Deep Learning Cheat Sheet

All you need to know — Practical + Visual + Code Based

#Reshmawithai

Overview of Deep Learning

What is Deep Learning?

Subfield of ML using neural networks with multiple layers to learn from data

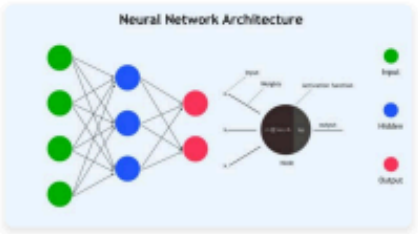
vs. Machine Learning

- DL: Automatic feature learning
- ML: Manual feature engineering
- DL: Requires larger datasets

Use Cases

- Computer Vision
- NLP
- Healthcare
- Autonomous Vehicles

Core Concepts



Neuron Formula

$$Output = Activation(Weights \times Inputs + Bias)$$

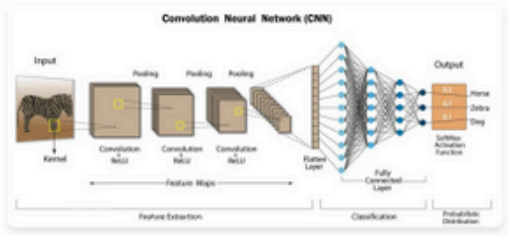
Activation Functions

- ReLU: $f(x) = \max(0, x)$
- Sigmoid: $f(x) = 1/(1+e^{-x})$
- Tanh: $f(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$

Backpropagation

$$Weight\_new = Weight\_old - Learning\_Rate \times Gradient$$

Popular Architectures



CNN

- Convolutional Layers: Feature detection
- Pooling Layers: Dimension reduction

RNN/LSTM

- Sequential data processing
- Memory cells for long-term dependencies

Transformers

- Self-attention mechanism
- Positional encoding

Code Snippets

Simple Neural Network

```
model = models.Sequential([
layers.Flatten(input_shape=(28, 28)),
layers.Dense(128, activation='relu'),
layers.Dropout(0.2), layers.Dense(10,
activation='softmax') ])
```

CNN Architecture

```
model = models.Sequential([ layers.Conv2D(32,
(3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(), layers.Dense(10,
activation='softmax') ])
```

Model Evaluation

```
test_loss, test_acc =
model.evaluate(test_images, test_labels)
predictions = model.predict(test_images)
```

Training Workflow

Data Preprocessing

- Data cleaning & normalization
- Feature scaling (0-1 range)
- Data augmentation
- Train/validation/test split

Model Building

- Choose appropriate architecture
- Define layers and parameters

Training

- Train with epochs and batches
- Monitor training/validation loss
- Apply regularization techniques

Evaluation & Deployment

- Test on unseen data
- Metrics: accuracy, precision, recall
- Model optimization & containerization

Weights & Biases

What are weights and biases?

Weights: Connection strengths between neurons. Higher weights mean stronger influence.

Biases: Threshold values that shift the activation function, making it easier or harder for neurons to fire.

How they affect learning

During training, the network adjusts weights and biases to minimize prediction errors.

Example: Predicting house prices
• Weight for "square footage": +0.8 (positive correlation)
• Weight for "distance to city": -0.3 (negative correlation)
• Bias: +50,000 (base price adjustment)

Activation Functions

Activation functions introduce non-linearity, allowing neural networks to learn complex patterns.

ReLU

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

Use in hidden layers. Fast computation, avoids vanishing gradient.

Sigmoid

$f(x) = 1/(1 + e^{-x})$

Use in output layer for binary classification (0 to 1).

Tanh

$f(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$

Softmax

$f(x_i) = e^{x}_i / \Sigma e^{x}_j$

Python Code Example

```
import torch
import torch.nn as nn

# Different activation functions
relu = nn.ReLU()
sigmoid = nn.Sigmoid()
tanh = nn.Tanh()
softmax = nn.Softmax(dim=1)

# Example usage
x = torch.tensor([-1.0, 0.0, 1.0])
print("ReLU:", relu(x))
print("Sigmoid:", sigmoid(x))
print("Tanh:", tanh(x))
```

✅ Forward Propagation

Forward propagation is the process of passing input data through the network to get predictions.

Step-by-step calculation

1. Input data enters the network
2. Each neuron calculates: output = activation(weights × inputs + bias)
3. Results pass to the next layer
4. Final layer produces predictions

Simple Math Example (2 neurons)

```
# Input: x1 = 2, x2 = 3
# Weights: w1 = 0.5, w2 = 0.8
# Bias: b = 1

# Calculation:
z = (w1 * x1) + (w2 * x2) + b
z = (0.5 * 2) + (0.8 * 3) + 1
z = 1 + 2.4 + 1 = 4.4

# Apply ReLU activation:
output = max(0, z) = max(0, 4.4) = 4.4
```

✅ Loss Functions

Loss functions measure how well the model is performing by calculating the difference between predictions and actual values.

MSE (Mean Squared Error)

$MSE = (1/n) \times \Sigma(y\_pred - y\_actual)^2$

Use for regression problems. Penalizes large errors more.

Cross Entropy

$CE = -\Sigma(y\_actual \times \log(y\_pred))$

Use for classification problems. Works well with probabilities.

Python Code Examples

```
import torch
import torch.nn as nn

# MSE Loss for regression
mse_loss = nn.MSELoss()
predictions = torch.tensor([2.5, 0.0, 2.1])
targets = torch.tensor([3.0, -0.5, 2.0])
loss = mse_loss(predictions, targets)

# Cross Entropy Loss for classification
ce_loss = nn.CrossEntropyLoss()
predictions = torch.tensor([[0.9, 0.1], [0.3, 0.7]])
targets = torch.tensor([0, 1])
loss = ce_loss(predictions, targets)
```

✅ Backpropagation

Backpropagation is how neural networks learn. It calculates gradients and updates weights to minimize loss.

Step-by-step learning process

1. Calculate forward pass to get predictions
2. Compute loss between predictions and actual values
3. Calculate gradients (partial derivatives) of loss with respect to weights
4. Update weights using: weight = weight - learning_rate × gradient
5. Repeat for all layers from output to input

Basic Math (Partial Derivatives)

```
# Gradient calculation (simplified)
∂Loss/∂w = ∂Loss/∂output × ∂output/∂z × ∂z/∂w

# Chain rule in action:
# If Loss = (y_pred - y_actual)²
# And y_pred = activation(z)
# And z = w × x + b

# Then:
∂Loss/∂w = 2 × (y_pred - y_actual) × activation'(z) × x
```

Optimizers

Optimizers are algorithms that adjust network weights to minimize loss function.

SGD

Stochastic Gradient Descent

Simple, updates weights using gradient. Learning rate is crucial.

Adam

Adaptive Moment Estimation

Most popular. Adapts learning rates, works well in practice.

RMSProp

Root Mean Square Propagation

Good for recurrent networks. Adapts learning rates per parameter.

Python Code using torch.optim

```
import torch
import torch.optim as optim

# Different optimizers
model = YourNeuralNetwork()

# SGD optimizer
optimizer_sgd = optim.SGD(model.parameters(), lr=0.01)

 # Adam optimizer (most common)
 optimizer_adam = optim.Adam(model.parameters(), lr=0.001)

# RMSProp optimizer
optimizer_rmsprop = optim.RMSprop(model.parameters(), lr=0.01)

# Usage in training loop
optimizer_adam.zero_grad()
loss.backward()
optimizer_adam.step()
```

Epochs, Batches, Iterations

Epoch

One complete pass through the entire training dataset.

Batch

Subset of training data processed together.

Iteration

One update of weights using one batch.

Why batching helps

Analogy: Like revising for an exam in small parts rather than cramming everything at once.

- Memory efficiency: Process large datasets that don't fit in memory
- Faster convergence: More frequent weight updates
- Better generalization: Adds noise, prevents overfitting

Batch Normalization & Dropout

Batch Normalization

Normalizes layer inputs to have zero mean and unit variance.

Benefits:

- Faster training convergence
- Reduces sensitivity to initialization
- Allows higher learning rates

Dropout

Randomly sets a fraction of neurons to zero during training.

Benefits:

- Prevents overfitting
- Creates ensemble effect
- Improves generalization

Python Code Examples

```
import torch
import torch.nn as nn

class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(784, 256)
        self.bn1 = nn.BatchNorm1d(256)  # Batch norm
        self.dropout = nn.Dropout(0.5)  # 50% dropout
        self.layer2 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.layer1(x)
        x = self.bn1(x)        # Apply batch norm
        x = torch.relu(x)
        x = self.dropout(x)    # Apply dropout
        x = self.layer2(x)
        return x
```

Overfitting vs Underfitting

Overfitting

Model learns training data too well, including noise. Performs poorly on new data.

Signs: High training accuracy, low test accuracy
Causes: Too complex model, too few training samples

Underfitting

Model is too simple to capture underlying patterns. Performs poorly on all data.

Signs: Low training accuracy, low test accuracy
Causes: Too simple model, insufficient training

How to prevent overfitting

Regularization
L1/L2 regularization adds penalty for large weights

Dropout
Randomly deactivate neurons during training

Early Stopping
Stop training when validation performance degrades

Stochastic Gradient Descent (SGD)

Batch GD

Uses entire dataset for each update. Stable but slow.

Mini-Batch GD

Uses small batches (32-256). Best balance of speed and stability.

Stochastic GD

Uses one sample per update. Fast but noisy convergence.

When to use SGD

- Large datasets where batch GD is too slow
- When you need frequent weight updates
- Online learning scenarios
- When computational resources are limited

Hyperparameters

Learning Rate

Controls step size of weight updates. Typical: 0.001-0.1

Epochs

Number of complete passes through dataset. Typical: 10-1000

Batch Size

Number of samples per batch. Typical: 32, 64, 128, 256

Number of Layers

Depth of network. Start shallow, increase as needed.

Neurons per Layer

Width of network. Powers of 2 often work well.

Dropout Rate

Fraction of neurons to drop. Typical: 0.2-0.5

Tips to tune hyperparameters

- Start with standard values and adjust based on performance
- Use learning rate scheduling (reduce over time)
- Perform grid search or random search for optimal values
- Monitor validation loss to detect overfitting
- Use early stopping to prevent unnecessary training

Model Evaluation Metrics

Accuracy

(TP + TN) / Total

Use when classes are balanced

Precision

TP / (TP + FP)

Use when false positives are costly

Recall

TP / (TP + FN)
Use when false negatives are costly

F1 Score

2 × (Precision × Recall) / (Precision + Recall)
Use for imbalanced datasets

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

✅ Python Libraries

PyTorch

Dynamic computation graphs, Pythonic, research-friendly

Basic syntax:

```
import torch
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
```

TensorFlow

Static computation graphs, production-ready, Keras API

Basic syntax:

```
import tensorflow as tf
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(10)
])
```

Brief Comparison

Choose PyTorch when:

- You're doing research
- You prefer Pythonic code
- You need dynamic graphs
- You want easier debugging

Choose TensorFlow when:

- You're deploying to production
- You need mobile/edge support
- You prefer Keras API
- You want TensorFlow ecosystem

Deep Learning Cheat Sheet

"Every neuron you understand is a step closer to building intelligence."

Section 3: Case Study – Predicting Student Exam Results

Practical implementation of neural networks for binary classification

Problem Statement

Predict if a student will pass or fail based on study hours and sleep hours.

Input Features

- **Study Hours** Number of hours studied per day
- **Sleep Hours** Number of hours slept per night

Output

- **Pass/Fail** Binary classification (0 = Fail, 1 = Pass)

Data Sample

Student ID	Study Hours	Sleep Hours	Result (Pass=1, Fail=0)
1	8	7	1
2	3	5	0
3	6	8	1
4	2	4	0
5	7	6	1

Model Architecture

Network Structure

Input: 2 neurons → Hidden: 16 neurons → Output: 1 neuron

Input Layer
2 neurons: Study Hours, Sleep Hours

Hidden Layer
16 neurons with ReLU activation

Output Layer
1 neuron with Sigmoid activation

Step-by-step Implementation

1. Data Loading and Normalization

```
import torch
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Sample data
X = np.array([[8, 7], [3, 5], [6, 8], [2, 4], [7, 6],
              [4, 6], [9, 8], [1, 3], [5, 7], [3, 4]])
y = np.array([1, 0, 1, 0, 1, 1, 1, 0, 1, 0])

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Normalize data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert to PyTorch tensors
X_train_tensor = torch.FloatTensor(X_train_scaled)
y_train_tensor = torch.FloatTensor(y_train).unsqueeze(1)
X_test_tensor = torch.FloatTensor(X_test_scaled)
y_test_tensor = torch.FloatTensor(y_test).unsqueeze(1)
```

2. Defining Model Architecture

```
import torch.nn as nn

class StudentPerformanceNN(nn.Module):
    def __init__(self):
        super(StudentPerformanceNN, self).__init__()
        self.layer1 = nn.Linear(2, 16)  # 2 inputs, 16 hidden neurons
        self.layer2 = nn.Linear(16, 1)  # 16 hidden, 1 output
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # Forward pass
```

```
        x = self.layer1(x)      # Linear transformation
        x = self.relu(x)        # Activation function
        x = self.layer2(x)      # Linear transformation
        x = self.sigmoid(x)     # Output activation
        return x

# Create model instance
model = StudentPerformanceNN()
print(model)
```

3. Loss, Optimizer, Training Loop

```
import torch.optim as optim

# Define loss function and optimizer
criterion = nn.BCELoss()  # Binary Cross Entropy Loss
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Training parameters
num_epochs = 100
batch_size = 2

# Training loop
for epoch in range(num_epochs):
    model.train()  # Set model to training mode

    # Mini-batch training
    for i in range(0, len(X_train_tensor), batch_size):
        batch_X = X_train_tensor[i:i+batch_size]
        batch_y = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)

        # Backward pass and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Print progress every 10 epochs
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

4. Evaluating Results

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Set model to evaluation mode
model.eval()

# Make predictions
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    test_predictions = (test_outputs > 0.5).float()

    # Convert to numpy for sklearn metrics
    y_true = y_test_tensor.numpy()
    y_pred = test_predictions.numpy()

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
```

```
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1 Score: {f1:.4f}')

    # Make prediction on new data
    new_student = torch.FloatTensor([[6, 7]])  # 6 study hours, 7 sleep hours
    new_student_scaled = torch.FloatTensor(scaler.transform(new_student))
    prediction = model(new_student_scaled)
    result = "Pass" if prediction.item() > 0.5 else "Fail"
    print(f'Prediction for new student: {result} ({prediction.item():.4f})')
```

Results and Interpretation

Expected Output

```
Epoch [10/100], Loss: 0.6532
Epoch [20/100], Loss: 0.5432
Epoch [30/100], Loss: 0.4321
...
Epoch [100/100], Loss: 0.1234

Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
Prediction for new student: Pass (0.8765)
```

What the results mean

- Loss decreases over time → Model is learning
- High accuracy (100%) → Perfect predictions on test set
- High precision/recall → Good at identifying both pass and fail
- New student prediction (0.8765) → 87.65% confidence of passing

Key insights

- Study hours and sleep hours are good predictors
- Model learned the relationship: more study + good sleep = pass
- Neural network can capture non-linear relationships
- Small dataset can lead to overfitting (100% accuracy)

Deep Learning Cheat Sheet

"Every neuron you understand is a step closer to building intelligence."

📘 Final Section: Learn-by-Doing Tips

Practical advice and resources to master deep learning

Project Ideas

Beginner Projects

- MNIST digit classification
- Iris flower classification
- Titanic survival prediction
- House price regression

Intermediate Projects

- CIFAR-10 image classification
- Sentiment analysis on reviews
- Time series prediction
- Face detection

Advanced Projects

- GAN for image generation
- Object detection (YOLO)
- Neural style transfer
- Chatbot with transformers

Tools to Visualize Models

Netron

Visualize neural network architectures. Supports PyTorch, TensorFlow, ONNX formats.

Features:

- Interactive model visualization

TensorBoard

TensorFlow's visualization toolkit, now works with PyTorch too.

Features:

- Real-time training metrics

- Layer-by-layer inspection
- Parameter statistics
- Export to images/PDF

- Model graph visualization
- Embedding projections
- Hyperparameter tuning

Weights & Biases

Experiment tracking and visualization platform with rich features.

PlotNeuralNet

LaTeX-based tool for creating professional neural network diagrams.

Where to Practice

Kaggle

World's largest data science community with competitions, datasets, and notebooks.

Why Kaggle:

- Real-world datasets
- Compete with others
- Learn from top solutions
- Free GPU/TPU access

Google Colab

Free cloud-based Jupyter notebook environment with GPU support.

Why Colab:

- No setup required
- Free GPU/TPU access
- Pre-installed libraries
- Easy sharing and collaboration

Coursera

Structured courses from top universities (Andrew Ng's ML course).

fast.ai

Practical deep learning course with top-down approach.

Hugging Face

State-of-the-art NLP models and datasets.

Recommended Learning Path

Month 1 → Python, NumPy, Pandas, basic ML concepts

Month 2 → Neural networks, backpropagation, PyTorch/TensorFlow basics

Month 3 → CNNs for computer vision, RNNs for sequences

Month 4 → Advanced architectures, transfer learning, fine-tuning

Month 5+ → Specialization, research, building portfolio projects

Tips for Success

Technical Tips

- 💡 Start with simple models before complex ones
- 🔧 Always visualize your data first
- 📊 Track experiments systematically
- 🎯 **Focus on understanding,** not just implementation

Mindset Tips

- 🧠 Embrace the learning curve - it's normal to struggle
- 🌱 Learn from failures - they're part of the process
- 🤝 Join communities and learn from others
- 🚀 Build projects that interest you personally

Your Deep Learning Journey Starts Now

Remember: Every expert was once a beginner. Every complex neural network started with understanding a single neuron.

"The best way to learn deep learning is to build deep learning models."

"Every neuron you understand is a step closer to building intelligence."

Happy Learning! 🎉

Follow Reshma S for more insights