Deep Learning

Nayel Bettache

Motivation for Neural Networks

- **Why Neural Networks?**
  - Solve complex problems with nonlinear relationships.
  - Handle large-scale data in fields like computer vision, NLP, and bioinformatics.
- **Applications:**
  - Image Recognition (e.g., autonomous vehicles, facial recognition).
  - Language Translation (e.g., Google Translate).
  - Predictive Analytics (e.g., stock market predictions, medical diagnostics).
- **Enablers of Deep Learning:**
  - Large datasets available online.
  - Powerful GPUs and TPUs for training models.
  - Improved algorithms (e.g., backpropagation, SGD).

Intuition for Neural Networks

- **Biological Analogy:**
  - Neurons process information and send signals to others.
  - Connections between neurons (synapses) determine how information is passed.
- **Neural Network Analogy:**
  - Artificial neurons (units) process inputs and produce activations.
  - Connections between units (weights) determine how data flows through the network.
- **Key Insight:**
  - Networks learn hierarchical patterns: low-level features (edges) to high-level patterns (faces, objects).

Intuition for Neural Networks

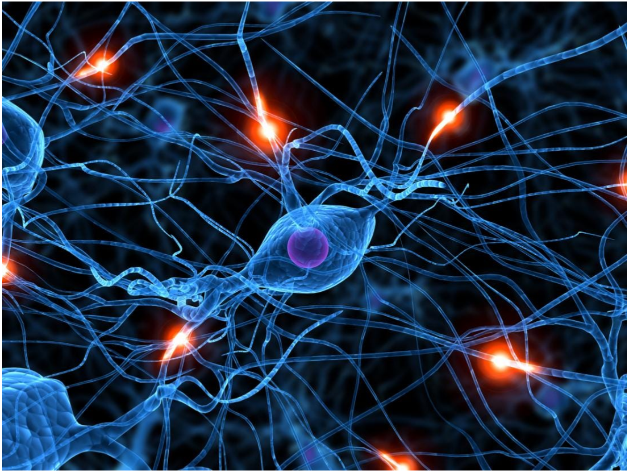

Figure: Neural networks mimic information flow in the brain.

Single Layer Neural Network: Concepts

- **Setting:** Predict a response variable Y using predictors $X = (X_1, \ldots, X_p)$.
- **Goal:** Learn a nonlinear function $f(X)$ to predict Y.
- **Hidden Layer Activations:**

$$A_k = h_k(X) = g(w_{k0} + w_{k1}X_1 + \cdots + w_{kp}X_p), \quad k = 1, \ldots, K,$$

  where $g(x)$ is a nonlinear activation function.
- **Activation Functions:**
  - Sigmoid: $g(x) = \frac{e^x}{1+e^x}$.
  - ReLU: $g(x) = \max(0, x)$.
- **Neural Network Model:**

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + w_{k1}X_1 + \cdots + w_{kp}X_p).$$

Single Layer Neural Network

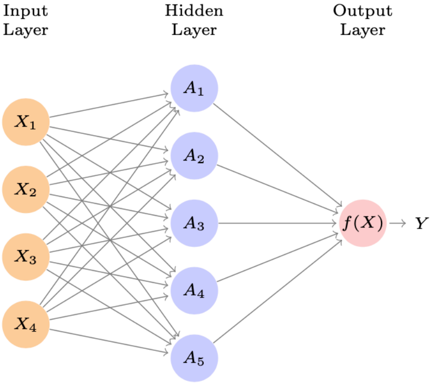

FIGURE 10.1. *Neural network with a single hidden layer. The hidden layer computes activations* $A_k = h_k(X)$ *that are nonlinear transformations of linear combinations of the inputs* $X_1, X_2, \ldots, X_p$. *Hence these* A_k *are not directly observed. The functions* $h_k(\cdot)$ *are not fixed in advance, but are learned during the training of the network. The output layer is a linear model that uses these activations* A_k *as inputs, resulting in a function* $f(X)$.

Multilayer Perceptron (MLP)

Goal: Learn complex, nonlinear relationships between inputs $X = (X_1, \dots, X_p)$ and target outputs Y.

Extend the single-layer model by introducing multiple hidden layers to capture hierarchical features.

- **Architecture:**
  - Input Layer: Accepts input features X.
  - Hidden Layers: Each layer transforms inputs through:

$$A_k^{(l)} = g\left(w_{k0}^{(l)} + \sum_{j=1}^{n^{(l-1)}} w_{kj}^{(l)} A_j^{(l-1)}\right), \quad l = 1, \dots, L,$$

    where $g(x)$ is an activation function (e.g., ReLU, Sigmoid).
  - Output Layer: Produces final predictions, $f(X)$.
- **Universal Approximation Theorem:** An MLP with sufficient hidden units can approximate any continuous function on a compact domain.
- Requires nonlinear activation functions (e.g., ReLU, Sigmoid) to model complex patterns.

MLP

- **Strengths:**
  - Capable of learning hierarchical representations of data.
  - Flexible for a wide range of applications: regression, classification, image recognition, etc.
- **Challenges:**
  - Requires significant computational power for deep architectures.
  - Sensitive to hyperparameters (e.g., learning rate, layer sizes, regularization).
- **Best Practices:**
  - Use regularization (e.g., L2, dropout) to prevent overfitting.
  - Employ batch normalization and optimization techniques like Adam for efficient training.
  - Tune hyperparameters systematically using grid search or Bayesian optimization.

Multilayer Perceptron

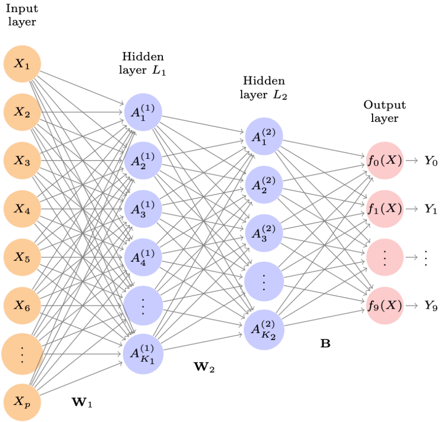

FIGURE 10.4. *Neural network diagram with two hidden layers and multiple outputs, suitable for the* MNIST *handwritten-digit problem. The input layer has* $p = 784$ *units, the two hidden layers* $K_1 = 256$ *and* $K_2 = 128$ *units respectively, and the output layer* 10 *units. Along with intercepts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

MLP - Performance

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.*

Fitting a Neural Network

- **Objective:** Given data $(x_i, y_i),\ i = 1, \ldots, n$, minimize the loss function:

$$\min_{w,\beta} \sum_{i=1}^{n} (y_i - f(x_i))^2,$$

where:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + w_{k1}X_1 + \cdots + w_{kp}X_p).$$

Fitting a Neural Network

- **Optimization Methods:**
  - Stochastic Gradient Descent (SGD).
  - Regularization: Minimize $RSS + \lambda \sum_j \theta_j^2$, where $\theta = (w, \beta)$.
- **Advanced Techniques:**
  - **Dropout:** Randomly drop units during training to prevent overfitting.
  - **Tuning:** Optimize parameters like:
    - Number of layers L.
    - Units per layer K.
    - Regularization parameter λ.
    - Learning rate in gradient descent.
    - Dropout probability.

Backpropagation: How Neural Networks Learn

- **Purpose:**
  - Efficiently compute gradients of the loss function with respect to all weights in the network.
  - Use these gradients to update weights via optimization (e.g., Gradient Descent).
- **Key Insight:**
  - Backpropagation uses the chain rule of calculus to efficiently compute gradients across multiple layers.
- **Challenges:**
  - Vanishing gradients: Gradients become very small in deep networks (mitigated by ReLU activation).
  - Computational cost: High-dimensional networks require significant resources.

Backpropagation: How Neural Networks Learn

- **Process:**
  1. **Forward Pass:**
     - Compute activations layer by layer to produce output $\hat{y}$.
     - Calculate the loss $\mathcal{L}(y, \hat{y})$ (e.g., MSE, cross-entropy).
  2. **Backward Pass:**
     - Start at the output layer and compute the gradient of the loss w.r.t. outputs:
       $$\frac{\partial \mathcal{L}}{\partial \hat{y}}$$
     - Propagate gradients backward through each layer using the chain rule:
       $$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial A} \cdot \frac{\partial A}{\partial w}$$
       where A represents activations.
  3. **Weight Update:**
     - Adjust weights using gradients:
       $$w \leftarrow w - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$
       where η is the learning rate.

Key Hyperparameters in Neural Networks

- **Learning Rate:**
  - Determines how fast weights are updated during training.
- **Batch Size:**
  - Number of samples processed before updating weights.
- **Number of Layers and Units:**
  - More layers/units increase model capacity but risk overfitting.
- **Regularization Parameter (λ):**
  - Controls penalty for large weights to avoid overfitting.
- **Dropout Probability:**
  - Fraction of neurons randomly dropped during training to improve generalization.

Tools for Deep Learning

- **Popular Frameworks:**
  - **TensorFlow:** High-performance library for scalable deep learning.
  - **PyTorch:** Flexible, dynamic computation graphs, widely used in research.
  - **Keras:** Simplified high-level API for neural networks.
- **Other Tools:**
  - Scikit-learn (for preprocessing).
  - Jupyter Notebooks (for experimentation).
- **Best Practices:**
  - Use GPU/TPU for faster training.
  - Leverage pre-trained models when appropriate.