Deep Learning: Unit 2 notes

1. Introduction to Neural Networks

- **Neural Networks (NN)** are computational models inspired by the human brain, used for tasks such as classification, regression, and pattern recognition.
- A **Feedforward Neural Network (FNN)** is a type of artificial neural network where information moves in one direction, from input to output, without any cycles or loops.
- FNNs are also known as **Multilayer Perceptrons (MLP)** when they contain multiple layers of neurons, including one or more hidden layers.

---

2. Structure of a Feedforward Neural Network

- **Input Layer**: Receives the raw input data in the form of vectors (e.g., pixel values for an image).
- **Hidden Layers**: Intermediate layers where neurons apply activation functions to the weighted input values to extract features from the data.
- **Output Layer**: Produces the final output of the network, such as class labels or predicted values.
- **Weights and Biases**:
  - Each connection between neurons has a weight, and each neuron has a bias term. These are the parameters the network learns during training.
  - The equation for the **pre-activation** is:

    $$a_i = W_i h_{i-1} + b_i$$

  - The **activation** function is then applied to the pre-activation output:

    $$h_i = g(a_i)$$

    where `g()` is the activation function like **ReLU**, **tanh**, or **sigmoid**.

---

3. Activation Functions

- **Sigmoid Function**:

$$g(x) = \frac{1}{1 + e^{-x}}$$

  It maps inputs to a range between 0 and 1, commonly used for binary classification.

- **ReLU (Rectified Linear Unit)**:

$$g(x) = \max(0, x)$$

  Popular in hidden layers due to its ability to mitigate the vanishing gradient problem and speed up training.

- **Tanh (Hyperbolic Tangent)**:

$$g(x) = \frac{2}{1 + e^{-2x}} - 1$$

  Produces outputs in the range [-1, 1], used when zero-centered outputs are desired.

- **Softmax Function**:

$$\hat{y}_j = \frac{e^{a_j}}{\sum_{k=1}^{K} e^{a_k}}$$

  It converts the outputs of the network into probabilities, mainly used in the output layer for multi-class classification.

---

4. Forward Propagation

- **Forward Propagation** involves passing the input through the network layers to compute the output.
- In forward propagation, the network computes the pre-activation `a_i` at each layer using the current weights and biases, applies the activation function `g(a_i)`, and forwards the result to the next layer.

5. Training Feedforward Neural Networks

- **Loss Function**:
  - The loss function quantifies the error between the predicted output and the true target. Common loss functions include:
    - **Mean Squared Error (MSE)** for regression tasks.
    - **Cross-Entropy Loss** for classification tasks:

$$L(\theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

    where `y` is the true label, and `\hat{y_i}` is the predicted probability.

- **Backpropagation Algorithm**:
  - Backpropagation is the key algorithm used to compute the gradient of the loss function with respect to the network's parameters (weights and biases).
  - **Gradient Descent** is an optimization algorithm used to update the weights in the direction of the steepest decrease in the loss function:

$$\theta_{t+1} = \theta_t - \eta \nabla \theta_t$$

  where `η` is the learning rate, and `\nabla \theta_t` is the gradient of the loss function with respect to the weights and biases.

6. Backpropagation in Detail

- **Chain Rule for Gradient Calculation**: The gradient of the loss function with respect to weights is computed using the chain rule:

$$\frac{\partial L(\theta)}{\partial W_{ij}} = \frac{\partial L(\theta)}{\partial h_j} \cdot \frac{\partial h_j}{\partial W_{ij}}$$

- **Gradient Descent Update**: For each weight `W`, the update rule is:

$$W_{ij} \leftarrow W_{ij} - \eta \frac{\partial L(\theta)}{\partial W_{ij}}$$

- **Two Phases of Backpropagation**:
  1. **Forward Pass:** Compute the activations and the final output of the network.
  2. **Backward Pass:** Propagate the error backward through the network and compute the gradients of the loss function with respect to each parameter.

7. Common Challenges and Solutions

- **Overfitting**:
  - The network performs well on the training data but poorly on unseen data.
  - Solutions:
    - **Regularization**: Techniques like **L2 regularization** or **dropout** can prevent overfitting by penalizing large weights or randomly deactivating neurons during training.
    - **Data Augmentation**: Increasing the diversity of the training data to improve generalization.
- **Underfitting**:
  - The model is too simple and cannot capture the underlying patterns in the data.
  - Solutions:
    - Use more complex architectures (e.g., adding more layers or neurons).
    - Increase the training time or use advanced optimizers like **Adam**.

---

8. Applications of Feedforward Neural Networks

- **Image Classification**: Identifying objects in images.
- **Speech Recognition**: Converting speech into text.
- **Financial Forecasting**: Predicting stock prices and economic trends.
- **Medical Diagnosis**: Assisting in diagnosing diseases based on medical data.
- **Natural Language Processing (NLP)**: Tasks such as sentiment analysis, machine translation, and chatbots.

---

9. Example Implementation using Python (Keras)

python
Copy code

```
from keras.models import Sequential
from keras.layers import Dense

# Create model
model = Sequential()
model.add(Dense(64, input_dim=8, activation='relu'))  # Input layer
model.add(Dense(32, activation='relu'))  # Hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer for binary
classification

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=10)
```

---

10. Conclusion

Feedforward Neural Networks are a foundational model in the field of deep learning. By using **forward propagation** to make predictions and **backpropagation** to learn from errors, they can solve a variety of tasks such as classification, regression, and more. Understanding these concepts is critical for advancing in neural network architectures, such as **Convolutional Neural Networks (CNNs)** and **Recurrent Neural Networks (RNNs)**, which are used for more complex tasks like image recognition and time series forecasting.