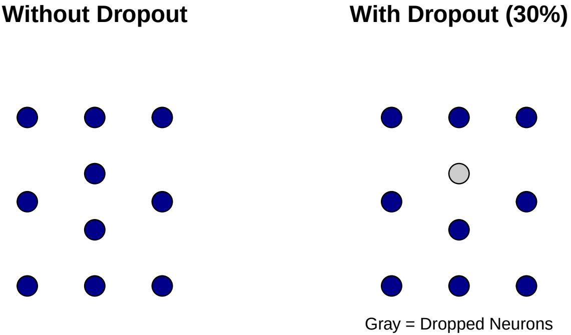
During Training: Randomly set neuron outputs to zero with probability p

During Testing: Use all neurons (scaled appropriately)

Intuition: Forces network to learn robust features (can't rely on any single neuron)

Typical value: $p = 0.2$ to 0.5

Slide 54: Dropout Visualization



Slide 55: Dropout in R/Keras

```r
model <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = "relu",
              input_shape = c(784)) %>%
  layer_dropout(rate = 0.3) %>%  # Drop 30% of neurons
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = "softmax")

# Note: Dropout automatically disabled during prediction!
```

Slide 56: Early Stopping

Strategy: Stop training when validation loss stops improving

Implementation:

1. Monitor validation loss every epoch
2. If no improvement for n epochs (patience), stop
3. Restore weights from best epoch

```r
early_stop <- callback_early_stopping(
  monitor = "val_loss",
  patience = 10,
  restore_best_weights = TRUE
)

history <- model %>% fit(
  x_train, y_train,
  validation_split = 0.2,
  epochs = 100,
```

Slide 57: Batch Normalization

Problem: Internal covariate shift (layer inputs change as previous layers update)

Solution: Normalize activations within each mini-batch

$$\hat{z} = \frac{z - \mu_{batch}}{\sqrt{\sigma^2_{batch} + \epsilon}}$$

$$\mathsf{BN}(z) = \gamma \hat{z} + \beta$$

Benefits:

- Faster training (can use higher learning rates)
- Reduces need for dropout
- Acts as regularization

Slide 58: Batch Normalization in Practice

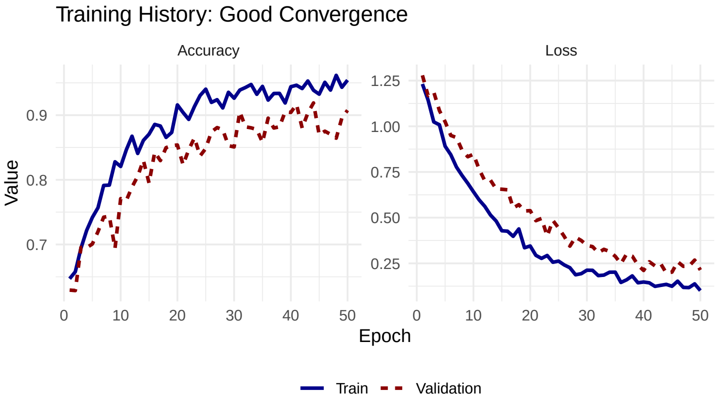
```r
model <- keras_model_sequential() %>%
  layer_dense(units = 128, input_shape = c(784)) %>%
  layer_batch_normalization() %>%  # Add BN here
  layer_activation("relu") %>%
  layer_dense(units = 64) %>%
  layer_batch_normalization() %>%
  layer_activation("relu") %>%
  layer_dense(units = 10, activation = "softmax")
```

Best Practice: Place BN before or after activation (debate ongoing!)

Slide 59: Complete Training Pipeline

```r
# 1. Build model with regularization
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu",
              kernel_regularizer = regularizer_l2(0.001),
              input_shape = c(784)) %>%
  layer_batch_normalization() %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 128, activation = "relu",
              kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 10, activation = "softmax")

# 2. Compile with Adam optimizer
model %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

Slide 60: Training Diagnostics



Signs of healthy training: Both metrics improving, no large gap between train/validation

Section 6

Convolutional Neural Networks (CNNs)

Slide 61: From Fully Connected to Convolutional

Problem with Fully Connected Networks for Images:

- MNIST image: $28 \times 28 = 784$ pixels
- First hidden layer (128 neurons): $784 \times 128 =$ **100,352 parameters**
- ImageNet image: $224 \times 224 \times 3 = 150{,}528$ pixels
- First hidden layer (1000 neurons): **150 million parameters!**

Issues:

1. Too many parameters (overfitting, memory)
2. Loses spatial structure (treats pixels as independent)
3. Not translation invariant (cat in top-left   cat in bottom-right)

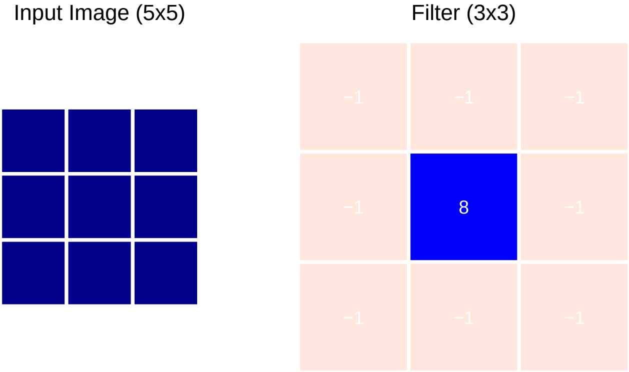
Slide 62: The Convolutional Solution

Key Ideas:

1. **Local Connectivity:** Each neuron connects to small region (receptive field)
2. **Parameter Sharing:** Same weights (filter) used across entire image
3. **Spatial Hierarchy:** Build from edges → shapes → objects

Result: Dramatically fewer parameters, preserves spatial structure

Slide 63: Convolution Operation - Visual Intuition



−1	−1	−1
−1	8	−1
−1	−1	−1

Convolution: Slide filter over image, compute dot product at each position

Slide 64: Convolution Mathematics

2D Convolution:

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

where: - I = Input image - K = Kernel (filter) - (i,j) = Output position

Intuition: Filter "scans" image, activating when it finds matching patterns

Slide 65: Convolution in R - Step by Step

```r
# Simple 2D convolution function
conv2d <- function(image, kernel) {
  img_rows <- nrow(image)
  img_cols <- ncol(image)
  ker_rows <- nrow(kernel)
  ker_cols <- ncol(kernel)

  # Output dimensions
  out_rows <- img_rows - ker_rows + 1
  out_cols <- img_cols - ker_cols + 1

  output <- matrix(0, out_rows, out_cols)

  for (i in 1:out_rows) {
    for (j in 1:out_cols) {
      # Extract patch
      patch <- image[i:(i+ker_rows-1), j:(j+ker_cols-1)]
```

Slide 66: Edge Detection Example

```
# Create simple image with vertical edge
image <- matrix(c(
  rep(0, 15),
  rep(1, 15)
), nrow = 6, byrow = FALSE)

# Vertical edge detector
vertical_filter <- matrix(c(
  -1, 0, 1,
  -1, 0, 1,
  -1, 0, 1
), nrow = 3, byrow = TRUE)

# Apply convolution
result <- conv2d(image, vertical_filter)

# Visualize
par(mfrow = c(1, 2))
```

Slide 67: Multiple Filters = Multiple Features

In Practice:

- Layer 1: 32 filters (3×3) → Detect 32 different patterns
- Layer 2: 64 filters (3×3) → Combine patterns into 64 features
- Layer 3: 128 filters (3×3) → Higher-level features

Each filter learns different features:

- Filter 1: Horizontal edges
- Filter 2: Vertical edges
- Filter 3: Corners
- Filter 4: Textures
- ... and so on

Slide 68: Padding and Stride

Padding: Add border of zeros to preserve spatial dimensions

- **Valid:** No padding (output shrinks)
- **Same:** Pad so output = input size

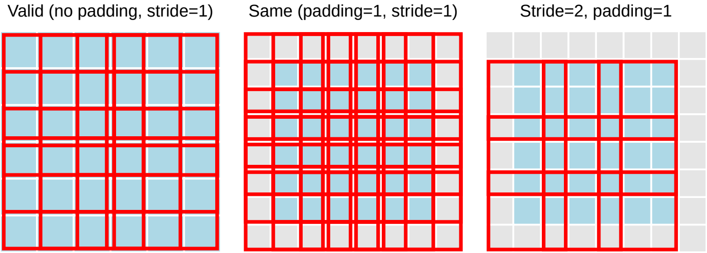
Stride: Step size when moving filter

- Stride = 1: Move 1 pixel at a time
- Stride = 2: Move 2 pixels (output half size)

Output size formula:

$$\text{Output} = \frac{\text{Input} - \text{Filter} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

Slide 69: Padding and Stride Visualization



Red boxes = filter positions

Slide 70: Pooling Layers

Purpose: Reduce spatial dimensions (downsample)

Max Pooling (most common):

- Take maximum value in each region
- Typical: 2×2 window, stride 2 (halves dimensions)

Average Pooling:

- Take average value in each region

Benefits:

- Reduces parameters
- Adds translation invariance
- Reduces overfitting

Slide 71: Max Pooling Visualization

```r
# Max pooling function
max_pool <- function(image, pool_size = 2, stride = 2) {
  rows <- nrow(image)
  cols <- ncol(image)

  out_rows <- floor((rows - pool_size) / stride) + 1
  out_cols <- floor((cols - pool_size) / stride) + 1

  output <- matrix(0, out_rows, out_cols)

  for (i in 1:out_rows) {
    for (j in 1:out_cols) {
      r_start <- (i - 1) * stride + 1
      c_start <- (j - 1) * stride + 1
      patch <- image[r_start:(r_start + pool_size - 1),
                     c_start:(c_start + pool_size - 1)]
      output[i, j] <- max(patch)
    }
```

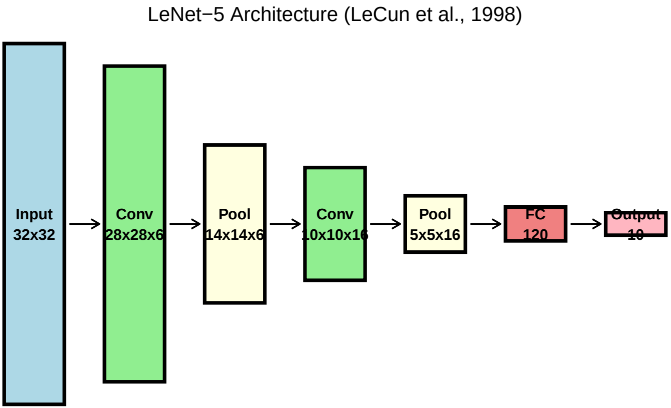
Slide 72: CNN Architecture Pattern

Standard CNN Structure:

1. **Convolutional Block:**
   - Conv Layer (with ReLU)
   - Optional: Batch Normalization
   - Pooling Layer
2. **Repeat 3-5 times** (increasing filter depth)
3. **Flatten** (convert 3D to 1D)
4. **Fully Connected Layers** (classification)

Example: Conv(32) → Pool → Conv(64) → Pool → Conv(128) → Pool → Flatten → Dense(128) → Dense(10)

Slide 73: Classic CNN Architecture - LeNet-5



Slide 74: Building CNN in Keras (R)

```r
library(keras)

# Build CNN for MNIST
model <- keras_model_sequential() %>%

  # First convolutional block
  layer_conv_2d(filters = 32, kernel_size = c(3, 3),
                activation = "relu",
                input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%

  # Second convolutional block
  layer_conv_2d(filters = 64, kernel_size = c(3, 3),
                activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%

  # Flatten and fully connected
  layer_flatten() %>%
```

Slide 75: CNN Summary Output

```
## Model: sequential

## -----------------------------------------------------------

## Layer (type)                Output Shape         Param #

## ===========================================================

## conv2d (Conv2D)             (None, 26, 26, 32)   320

## max_pooling2d (MaxPool2D) (None, 13, 13, 32)   0

## conv2d_1 (Conv2D)           (None, 11, 11, 64)   18,496

## max_pooling2d_1             (None, 5, 5, 64)     0

## flatten (Flatten)           (None, 1600)         0

## dense (Dense)               (None, 128)          204,928

## dropout (Dropout)           (None, 128)          0
```

Slide 76: What Do CNN Filters Learn?

Hierarchical Feature Learning:

- **Layer 1:** Low-level features (edges, colors, textures)
- **Layer 2:** Medium-level (corners, curves, simple shapes)
- **Layer 3:** High-level (object parts, patterns)
- **Final Layers:** Complete objects

This hierarchy emerges automatically from training!

Slide 77: Visualizing Learned Filters

```r
# Extract first convolutional layer weights
layer1_weights <- get_weights(model$layers[[1]])
filters <- layer1_weights[[1]]  # Shape: (3, 3, 1, 32)

# Visualize first 16 filters
par(mfrow = c(4, 4), mar = c(0.5, 0.5, 0.5, 0.5))
for (i in 1:16) {
  filter <- filters[, , 1, i]
  image(t(filter[nrow(filter):1, ]),
        col = gray.colors(20),
        axes = FALSE)
}
```

Typical patterns: Edge detectors at various angles, blob detectors

Slide 78: Data Augmentation for CNNs

Problem: Deep CNNs need lots of data

Solution: Artificially expand dataset with transformations

Common Augmentations:

- Random rotations (±15 degrees)
- Random horizontal flips
- Random crops and zooms
- Brightness/contrast adjustments
- Random shifts

Benefit: Network learns invariance to these transformations

Slide 79: Data Augmentation in Keras

```
# Create data augmentation generator
datagen <- image_data_generator(
  rotation_range = 15,
  width_shift_range = 0.1,
  height_shift_range = 0.1,
  horizontal_flip = TRUE,
  zoom_range = 0.1,
  fill_mode = "nearest"
)

# Fit generator to training data
datagen %>% fit_image_data_generator(x_train)

# Train with augmented data
model %>% fit_generator(
  datagen %>% flow_images_from_data(
    x_train, y_train,
    batch_size = 32,
```

Slide 80: Transfer Learning - Standing on Giants' Shoulders

Problem: Training deep CNNs from scratch requires: - Millions of labeled images - Days/weeks of GPU time - Specialized expertise

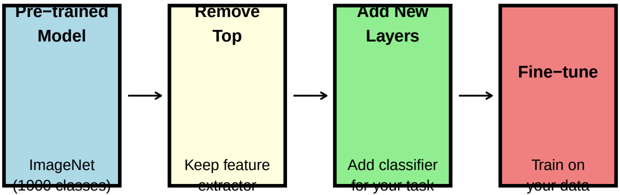
Solution: Transfer Learning

1. Take pre-trained model (trained on ImageNet)
2. Remove final layer
3. Add new layer for your task
4. Fine-tune on your data

Intuition: Low-level features (edges, textures) are universal!

Slide 81: Transfer Learning Visualization

Transfer Learning Workflow



Pre−trained Model

ImageNet (1000 classes)

Remove Top

Keep feature extractor

Add New Layers

Add classifier for your task

Fine−tune

Train on your data

Slide 82: Transfer Learning in R

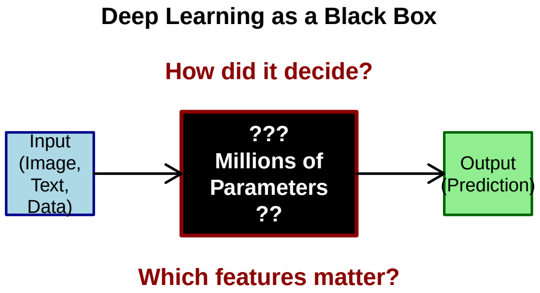
```r
# Load pre-trained VGG16 (trained on ImageNet)
base_model <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,  # Remove classification layer
  input_shape = c(224, 224, 3)
)

# Freeze base model weights (don't retrain)
freeze_weights(base_model)

# Build new model
model <- keras_model_sequential() %>%
  base_model %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(0.5) %>%
  layer_dense(units = 5, activation = "softmax")  # 5 classes
```

Section 7

Introduction to Explainable AI (XAI)

Slide 83: The Black Box Problem

We've built powerful models, but...


Deep Learning as a Black Box
How did it decide?
Input (Image, Text, Data)
??? Millions of Parameters ??
Output (Prediction)
Which features matter?

Slide 84: Why Do We Need XAI?

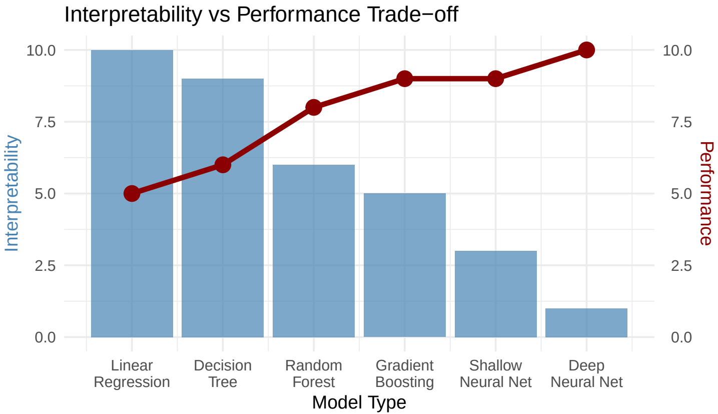
Critical Applications:

1. **Healthcare:** “Why did the model diagnose cancer?”
2. **Finance:** “Why was the loan rejected?” (legal requirement)
3. **Criminal Justice:** “Why higher recidivism risk?”
4. **Autonomous Vehicles:** “Why did it brake suddenly?”

Requirements:

- **Trust:** Users need to understand decisions
- **Debugging:** Find model errors and biases
- **Compliance:** Regulations (GDPR, etc.)
- **Scientific Discovery:** Learn from model insights

Slide 85: The Interpretability Spectrum



Blue = Interpretability, Red = Performance

Slide 86: Two Types of Interpretability

1. **Intrinsic (Model-Specific):**
   - Built into model architecture
   - Examples: Linear regression coefficients, decision tree rules
   - **Deep Learning:** Attention mechanisms, prototype networks

2. **Post-Hoc (Model-Agnostic):**
   - Applied after training
   - Works with any model
   - Examples: LIME, SHAP, Feature Importance
   - **Focus for Deep Learning**

Slide 87: Local vs Global Explanations

Global Explanations:

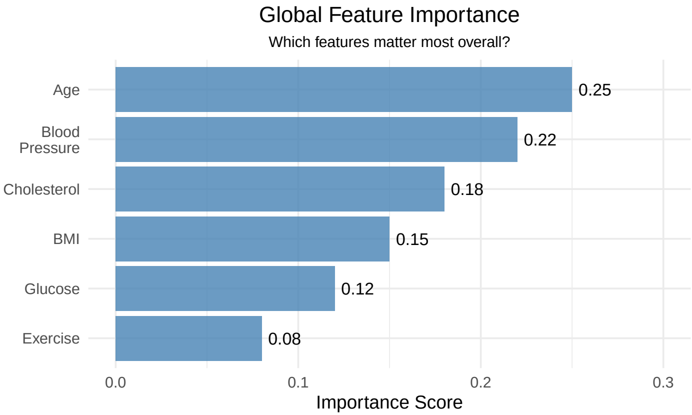
- “Overall, what does the model rely on?”
- Feature importance across all predictions
- Example: “Age is the most important feature”

Local Explanations:

- “Why this specific prediction?”
- Feature importance for one instance
- Example: “For THIS patient, high blood pressure drove the diagnosis”

Both are needed!

Slide 88: Feature Importance - The Starting Point



Slide 89: Permutation Feature Importance

Algorithm:

1. Train model and measure baseline performance
2. For each feature:
    - Randomly shuffle that feature's values
    - Measure new performance
    - Importance = drop in performance
3. Features causing big drops are important

Advantages:

- Model-agnostic (works with any model)
- Easy to understand
- Captures non-linear relationships

Slide 90: Permutation Importance in R

```r
# Permutation importance function
permutation_importance <- function(model, X, y,
                                   metric = "accuracy",
                                   n_repeats = 10) {
  # Baseline performance
  baseline <- mean(predict(model, X) == y)

  importance <- numeric(ncol(X))
  names(importance) <- colnames(X)

  for (feat in 1:ncol(X)) {
    scores <- numeric(n_repeats)

    for (i in 1:n_repeats) {
      # Shuffle feature
      X_permuted <- X
      X_permuted[, feat] <- sample(X_permuted[, feat])
```

Section 8

Advanced XAI Techniques

Slide 91: LIME - Local Interpretable Model-agnostic Explanations

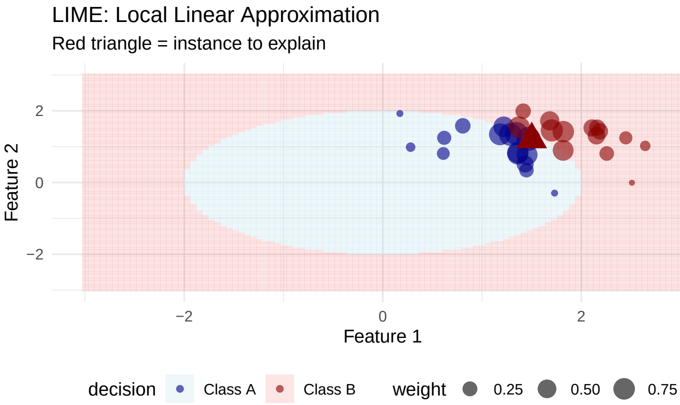
Core Idea: Explain individual predictions by approximating the complex model locally with a simple, interpretable model.

The LIME Process:

1. **Select** an instance to explain
2. **Perturb** the instance (create similar examples)
3. **Get predictions** from black-box model on perturbed samples
4. **Fit** a simple model (e.g., linear) weighted by proximity
5. **Explain** using the simple model's coefficients

Intuition: Complex model may be linear in a small neighborhood around one instance

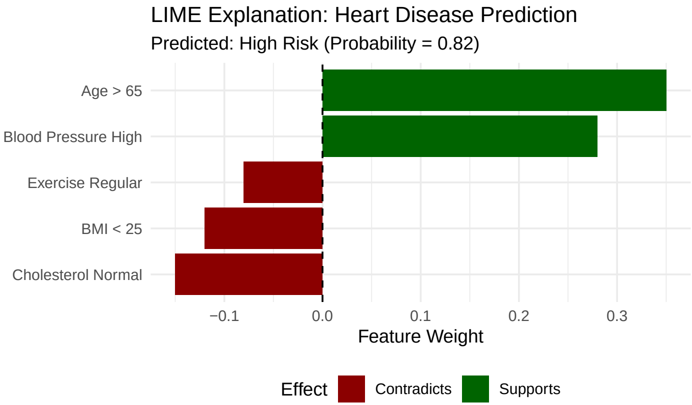
Slide 92: LIME Algorithm Visualization



Slide 93: LIME for Tabular Data in R

```r
library(lime)

# Assuming you have a trained model and data
# model <- trained_classification_model
# x_train <- training_data
# x_test <- test_data

# Create explainer
explainer <- lime(x_train, model)

# Explain a single prediction
explanation <- explain(
  x_test[1, ],              # Instance to explain
  explainer,
  n_features = 5,           # Top 5 features
  n_permutations = 1000  # Perturbed samples
)
```

Slide 94: LIME Results Interpretation



Interpretation: Age and blood pressure strongly support high-risk prediction

Slide 95: SHAP - SHapley Additive exPlanations

Foundation: Game theory (Shapley values from cooperative games)

Key Idea: Each feature is a "player" contributing to the prediction. SHAP values = fair distribution of the prediction among features.

SHAP Value Formula:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f(S \cup \{i\}) - f(S)]$$

In plain English: Average marginal contribution of feature i across all possible feature combinations

Slide 96: SHAP Properties - Why It's Special

Desirable Properties (Axioms):

1. **Local Accuracy:** Explanations sum to actual prediction

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i$$

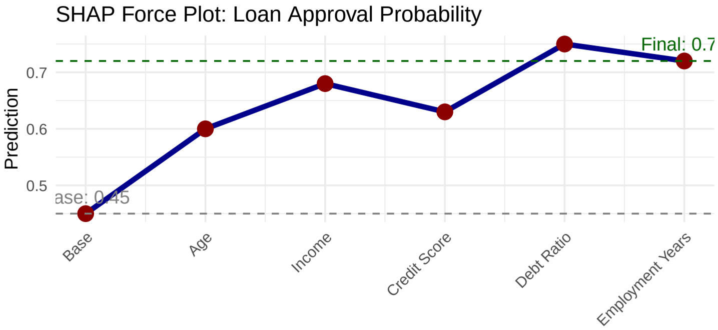
2. **Missingness:** Missing features have zero impact
3. **Consistency:** If a feature's contribution increases, its SHAP value shouldn't decrease

SHAP is the ONLY explanation method satisfying all three!

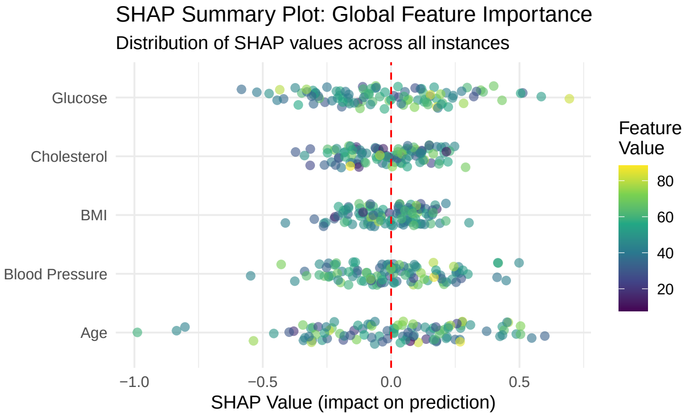
Slide 97: SHAP in R - Basic Implementation

```r
library(shapr)

# Prepare data and model
# model <- your_trained_model
# x_train <- training_features
# x_explain <- test_instance

# Create explainer
explainer <- shapr(x_train, model)

# Compute SHAP values
shap_values <- explain(
  x_explain,
  approach = "empirical",  # or "gaussian", "ctree"
  explainer = explainer,
  prediction_zero = mean(predictions_train)
)
```

Slide 98: SHAP Force Plot Concept



Slide 99: SHAP Summary Plot - Global Importance



Each point = one instance. Position = impact magnitude

Slide 100: SHAP vs LIME Comparison

Aspect	LIME	SHAP
Theoretical Foundation	Sparse linear model	Game theory (Shapley values)
Computation	Faster	Slower (exact: exponential)
Consistency	No guarantees	Mathematically consistent
Global + Local	Local only	Both
Additivity	Approximate	Exact

Recommendation: - **LIME:** Quick exploration, high-dimensional data - **SHAP:** Rigorous analysis, when computational resources allow

Slide 101: Saliency Maps for Images

Goal: Which pixels matter most for the prediction?

Gradient-based Saliency (Simonyan et al., 2014):

$$S_c(x) = \left| \frac{\partial f_c(x)}{\partial x} \right|$$

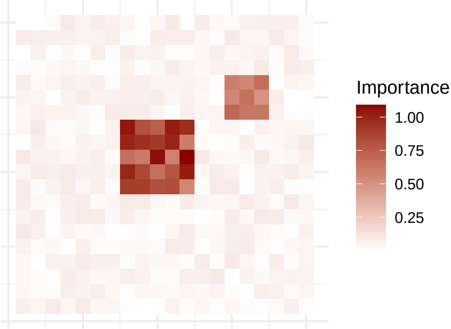
where $f_c(x)$ is the class score for class c

Intuition: If changing a pixel affects prediction a lot, that pixel is important

Slide 102: Computing Saliency Maps in R

```r
library(keras)

# Load pre-trained model
model <- application_vgg16(weights = "imagenet")

# Load and preprocess image
img <- image_load("path/to/image.jpg",
                  target_size = c(224, 224))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

# Get predicted class
preds <- model %>% predict(x)
class_idx <- which.max(preds[1, ])

# Compute gradients
with(tf$GradientTape() %as% tape, {
```

Slide 103: Saliency Map Visualization Concept



Saliency Map: Which Pixels Matter?

Brighter = More important for prediction

Slide 104: Grad-CAM - Class Activation Mapping

Improvement over basic saliency: Shows class-discriminative regions

Grad-CAM Formula:

$$L^c_{Grad-CAM} = ReLU\left(\sum_k \alpha^c_k A^k\right)$$

where: - $A^k =$ activation maps from last conv layer - $\alpha^c_k =$ importance weights (from gradients)

Advantage: Produces visual explanations showing “where the model looks”

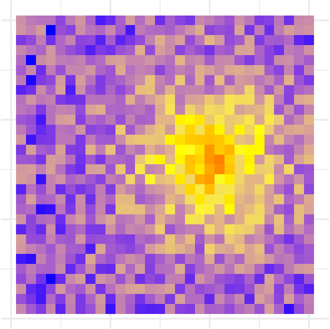
Slide 105: Grad-CAM Implementation in R

```r
# Grad-CAM implementation
gradcam <- function(model, img_array, class_idx,
                    layer_name = "block5_conv3") {

  # Get last conv layer
  grad_model <- keras_model(
    inputs = model$input,
    outputs = list(
      model$get_layer(layer_name)$output,
      model$output
    )
  )

  # Compute gradients
  with(tf$GradientTape() %as% tape, {
    conv_outputs <- grad_model(img_array)
    predictions <- conv_outputs[[2]]
```

Slide 106: Grad-CAM Visualization Example

Grad−CAM: Where the Model Looks

Red = High attention for this prediction



Slide 107: Integrated Gradients

Problem with basic gradients: Can be noisy and miss important features

Integrated Gradients (Sundararajan et al., 2017):

$$IG_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

Key Idea: - Start from baseline input x' (e.g., all zeros) - Gradually interpolate to actual input x - Accumulate gradients along this path

Slide 108: Integrated Gradients Properties

Advantages:

1. **Completeness:** Attribution scores sum to difference from baseline

$$\sum_i IG_i(x) = f(x) - f(x')$$

2. **Sensitivity:** Non-zero gradient when feature matters
3. **Implementation Invariance:** Same results for functionally equivalent networks

Use Case: More reliable than vanilla gradients for feature attribution

Slide 109: Attention Mechanisms as Built-in XAI

Attention Weights = Interpretability

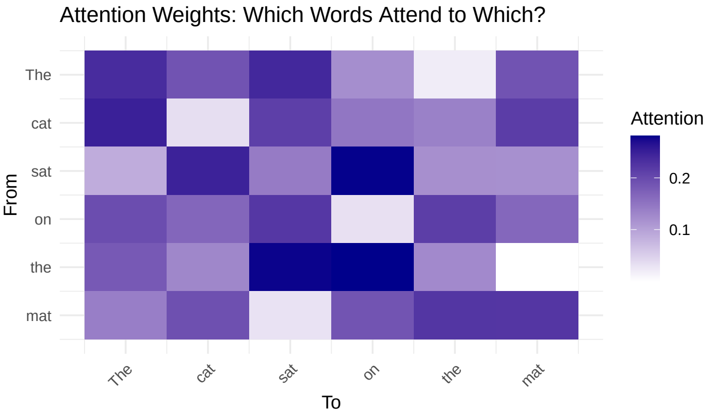
For Transformer models, attention scores show which parts of input the model focuses on.

Self-Attention Formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Attention weights (from softmax) directly indicate importance!

Slide 110: Visualizing Attention Weights



Slide 111: Partial Dependence Plots (PDP)

Goal: Show marginal effect of one or two features on predictions

PDP Formula:

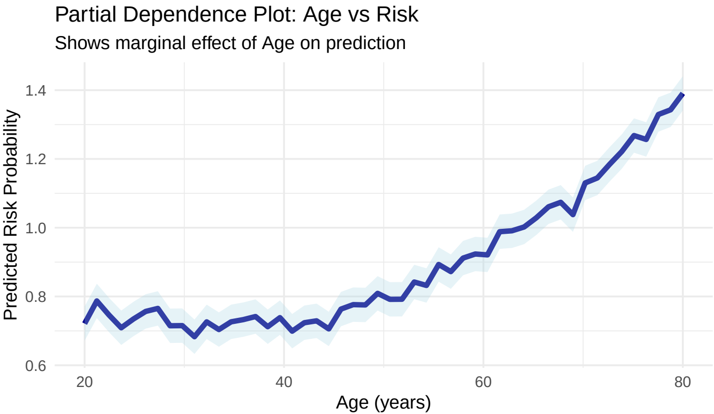
$$\hat{f}_S(x_S) = E_{X_C}[\hat{f}(x_S, X_C)] = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)})$$

Process: 1. Choose feature(s) of interest 2. For each value of feature, average predictions across all other feature combinations 3. Plot the average prediction

Slide 112: Computing PDP in R

```r
library(pdp)

# Assuming you have a trained model
# model <- your_trained_model
# data <- your_training_data

# Single feature PDP
partial_age <- partial(
  model,
  pred.var = "Age",
  train = data,
  plot = TRUE
)

# Two-feature interaction PDP
partial_age_bp <- partial(
  model,
```

Slide 113: PDP Example Visualization



Interpretation: Risk increases with age, especially after 50

Slide 114: Individual Conditional Expectation (ICE)

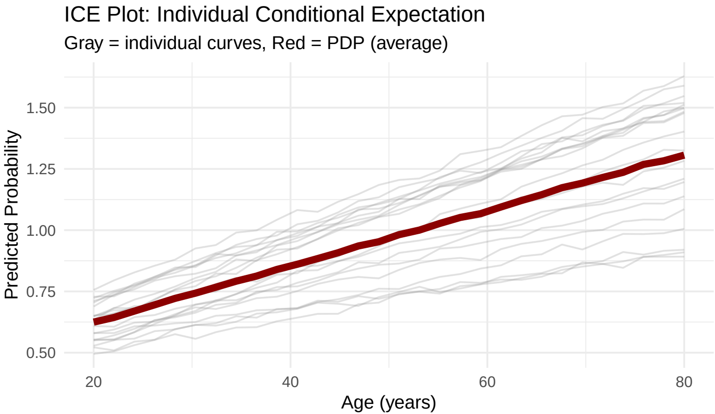
Problem with PDP: Hides heterogeneous effects (different patterns for different instances)

ICE Solution: Plot prediction vs feature for EACH instance separately

$$\hat{f}_i(x_S) = \hat{f}(x_S, x_C^{(i)})$$

PDP = Average of all ICE curves

Slide 115: ICE Plot Visualization



Shows variation: Some instances more sensitive to age than others

Slide 116: Counterfactual Explanations

Question: "What is the smallest change to inputs that would flip the prediction?"

Example: - **Current:** Loan rejected (income=$45k, debt=$30k) - **Counterfactual:** Loan approved if income=$52k OR debt=$22k

Value: Actionable insights—tells users what to change

Slide 117: Finding Counterfactuals

Optimization Problem:

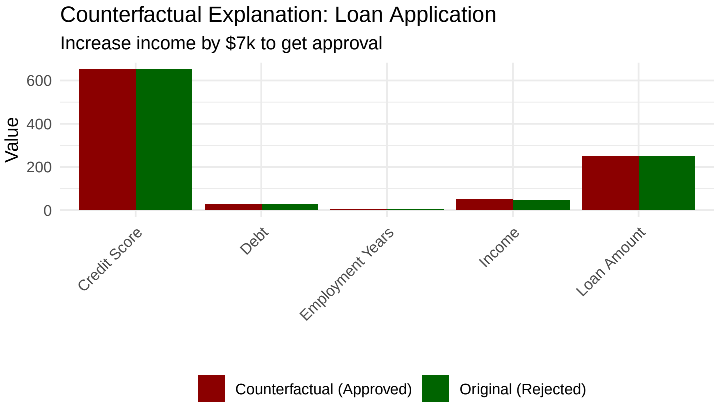
$$\min_{x'} \text{distance}(x, x') \quad \text{s.t.} \quad f(x') \neq f(x)$$

Constraints: - Feasible changes only (can't change age, race, etc.) - Realistic ranges - Sparse changes (modify few features)

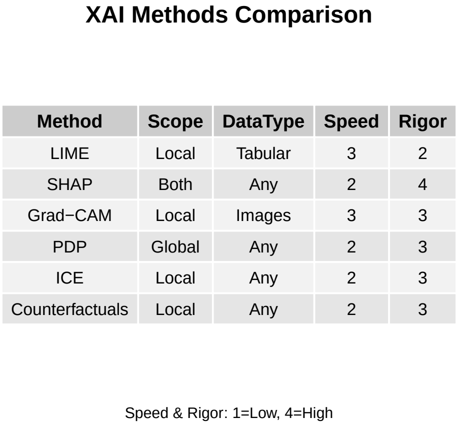
Slide 118: Counterfactual Example in R

```r
# Conceptual counterfactual generation
find_counterfactual <- function(model, instance,
                                target_class,
                                mutable_features) {

  # Initialize with current instance
  counterfactual <- instance

  # Iteratively adjust mutable features
  for (iter in 1:100) {
    prediction <- predict(model, counterfactual)

    if (prediction == target_class) {
      return(counterfactual)
    }

    # Adjust features toward target
```

Slide 119: Counterfactual Visualization



Slide 120: XAI Methods Summary and Selection Guide



XAI Methods Comparison

Method	Scope	DataType	Speed	Rigor
LIME	Local	Tabular	3	2
SHAP	Both	Any	2	4
Grad−CAM	Local	Images	3	3
PDP	Global	Any	2	3
ICE	Local	Any	2	3
Counterfactuals	Local	Any	2	3

Speed & Rigor: 1=Low, 4=High

Selection Guide: - **Need guarantees?** → SHAP - **Fast exploration?** → LIME - **Images?** → Grad-CAM - **Global trends?** → PDP - **Actionable advice?** → Counterfactuals # Practical XAI Implementation

Slide 121: XAI Workflow - From Model to Explanation

Standard XAI Pipeline:

1. **Train Model:** Build and validate your deep learning model
2. **Select XAI Method:** Based on task and requirements
3. **Generate Explanations:** Apply chosen method(s)
4. **Validate Explanations:** Do they make sense?
5. **Communicate Results:** Present to stakeholders
6. **Iterate:** Refine based on feedback

Key Principle: XAI is not one-time—it's an ongoing process

Slide 122: Complete XAI Example - Setup

```
library(keras)
library(lime)
library(ggplot2)
library(dplyr)

# Load dataset (using iris as example)
data(iris)
set.seed(123)

# Prepare data
train_idx <- sample(1:nrow(iris), 0.8 * nrow(iris))
x_train <- as.matrix(iris[train_idx, 1:4])
y_train <- as.integer(iris[train_idx, 5]) - 1
x_test <- as.matrix(iris[-train_idx, 1:4])
y_test <- as.integer(iris[-train_idx, 5]) - 1

# One-hot encode labels
```

Slide 123: Build and Train Model

```r
# Build neural network
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = c(4)) %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 8, activation = "relu") %>%
  layer_dense(units = 3, activation = "softmax")

# Compile
model %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

# Train
history <- model %>% fit(
```

Slide 124: Apply LIME Explanation

```
# Create LIME explainer
explainer <- lime(
  x = as.data.frame(x_train),
  model = model,
  bin_continuous = TRUE,
  n_bins = 4
)

# Explain a single prediction
instance_to_explain <- 1
explanation <- explain(
  x = as.data.frame(x_test[instance_to_explain, , drop = FALSE
  explainer = explainer,
  n_labels = 1,
  n_features = 4,
  n_permutations = 1000
)
```

Slide 125: Custom Feature Importance Function

```r
# Compute feature importance by permutation
compute_feature_importance <- function(model, X, y,
                                       n_repeats = 10) {
  # Baseline accuracy
  preds_baseline <- model %>% predict(X) %>% k_argmax() %>%
                    as.integer()
  baseline_acc <- mean(preds_baseline == y)

  # Initialize importance scores
  importance <- numeric(ncol(X))
  names(importance) <- colnames(X)

  for (feat_idx in 1:ncol(X)) {
    scores <- numeric(n_repeats)

    for (rep in 1:n_repeats) {
      # Permute feature
```

Slide 126: Visualize Feature Importance

```r
# Compute importance
importance_scores <- compute_feature_importance(
  model, x_test, y_test, n_repeats = 20
)

# Create visualization
importance_df <- data.frame(
  Feature = names(importance_scores),
  Importance = importance_scores
)

ggplot(importance_df, aes(x = reorder(Feature, Importance),
                          y = Importance)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Global Feature Importance",
       subtitle = "Permutation-based importance scores",
       x = "", y = "Importance (Accuracy Drop)") +
```

Slide 127: Case Study 1 - Medical Diagnosis

Scenario: Predicting diabetes risk from patient data

Dataset: 768 patients, 8 features (glucose, BMI, age, etc.)

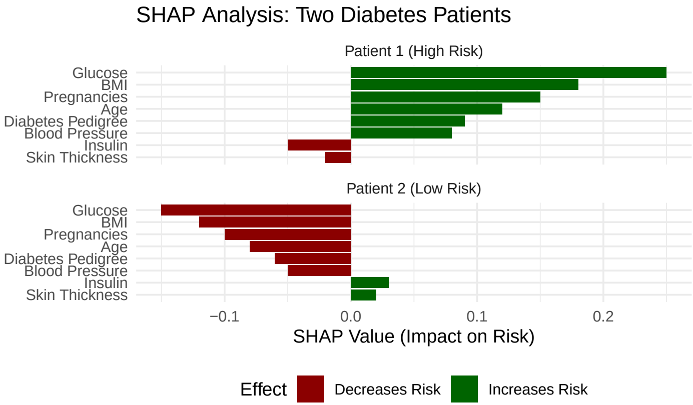
Model: 3-layer neural network, 85% accuracy

XAI Requirements: - **Doctors need:** Feature importance for individual patients - **Compliance:** Explain rejected insurance claims - **Research:** Discover unexpected risk factors

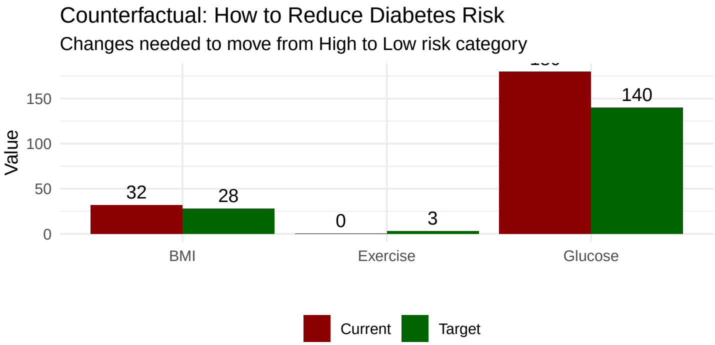
Slide 128: Medical Diagnosis - Model Architecture

```
# Load diabetes dataset
# data <- read.csv("diabetes.csv")

# Build medical diagnosis model
medical_model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu",
              input_shape = c(8)) %>%
  layer_batch_normalization() %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_batch_normalization() %>%
  layer_dropout(0.3) %>%
  layer_dense(units = 1, activation = "sigmoid")

medical_model %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  loss = "binary_crossentropy",
  metrics = c("accuracy", "AUC"))
```

Slide 129: Medical Diagnosis - SHAP Analysis



Slide 130: Medical Diagnosis - Counterfactual Advice


Counterfactual: How to Reduce Diabetes Risk

Changes needed to move from High to Low risk category

Actionable advice: Lower glucose, reduce BMI, exercise 3x/week

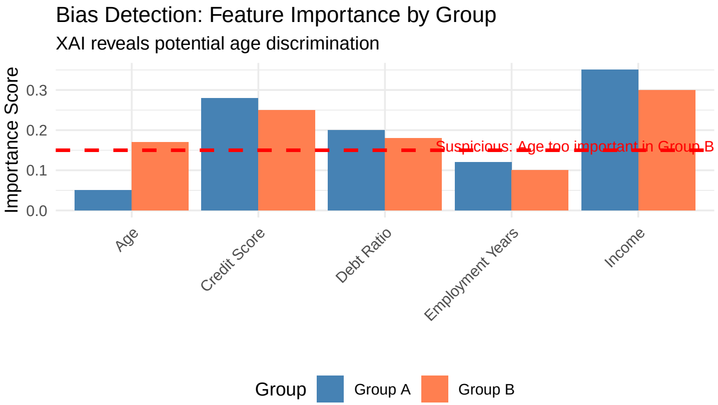
Slide 131: Case Study 2 - Credit Scoring

Scenario: Loan approval prediction

Challenges: - **Legal requirement:** Explain rejections (Fair Credit Reporting Act) - **Bias detection:** Ensure no discrimination - **Customer trust:** Transparent decisions

XAI Solution: LIME + Counterfactuals for each rejected application

Slide 132: Credit Scoring - Bias Detection with XAI



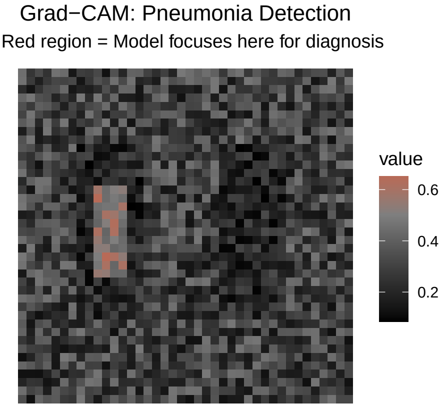
Slide 133: Case Study 3 - Image Classification

Scenario: Medical image analysis (X-ray classification)

Task: Detect pneumonia from chest X-rays

XAI Methods: - **Grad-CAM:** Show which regions indicate disease - **Saliency Maps:** Highlight important pixels - **Validation:** Radiologist reviews explanations

Slide 134: Image XAI - Grad-CAM for Pneumonia Detection



Clinical validation: Heatmap matches known infection location

Slide 135: Validating XAI Explanations

How do we know explanations are correct?

Validation Strategies:

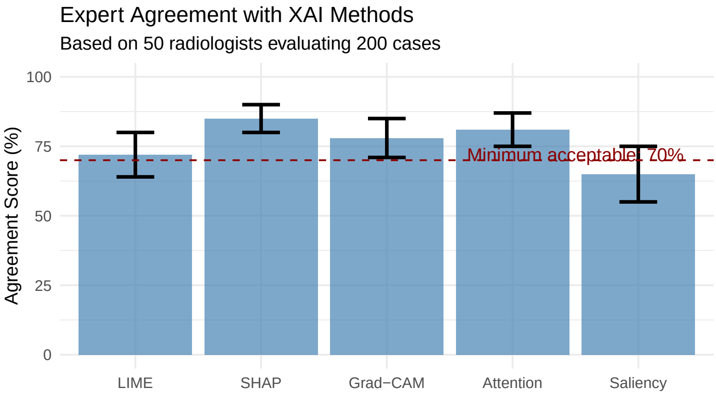
1. **Sanity Checks:** Do explanations change when model changes?
2. **Human Agreement:** Do experts agree with explanations?
3. **Perturbation Tests:** Remove important features → prediction changes?
4. **Known Ground Truth:** Test on synthetic data with known answers

Warning: XAI can be misleading if not validated!

Slide 136: Sanity Check - Model Randomization Test

```r
# Sanity check: Explanations should change with random model
sanity_check <- function(model, explainer, instance,
                         method = "lime") {

  # Get explanation from trained model
  explanation_trained <- explain(instance, explainer)
  importance_trained <- explanation_trained$feature_weight

  # Randomize model weights
  model_random <- model
  for (layer in model_random$layers) {
    if (length(layer$get_weights()) > 0) {
      weights <- layer$get_weights()
      weights <- lapply(weights, function(w) {
        array(rnorm(length(w)), dim = dim(w))
      })
      layer$set_weights(weights)
    }
```

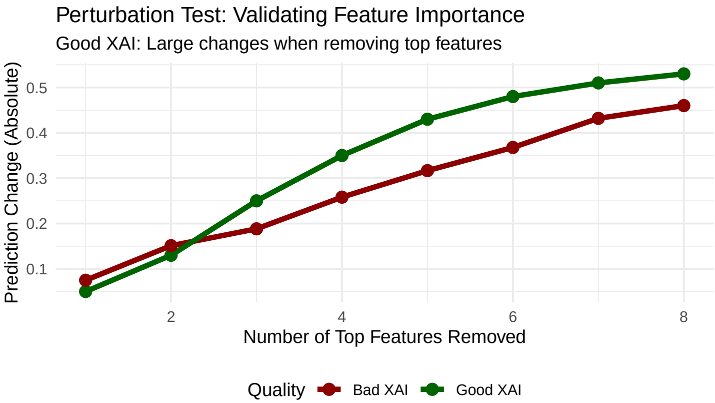
Slide 137: Human Evaluation of Explanations



Slide 138: Perturbation Test - Feature Removal

```r
# Test: Remove top features, prediction should change
perturbation_test <- function(model, instance,
                              feature_importance) {

  # Get baseline prediction
  pred_baseline <- model %>% predict(instance)

  # Sort features by importance
  features_sorted <- names(sort(feature_importance,
                                decreasing = TRUE))

  results <- data.frame(
    n_removed = integer(),
    prediction_change = numeric()
  )

  # Progressively remove top features
  for (i in 1:length(features_sorted)) {
```

Slide 139: Perturbation Test Visualization



Slide 140: Debugging Models with XAI

XAI reveals model failures:

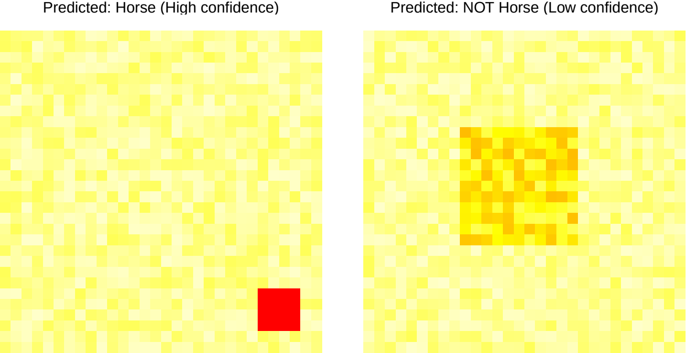
1. **Clever Hans:** Model uses spurious correlations
2. **Shortcut Learning:** Relies on dataset artifacts
3. **Bias:** Discriminates based on protected attributes
4. **Brittleness:** Small input changes cause large output changes

Example: Image classifier using image borders instead of objects

Slide 141: Detecting Clever Hans Behavior

Clever Hans Detection: Model Uses Watermark!

Red = Model attention. Should focus on center, not corner



Predicted: Horse (High confidence)

Predicted: NOT Horse (Low confidence)

Slide 142: XAI for Model Debugging - Action Plan

When XAI reveals problems:

1. **Identify issue:** What's the model learning?
2. **Check data:** Is there a dataset bias?
3. **Data augmentation:** Add examples without spurious features
4. **Architecture change:** Add regularization, change receptive field
5. **Re-validate:** Use XAI again to confirm fix

Iterative process: XAI → Debug → Retrain → XAI → ...

Slide 143: Communicating XAI to Non-Technical Stakeholders

Different audiences need different explanations:

Audience	What They Need	Best Method
Data Scientists	Technical details, metrics	SHAP values, equations
Domain Experts	Feature relevance	Feature importance plots
Executives	Business impact	Counterfactuals, summary stats
End Users	Simple reason for decision	LIME with 2-3 features
Regulators	Compliance proof	Audit trails, documentation

Slide 144: Effective XAI Visualization Principles

Design Guidelines:

1. **Simplicity:** Show 3-5 most important features
2. **Color:** Red = increases risk, Green = decreases risk
3. **Context:** Include prediction confidence
4. **Comparisons:** Show baseline or alternative scenarios
5. **Actionability:** Highlight what can be changed

Anti-pattern: Overwhelming users with all features and technical jargon

Slide 145: XAI Dashboard Example

Loan Decision Dashboard

APPROVED

Confidence: 87%

Key Factors:

Feature	Impact	Value
Credit Score	+15%	720
Income	+12%	$65k
Debt Ratio	−8%	0.42

To improve approval odds:

- Increase income by $5k
- Reduce debt ratio to 0.35

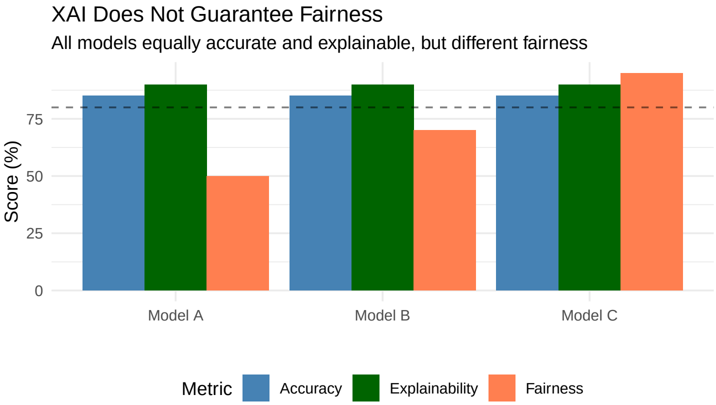
Slide 146: Ethical Considerations in XAI

Key Ethical Issues:

1. **Misleading Explanations:** XAI can give false confidence
2. **Gaming the System:** Users manipulate features to change predictions
3. **Privacy:** Explanations may reveal training data
4. **Fairness:** Explanations don't guarantee fair decisions
5. **Responsibility:** Who's accountable when XAI is wrong?

Critical principle: Explanations   Justifications

Slide 147: XAI and Fairness



Lesson: Always evaluate fairness separately from explainability

Slide 148: Best Practices for Responsible XAI

10 Best Practices:

1. **Multiple methods:** Use 2+ XAI techniques
2. **Validation:** Always validate explanations
3. **Documentation:** Record XAI methodology
4. **Uncertainty:** Communicate confidence in explanations
5. **Simplicity:** Prefer simpler explanations when possible
6. **Human-in-loop:** Expert review of explanations
7. **Bias testing:** Check for discrimination
8. **Update regularly:** Re-explain as model changes
9. **Privacy:** Protect sensitive information in explanations
10. **Humility:** Acknowledge XAI limitations

Slide 149: XAI Limitations and Future Directions

Current Limitations:

- **Computational cost:** SHAP exponential in features
- **Instability:** Small input changes → different explanations
- **Local only:** Most methods explain one instance
- **No guarantees:** Explanations may be misleading

Future Research:

- Causal explanations (beyond correlations)
- Interactive explanations (user can ask “what if”)
- Certified explanations (with guarantees)
- Explanations for generation and RL

Slide 150: Putting It All Together - XAI Checklist

Before Deployment:

- □ Model trained and validated
- □ At least 2 XAI methods applied
- □ Explanations validated by domain experts
- □ Sanity checks passed
- □ Bias and fairness tested
- □ Documentation complete
- □ Stakeholder communication plan
- □ Monitoring system for ongoing explanations

Remember: XAI is a journey, not a destination