DL Demystified 15 - Evaluating ML Systems - MMLU Benchmarks

January 29, 2025

[1]:
```
######################################################################
#                                                                    #
#  CS435 Generative AI: Security, Ethics and Governance              #
#                                                                    #
#  Instructor: Dr. Adnan Masood                                      #
#  Contact:    adnanmasood@gmail.com                                 #
#                                                                    #
#  Notebook is MIT Licensed                                          #
######################################################################
```

1 LLM Evaluations with MMLU and Other Benchmarks

1.0.1 *A Comprehensive, Step-by-Step Explanation and Example with PyTorch*

By: **Dr. Adnan Masood**

1.1 Notebook Overview

1. **Building an Intuitive Understanding** (From "Curious Adventurer" to "Doctoral Explorer")
2. **Intuition Behind LLM Evaluation**
3. **Brief History and Underlying Tech**
4. **Math Behind It**
5. **Illustrative Example (Code + Explanation)**
6. **Example Calculations** (weights, biases, etc.)
7. **Step-by-Step Example: Building From Scratch**
8. **Illustrative Problem It Solves**
9. **Practical, Real-World Problem It Solves**
10. **How to Solve a Real-World Problem Using This Tech**
11. **Questions to Ask & Their Answers (with Code)**
12. **A Sample Exercise**
13. **Glossary**

1.2 1. Building an Intuitive Understanding

We'll explain **LLM (Large Language Model) Evaluations** with a focus on the **MMLU benchmark** (Massive Multitask Language Understanding) and other benchmarks, at different depths. Think of it as layering new insights each time.

- **What's an LLM?** A Large Language Model is like a very big "predictive text" that can answer questions, write stories, or have a conversation.
- **What's MMLU?** It's a big test for these models. It has lots of questions on different subjects (math, history, science, etc.), and we see how well the model answers.
- **Why do we test them?** Just like you take tests at school, we give tests to AI models to check what they know and how good they are at answering questions.
- **LLMs** use huge amounts of text to "learn" patterns in language. They can complete sentences or answer queries.
- **Evaluation**: We measure how well the model does on different benchmarks (like MMLU). MMLU is a set of exam-style questions across many academic subjects.
- **Goal**: If an LLM can answer a wide variety of questions correctly, it shows it "understands" or can replicate knowledge from various fields.
- **LLMs** rely on deep neural networks (e.g., Transformers) that learn word relationships.
- **Benchmarks** like MMLU test the model's ability to handle tasks across numerous academic disciplines, checking both factual recall and reasoning.
- **Metrics**: Typically, we measure accuracy, or how often the model picks the correct answer. We may also measure other metrics like F1-score, perplexity, etc.
- **Why MMLU?** MMLU covers 57 subjects, reflecting real-world academic tests. It's a stress test for a model's multitask knowledge.
- **Transformer Architecture**: The LLM is built on multi-head self-attention layers, enabling it to weigh different parts of text differently.
- **Fine-tuning & Prompt Engineering**: MMLU is used as a downstream evaluation benchmark. Models can be either fine-tuned or prompted to perform well.
- **Domain Generalization**: The ability of a model to handle varied subjects with minimal parameter adjustments is key to success on MMLU.
- **Evaluation Nuance**: We also consider calibrations (confidence measures), multi-step reasoning, and chain-of-thought prompts.
- **Detailed Architecture**: LLMs typically rely on pretraining objectives (e.g., masked language modeling, next-token prediction) on massive corpora. They exploit emergent capabilities when scaled.
- **Evaluation Techniques**: MMLU involves multiple-choice questions that gauge knowledge from high school to professional exam level. The choice of prompting strategy (few-shot, zero-shot, chain-of-thought) can drastically affect performance.

- **Research Implications**: Because MMLU covers broad disciplines, it challenges the model's reasoning, knowledge retrieval, and ability to handle out-of-distribution questions.
- **Limitations**: Models might achieve good MMLU performance yet fail in interpretability, calibration, or real-world application constraints.

1.3 2. Intuition Behind LLM Evaluation

- Large Language Models learn patterns by seeing tons of text examples.
- We want to test if they can answer new questions they haven't seen before.
- Think of MMLU as a giant trivia quiz for the model—covering many domains. If the model truly "understands" or at least can replicate knowledge well, it will do well on these varied questions.
- Evaluation helps us measure progress and compare different models or versions of models.

1.4 3. Brief History, Invention, and Underlying Tech

- **Deep Neural Networks**: Before LLMs, we had simpler neural nets. But with the introduction of Transformers (Vaswani et al., 2017), we could handle large contexts.
- **Pretraining & Fine-tuning**: GPT (Generative Pretrained Transformer) models from OpenAI, BERT from Google, and others popularized large-scale pretraining.
- **MMLU**: Proposed as a comprehensive benchmark covering 57 subjects, from elementary to professional knowledge. The idea is to test a model's broad knowledge.
- **Underlying Tech**: High-dimensional vector representations (embeddings), attention mechanisms, massive parallel computing with GPUs/TPUs.

1.5 4. Math Behind It

Core Concept: We use a neural network (often a Transformer) which does something like this:

1. **Embedding**: Convert each word into a vector of numbers. Suppose a word `w` is mapped to a vector $\mathbf{x}$ of size d.
2. **Attention**: The model calculates how important each word is to every other word in the input. This can be written (in simplified form) as:

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

where

- Q = Query matrix
- K = Key matrix
- V = Value matrix
- d_k = dimension of the key vectors

3. **Feed-Forward**: After attention, we pass the result through linear layers with activation functions (like ReLU or GELU).

$$\mathbf{z} = \text{ReLU}(W_2\,(\text{ReLU}(W_1\,\mathbf{h} + b_1)) + b_2)$$

This is a typical 2-layer feed-forward used in Transformers.

4. **Output**: Eventually, the model learns to predict the next word or fill in masked words. For multiple choice, it can rank which answer is likely correct.

5. **Evaluation**: We check if the highest probability answer is the correct one. Accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{Number of correct answers}}{\text{Total questions}}.$$

In MMLU, we have many different subjects, so we get an overall accuracy across all tasks.

1.6 5. Illustrative Example (Code + Explanation)

Imagine a simpler scenario. We'll create a toy neural network that tries to choose the correct label for small text inputs. Instead of 57 subjects, we'll just do a mini classification example to illustrate how we might evaluate correctness.

---

[2]:
```
# We'll simulate a tiny dataset of text, each labeled as category 0 or 1.
# Then we'll build a simple feed-forward network to classify them.

import torch
import torch.nn as nn
import torch.optim as optim

torch.manual_seed(42)

# Suppose we have a small vocabulary and each word is a simple integer.
# We'll create a synthetic dataset (text_samples, labels) just for illustration.

text_samples = [
    [1, 2, 3],   # e.g. "I love math"
    [4, 5, 6],   # e.g. "Dogs are cute"
    [1, 5, 7],   # e.g. "I are cats" (nonsense example)
    [2, 3, 7],   # e.g. "love math cats" (nonsense)
]

# Labels: 0 or 1
labels = [0, 1, 1, 0]

# Convert to tensors
text_tensor = torch.tensor(text_samples, dtype=torch.long)
labels_tensor = torch.tensor(labels, dtype=torch.long)
```

```
class SimpleClassifier(nn.Module):
    def __init__(self, vocab_size=8, embed_dim=4, hidden_dim=8, num_classes=2):
        super(SimpleClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        # Simple feed-forward
        self.fc1 = nn.Linear(embed_dim * 3, hidden_dim)  # 3 words each of␣
 ↪embed_dim
        self.fc2 = nn.Linear(hidden_dim, num_classes)
        self.relu = nn.ReLU()
    def forward(self, x):
        # x shape: (batch_size, seq_len)
        embeds = self.embedding(x)  # (batch_size, seq_len, embed_dim)
        # Flatten
        embeds = embeds.view(embeds.size(0), -1)  # (batch_size,␣
 ↪seq_len*embed_dim)
        out = self.fc1(embeds)
        out = self.relu(out)
        out = self.fc2(out)
        return out

# Initialize model
model = SimpleClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# We'll do a simple training loop
num_epochs = 20

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(text_tensor)
    loss = criterion(outputs, labels_tensor)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 5 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

# Evaluate accuracy on the training set (since we have a tiny dataset)
with torch.no_grad():
    outputs = model(text_tensor)
    _, predicted = torch.max(outputs, dim=1)
    correct = (predicted == labels_tensor).sum().item()
    accuracy = correct / len(labels_tensor) * 100
    print(f"Training Accuracy: {accuracy:.2f}%")
```

```
Epoch [5/20], Loss: 0.6232
```

```
Epoch [10/20], Loss: 0.5231
Epoch [15/20], Loss: 0.3873
Epoch [20/20], Loss: 0.2444
Training Accuracy: 100.00%
```

Interpretation: - We used a dummy dataset and a small embedding + feed-forward network. - The model tries to classify text samples into 0 or 1. - We measure the accuracy. This is analogous (in a very tiny way) to how we would see if our model picks the correct answer among choices.

For MMLU, we'd do something similar but with a much larger dataset and more sophisticated prompts or fine-tuning strategies.

1.7 6. Example Calculations

1.7.1 Weights (W)

- These are the parameters the network learns. For example, in our code, `self.fc1.weight` and `self.fc2.weight` are weight matrices.
- In a Transformer, we have weights in the embedding layer, attention matrices Q, K, V, and feed-forward layers.

1.7.2 Bias (b)

- Added to the linear transformation. For instance, `self.fc1.bias`.
- Helps the model shift predictions up or down.

1.7.3 Matrix Multiplication

- If we have x as an input vector (e.g., embeddings), a linear layer does $Wx + b$. Suppose W is 2x3 and x is 3x1, the result is a 2x1 vector.

1.7.4 Softmax

- Used to convert raw scores into probabilities. For classification, we check which probability is highest.

1.7.5 Loss Calculation

- If the correct label is 1, but the model's probability for label 1 is 0.2, it gets penalized. The network adjusts weights via backpropagation to reduce this penalty.

1.8 7. Step-by-Step Example: Creating the Technology from Scratch

Step 1: Collect or define a dataset - For MMLU, it's a big set of multiple-choice questions across 57 subjects. - For our mini example, we just made a few text samples.

Step 2: Build or define the model architecture - This could be a Transformer-based model for large-scale tasks. - In our mini example, it's an embedding + feed-forward.

Step 3: Choose a training objective - For classification, it's usually cross-entropy. - For MMLU or multiple-choice QA, we compare the logits for each choice.

Step 4: Train the model - Provide data in batches, do forward pass, compute loss, backpropagate, update weights.

Step 5: Evaluate - Compare model predictions vs. correct answers. - Calculate accuracy or other metrics.

Step 6: Iterate - Tweak hyperparameters, architecture, or training steps to improve performance.

1.9 8. Illustrative Problem It Solves

In Academics: - MMLU can test if a model "knows" or can replicate knowledge from, say, chemistry or math. If it can pass those tests, it implies broad coverage.

In Our Mini Example: - Our tiny classifier can differentiate text into categories (0 or 1). This is akin to seeing if it can pick the right label from a set of choices.

1.10 9. Practical, Real-World Problem It Solves

Chatbots and Virtual Assistants: - Tools like ChatGPT are evaluated on tasks akin to MMLU to see if they can answer professional-level and academic queries reliably.

Diagnostics: - If a model is deployed in a tutoring app, we want to ensure it can handle different subject questions well. MMLU-like evaluations help gauge that.

1.11 10. How to Solve a Real-World Problem Using This Tech

1. **Identify the Task**: e.g., build a Q&A system for a medical domain.
2. **Collect Data**: gather medical questions and correct answers.
3. **Train or Fine-tune**: pick a large pre-trained LLM, fine-tune it or prompt it to handle medical Q&A.
4. **Evaluate**: use an MMLU-like benchmark (or specialized medical exam questions) to see how well the system does.
5. **Deploy & Monitor**: put the system in a real setting, monitor performance, gather feedback for improvements.

1.12 11. Questions to Ask & Their Answers (with Code Examples)

1.12.1 Q1. *How do we add new subjects or tasks to an MMLU-like benchmark?*

Answer: You'd create or source multiple-choice questions from the new subject, standardize them (similar to the existing format), and then incorporate them into the evaluation pipeline.

1.12.2 Q2. *What if my model is performing poorly on certain subjects?*

Answer: You can gather more training data relevant to that subject, do targeted fine-tuning, or employ prompt engineering to help the model reason more effectively.

1.12.3 Q3. *How do we measure success on MMLU?*

Answer: Typically, we measure the percentage of questions answered correctly across all subjects. We may also look at performance by subject.

1.12.4 Q4. *Can we do a quick code snippet that checks multiple-choice answers?*

Answer: Yes, see below for a minimal illustration:

[3]:
```
# Minimal multiple-choice style check
import torch

# Suppose we have 1 question with 3 possible answers.
question_embedding = torch.randn(1, 4)  # random question representation
choice_embeddings = torch.randn(3, 4)  # 3 choices, each a 4-d vector

# Let's do a simple dot product to see which answer is most similar to the␣
 ↪question.
scores = torch.matmul(choice_embeddings, question_embedding.transpose(0, 1))
scores = scores.view(-1)  # flatten
best_choice = torch.argmax(scores).item()
print("Scores:", scores)
print("Best choice index:", best_choice)
```

```
Scores: tensor([ 2.5861, -0.3854, -1.7413])
Best choice index: 0
```

Interpretation: In a real scenario, the model outputs logits for each choice, and the highest logit is the predicted answer.

1.13 12. A Sample Exercise

Below is a more elaborate example. The idea is to have students: 1. **Complete the TODOs**. 2. **Run the cells**. 3. **Observe the results**.

We'll build a very small text classification model (similar to before but with some missing pieces for you to fill). We'll imagine each piece of text is an exam question, and the label is the correct subject category (just for demonstration).

---

[4]:
```
# TODO CODE EXAMPLE
# Follow the hints and fill in the missing pieces marked as TODO.

import torch
```

```
import torch.nn as nn
import torch.optim as optim

################################
# 1. Prepare some synthetic data
################################

# Suppose we have 6 text samples with 2 different "subject categories" (label 0␣
 ↪or 1)
text_samples = [
    [1, 2, 2, 3],   # e.g. question tokens
    [2, 2, 3, 1],
    [4, 1, 5, 5],
    [4, 1, 1, 5],
    [7, 2, 2, 3],
    [7, 1, 1, 2]
]
labels = [0, 0, 1, 1, 0, 1]

# Convert to tensors
text_tensor = torch.tensor(text_samples, dtype=torch.long)
labels_tensor = torch.tensor(labels, dtype=torch.long)

################################
# 2. Define a simple network
################################

class MyTinyModel(nn.Module):
    def __init__(self, vocab_size=10, embed_dim=4, hidden_dim=8, num_classes=2):
        super(MyTinyModel, self).__init__()
        # TODO: Create an embedding layer called self.embedding
        self.embedding = nn.Embedding(num_embeddings=vocab_size,␣
 ↪embedding_dim=embed_dim)

        # We assume each sample has length 4, so final embedding size is␣
 ↪embed_dim * 4
        # TODO: Create a linear layer self.fc1 from embed_dim*4 -> hidden_dim
        self.fc1 = nn.Linear(embed_dim*4, hidden_dim)

        # TODO: Create a second linear layer self.fc2 from hidden_dim ->␣
 ↪num_classes
        self.fc2 = nn.Linear(hidden_dim, num_classes)

        self.relu = nn.ReLU()

    def forward(self, x):
        # x shape: (batch_size, seq_len)
```

```
        embeds = self.embedding(x)
        # flatten
        embeds = embeds.view(embeds.size(0), -1)
        out = self.fc1(embeds)
        out = self.relu(out)
        out = self.fc2(out)
        return out

################################
# 3. Training loop
################################

# TODO: Instantiate the model, define criterion (CrossEntropyLoss), and␣
 ↪optimizer (Adam or SGD)
model = MyTinyModel()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# TODO: define num_epochs
num_epochs = 20

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(text_tensor)
    loss = criterion(outputs, labels_tensor)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 5 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

# Evaluate
with torch.no_grad():
    outputs = model(text_tensor)
    _, preds = torch.max(outputs, dim=1)
    correct = (preds == labels_tensor).sum().item()
    accuracy = correct / len(labels_tensor) * 100
print(f"Final Training Accuracy: {accuracy:.2f}%")
```

```
Epoch [5/20], Loss: 0.5779
Epoch [10/20], Loss: 0.4137
Epoch [15/20], Loss: 0.2359
Epoch [20/20], Loss: 0.0914
Final Training Accuracy: 100.00%
```

1.13.1 Hints:

- The embedding layer is `nn.Embedding(num_embeddings, embedding_dim)`.
- The linear layers are `nn.Linear(in_features, out_features)`.

- The CrossEntropyLoss expects raw logits, so don't apply softmax inside the model (PyTorch automatically does it in `nn.CrossEntropyLoss`).

1.14 13. Glossary

- **LLM (Large Language Model)**: A neural network model trained on massive text data, capable of generating or understanding human-like text.
- **MMLU (Massive Multitask Language Understanding)**: A comprehensive benchmark testing LLMs across 57 subjects.
- **Benchmark**: A standard test set or suite of tasks for comparing model performance.
- **Transformer**: A neural network architecture using attention mechanisms to handle sequential data.
- **Attention**: Method to focus on different parts of input text selectively.
- **Weights and Biases**: Parameters the network learns. Weights are multiplicative factors, biases are additive offsets.
- **Loss Function**: A measure of how far off the model's predictions are from the correct labels.
- **Accuracy**: Ratio of correct predictions to total predictions.
- **Logits**: The raw, unnormalized scores output by a model before a softmax.
- **Embeddings**: Vector representations of words or tokens.
- **Fine-tuning**: Adapting a pre-trained model to a specific task.
- **Chain-of-thought**: A method of prompting that shows the model detailed reasoning steps.

[5]:
```
import os, sys, platform, datetime, uuid, socket

def signoff(name="Anonymous"):
    colab_check = "Yes" if 'google.colab' in sys.modules else "No"
    mac_addr = ':'.join(format((uuid.getnode() >> i) & 0xff, '02x') for i in␣
 ↪reversed(range(0, 48, 8)))
    print("+++ Acknowledgement +++")
    print(f"Executed on: {datetime.datetime.now()}")
    print(f"In Google Colab: {colab_check}")
    print(f"System info: {platform.system()} {platform.release()}")
    print(f"Node name: {platform.node()}")
    print(f"MAC address: {mac_addr}")
    try:
        print(f"IP address: {socket.gethostbyname(socket.gethostname())}")
    except:
        print("IP address: Unknown")
    print(f"Signing off, {name}")

signoff("Ali Muhammad Asad")
```

```
+++ Acknowledgement +++
Executed on: 2025-01-29 01:47:11.438630
In Google Colab: No
System info: Linux 6.8.0-51-generic
Node name: alimuhammad-Inspiron-7559
```

```
MAC address: 20:47:47:74:94:05
IP address: 127.0.1.1
Signing off, Ali Muhammad Asad
```

[ ]: