18-661 Introduction to Machine Learning

Neural Networks-III

Spring 2025

ECE – Carnegie Mellon University

Outline

1. Language Models and RNNs
2. Transformer Language Models
3. Stochastic Gradient Descent Convergence

Language Models and RNNs

What is Generative AI?

- So far, we have considered supervised learning tasks where we are given a training dataset of feature-label pairs $(\mathbf{x}_n, y_n)$, for $n = 1, \ldots, N$. Our goal is to learn a function $f(\mathbf{x}_n) \approx y_n$ that maps features to targets/labels
- In generative AI, we do not have explicit labels. Given a sequence of inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_t$ our goal is to predict the next element of the sequence $\mathbf{x}_{t+1}$.

What is Generative AI?

- So far, we have considered supervised learning tasks where we are given a training dataset of feature-label pairs $(\mathbf{x}_n, y_n)$, for $n = 1, \ldots, N$. Our goal is to learn a function $f(\mathbf{x}_n) \approx y_n$ that maps features to targets/labels
- In generative AI, we do not have explicit labels. Given a sequence of inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_t$ our goal is to predict the next element of the sequence $\mathbf{x}_{t+1}$.
- Examples:
    - Next word prediction
    - Text generation given a prompt
    - Machine translation
    - Image/video generation from a description

How does Generative AI work?

- We model the conditional distribution of the next token given the previous tokens:

$$\Pr(\mathbf{x}_{t+1}|\mathbf{x}_t, \ldots \mathbf{x}_1)$$

  using a neural network such as an RNN or transformer
- Then we sample from this probability distribution to generate $\mathbf{x}_{t+1}$

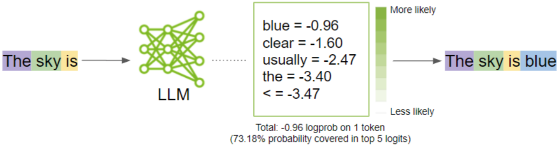
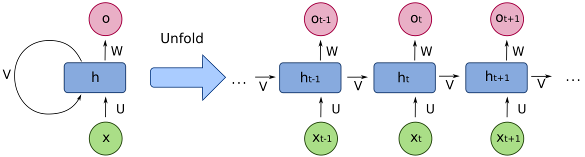


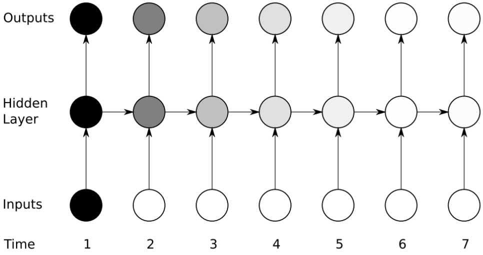
Figure source: NVIDIA technical blog

Recurrent Neural Networks (RNNs)



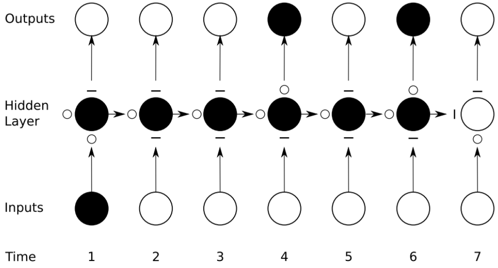
- Precursors to transformers, RNNs were widely used to model temporal or sequential data (e.g., natural language).
- Sequence of hidden states $\mathbf{h}_t$ that depend on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$
- Output computation: $\mathbf{o}_t = \psi(\mathbf{W}\mathbf{h}_t + b)$
- Hidden state computation: $\mathbf{h}_t = \phi(\mathbf{V}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + c)$
- The weight matrices $\mathbf{W}, \mathbf{V}, \mathbf{U}$ and biases b and c are trained using backpropagation on a dataset of sequences of varying lengths
- The predicted output $\mathbf{o}_t$ becomes the next input $\mathbf{x}_{t+1}$

RNNs and Forgetting



- RNNs tend to forget information as they progress forward through the sequence
- This is due to weak or vanishing gradients as we move longer distance through the model
- For a long sentence where the beginning of the sentence has information about the subject, such forgetfulness can be catastrophic

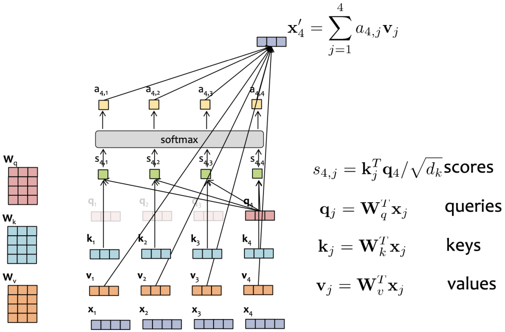
Long and Short-term Memory (LSTM)



- They combat the RNN forgetting issue via gates that decide whether to remember or forget information about the hidden states.
- But they still have drawbacks such as:
  - Difficulty with long-range dependencies (albeit less than RNNs)
  - Even though they solve the vanishing gradient problem, they suffer from exploding gradients
  - Inherently serial computation makes it harder to train

Transformer Language Models

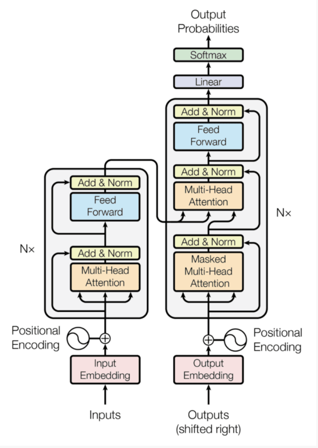
Transformers: Attention Mechanism



- RNNs and LSTM maintain a fixed length hidden state to represent the history of a sequence.
- Instead, the attention mechanism in Transformers looks at all previous token when predicting the next token
- The 'attention' that it pays to each token is computed using the attention mechanism

Transformers

- Encoder-decoder architecture: learn a representation of each input in the sequence, then decode to predict next entry in the output sequence
  - Autoregressive structure: takes previously generated outputs as inputs
  - Attach an attention weight to each entry of each input representation
  - Self-attention between each layer
- Much larger and slower to train, but usually gives good performance



Large Language Models

- "Generative pre-trained transformer" models: *generate* language outputs based on *pre-training* of *transformer*-based architectures on a massive corpus of language data
- Classification task: next-word prediction (run many times)
  - Tokenization: divide text into 'tokens of similar length/information
  - Predict the next token based on the preceding sequence of tokens (typically 1M long)
- Self-/semi-supervised models: generate supervisory signals ("labels") based on output of currently trained model
- Other generative models can generate images, videos, etc. Multi-modal models can, e.g., use a text input to generate an image.

Pre-Training and Finetuning

- Modern deep learning models are too expensive to train from scratch (GPT-4 likely cost millions of dollars to train!) As an example, Llama-7B, 13-B, etc. have billions of parameters.
- Pre-trained foundation models capture essential patterns and can be finetuned to specific datasets
  - Types of language, e.g., coding tools or translation tasks
  - Types of images, e.g., generating images of a certain style
- Foundation models can be trained further on a new dataset
  - Layer freezing or prompt engineering
  - "Warm start" initialization to the usual SGD-based training steps
- Prune, quantize, or compress foundation models to fit them or train them on smaller devices.

Outline

1. Language Models and RNNs

2. Transformer Language Models

3. Stochastic Gradient Descent Convergence

Stochastic Gradient Descent Convergence

SGD is at the core of Machine Learning!

We use it to train the model parameters $\mathbf{w}$ in

- Linear Regression: $y = \mathbf{w}^\top \mathbf{x}$
- Logistic Regression: $y = \sigma(\mathbf{w}^\top \mathbf{x})$
- Neural Networks: $y = NN(\mathbf{x}; \mathbf{w})$

For each problem, we define a loss function $F(\mathbf{w})$ to measure the error in the predicted output, and then update $\mathbf{w}$ according to:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w})$$

SGD is at the core of Machine Learning!

For each problem, we define a loss function $F(\mathbf{w})$ to measure the error in the predicted output, and then update $\mathbf{w}$ according to:

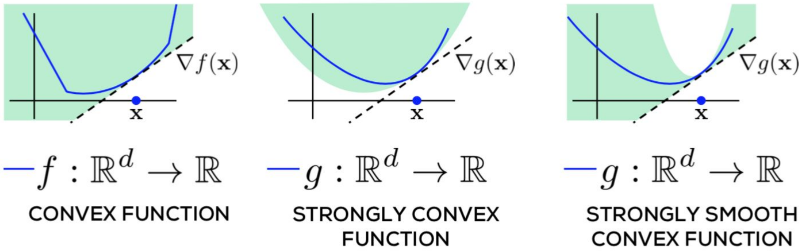
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta g(\mathbf{w})$$

The gradient $g(\mathbf{w})$ can be

- Full gradient, $\nabla F(\mathbf{w}_t) = \frac{1}{N}\sum_{i=1}^{N} \nabla f(\mathbf{w}_t; \xi)$ computed over the whole dataset
- Stochastic gradient $\nabla f(\mathbf{w}_t; \xi)$ for a randomly chosen sample ξ
- Mini-batch stochastic gradient $g(\mathbf{w}; \xi) = \frac{1}{b}\sum_{i=1}^{b} \nabla f(\mathbf{w}; \xi_i)$, computed using a batch ξ of b samples chosen at random

Let us analyze the time SGD takes to reach an ϵ error

A c-strongly Convex and L-Smooth Function



Satisfies the upper and lower bounds given by

$$F(\mathbf{w}) \leq F(\mathbf{y}) + \nabla F(\mathbf{y})^\top(\mathbf{w} - \mathbf{y}) + \frac{L}{2}\|\mathbf{w} - \mathbf{y}\|^2$$

$$F(\mathbf{w}) \geq F(\mathbf{y}) + \nabla F(\mathbf{y})^\top(\mathbf{w} - \mathbf{y}) + \frac{1}{2}c\|\mathbf{w} - \mathbf{y}\|^2 \text{ for all } \mathbf{w}, \mathbf{y} \in \mathbb{R}^d$$

Convergence Analysis of GD

Convergence of GD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$F(\mathbf{w}_t) - F(\mathbf{w}^*) \leq (1 - \eta c)^t (F(\mathbf{w}_0) - F(\mathbf{w}^*))$$

How many iterations do we need to converge to reach error $F(\mathbf{w}_t) - F(\mathbf{w}^*) = \epsilon$?

$$\begin{aligned}
(1 - \eta c)^t (F(\mathbf{w}_0) - F(\mathbf{w}^*)) &\leq \epsilon \\
t \log(1 - \eta c) + \log(F(\mathbf{w}_0) - F(\mathbf{w}^*)) &\leq \log(\epsilon) \\
t \log(1/(1 - \eta c)) - \log(F(\mathbf{w}_0) - F(\mathbf{w}^*)) &\geq \log(\frac{1}{\epsilon}) \\
t &= O(\log(\frac{1}{\epsilon}))
\end{aligned}$$

Convergence Analysis of GD

Convergence of GD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$F(\mathbf{w}_t) - F(\mathbf{w}^*) \leq (1 - \eta c)^t (F(\mathbf{w}_0) - F(\mathbf{w}^*))$$

How does the convergence speed depend on

- Learning rate η – Converges faster for larger η as long as $\eta < \frac{1}{L}$
- Lipschitz smoothness L – Converges faster for smaller L because we can set a higher η
- Strong convexity parameter c – Converges faster for larger c

Convergence Analysis of Mini-batch SGD

Assumptions on the Stochastic Gradients

Since we are using noisy gradients, we need the following assumptions on them

- Unbiased Gradients: The stochastic gradient $\nabla f(\mathbf{w};\xi)$ is an unbiased estimate of $\nabla F(\mathbf{w})$, that is,

$$\mathbb{E}_{\xi}[\nabla f(\mathbf{w};\xi)] = \nabla F(\mathbf{w})$$

- Bounded Variance: The stochastic gradient $\nabla f(\mathbf{w};\xi)$ has bounded variance, that is,

$$\mathrm{Var}(\nabla f(\mathbf{w};\xi)) \leq \sigma^2$$

  which implies that the variance of a mini-batch gradient is:

$$\mathrm{Var}(g(\mathbf{w};\xi)) \leq \frac{\sigma^2}{b}$$

Convergence Analysis of Mini-batch SGD

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \right)$$

- For batch *GD*, as $t \to \infty$, the objective $F(\mathbf{w}_t) \to F(\mathbf{w}^*)$
- For mini-batch SGD, as $t \to \infty$, we will be left with an error floor $\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \to \frac{\eta L \sigma^2}{2cb}$.
- This is the price that we pay for noisy gradients, that is the variance bound being $\sigma^2 \geq 0$

Effect of Mini-batch Size on the Error Floor

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2bc} \right)$$

- Recall that

$$g(\mathbf{w}, \xi) = \frac{1}{b} \sum_{n \in S} \nabla f(\mathbf{w}),$$

- And the bounded variance assumption is $\mathrm{Var}(g(\mathbf{w}; \xi)) \leq \sigma^2 / b$
- When we increase the mini-batch size b, the error floor $\frac{\eta L \sigma^2}{2cb}$ reduces.

Effect of Learning Rate on Convergence Speed and Error Floor

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t gradient descent iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \right)$$

- The convergence speed, represents by $(1 - \eta c)$ only depends on the strong convexity parameter c and learning rate η, not on the mini-batch size b
- As η increases, the algorithm converges faster
- But, as η increases, the error floor $\frac{\eta L \sigma^2}{2cb}$ also increases

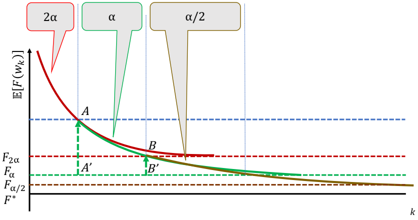
How do we achieve zero error floor?

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t mini-batch SGD iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1-\eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb}\right)$$

KEY IDEA: Decay the learning rate η (denoted by α in the figure below) by 2 whenever F is 2 times its error floor $\frac{\eta L \sigma^2}{2cb}$.



How do we achieve zero error floor?

Convergence of Mini-batch SGD

For a c-strongly convex and L-smooth function, if the learning rate $\eta < \frac{1}{L}$ and the starting point is $\mathbf{w}_0$ then $F(\mathbf{w}_t)$ after t mini-batch SGD iterations is bounded as

$$\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \leq (1 - \eta c)^t \left(\mathbb{E}[F(\mathbf{w}_0)] - F(\mathbf{w}^*) - \frac{\eta L \sigma^2}{2cb} \right)$$

KEY IDEA: Decay the learning rate η

- If $\eta = \eta_0 / t$, then we can show that $\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \leq O(1/t)$
- Thus, the number of iterations required an error ϵ is $O(1/\epsilon)$
- In contrast, with GD we need only $O(\log(1/\epsilon))$ iterations to reach an ϵ error

Summary

You should know:

- Language models and RNNs
- Transformer language models