Data Science Collective

Member-only story

Attention-based Neural Network Distillation: Enhancing Performance through Learned Weighted Aggregation of Ensemble Models

Shenggang Li · Subscribed
Published in Data Science Collective · 8 min read · 12 hours ago

2

A Novel Framework Demonstrating Empirical Benefits and Probabilistic Interpretations via Neural Attention in Teacher-Student Distillation

Photo by Girl with red hat on Unsplash

Introduction

Can we boost prediction accuracy by combining multiple machine learning models? Most traditional distillation methods stick to just one teacher model, meaning they miss out on the strengths (and uncertainties) of others. In this post, I tackle that limitation by introducing an attention-based distillation approach that pulls together insights from multiple teachers — so the final model gets the best of all worlds.

Why use attention? Standard distillation doesn't handle multiple teachers well — it treats them equally or uses fixed weights, which doesn't work in real-world scenarios. My method uses attention to dynamically consider which teacher should have more say in each prediction. For example, in marketing data, one model might be great at handling holiday behavior while another is better for regular weekdays. The attention mechanism adjusts on the fly based on the context, leading to better performance overall.

I used powerful teacher models — *XGBoost* and *Random Forest* — within a neural network student model. The code experiments show the improved performance through metrics such as *AUC*, accuracy, and *KS* scores. These promising results highlight the potential of attention-based distillation methods. Moving forward, adding uncertainty modeling (like Bayesian methods) could make this even more powerful, especially in areas like finance, healthcare, and e-commerce, where accurate predictions matter.

Algorithmic Framework and Mechanisms

Attention-Based Neural Distillation Framework

In traditional teacher-student distillation, we typically transfer knowledge from a single, high-performance "teacher" model to a "student" model by leveraging soft predictions from the teacher. However, this method is limited because it doesn't adequately utilize the diverse strengths of multiple teacher models. My approach enhances the traditional teacher-student framework by introducing an attention-based neural network mechanism, allowing the student model to dynamically and adaptively aggregate information from multiple teachers.

Attention Mechanism and Weighted Aggregation

The core idea behind attention mechanisms in neural networks is inspired by human cognitive processes, where focus shifts dynamically based on contextual importance. Given a set of teacher predictions *Y_j*, the attention mechanism learns to assign weights *a_j(X)* adaptively based on input features *X*. These weights are computed via a *softmax* function:

$$a_j(X) = \frac{\exp(F_j(X))}{\sum_{k=1}^{m} \exp(F_k(X))}$$

where *F_j(x)* represents the output of an intermediate neural layer specifically trained to gauge each teacher model's predictive reliability given input *X*. This ensures that the aggregation dynamically emphasizes the most reliable predictions for each unique scenario, greatly enhancing predictive robustness.

For instance, in marketing applications, if one teacher excels in predicting customer behavior during holiday seasons while another teacher better captures customer dynamics on weekdays, the attention weights will adapt accordingly, optimizing predictions by appropriately blending these strengths.

Student Model Formulation and Distillation Loss

The student model itself is structured as a neural network with a shared embedding representation *h = G(x)*, followed by two distinct outputs: one directly predicting the target and another performing weighted aggregation of the teacher predictions. This combined output ensures a balance between the student's independent predictions and the teachers' aggregated wisdom. Formally, the final prediction *y^(X)* of the student model is defined as:

$$\hat{y}(X) = \frac{1}{2}\left[\sigma(W_s \cdot h) + \sum_{j=1}^{m} a_j(X) \cdot Y_j(X)\right]$$

where *σ* is the sigmoid activation, *W_s* represents the student's prediction layer weights, and *Y_j(X)* denotes the predictions from the teacher model.

The training of this student model involves a combination of two key losses: Binary Cross Entropy (*BCE*) and Kullback-Leibler (*KL*) divergence. The BCE loss encourages accuracy in direct predictions from the student, defined as:

$$L_{BCE}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})]$$

while the *KL* divergence loss guides the student model toward the distribution provided by the teachers, fostering a meaningful knowledge transfer:

$$L_{KL}(Y_{mix}, \hat{y}) = \sum Y_{mix} \log\left(\frac{Y_{mix}}{\hat{y}}\right)$$

where *Y_mix* represents the dynamically aggregated teacher predictions weighted by attention scores. Balancing these losses ensures that the model captures both the accuracy and consistency of learned knowledge.

Justification and Applications

The rationale for designing the attention-based aggregation is its flexibility and adaptivity compared to static aggregation methods. Rather than relying on fixed weights or averaging, this framework intelligently learns from data, determining how much each teacher contributes under different conditions. Mathematically, the adaptiveness of attention weights is akin to introducing a learned gating mechanism similar to mixture-of-experts models but simplified into a distillation context.

Code Experiment

I created a synthetic dataset inspired by promotional campaign data to test the attention-based neural distillation framework. It includes features like age, income, days since last interaction, loyalty score, holiday flag, and marketing channel. The data structure mimics e-commerce behavior while protecting privacy.

I will combine predictions from two strong teacher models — *XGBoost* and *Random Forest* — into a single student model using an attention-based neural network. The student learns to weigh each teacher's output differently depending on the input. I train on non-promo data to avoid bias and test on promo data. Key metrics like *AUC*, accuracy, and *KS* score help us measure how well the model performs and how attention shifts based on different inputs.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import roc_auc_score, accuracy_score
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import ks_2samp
import torch
import torch.nn as nn
import torch.optim as optim

# Load data
df = pd.read_csv('sales.csv')

train_df = df[df.Promo == 0]
test_df = df[df.Promo == 1]

# Preprocess
ohe = OneHotEncoder()
ch_train = ohe.fit_transform(train_df[["Channel"]]).toarray()
ch_test = ohe.transform(test_df[["Channel"]]).toarray()

X_train = pd.concat([
    train_df[["Age", "Income", "Days", "Holiday", "Loyalty"]].reset_index(drop=T
    pd.DataFrame(ch_train).reset_index(drop=True)
], axis=1)
y_train = train_df["Purchase"].values

X_test = pd.concat([
    test_df[["Age", "Income", "Days", "Holiday", "Loyalty"]].reset_index(drop=Tr
    pd.DataFrame(ch_test).reset_index(drop=True)
], axis=1)
y_test = test_df["Purchase"].values

X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.2, ran

# Teacher models: XGB + RF
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss').fit(X_tr, y_
rf = RandomForestClassifier().fit(X_tr, y_tr)
teachers = {"xgb": xgb, "rf": rf}
```

```
def get_teacher_preds(X):
    return np.vstack([
        xgb.predict_proba(X)[:, 1],
        rf.predict_proba(X)[:, 1]
    ]).T

tp_tr = get_teacher_preds(X_tr)
tp_val = get_teacher_preds(X_val)
tp_test = get_teacher_preds(X_test)

# Bayesian Distillation Model
class BayesianDistillNN(nn.Module):
    def __init__(self, input_dim, n_teachers):
        super().__init__()
        self.shared = nn.Sequential(nn.Linear(input_dim, 64), nn.ReLU(), nn.Line
        self.attn = nn.Sequential(nn.Linear(32, n_teachers), nn.Softmax(dim=1))
        self.student = nn.Sequential(nn.Linear(32, 1), nn.Sigmoid())

    def forward(self, x, teacher_probs):
        h = self.shared(x)
        a = self.attn(h)
        t_out = torch.sum(a * teacher_probs, dim=1, keepdim=True)
        s_out = self.student(h)
        out = 0.5 * (s_out + t_out)
        return out, a

# 6. Torch data
X_tr_t = torch.tensor(X_tr, dtype=torch.float32)
y_tr_t = torch.tensor(y_tr, dtype=torch.float32).unsqueeze(1)
tp_tr_t = torch.tensor(tp_tr, dtype=torch.float32)
X_val_t = torch.tensor(X_val, dtype=torch.float32)
y_val_t = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)
tp_val_t = torch.tensor(tp_val, dtype=torch.float32)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
tp_test_t = torch.tensor(tp_test, dtype=torch.float32)

# Train Distill Model
model = BayesianDistillNN(X_tr.shape[1], 2)
optimizer = optim.Adam(model.parameters(), lr=0.0005)
bce = nn.BCELoss()
best_auc, patience = 0, 10
for epoch in range(50):
    model.train()
    optimizer.zero_grad()
    student_pred, att_w = model(X_tr_t, tp_tr_t)
    teacher_mix = torch.sum(att_w * tp_tr_t, dim=1, keepdim=True)
    kl_loss = torch.mean(teacher_mix * torch.log((teacher_mix + 1e-7) / (student
    loss = bce(student_pred, y_tr_t) + 0.3 * kl_loss
    loss.backward()
```

```
        optimizer.step()
        with torch.no_grad():
            model.eval()
            val_pred, _ = model(X_val_t, tp_val_t)
            val_auc = roc_auc_score(y_val, val_pred.numpy())
            if val_auc > best_auc:
                best_auc = val_auc
                torch.save(model.state_dict(), 'best_bayes.pt')
                patience = 10
            else:
                patience -= 1
                if patience == 0: break
        print(f"Epoch {epoch+1}, Loss {loss.item():.4f}, Val AUC {val_auc:.4f}")

    # Evaluate Final Model
    model.load_state_dict(torch.load('best_bayes.pt'))
    model.eval()
    with torch.no_grad():
        test_pred, att = model(X_test_t, tp_test_t)
        pred_np = test_pred.numpy()
        auc = float(roc_auc_score(y_test, pred_np))
        acc = float(accuracy_score(y_test, pred_np > 0.5))
        ks = float(ks_2samp(pred_np[y_test == 1], pred_np[y_test == 0])[0])
        att_mean = att.numpy().mean(axis=0)

    print(f"\nBayesian Distillation Test Results\nAUC: {auc:.4f}, Accuracy: {acc:.4f
    print(f"Avg Teacher Attention Weights (XGB, RF): {att_mean}")

    # Evaluate Each Teacher
    print("\n--- Individual Teacher Model Performance ---")
    for name, model_t in teachers.items():
        prob = model_t.predict_proba(X_test)[:, 1]
        auc = roc_auc_score(y_test, prob)
        acc = accuracy_score(y_test, prob > 0.5)
        ks_stat = ks_2samp(prob[y_test == 1], prob[y_test == 0])[0]
        print(f"{name.upper()} - AUC: {auc:.4f}, Accuracy: {acc:.4f}, KS: {ks_stat:.
```

Here are the results:

```
###########Bayesian Distillation model#########################
Epoch 1, Loss 0.4821, Val AUC 0.7915
Epoch 2, Loss 0.4796, Val AUC 0.7916
Epoch 3, Loss 0.4771, Val AUC 0.7916
Epoch 4, Loss 0.4746, Val AUC 0.7916
Epoch 5, Loss 0.4722, Val AUC 0.7916
Epoch 6, Loss 0.4697, Val AUC 0.7915
Epoch 7, Loss 0.4672, Val AUC 0.7915
Epoch 8, Loss 0.4648, Val AUC 0.7915
Epoch 9, Loss 0.4623, Val AUC 0.7914
Epoch 10, Loss 0.4599, Val AUC 0.7913
Epoch 11, Loss 0.4575, Val AUC 0.7913
Epoch 12, Loss 0.4550, Val AUC 0.7912
Epoch 13, Loss 0.4526, Val AUC 0.7911

Bayesian Distillation Test Results
AUC: 0.7928, Accuracy: 0.8996, KS: 0.4505

Avg Teacher Attention Weights (XGB, RF): [0.48274437 0.51725656]:

--- Individual Teacher Model Performance ---
XGB - AUC: 0.7889, Accuracy: 0.5143, KS: 0.4352
RF - AUC: 0.7691, Accuracy: 0.5077, KS: 0.4079
```

Model Performance and Insights

The attention-based Bayesian distillation model achieved strong results, with a test *AUC* of *0.7928*, an accuracy of *0.8996*, and a *KS* score of *0.4505*. These metrics indicate that the student model not only matches but also slightly outperforms both teacher models individually. The learning curve also shows consistent loss reduction and stable validation *AUC*, which reflects good generalization without overfitting.

Comparing individual teachers, *XGBoost* reached an *AUC* of *0.7889* and *KS* of

with threshold-based classification despite good probabilistic ranking.

Interestingly, the learned attention weights *48%* for *XGB* and *52%* for *RF* — showing the model treated both teachers as equally valuable, with a slight preference for RF. This dynamic weighting allowed the student model to adaptively blend strengths from both sources, leading to better-balanced and more reliable predictions overall. This result shows the power of learned aggregation over fixed ensemble methods.

Final Thoughts

This framework has strong real-world potential. It improves prediction accuracy in areas like customer analytics, fraud detection, credit scoring, and personalized marketing. Our experiments show clear gains in *AUC*, accuracy, and *KS* scores.

By using distillation, the student model doesn't treat all teacher predictions the same — it learns which teacher to trust in different situations. Think of it as a smart assistant that picks the right expert for each task.

The result is a compact model that captures the strengths of larger, more complex models. It performs well, generalizes better, and is suitable for real-time applications where speed and size matter.

Looking ahead, the framework can be enhanced with uncertainty estimation, domain adaptation, or online learning, where it keeps adjusting over time. Bayesian distillation isn't just averaging — it's learning how to blend intelligently.

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

Machine Learning · Distillation · Xgboost · Random Forest · Python

Published in Data Science Collective

835K Followers · Last published 12 hours ago

Advice, insights, and ideas from the Medium data science community

Follow

Written by Shenggang Li

2.3K Followers · 77 Following

Subscribed

No responses yet

Alex Mylnikov

What are your thoughts?

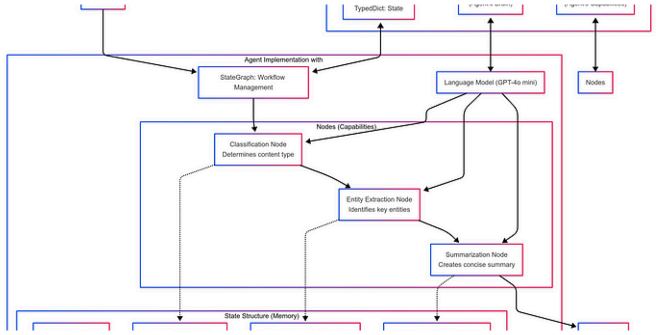
More from Shenggang Li and Data Science Collective

In Towards AI by Shenggang Li

Reinforcement Learning for Business Optimization: A Genetic...

Applying PPO and Genetic Algorithms to Dynamic Pricing in Competitive Markets

Mar 17 187 3


TypedDict: State
Agent Implementation with
StateGraph: Workflow Management
Language Model (GPT-4o mini)
Nodes
Nodes (Capabilities)
Classification Node Determines content type
Entity Extraction Node Identifies key entities
Summarization Node Creates concise summary
State Structure (Memory)

In Data Science Collective by Paolo Perrone

The Complete Guide to Building Your First AI Agent with...

Three months into building my first commercial AI agent, everything collapsed...

Mar 11 2.8K 56

In Data Science Collective by Buse Şenol

Model Context Protocol (MCP): An End-To-End Tutorial With Hands-...

What is MCP? How to create an MPC Server that brings news from a web site with Claude...

In Towards AI by Shenggang Li

Reinforcement Learning-Enhanced Gradient Boosting Machines

A Novel Approach to Integrating Reinforcement Learning within Gradient...

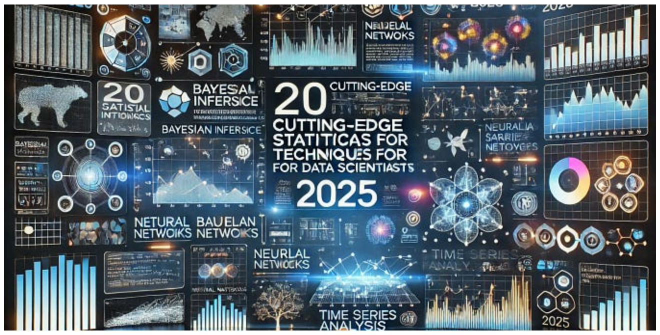
Mar 18 1.1K 12

Apr 1 276 3

See all from Shenggang Li

See all from Data Science Collective

Recommended from Medium


20 CUTTING-EDGE STATITICAS FOR TECHNIQUES FOR FOR DATA SCIENTIASTS 2025
BAYESAL INFERSICE
BAYESIAN INFERSICE
NETURAL BAUELAN NETWOIKS NETWOKS
NEURLAL NETWOKS
TIME SERIES ANALYSIS

The Data Beast

20 Cutting-Edge Statistical Techniques Every Data Scientist…

In today's fast-paced data world, traditional methods are evolving rapidly. In 2025, the…

Mar 7 211 2

In Data Science Collecti… by Bradley Stephen Sh…

Teach Your GBM to Extrapolate with Model Stacking

Background

5d ago 170 4

In Towards AI by Shenggang Li

Reinforcement Learning-Enhanced Gradient Boosting Machines

A Novel Approach to Integrating Reinforcement Learning within Gradient...

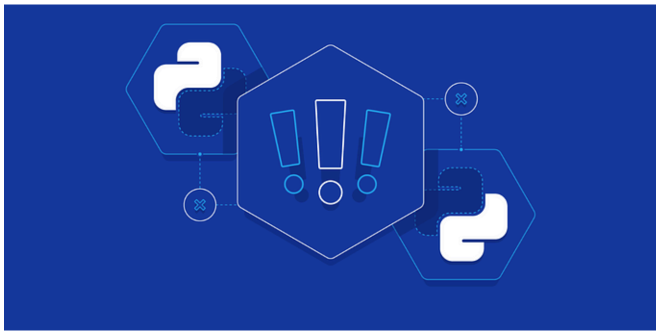
Apr 1 276 3

In The Pythoneers by Abhay Parashar

How to Explain Each Core Machine Learning Model in an Interview

From Regression to Clustering to CNNs: A Brief Guide to 25+ Machine Learning Models

Mar 12 1.97K 26



In Coding Nexus by Algo Insights

8 Confusing Python Concepts That Frustrate Most Developers

Python is one of the most beginner-friendly programming languages. But even...

Feb 13 243 4

In Into Quantum by Dr. Ashish Bamania

An Introduction To Eigenvectors & Eigenvalues Towards Quantum...

Learn the mathematical basis of quantum measurements by performing them by hand.

5d ago 11 1

See more recommendations