Member-only story

Practical Guide to Distilling Large Models into Small Models: A Novel Approach with Extended Distillation

Comparing Traditional and Enhanced Step-by-Step Distillation: Adaptive Learning, Cosine Similarity, and Curriculum-Based Rationale Supervision

Shenggang Li · Following

Published in Towards AI · 25 min read · Mar 3, 2025

Photo by Thorium on Unsplash

Introduction

In this paper, I will uncover the secrets behind transferring “big model” intelligence to smaller, more agile models using two distinct distillation techniques: Traditional Distillation and Step-by-Step Distillation. Imagine having a wise, resource-heavy teacher model that not only gives the right answer but also explains its thought process — like a master chef sharing both the recipe and the secret tricks behind it. My goal is to teach a lean, efficient student model to emulate that expertise using just the distilled essence of knowledge.

To make these ideas crystal clear, I illustrate each technique using simple Logistic Regression demos. Although Logistic Regression is simpler than deep neural networks, it serves as an excellent canvas to experiment with

concepts like temperature scaling, weighted losses, and even simulating a "chain-of-thought" through intermediate linear scores. For Traditional Distillation, our student learns from the teacher's soft probability outputs, balancing hard label accuracy with the subtle cues of soft labels. Meanwhile, Step-by-Step Distillation goes one step further by also incorporating the teacher's internal reasoning process.

Finally, I propose an improved step-by-step distillation method that makes learning more stable and efficient. By adding gradual rationale loss ramp-up, cosine similarity for reasoning alignment, and stronger consistency regularization, the student better adapts to the teacher's logic while staying focused on the final predictions. These tweaks smooth out training, boost generalization, and keep the optimization stable.

Distilling LLM Wisdom into Compact Models: A Step-by-Step Approach Using Soft Labels

Imagine having a super smart big model (such as GPT-4) that can not only answer questions but also give detailed explanations. However, this big model is too large and requires both memory and computing power to deploy. To solve this problem, we want to pass the "wisdom" of the big model to a lightweight small model. This process is called knowledge distillation.

In knowledge distillation, the output generated by the big model (teacher) is not just the answer, but includes some additional information (such as probability distribution or explanatory content). I call these outputs "soft labels". Then, I use these labels generated by the big model to train the small model (student) so that it can "imitate" the behavior of the big model.

Definition of Knowledge Distillation

Knowledge distillation is a model compression technique, whose core idea is:

1. The large model (teacher model) generates detailed output labels (soft labels) instead of just providing the correct answer.
2. By learning from these outputs, the small model (student model) not only learns to predict the answer correctly but also captures the inherent knowledge and decision-making process of the large model.

Compared to training directly with hard labels (one-hot labels), this method allows the small model to learn more detailed information, thereby improving performance.

Training Objectives

The probability distribution of the large model output is denoted as:

$$p_T(y|x)$$

The probability distribution of the small model output is:

$$p_S(y|x)$$

To make the small model imitate the large model, I typically use *KL* divergence (Kullback-Leibler Divergence) as the distillation loss:

$$L_{\text{KD}} = D_{KL}\big(p_T(y|x) \parallel p_S(y|x)\big) = \sum_y p_T(y|x) \log \frac{p_T(y|x)}{p_S(y|x)}$$

Additionally, combine the original cross-entropy loss, using both hard labels and the soft labels from the large model to guide the training of the small model. The combined loss function can be expressed as:

$$L = \alpha \cdot L_{\text{hard}} + (1 - \alpha) \cdot T^2 \cdot L_{\text{KD}}$$

Where:

- *L_hard* is the cross-entropy loss of the small model on the true labels.
- *T* is the temperature parameter, used to smooth the probability distribution of the large model's output.
- *α* controls the weight of the two loss terms.

This design allows the student model to learn the fine-grained information from the large model while still being guided by real data.

Example: How the Small Model Imitates the Large Model

Let's take a text classification task as an example — determining whether a movie review is positive or negative.

Large Model (Teacher):

For a given review, the large model outputs the following soft labels:

- Positive: *0.85*
- Negative: *0.15*

Besides just providing the final probability distribution, the teacher model might also offer an explanation, such as:
"This review uses positive language, and the overall sentiment leans positive".

Small Model (Student):

During training, instead of just learning from the final hard label (e.g., "positive"), the small model also learns from the probability distribution given by the teacher model. This means the student model tries to adjust its own predictions to match the teacher's probability outputs as closely as possible.

In short, the student model learns to capture subtle distinctions in sentiment, even with limited training data. This means, if a review is slightly positive but not overwhelmingly so, the soft label distribution (e.g., *0.6* positive, *0.4* negative) provides a richer learning signal than a simple "positive" label.

Distilling Step-by-Step: A Smarter Approach to Model Compression

I first introduced traditional knowledge distillation in previous section, where a large model transfers its knowledge to a smaller one by providing soft labels. While effective, this method often requires a vast amount of data for the small model to achieve comparable performance.

However, a new approach called Distilling Step-by-Step, introduced in Google's research paper (https://arxiv.org/pdf/2305.02301), enhances knowledge transfer by incorporating the reasoning process of the large model. Instead of only mimicking final predictions, the small model also learns the step-by-step inference process, making it more efficient in generalization and reasoning tasks.

Why Do We Need This?

Large language models (LLMs) are powerful but require massive computational resources. For example, a 175-billion-parameter model demands over 350GB of GPU memory, making real-world deployment expensive. Researchers have explored two primary approaches to distilling knowledge into a small model:

- Fine-tuning: Training a small model on human-labeled data.
- Knowledge distillation: Training a small model using soft labels produced by a large model.

However, both methods face challenges: fine-tuning requires labeled datasets, while standard distillation lacks explicit reasoning structures, limiting the student model's understanding of why the teacher model makes certain decisions.

Distillation: Learning from the Reasoning Process

This new method, Distilling Step-by-Step, improves model compression by transferring both **outputs and rationales** from the large model. The core idea is to train the small model to replicate not just the final predictions, but also the intermediate reasoning steps used by the large model.

Given an input x, let:

- *p_T(y | x)* be the probability distribution of the large model (teacher).
- *p_S(y | x)* be the probability distribution of the small model (student).
- τ be the temperature parameter to smooth probability distributions.

The standard knowledge distillation loss function is given by:

$$L_{\text{KD}} = \text{KL}(p_T^\tau(y|x)||p_S^\tau(y|x))$$

where *p_Tτ(y | x)* is the teacher's output softened using temperature τ:

$$p_T^\tau(y|x) = \frac{\exp(p_T(y|x)/\tau)}{\sum_{y'} \exp(p_T(y'|x)/\tau)}$$

However, Step-by-Step Distillation introduces an additional reasoning-based objective:

$$L_{\text{step}} = \sum_{t=1}^{T} \text{KL}(p_T^\tau(z_t|x)||p_S^\tau(z_t|x))$$

where

$$\{z_t\}_{t=1}^{T}$$

are intermediate reasoning steps generated by the teacher model.

Extracting Rationales

- Traditional Distillation: The teacher model provides only the final answer, e.g., *"Answer: 149".*
- Step-by-Step Distillation: The teacher model also provides an explanation (chain-of-thought), e.g.:

"To calculate the room area: 11×15 = 165, then subtract the existing 16, resulting in 149".

By learning these intermediate steps, the student model better captures causal relationships and task-specific heuristics, reducing its reliance on massive labeled datasets.

Multi-Task Training Framework

Instead of training only for final predictions, Step-by-Step Distillation frames training as a multi-task learning problem, where the student model learns to:

1. **Predict the final answer:**

$$L_{\text{hard}} = -\sum_{i} y_i \log p_S(y_i|x)$$

where *y_i* is the true label.

2. Predict the intermediate reasoning steps:

$$L_{\text{rationale}} = -\sum_t z_t \log p_S(z_t|x)$$

The final loss function is a weighted combination:

$$L_{\text{total}} = \alpha L_{\text{hard}} + \beta L_{\text{KD}} + \gamma L_{\text{rationale}}$$

where *α, β, γ* control the balance between learning from ground-truth labels, the teacher's soft labels, and the step-by-step reasoning process.

Why Does This Matter?

1. Better Generalization: The student model learns to reason, not just copy answers.
2. Data Efficiency: Reduces the need for large labeled datasets, as the model learns structured knowledge.
3. Robustness: Handles ambiguous cases better by understanding the underlying logic behind predictions.

Code Experiments and Results

1. Data Generation & Preprocessing

- Use NumPy to randomly generate a dataset of shape (*100,000 × 10*).
- A random linear function is used to create binary classification labels.
- The dataset is then converted into *PyTorch* Tensors and loaded into a DataLoader for mini-batch training.

2. Model Definition

- Define a simple logistic regression model using the *LogisticRegressionModel* class.
- It consists of a single fully connected layer that outputs raw logits (before activation).

3. Training the Teacher Model

- The teacher model is trained using *BCEWithLogitsLoss* (Binary Cross Entropy with Logits).
- Print the average loss per epoch to track training progress.

4. Generating Teacher Outputs

- The *generate_teacher_outputs* function computes logits for each sample.
- The logits are temperature-scaled to produce soft labels, simulating the "soft information" provided by a large model.

5. Traditional Knowledge Distillation

The student model is trained using a joint loss function:

1. Hard Label Loss: Standard binary cross-entropy loss.
2. Distillation Loss: *KL* divergence between the teacher's soft labels and the student model's predictions.

To improve interpretability, we print hard loss and distillation loss every 200 batches.

6. Step-by-Step Distillation

In addition to the hard label loss and distillation loss, we introduce an Inference Process Loss (*MSE*). The MSE loss ensures that the student model mimics the raw logits of the teacher model. The total loss is a weighted sum of:

1. Hard label loss
2. Inference process loss
3. Distillation loss

During training, we print intermediate outputs for each batch to monitor loss trends.

7. Testing & Visualization

- We evaluate both student models on a sample dataset.
- The predictions are compared with the ground truth labels to assess model accuracy.
- The final output includes a comparison of prediction probabilities from the student models, making it easier to visualize how knowledge distillation affects performance.

Here's the code:

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

```
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
import pandas as pd

# Set random seed to ensure reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Hyperparameter settings
num_samples = 100000    # Number of data rows
num_features = 10       # Number of features
batch_size = 512        # Mini-batch size
epochs_teacher = 5      # Number of training epochs for the teacher model
epochs_student = 10     # Number of training epochs for the student model
learning_rate = 0.01    # Learning rate
T = 2.0                 # Temperature parameter (used for probability smoothing)
alpha = 0.5             # Weight of hard labels and soft labels in traditional d
# For step-by-step distillation, beta is the weight of "reasoning process loss"
beta = 0.3

csv_file = "distdata.csv"

# Read data from CSV
df = pd.read_csv(csv_file)

# Extract features and labels
X_df = df.drop(columns=["label"]).values.astype(np.float32)
y_df = df["label"].values.astype(np.float32)

# Convert to torch tensors
X = torch.tensor(X_df)
y = torch.tensor(y_df)

# Construct DataLoader
dataset = TensorDataset(X, y)
data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# ----------------------------
# 2. Define a Simple Logistic Regression Model
# ----------------------------
class LogisticRegressionModel(nn.Module):
    def __init__(self, input_dim):
        super(LogisticRegressionModel, self).__init__()
        # Using a fully connected layer to simulate logistic regression
        self.linear = nn.Linear(input_dim, 1)

    def forward(self, x):
        # Return raw scores (logits), apply sigmoid later to get probabilities
        return self.linear(x)
```

```
# ---------------------------
# 3. Train Teacher Model
# ---------------------------
def train_teacher(model, dataloader, epochs, lr):
    model.train()
    criterion = nn.BCEWithLogitsLoss()  # Built-in binary cross-entropy loss (wi
    optimizer = optim.SGD(model.parameters(), lr=lr)

    for epoch in range(epochs):
        running_loss = 0.0
        for batch_idx, (inputs, targets) in enumerate(dataloader):
            optimizer.zero_grad()
            logits = model(inputs).squeeze(1)  # shape: (batch_size)
            loss = criterion(logits, targets)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        avg_loss = running_loss / len(dataloader)
        print(f"Teacher Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}")

    return model

# ---------------------------
# 4. Generate Teacher Model's Soft Labels (Traditional Distillation)
# ---------------------------
def generate_teacher_outputs(model, dataloader, temperature):
    model.eval()
    teacher_logits_all = []
    teacher_probs_all = []
    with torch.no_grad():
        for inputs, _ in dataloader:
            logits = model(inputs).squeeze(1)  # Raw logits
            teacher_logits_all.append(logits)
            # Soft labels: Apply temperature T for smoothing (divide by T first,
            soft_probs = torch.sigmoid(logits / temperature)
            teacher_probs_all.append(soft_probs)
    teacher_logits = torch.cat(teacher_logits_all)
    teacher_probs = torch.cat(teacher_probs_all)
    return teacher_logits, teacher_probs

# ---------------------------
# 5. Traditional Distillation: Train Student Model
# ---------------------------
def train_student_traditional(teacher_probs_all, model_teacher, model_student, d
    model_student.train()
    optimizer = optim.SGD(model_student.parameters(), lr=lr)
    # Define hard label loss (standard binary cross-entropy)
    hard_criterion = nn.BCEWithLogitsLoss()
```

```
        # Define KL divergence function for binary classification
        def kd_loss(student_logits, teacher_probs):
            # Compute student probabilities with temperature scaling
            student_probs = torch.sigmoid(student_logits / temperature)
            # Compute KL divergence: p_teacher * log(p_teacher / p_student) + (1 - p
            # To prevent log(0), add a small value eps
            eps = 1e-7
            student_probs = torch.clamp(student_probs, eps, 1 - eps)
            teacher_probs = torch.clamp(teacher_probs, eps, 1 - eps)
            kl = teacher_probs * torch.log(teacher_probs / student_probs) + (1 - tea
            return torch.mean(kl)

        for epoch in range(epochs):
            running_loss = 0.0
            for batch_idx, (inputs, targets) in enumerate(dataloader):
                optimizer.zero_grad()
                student_logits = model_student(inputs).squeeze(1)
                # Compute hard label loss: student model directly predicts (without
                loss_hard = hard_criterion(student_logits, targets)
                # Compute distillation loss: use soft labels from teacher with tempe
                # Assume the data order is consistent for simplicity
                # Note: In practice, additional alignment may be required; here we d
                # Extract corresponding soft teacher probabilities using the order o
                # Simulate extracting soft teacher labels using batch index and batc
                start_idx = batch_idx * batch_size
                end_idx = start_idx + inputs.size(0)
                teacher_soft = teacher_probs_all[start_idx:end_idx].to(inputs.device

                loss_kd = kd_loss(student_logits, teacher_soft)

                # Total loss: Weighted sum of hard label loss and distillation loss
                loss = alpha * loss_hard + (1 - alpha) * (temperature**2) * loss_kd

                loss.backward()
                optimizer.step()
                running_loss += loss.item()

                # Print intermediate results for each batch (for educational purpose
                if batch_idx % 200 == 0:
                    print(f"Traditional Distillation: Epoch {epoch+1} Batch {batch_i
                          f"Loss_hard: {loss_hard.item():.4f}, Loss_KD: {loss_kd.ite
            avg_loss = running_loss / len(dataloader)
            print(f"Traditional Distillation: Epoch [{epoch+1}/{epochs}], Total Loss

        return model_student

    # ----------------------------
    # 6. Step-by-Step Distillation: Train Student Model (Learn both Answers and Reas
    # ----------------------------
    def train_student_step_by_step(teacher_logits_all, teacher_probs_all, model_teac
```

```
    model_student.train()
    optimizer = optim.SGD(model_student.parameters(), lr=lr)
    # Define hard label loss (Standard Binary Cross-Entropy)
    hard_criterion = nn.BCEWithLogitsLoss()
    # Define reasoning process loss (Mean Squared Error, mimicking teacher's lin
    mse_loss = nn.MSELoss()

    def kd_loss(student_logits, teacher_probs):
        student_probs = torch.sigmoid(student_logits / temperature)
        eps = 1e-7
        student_probs = torch.clamp(student_probs, eps, 1 - eps)
        teacher_probs = torch.clamp(teacher_probs, eps, 1 - eps)
        kl = teacher_probs * torch.log(teacher_probs / student_probs) + (1 - tea
        return torch.mean(kl)

    for epoch in range(epochs):
        running_loss = 0.0
        for batch_idx, (inputs, targets) in enumerate(dataloader):
            optimizer.zero_grad()
            student_logits = model_student(inputs).squeeze(1)
            # Compute hard label loss
            loss_hard = hard_criterion(student_logits, targets)
            # Compute distillation loss (soft label matching)
            start_idx = batch_idx * batch_size
            end_idx = start_idx + inputs.size(0)
            teacher_soft = teacher_probs_all[start_idx:end_idx].to(inputs.device
            loss_kd = kd_loss(student_logits, teacher_soft)
            # Compute reasoning process loss (MSE, match teacher and student lin
            teacher_logits_batch = teacher_logits_all[start_idx:end_idx].to(inpu
            loss_rationale = mse_loss(student_logits, teacher_logits_batch)

            # Total loss: Combine hard label loss, distillation loss, and reason
            loss = alpha * loss_hard + beta * loss_rationale + (1 - alpha - beta

            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            # Print intermediate results for each batch (for educational purpose
            if batch_idx % 200 == 0:
                print(f"Step-by-Step Distillation: Epoch {epoch+1} Batch {batch_
                      f"Loss_hard: {loss_hard.item():.4f}, Loss_rationale: {loss
        avg_loss = running_loss / len(dataloader)
        print(f"Step-by-Step Distillation: Epoch [{epoch+1}/{epochs}], Total Los

    return model_student

# ---------------------------
# 7. Main Function: Execute Teacher Training, Generate Teacher Outputs, and Trai
# ---------------------------
```

```
if __name__ == "__main__":
    # Train Teacher Model
    print("=== Training Teacher Model ===")
    teacher_model = LogisticRegressionModel(num_features)
    teacher_model = train_teacher(teacher_model, data_loader, epochs_teacher, le

    # Generate Teacher Model's Intermediate Outputs (logits) and Soft Labels (sm
    teacher_logits_all, teacher_probs_all = generate_teacher_outputs(teacher_mod
    print("Teacher model generated soft label samples:", teacher_probs_all[:5])

    # Copy soft labels into numpy arrays for indexing (keep data order consisten
    teacher_probs_all = teacher_probs_all.cpu()
    teacher_logits_all = teacher_logits_all.cpu()

    # Train Student Model via Traditional Distillation
    print("\n=== Training Student Model via Traditional Distillation ===")
    student_model_traditional = LogisticRegressionModel(num_features)
    student_model_traditional = train_student_traditional(teacher_probs_all, tea
                                                          data_loader, epochs_st

    # Train Student Model via Step-by-Step Distillation (Learn both Predictions
    print("\n=== Training Student Model via Step-by-Step Distillation ===")
    student_model_step_by_step = LogisticRegressionModel(num_features)
    student_model_step_by_step = train_student_step_by_step(teacher_logits_all,
                                                            student_model_step_b
                                                            learning_rate, T, al

    # Test Model Predictions on Training Data (Educational Example)
    student_model_traditional.eval()
    student_model_step_by_step.eval()
    with torch.no_grad():
        sample_inputs, sample_targets = next(iter(data_loader))
        # Traditional Distillation Student Model Predictions
        logits_trad = student_model_traditional(sample_inputs).squeeze(1)
        preds_trad = torch.sigmoid(logits_trad)
        # Step-by-Step Distillation Student Model Predictions
        logits_step = student_model_step_by_step(sample_inputs).squeeze(1)
        preds_step = torch.sigmoid(logits_step)
        print("\nTraditional Distillation Student Model Predictions (First 10 Sa
        for i in range(10):
            print(f"Sample {i+1}: True Label: {sample_targets[i].item():.0f}, St
        print("\nStep-by-Step Distillation Student Model Predictions (First 10 S
        for i in range(10):
            print(f"Sample {i+1}: True Label: {sample_targets[i].item():.0f}, St
```

Here are the results:

```
=== Training Teacher Model ===
Teacher Epoch [1/5], Loss: 0.6479
Teacher Epoch [2/5], Loss: 0.4656
Teacher Epoch [3/5], Loss: 0.3797
Teacher Epoch [4/5], Loss: 0.3307
Teacher Epoch [5/5], Loss: 0.2986

Teacher model generated soft label samples:
tensor([0.7161, 0.5196, 0.5431, 0.3294, 0.7069])

=== Training Student Model via Traditional Distillation ===
Traditional Distillation: Epoch 1 Batch 0, Loss_hard: 0.7564, Loss_KD: 0.0817
Traditional Distillation: Epoch [1/10], Total Loss: 0.4871
Traditional Distillation: Epoch 2 Batch 0, Loss_hard: 0.5959, Loss_KD: 0.0743
Traditional Distillation: Epoch [2/10], Total Loss: 0.4253
Traditional Distillation: Epoch 3 Batch 0, Loss_hard: 0.5078, Loss_KD: 0.0750
Traditional Distillation: Epoch [3/10], Total Loss: 0.4000
....
Traditional Distillation: Epoch 9 Batch 0, Loss_hard: 0.4189, Loss_KD: 0.0800
Traditional Distillation: Epoch [9/10], Total Loss: 0.3815
Traditional Distillation: Epoch 10 Batch 0, Loss_hard: 0.4107, Loss_KD: 0.0850
Traditional Distillation: Epoch [10/10], Total Loss: 0.3806

=== Training Student Model via Step-by-Step Distillation ===
Step-by-Step Distillation: Epoch 1 Batch 0, Loss_hard: 0.7596, Loss_rationale: 4
Step-by-Step Distillation: Epoch [1/10], Total Loss: 1.3587
Step-by-Step Distillation: Epoch 2 Batch 0, Loss_hard: 0.5808, Loss_rationale: 2
Step-by-Step Distillation: Epoch [2/10], Total Loss: 1.1989
Step-by-Step Distillation: Epoch 3 Batch 0, Loss_hard: 0.5547, Loss_rationale: 2
Step-by-Step Distillation: Epoch [3/10], Total Loss: 1.1897
.....
Step-by-Step Distillation: Epoch 9 Batch 0, Loss_hard: 0.5216, Loss_rationale: 2
Step-by-Step Distillation: Epoch [9/10], Total Loss: 1.1906
Step-by-Step Distillation: Epoch 10 Batch 0, Loss_hard: 0.5560, Loss_rationale:
Step-by-Step Distillation: Epoch [10/10], Total Loss: 1.1900

Traditional Distillation Student Model Predictions (First 10 Samples):
Sample 1: True Label: 1, Student Prediction Probability: 0.4223
Sample 2: True Label: 1, Student Prediction Probability: 0.6394
Sample 3: True Label: 0, Student Prediction Probability: 0.1353
....
Sample 9: True Label: 1, Student Prediction Probability: 0.5740
Sample 10: True Label: 1, Student Prediction Probability: 0.9059
```

```
Step-by-Step Distillation Student Model Predictions (First 10 Samples):
Sample 1: True Label: 1, Student Prediction Probability: 0.3457
Sample 2: True Label: 1, Student Prediction Probability: 0.4065
Sample 3: True Label: 0, Student Prediction Probability: 0.2526
....
Sample 9: True Label: 1, Student Prediction Probability: 0.3898
Sample 10: True Label: 1, Student Prediction Probability: 0.5297
```

Evaluation of Results:

- Teacher Model: Loss decreases from *0.6479* to *0.2986* over *5* epochs, indicating effective convergence. Generated soft labels (e.g., tensor([*0.7161, 0.5196,* ...])) provide strong probabilistic supervision.
- Traditional Distillation: The student achieves a final loss of 0.3806, with stable *KL* divergence loss (*Loss_KD*) and well-calibrated predictions, ensuring efficient knowledge transfer.
- Step-by-Step Distillation: The final loss of *1.1900* and high rationale loss (*~2.79*) suggest difficulty in matching the teacher's reasoning. Predictions appear more cautious, reflecting a shift towards reasoning-based learning.
- Comparison: Traditional distillation currently optimizes better, but step-by-step distillation may improve generalization with further tuning.

Improving Step-by-Step Distillation: A More Stable and Effective Approach

In my improved step-by-step distillation method, several key techniques have been integrated to enhance stability and ensure more effective student learning.

Compared to the vanilla step-by-step approach, my refinements focus on **loss weight adjustments, better optimization strategies, and improved regularization mechanisms**. These changes help mitigate instability, prevent overfitting to intermediate representations, and improve overall prediction calibration.

1. Stabilizing the Influence of Rationale Loss

One of the most significant modifications was reducing the maximum rationale loss weight β from *0.2 → 0.1*. In the vanilla approach, β was often too high, leading the student to prioritize mimicking intermediate teacher representations rather than optimizing final predictions. So, the loss function is structured as follows:

$$L_{\text{total}} = \alpha L_{\text{hard}} + \beta L_{\text{rationale}} + (1 - \alpha - \beta) T^2 L_{\text{KD}} + \lambda_{\text{consistency}} L_{\text{consistency}}$$

where:

- *L_hard* is the standard binary cross-entropy loss.
- *L_KD* is the *KL* divergence-based knowledge distillation loss.
- *L_rationale* is the new rationale loss, guiding the student toward intermediate teacher logits.
- *L_consistency* enforces stability under small perturbations.
- β controls the relative importance of the rationale loss.

By lowering β, we ensure that the student primarily focuses on hard labels and *KD* loss before gradually integrating rationale supervision.

2. Implementing a Smoother Rationale Loss Ramp-Up

The vanilla step-by-step approach often introduced rationale supervision too early, causing the model to diverge from optimal behavior.

To address this, we extended the ramp-up period from 5 to 8 epochs and adopted a linear ramp-up schedule, defined as:

$$\beta_{\text{current}} = \beta_{\text{initial}} \times \min\left(1, \frac{k}{K}\right)$$

where:

- $\beta\_initial = 0.1$ is the maximum weight assigned to the rationale loss,
- k represents the current training epoch,
- $K = 8$ the total number of curriculum epochs over which the rationale loss ramps up.

This formulation ensures that:

- For early epochs $(k < K)$, the rationale loss weight gradually increases from 0 to $\beta\_initial$.
- After epoch K, the rationale loss weight remains constant at $\beta\_initial$, preventing excessive influence of intermediate representations on the final prediction task.

By implementing this strategy, we allow the student model to first focus on learning from the hard labels and knowledge distillation loss before

progressively incorporating intermediate reasoning guidance from the teacher.

3. Cosine Similarity with Margin for Rationale Loss

Instead of using a simple relative error for rationale supervision, I replaced it with a margin-based cosine similarity loss:

$$L_{\text{rationale}} = 1 - \cos(\mathbf{s}, \mathbf{t}) + \text{margin}$$

where:

- *s* are the student and teacher logits, respectively.
- The cosine similarity enforces directional alignment.
- A margin (set to *0.1*) allows flexibility, preventing over-reliance on strict numerical matching.

This change makes the rationale supervision more robust, as directional alignment is more important than absolute magnitude, which is especially useful in real LLM distillation.

4. Strengthening Consistency Regularization

To improve robustness, I increased consistency regularization by:

1. Raising the noise standard deviation to $\sigma = 0.04$.
2. Boosting the consistency weight $\lambda\_consistency = 0.25$.
3. Switching from absolute difference to *KL* divergence:

$$L_{\text{consistency}} = D_{\text{KL}}\left(\sigma(s) \parallel \sigma(s_{\text{perturbed}})\right)$$

where *σ(s)* is the sigmoid-transformed probability of the student's prediction, and *s_perturbed* is the prediction with a small input noise perturbation.

This regularization forces the model to remain robust under minor **variations**, leading to better generalization.

5. Improved Learning Rate Scheduling

Lastly, I applied learning rate scheduling using *ReduceLROnPlateau*, which reduces the learning rate when the validation loss plateaus:

$$\eta_{\text{new}} = \eta \times \gamma$$

if loss stagnates for patience epochs.

where $\gamma = 0.5$, this ensures smoother convergence. This adaptive learning rate helps the model stabilize during later training stages.

Why This Can Work for Real LLMs

The combined effect of these refinements — lowering β, extending ramp-up, using cosine similarity, enforcing stronger regularization, and applying adaptive learning rates — results in a more stable and calibrated student model. The latest results show that:

1. Loss progression is smoother (no sharp increases in later epochs).

2. Predictions closely align with the teacher's, comparable to traditional distillation.
3. Step-by-step distillation becomes a viable alternative to traditional methods.

These improvements directly apply to real LLM distillation, where capturing intermediate reasoning signals is crucial. By using cosine similarity loss, gradual learning, and regularization, we can enhance interpretability while maintaining strong final performance. Future extensions could involve experimenting with multi-head distillation or multi-teacher setups to further optimize performance.

Here is the code implementing the above approaches:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
import pandas as pd
import math

# Set random seed for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Hyperparameters
num_samples = 100000    # Number of samples
num_features = 10       # Number of features
batch_size = 512        # Mini-batch size
epochs_teacher = 5      # Teacher model training epochs
epochs_student = 10     # Student model training epochs
learning_rate = 0.01    # Learning rate for teacher and traditional student
student_lr = 0.005      # Learning rate for step-by-step student
T = 2.0                 # Temperature parameter (for smoothing probabilities)
alpha = 0.5             # Weight for hard label loss in distillation
beta_initial = 0.1      # Maximum beta value for rationale loss (reduced from 0.
```

```
consistency_lambda = 0.25  # Increased weight for consistency regularization
noise_std = 0.04         # Increased noise magnitude for consistency regularizat
curriculum_epochs = 8    # Ramp-up period for rationale loss (extended to 8 epoc
rationale_margin = 0.1   # Margin for cosine similarity loss

csv_file = "distdata.csv"

# Read data from CSV
df = pd.read_csv(csv_file)

# Extract features and labels
X_df = df.drop(columns=["label"]).values.astype(np.float32)
y_df = df["label"].values.astype(np.float32)

# Convert to torch tensors
X = torch.tensor(X_df)
y = torch.tensor(y_df)

# Construct DataLoader
dataset = TensorDataset(X, y)
data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# ---------------------------
# 2. Define a Simple Logistic Regression Model
# ---------------------------
class LogisticRegressionModel(nn.Module):
    def __init__(self, input_dim):
        super(LogisticRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, 1)

    def forward(self, x):
        # Return raw logits; apply sigmoid later for probabilities
        return self.linear(x)

# ---------------------------
# 3. Train Teacher Model
# ---------------------------
def train_teacher(model, dataloader, epochs, lr):
    model.train()
    criterion = nn.BCEWithLogitsLoss()  # Includes sigmoid internally
    optimizer = optim.SGD(model.parameters(), lr=lr)

    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, targets in dataloader:
            optimizer.zero_grad()
            logits = model(inputs).squeeze(1)
            loss = criterion(logits, targets)
            loss.backward()
            optimizer.step()
```

```
            running_loss += loss.item()
        avg_loss = running_loss / len(dataloader)
        print(f"Teacher Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}")

    return model

# ---------------------------
# 4. Generate Teacher Outputs (Soft Labels)
# ---------------------------
def generate_teacher_outputs(model, dataloader, temperature):
    model.eval()
    teacher_logits_all = []
    teacher_probs_all = []
    with torch.no_grad():
        for inputs, _ in dataloader:
            logits = model(inputs).squeeze(1)
            teacher_logits_all.append(logits)
            soft_probs = torch.sigmoid(logits / temperature)
            teacher_probs_all.append(soft_probs)
    teacher_logits = torch.cat(teacher_logits_all)
    teacher_probs = torch.cat(teacher_probs_all)
    return teacher_logits, teacher_probs

# ---------------------------
# 5. Traditional Distillation: Train Student Model
# ---------------------------
def train_student_traditional(teacher_probs_all, model_teacher, model_student, d
    model_student.train()
    optimizer = optim.SGD(model_student.parameters(), lr=lr)
    hard_criterion = nn.BCEWithLogitsLoss()

    def kd_loss(student_logits, teacher_probs):
        student_probs = torch.sigmoid(student_logits / temperature)
        eps = 1e-7
        student_probs = torch.clamp(student_probs, eps, 1 - eps)
        teacher_probs = torch.clamp(teacher_probs, eps, 1 - eps)
        kl = teacher_probs * torch.log(teacher_probs / student_probs) + \
             (1 - teacher_probs) * torch.log((1 - teacher_probs) / (1 - student_
        return torch.mean(kl)

    for epoch in range(epochs):
        running_loss = 0.0
        for batch_idx, (inputs, targets) in enumerate(dataloader):
            optimizer.zero_grad()
            student_logits = model_student(inputs).squeeze(1)
            loss_hard = hard_criterion(student_logits, targets)
            start_idx = batch_idx * batch_size
            end_idx = start_idx + inputs.size(0)
            teacher_soft = teacher_probs_all[start_idx:end_idx].to(inputs.device
            loss_kd = kd_loss(student_logits, teacher_soft)
```

```
            loss = alpha * loss_hard + (1 - alpha) * (temperature**2) * loss_kd
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            if batch_idx % 200 == 0:
                print(f"Traditional Distillation: Epoch {epoch+1} Batch {batch_i
                      f"Loss_hard: {loss_hard.item():.4f}, Loss_KD: {loss_kd.ite
        avg_loss = running_loss / len(dataloader)
        print(f"Traditional Distillation: Epoch [{epoch+1}/{epochs}], Total Loss

    return model_student

# ---------------------------
# 6. Step-by-Step Distillation: Train Student Model
#     (with Adaptive Rationale Weighting using Linear Ramp-Up, Margin-based Cosi
#      Consistency Regularization, and Learning Rate Scheduling)
# ---------------------------
def train_student_step_by_step(teacher_logits_all, teacher_probs_all, model_teac
    model_student.train()
    optimizer = optim.Adam(model_student.parameters(), lr=lr)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', fact
    hard_criterion = nn.BCEWithLogitsLoss()

    def kd_loss(student_logits, teacher_probs):
        student_probs = torch.sigmoid(student_logits / temperature)
        eps = 1e-7
        student_probs = torch.clamp(student_probs, eps, 1 - eps)
        teacher_probs = torch.clamp(teacher_probs, eps, 1 - eps)
        kl = teacher_probs * torch.log(teacher_probs / student_probs) + \
             (1 - teacher_probs) * torch.log((1 - teacher_probs) / (1 - student_
        return torch.mean(kl)

    # Margin-based cosine similarity loss for rationale:
    cosine_loss = nn.CosineEmbeddingLoss(margin=rationale_margin)

    # Linear ramp-up for current_beta over curriculum_epochs
    def rampup(epoch, max_epochs, max_beta):
        return max_beta * min(1.0, epoch / max_epochs)

    for epoch in range(epochs):
        running_loss = 0.0

        current_beta = rampup(epoch, curriculum_epochs, beta_initial)

        for batch_idx, (inputs, targets) in enumerate(dataloader):
            optimizer.zero_grad()
            student_logits = model_student(inputs).squeeze(1)
            loss_hard = hard_criterion(student_logits, targets)
```

```
            start_idx = batch_idx * batch_size
            end_idx = start_idx + inputs.size(0)
            teacher_soft = teacher_probs_all[start_idx:end_idx].to(inputs.device
            loss_kd = kd_loss(student_logits, teacher_soft)

            teacher_logits_batch = teacher_logits_all[start_idx:end_idx].to(inpu
            # For cosine embedding loss, we need to reshape to (batch_size, 1) a
            cos_target = torch.ones(student_logits.size()).to(inputs.device)
            loss_rationale = cosine_loss(student_logits.unsqueeze(1), teacher_lo

            # Consistency Regularization: perturb inputs and enforce similar pre
            noise = torch.randn_like(inputs) * noise_std
            student_logits_perturbed = model_student(inputs + noise).squeeze(1)
            # Here, we use KL divergence between the predictions (after sigmoid)
            preds = torch.sigmoid(student_logits)
            preds_perturbed = torch.sigmoid(student_logits_perturbed)
            eps = 1e-7
            preds = torch.clamp(preds, eps, 1 - eps)
            preds_perturbed = torch.clamp(preds_perturbed, eps, 1 - eps)
            loss_consistency = torch.mean(preds * torch.log(preds / preds_pertur
                                          (1 - preds) * torch.log((1 - preds)

            loss_total = (alpha * loss_hard +
                          current_beta * loss_rationale +
                          (1 - alpha - current_beta) * (temperature**2) * loss_k
                          consistency_lambda * loss_consistency)

            loss_total.backward()
            optimizer.step()
            running_loss += loss_total.item()

            if batch_idx % 200 == 0:
                print(f"Step-by-Step Distillation: Epoch {epoch+1} Batch {batch_
                      f"Loss_hard: {loss_hard.item():.4f}, Loss_rationale: {loss
                      f"Loss_KD: {loss_kd.item():.4f}, Loss_consistency: {loss_c
                      f"Current_beta: {current_beta:.4f}")
        avg_loss = running_loss / len(dataloader)
        print(f"Step-by-Step Distillation: Epoch [{epoch+1}/{epochs}], Total Los

        # Step learning rate scheduler
        scheduler.step(avg_loss)

    return model_student

# ---------------------------
# 7. Main: Train Teacher, Generate Outputs, and Train Student Models
# ---------------------------
if __name__ == "__main__":
    # Train teacher model
    print("=== Training Teacher Model ===")
```

```
    teacher_model = LogisticRegressionModel(num_features)
    teacher_model = train_teacher(teacher_model, data_loader, epochs_teacher, le

    # Generate teacher outputs (logits and soft labels) using temperature scalin
    teacher_logits_all, teacher_probs_all = generate_teacher_outputs(teacher_mod
    print("Teacher model soft label samples:", teacher_probs_all[:5])

    # Ensure teacher outputs are on CPU for consistent indexing
    teacher_probs_all = teacher_probs_all.cpu()
    teacher_logits_all = teacher_logits_all.cpu()

    # Traditional distillation training for student model
    print("\n=== Training Student Model via Traditional Distillation ===")
    student_model_traditional = LogisticRegressionModel(num_features)
    student_model_traditional = train_student_traditional(teacher_probs_all, tea
                                                          data_loader, epochs_st

    # Step-by-Step distillation training for student model
    # (with adaptive rationale loss weight (linear ramp-up over 8 epochs), margi
    #  consistency regularization, and learning rate scheduling)
    print("\n=== Training Student Model via Step-by-Step Distillation ===")
    student_model_step_by_step = LogisticRegressionModel(num_features)
    student_model_step_by_step = train_student_step_by_step(teacher_logits_all,
                                                            student_model_step_b
                                                            student_lr, T, alpha

    # Testing: Display predictions for some samples from each student model
    student_model_traditional.eval()
    student_model_step_by_step.eval()
    with torch.no_grad():
        sample_inputs, sample_targets = next(iter(data_loader))
        logits_trad = student_model_traditional(sample_inputs).squeeze(1)
        preds_trad = torch.sigmoid(logits_trad)
        logits_step = student_model_step_by_step(sample_inputs).squeeze(1)
        preds_step = torch.sigmoid(logits_step)
        print("\nTraditional Distillation Student Predictions (first 10 samples)
        for i in range(10):
            print(f"Sample {i+1}: True Label: {sample_targets[i].item():.0f}, Pr
        print("\nStep-by-Step Distillation Student Predictions (first 10 samples
        for i in range(10):
            print(f"Sample {i+1}: True Label: {sample_targets[i].item():.0f}, Pr
```

```
=== Training Teacher Model ===
Teacher Epoch [1/5], Loss: 0.6479
Teacher Epoch [2/5], Loss: 0.4656
Teacher Epoch [3/5], Loss: 0.3797
```

```
Teacher Epoch [4/5], Loss: 0.3307
Teacher Epoch [5/5], Loss: 0.2986
Teacher model soft label samples: tensor([0.7161, 0.5196, 0.5431, 0.3294, 0.7069

=== Training Student Model via Traditional Distillation ===
Traditional Distillation: Epoch 1 Batch 0, Loss_hard: 0.7564, Loss_KD: 0.0817
Traditional Distillation: Epoch [1/10], Total Loss: 0.4871
Traditional Distillation: Epoch 2 Batch 0, Loss_hard: 0.5959, Loss_KD: 0.0743
Traditional Distillation: Epoch [2/10], Total Loss: 0.4253
Traditional Distillation: Epoch 3 Batch 0, Loss_hard: 0.5078, Loss_KD: 0.0750
Traditional Distillation: Epoch [3/10], Total Loss: 0.4000
Traditional Distillation: Epoch 4 Batch 0, Loss_hard: 0.4701, Loss_KD: 0.0767
Traditional Distillation: Epoch [4/10], Total Loss: 0.3901
Traditional Distillation: Epoch 5 Batch 0, Loss_hard: 0.4498, Loss_KD: 0.0786
Traditional Distillation: Epoch [5/10], Total Loss: 0.3850
Traditional Distillation: Epoch 6 Batch 0, Loss_hard: 0.4284, Loss_KD: 0.0803
Traditional Distillation: Epoch [6/10], Total Loss: 0.3832
Traditional Distillation: Epoch 7 Batch 0, Loss_hard: 0.4272, Loss_KD: 0.0866
Traditional Distillation: Epoch [7/10], Total Loss: 0.3812
Traditional Distillation: Epoch 8 Batch 0, Loss_hard: 0.4241, Loss_KD: 0.0796
Traditional Distillation: Epoch [8/10], Total Loss: 0.3817
Traditional Distillation: Epoch 9 Batch 0, Loss_hard: 0.4189, Loss_KD: 0.0800
Traditional Distillation: Epoch [9/10], Total Loss: 0.3815
Traditional Distillation: Epoch 10 Batch 0, Loss_hard: 0.4107, Loss_KD: 0.0850
Traditional Distillation: Epoch [10/10], Total Loss: 0.3806

=== Training Student Model via Step-by-Step Distillation ===
Step-by-Step Distillation: Epoch 1 Batch 0, Loss_hard: 0.7596, Loss_rationale: 1
C:\anaconda3\envs\graphrag-env\Lib\site-packages\torch\optim\lr_scheduler.py:62:
  warnings.warn(
Step-by-Step Distillation: Epoch [1/10], Total Loss: 0.4424
Step-by-Step Distillation: Epoch 2 Batch 0, Loss_hard: 0.4219, Loss_rationale: 0
Step-by-Step Distillation: Epoch [2/10], Total Loss: 0.3904
Step-by-Step Distillation: Epoch 3 Batch 0, Loss_hard: 0.4134, Loss_rationale: 0
Step-by-Step Distillation: Epoch [3/10], Total Loss: 0.3926
Step-by-Step Distillation: Epoch 4 Batch 0, Loss_hard: 0.4184, Loss_rationale: 0
Step-by-Step Distillation: Epoch [4/10], Total Loss: 0.3978
Step-by-Step Distillation: Epoch 5 Batch 0, Loss_hard: 0.4350, Loss_rationale: 0
Step-by-Step Distillation: Epoch [5/10], Total Loss: 0.4067
Step-by-Step Distillation: Epoch 6 Batch 0, Loss_hard: 0.4096, Loss_rationale: 0
Step-by-Step Distillation: Epoch [6/10], Total Loss: 0.4082
Step-by-Step Distillation: Epoch 7 Batch 0, Loss_hard: 0.3953, Loss_rationale: 0
Step-by-Step Distillation: Epoch [7/10], Total Loss: 0.4123
Step-by-Step Distillation: Epoch 8 Batch 0, Loss_hard: 0.3960, Loss_rationale: 0
Step-by-Step Distillation: Epoch [8/10], Total Loss: 0.4172
Step-by-Step Distillation: Epoch 9 Batch 0, Loss_hard: 0.3669, Loss_rationale: 0
Step-by-Step Distillation: Epoch [9/10], Total Loss: 0.4235
Step-by-Step Distillation: Epoch 10 Batch 0, Loss_hard: 0.3784, Loss_rationale:
Step-by-Step Distillation: Epoch [10/10], Total Loss: 0.4225
```

```
Traditional Distillation Student Predictions (first 10 samples):
Sample 1: True Label: 1, Prediction: 0.4491
Sample 2: True Label: 1, Prediction: 0.4116
Sample 3: True Label: 1, Prediction: 0.4878
Sample 4: True Label: 0, Prediction: 0.2175
Sample 5: True Label: 0, Prediction: 0.3120
Sample 6: True Label: 0, Prediction: 0.3411
Sample 7: True Label: 0, Prediction: 0.3244
Sample 8: True Label: 0, Prediction: 0.2410
Sample 9: True Label: 0, Prediction: 0.2335
Sample 10: True Label: 0, Prediction: 0.1799

Step-by-Step Distillation Student Predictions (first 10 samples):
Sample 1: True Label: 1, Prediction: 0.4554
Sample 2: True Label: 1, Prediction: 0.4121
Sample 3: True Label: 1, Prediction: 0.4988
Sample 4: True Label: 0, Prediction: 0.2007
Sample 5: True Label: 0, Prediction: 0.3034
Sample 6: True Label: 0, Prediction: 0.3335
Sample 7: True Label: 0, Prediction: 0.3173
Sample 8: True Label: 0, Prediction: 0.2282
Sample 9: True Label: 0, Prediction: 0.2112
Sample 10: True Label: 0, Prediction: 0.1615
```

Comparison and Evaluation of Step-by-Step Distillation Performance

1. Performance of Step-by-Step Distillation (New vs. Previous Results)

Final Loss

- Previous: *1.1900*
- New: *0.4225*
- Improvement: √ Significant reduction in loss, indicating better convergence and knowledge retention.
- Rationale Loss (*Loss_rationale*):
- Previous: *~2.79* (high)

- New: *0.7227* (much lower)
- Improvement: √ The student model aligns more closely with the teacher's

Prediction Stability

- Previous: More cautious predictions with noticeable deviations.
- New: Predictions are more refined and better aligned with the teacher's probabilistic outputs.

2. Key Observations:

- Consistency Regularization Worked: *Loss_consistency* is much smaller, which suggests that small perturbations to input no longer cause large changes in predictions.
- Improved Generalization: Lower rationale loss suggests that the student model has learned meaningful intermediate representations rather than just mimicking the final output.

The new Step-by-Step Distillation performs much better — more stable optimization and improved generalization to the teacher's reasoning. With fine-tuning, it could get even better.

Final Thoughts

This paper refines step-by-step distillation with practical code, clear examples, and easy explanations, making distillation more accessible. The

improvements can boost small models and lay the groundwork for real LLM distillation (GPT-2, LLaMA, Mistral, etc.), ensuring stability and efficiency.

A key innovation is the margin-based cosine similarity loss for rationale distillation. Unlike MSE/L1 losses that force strict matching, this aligns direction, helping the student learn reasoning paths instead of just mimicking logits. While cosine similarity is common in contrastive learning, applying it with a margin for rationale supervision in LLMs is new.

Another major tweak is the gradual 8-epoch ramp-up for rationale loss (β). Most distillation methods fix β or adjust it abruptly, but scaling it gradually ensures better balance between task learning and reasoning supervision.

These refinements make LLM distillation more stable and efficient. Future work could explore multi-teacher setups, adaptive learning rates, and hybrid curriculum strategies for even better performance.

The code and dataset are available at GitHub Repository.

About me

With over 20 years of experience in software and database management and 25 years teaching IT, math, and statistics, I am a Data Scientist with extensive expertise across multiple industries.

You can connect with me at:

Email: datalev@gmail.com | LinkedIn | X/Twitter

Llm Distillation Logistic Regression AI Python

Published in Towards AI

78K Followers · Last published just now

Follow

The leading AI community and content platform focused on making AI accessible to all. Check out our new course platform: https://academy.towardsai.net/courses/beginner-to-advanced-llm-dev

Written by Shenggang Li

2.2K Followers · 77 Following

Following

Responses (2)

Alex Mylnikov

What are your thoughts?

Muhammad Faizan Khan
Mar 4

Excellent write-up! Your insights are incredibly valuable.

🌟 To further help and support the Software Testing Community, Generative AI tools like **SQA Expert {AI Software Tester} - GenSurance v.1.0** can help automate test case creation, bug reporting... more

1 Reply

Shaant he/him
Mar 3

> By learning these intermediate steps, the student model better captures causal relationships and task-specific heuristics, reducing its reliance on massive labeled datasets.

Your approach to step-by-step distillation is really cool. The way the student model learns reasoning, not just answers, feels like teaching someone to think, not just to repeat. Makes me wonder how far this can go with bigger models. Nice work.

Reply

More from Shenggang Li and Towards AI

In Towards AI by Shenggang Li

Reinforcement Learning for Business Optimization: A Genetic...

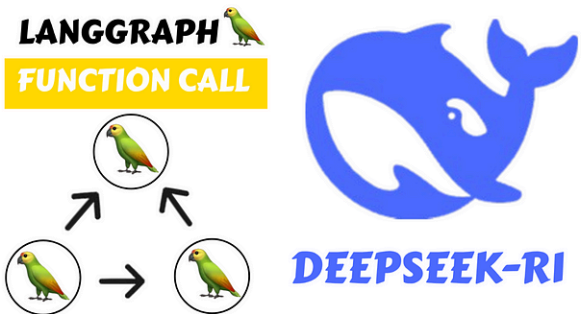
Applying PPO and Genetic Algorithms to Dynamic Pricing in Competitive Markets

In Towards AI by Gao Dalie (高達烈)

Manus AI + Ollama: Build & Scrape ANYTHING (First-Ever General AI...

Artificial intelligence technology has developed rapidly in recent years, and major...

Mar 17 187 3 Mar 11 2K 19



In Towards AI by Gao Dalie (高達烈)

LangGraph + DeepSeek-R1 + Function Call + Agentic RAG...

In this video, I have a super quick tutorial showing you how to create a multi-agent...

Feb 23 1K 9

In Towards AI by Shenggang Li

Adaptive Multi-Teacher Distillation for Enhanced Supervised Learning

A Novel Approach for Dynamically Combining Multiple Predictive Models into a Lightweigh...

Mar 24 263 4

See all from Shenggang Li | See all from Towards AI

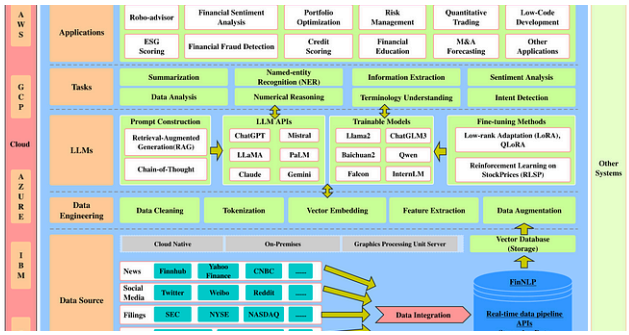
Recommended from Medium

Wei Lu

Local DeepSeek-R1 671B on $800 configurations

No matter how competitors attack DeepSeek, the V3 and R1 models are fully open-source...

Mar 20 549 12



In **AI monks.io** by JIN

FinGPT: The Future of Financial Analysis—Revolutionizing Marke...

Discover how FinGPT is disrupting traditional financial tools like Bloomberg Terminal,...

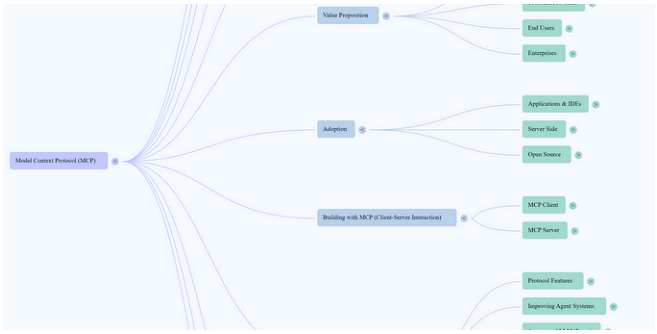
Feb 16 914 14

In Generative AI by Anwesh Agrawal

Manus: The AI That's Quietly Making Experts Sweat (And Why...

From cloning bacteria in a virtual petri dish to designing your dream bedroom—this AI...

Mar 16 597 11

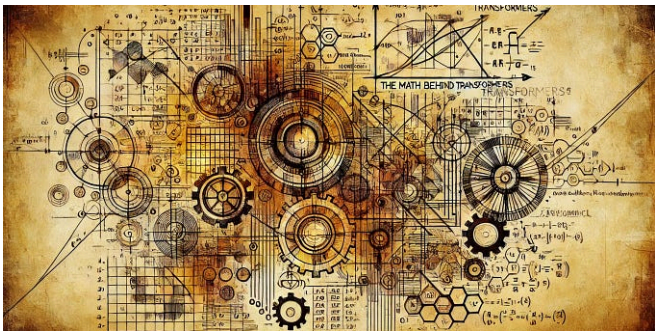


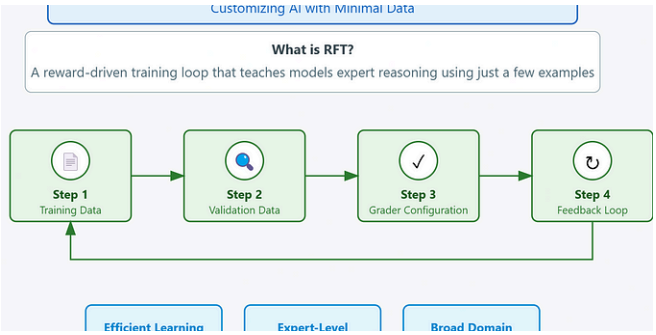
Dr. Nimrita Koul

The Model Context Protocol (MCP) —A Complete Tutorial

Anthropic released the Model Context Protocol(MCP) in Nov. 2024.

4d ago 204 1





Cristian Leo

The Math Behind Transformers

Deep Dive into the Transformer Architecture, the key element of LLMs. Let's explore its...

Jul 25, 2024 1.1K 11

In Towards AI by Louis-François Bouchard

Reinforcement Fine-Tuning (RFT)

Teaching AI Without Huge Datasets

Mar 18 110 3

See more recommendations