Sentence Prediction AI

- Understanding Neural Networks and the basic Fundamentals of Machine Learning and AI

By Kobe Taylor

Slideshow Overview

Project Scope and Overview

Understand the project incrementally

- Learn n-grams
- Learn Tokenization/Regex
- Learn new approaches to coding - Streamlit and SQLite
- PyTorch

Basic N-Gram Model → Neural Network (GRU)

Why This Direction and Idea?

- AI is inevitable and growing
- Challenge Myself
- Curiosity
- Relevance

Early Project History and Fundamentals

Began with the most basic predictive algorithm

- N-Grams
- Storing my sentences and effective handling
  - SQLite and Upserting
- Tokenization/Regex
  - Important for my predictions
- Frequency Based over Vector Based

Basic Implementation of UI

- Built-in Python Framework → Streamlit
- First instance of tangibility of my code and visualization of it as well

The Mathematical Structure of Machine Learning - Vectors, Weights, Embedding, BackPropagation

- Vectors
  - Vectors are just lists of numbers the model uses to represent information.
  - Every word gets turned into a numerical vector so the neural network can do math on it.
- Gradient
  - measures how much the loss would change if a weight changed slightly.
  - Think of it as the model asking:
  - "Which direction should I move this weight to make the prediction less wrong?"
  - During training, gradients guide how far and in what direction each weight should be updated.
- Weights
  - Weights are the adjustable parameters inside the neural network that determine how strongly inputs influence outputs.
  - During training, these weights change to reduce prediction errors.
- Embeddings
  - An embedding is a learned vector that represents a word's meaning.
  - Similar words (cat/dog) end up with similar vectors because the model learns patterns from the training data.
- Backpropagation
  - The process where the model looks at its mistakes and adjusts its weights in the right direction.
  - PyTorch handles the math automatically, but understanding it explains how the model learns over time.

Transition into Neural Networks - GRU and PyTorch

- What is GRU?
  - A recurrent neural network that uses update and reset gates to remember context
    - Update Gate: decides how much of the previous state should be kept.
    - Reset Gate: decides how much of the old information to forget.
  - Learns patterns across sequences (ex: "I like …")
  - Like backpropagation, python does this for us via nn.GRU
- What is PyTorch?
  - Deep learning framework used to build and train neural networks
  - Handles tensors, automatic differentiation, GPU acceleration
    - A tensor is just a multi-dimensional container for numbers — like a generalization for our vectors
      - how the GRU actually receives and processes language. Words like "cat" or "pizza" cannot be fed into the model directly, so each word is converted into an integer ID, and those IDs are placed into tensors.
      - Without tensors, none of the math behind training or prediction would be possible. They are the basic data format that makes the entire neural network operate.
      - PyTorch expects all model inputs, weights, and outputs to be in tensor form — this is how it performs the massive amount of math required for training. Regular Python types like integers, lists, or strings can't store gradients, can't run on the GPU, and can't be optimized during training. Without tensors, the model would not be able to compute predictions, measure loss, or update weights.
  - Used to build embeddings, GRU layers, calculation of loss

Transition into Neural Networks - GRU and PyTorch cont.

Layering

- Embedding layer
- GRU layer
    - Layering refers to how different parts of the neural network stack together, each transforming the input in a different way before passing it to the next stage. In my project, the first layer is the Embedding layer, which converts each word ID into a dense vector so the model can understand words as mathematical relationships instead of plain numbers. The next layer is the GRU layer, which reads these word vectors one at a time and learns how the words relate to each other across the sequence — essentially forming a memory of the two-word context.

Training Loop Structure

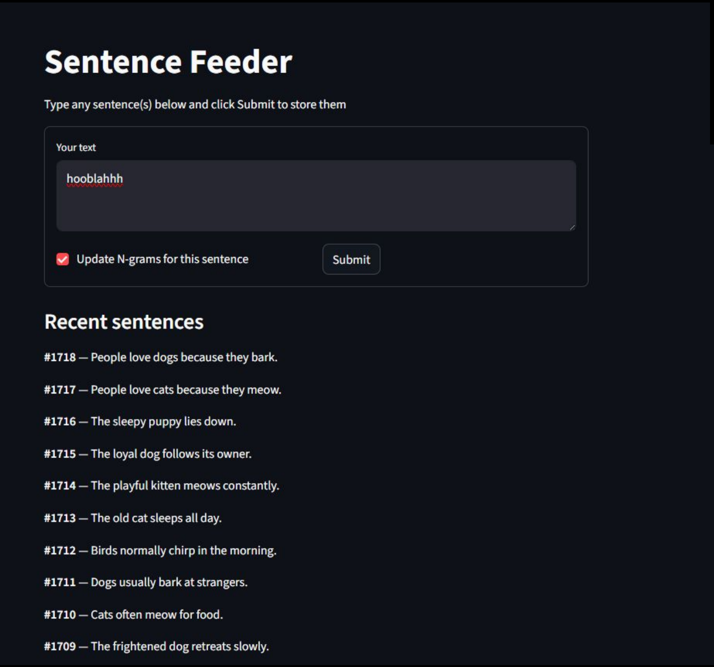
- Forward pass → model predicts next word
- Compute loss → compare prediction vs true word
- Backward pass → compute gradients with autograd
- Update weights
- Repeat over many epochs

Beginning to Train/ Data Sets

Adding data and finding earlier signs of success

Sentence Feeder
- Our first app to input data

Neural GRU Word Predicator
- Two Predictions
  - N-Gram
    - Our basic model, frequency based
  - Neural
    - Probabilities based on our vectors based on our training model and smoothing



Sentence Feeder

Type any sentence(s) below and click Submit to store them

Your text

hooblahhh

[✓ Update N-grams for this sentence]

Recent sentences

#1718 — People love dogs because they bark.

#1717 — People love cats because they meow.

#1716 — The sleepy puppy lies down.

#1715 — The loyal dog follows its owner.

#1714 — The playful kitten meows constantly.

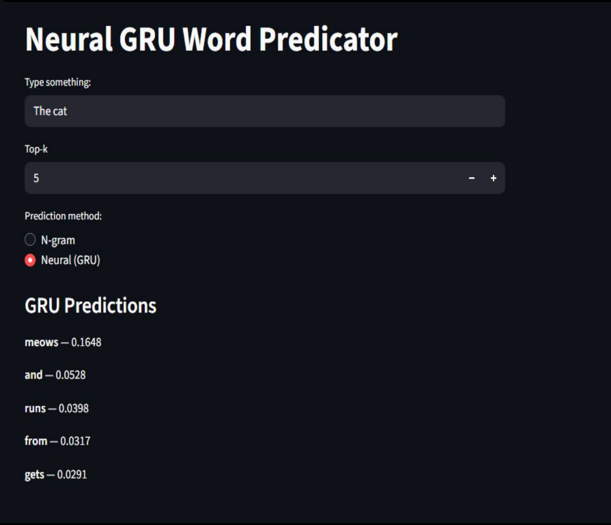
#1713 — The old cat sleeps all day.

#1712 — Birds normally chirp in the morning.

#1711 — Dogs usually bark at strangers.

#1710 — Cats often meow for food.

#1709 — The frightened dog retreats slowly.



Neural GRU Word Predicator

Type something: The cat

Top-k: 5

Prediction method: N-gram [✓ Neural (GRU)]

GRU Predictions

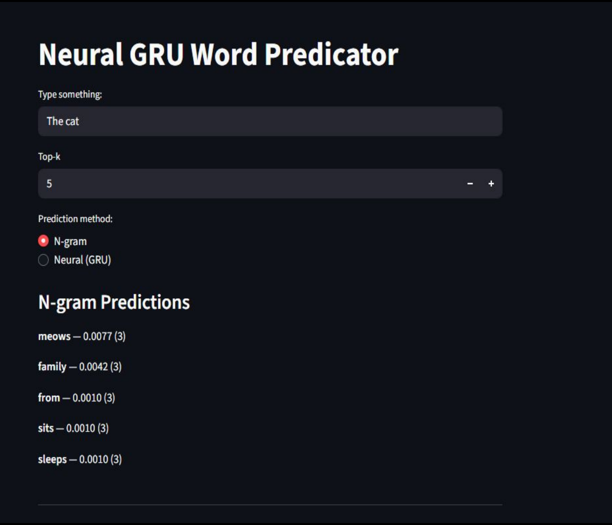
meows — 0.1648

and — 0.0528

runs — 0.0398

from — 0.0317

gets — 0.0291



Neural GRU Word Predicator

Type something: The cat

Top-k: 5

Prediction method: [✓ N-gram] Neural (GRU)

N-gram Predictions

meows — 0.0077 (3)

family — 0.0042 (3)

from — 0.0010 (3)

sits — 0.0010 (3)

sleeps — 0.0010 (3)

More Problems and Complexity of Vocabulary

So much data...

- Why My Project Needs So Much Data
- Language Is Enormously Diverse
  - Millions of possible word combinations
  - N-grams quickly explode in size (trigrams especially)
  - Most combinations will not appear unless the dataset is huge
- Neural Networks Need Dense Examples
  - GRUs learn from statistical patterns, not exact matches
  - With too little data, embeddings are bad and predictions become noisy

Takeaways

1. Simple Models → Deep Understanding

Starting with N-grams taught me how language prediction begins at the statistical level.

Building it basically first (tokenizing, counting, smoothing) made the fundamentals clear before jumping into neural networks.

2. Data Quality Determines Model Quality

Both N-grams and GRUs struggled with small datasets.

The project demonstrated firsthand why modern AI systems require thousands to millions of sentences.

3. Make your Model Clean
   - Tokenization again, really important

4. Why it Matters?
   - Introduced me to AI, its functionalities, and gave me insight on the behind the scenes of certain algorithms or machine learning such as a simple auto-complete