Token'lar ve Context Window'lar

Token'ların ve tokenization'ın ne olduğu, context window'un sert sınırı, context dolduğunda ne olduğu, ve truncation/summarization/retrieval ile sınırlı context'i yönetme stratejileri.

Orta 18 dk

Son güncelleme: 6 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/tokens-and-context-windows

Token'lar ve Context Window'lar

"Large Language Model'ler Nasıl Çalışır?" bu dersin çok daha derinlemesine gireceği iki fikri tanıttı: **token'lar** (bir modelin birer birer tahmin ettiği küçük birimler) ve **context** (modelin üretmeden önce gördüğü her şey). İkisinin de, bir LLM ile çalışmanın neredeyse her yönünü şekillendiren sert, pratik sınırları olduğu ortaya çıkıyor -- ona bir kerede ne kadar bilgi verebileceğiniz, bir isteğin ne kadara mal olduğu, ve bir modelin çok uzun bir sohbetin başındaki bir şeyi neden "unutabildiği." Bu sınırları anlamak, hemen sonraki "Prompting and Prompt Engineering" dersi ve bu kursun ilerleyen "Tool'lar ve MCP" kategorisi için gerekli bir zemindir -- o kategori büyük ölçüde bir modelin sınırlı context'ine neyin gireceğini yönetmek üzerine kurulu.

Token Tam Olarak Nedir?

"Large Language Model'ler Nasıl Çalışır?" bir token'ı bir kelimeden "daha küçük bir birim" olarak tarif etmişti, bunun tam olarak ne anlama geldiğini söylemeden -- bu ders bunu netleştiriyor. Token, bir LLM'in gerçekte okuduğu ve ürettiği birimdir; genellikle, ama her zaman değil, tam bir kelimedir. Yaygın kelimeler ("the," "cat") sıklıkla tek bir token'dır; daha az yaygın ya da daha uzun kelimeler genellikle birden fazla token'a bölünür (örneğin, "tokenization" "token" + "ization" olabilir); noktalama işaretleri, boşluklar, ve hatta emoji ya da İngilizce olmayan metin parçaları da token'dır. Bu bölme sürecine **tokenization** denir, ve modelin eğitildiği sabit bir kelime haznesi (vocabulary) aracılığıyla gerçekleşir -- İngilizce metin için kabaca bir kural, bir token'ın yaklaşık 4 karakter olduğu, ya da 100 token'ın kabaca 75 kelime olduğudur, ama bu dile ve içerik türüne göre belirgin biçimde değişir (kod ve İngilizce olmayan metin genellikle kelime başına İngilizce düz yazıdan daha fazla token kullanır).

> Tokenization dile göre değiştiği için, aynı cümle bir dilde diğerinden belirgin biçimde daha fazla token'a mal olabilir -- birden fazla dilde bir şey inşa ediyorsanız, bu yalnızca akademik bir ayrıntı değil, maliyet ve context bütçesi için gerçek, pratik bir husustur.

Neden Tam Kelimeler Yerine Token?

"Large Language Model'ler Nasıl Çalışır?"tan, pretraining'in bir modeli, kendinden önceki her şeye dayanarak sıradaki token'ı tahmin etmesi için eğittiğini hatırlayın -- modelin, tahmin edeceği sabit, sonlu bir olası token kelime haznesine ihtiyacı vardır. Yalnızca tam kelimelerden oluşan bir kelime haznesi ya devasa olurdu (her dildeki her kelimeyi, isimleri, yazım hatalarını ve uydurma kelimeleri kapsamak için) ya da hiç görmediği kelimelerde sürekli başarısız olurdu. Metni daha küçük, yeniden kullanılabilir parçalara bölmek her iki sorunu da çözer: yaygın kelimelerin, kelime parçalarının ve tek tek karakterlerin mütevazı bir kelime haznesi (genellikle on binlerce giriş), tokenizer'ı tasarlayanların hiç öngörmediği kelimeler dahil, kelimenin tam anlamıyla her metni, daha küçük parçalara geri dönerek temsil edebilir.

Context Window: Sert Bir Sınır

Önceki dersteki "Context'e İlk Bakış," context'i modelin sıradaki token'ını üretmeden önce gördüğü her şey olarak tarif etmişti. O "her şey"in **context window** denen sert bir tavanı vardır -- belirli bir modelin bir kerede işleyebileceği maksimum token sayısı, talimatlarınızı, sağlanan her türlü arka plan bilgisini, sohbet geçmişini ve üretilmekte olan yanıtı birleştirir. Context window'lar modele göre değişir, erken LLM'lerdeki birkaç bin token'dan modern olanlardaki yüz binlerce (ya da daha fazla) token'a kadar -- ama her modelin *bir* sınırı vardır. Bir sohbetin toplam token sayısı bu sınırı aşacak hale geldiğinde, bir şeyin feragat etmesi gerekir: modeli çağıran sistemin nasıl kurulduğuna bağlı olarak, daha eski içerik düşürülür, özetlenir, ya da istek doğrudan reddedilir.

Context Dolduğunda Ne Olur?

Çoğu insanın karşılaştığı "model uzun bir sohbetin daha önceki kısımlarını unutuyor" deneyiminin gerçekte nereden geldiği tam olarak burasıdır. "In-Context Learning: Bir Model Ona Verdiğinizi Nasıl Kullanır?"dan, bir modelin kendi belleği olmadığını hatırlayın -- mevcut sohbet hakkında "bildiği" her şey, şu anda context'inde bulunan metinden ibarettir. Modeli besleyen sistem, context window sınırının altında kalmak için en erken mesajları düşürmek zorunda kalırsa, bu bilgi model tarafından daha derin bir anlamda "unutulmaz" -- yalnızca artık modelin okuduğu metnin bir parçası değildir, bu yüzden onu kullanmasının hiçbir yolu yoktur, tıpkı hiç söylenmemiş gibi. Bu, belirli bir ürüne özgü bir kusur değildir; context window'un sonlu olmasının doğrudan, kaçınılmaz bir sonucudur.

Maliyet ve Gecikme Birimi Olarak Token'lar

Context window sınırının ötesinde, token'lar aynı zamanda LLM kullanımının ölçüldüğü ve genellikle faturalandırıldığı pratik birimdir -- hem gönderdiğiniz token'lar (girdi) hem modelin ürettiği token'lar (çıktı). Bunun bir LLM'in üzerine bir şey inşa eden herkes için çok somut iki sonucu vardır: daha uzun prompt'lar ve daha uzun sohbet geçmişleri istek başına daha pahalıya mal olur, ve daha uzun bir yanıt üretmek ölçülebilir biçimde daha uzun sürer, çünkü ("In-Context Learning: Bir Model Ona Verdiğinizi Nasıl Kullanır?"ı hatırlayın) model her tek yanıtta context'in tamamını yeniden okur. Context'te gerçekten olması gereken şey konusunda bilinçli olmak -- "ne olur ne olmaz" diye her şeyi dahil etmek yerine -- bu yüzden yalnızca bir doğruluk sorunu değil, gerçek bir maliyet ve hız sorunudur; "Tool'lar ve MCP" kategorisi, sistemlerin bir modele hangi bilgiyi vereceğini nasıl seçtiğini kapsarken bu temaya geri dönüyor.

Sınırlı Bir Context Window'u Yönetmek

Her modelin sert bir token tavanı olduğu için, LLM üzerine inşa edilmiş gerçek sistemler, her birinin gerçek bir ödünleşimi olan birkaç tekrar eden strateji kullanır:

- **Truncation (kesme):** sınıra yaklaşıldığında yalnızca en eski mesajları düşürmek -- basittir, ama gerçekten önemli daha eski context'i kaybetme riski taşır ("Context Dolduğunda Ne Olur?"a bakın).
- **Summarization (özetleme):** daha eski sohbet geçmişini periyodik olarak onun daha kısa bir özetiyle değiştirmek -- tam kelime seçimini ve ayrıntıyı kaybetmek pahasına, çok daha az token kullanarak özü korur.
- **Retrieval (getirme):** her şeyi sürekli context'te tutmak yerine, yalnızca mevcut istekle ilgili belirli bilgi parçalarını getirip yalnızca onları context'e eklemek -- bu, bu kursun ilerleyen kategorilerinin kapsamlı biçimde üzerine inşa ettiği tekniklerin temel fikridir.

Hiçbir strateji evrensel olarak doğru değildir -- hangisinin (ya da hangi kombinasyonun) mantıklı olduğu, belirli uygulamanın gerçekte neyi hatırlaması gerektiğine tamamen bağlıdır.

Best Practices

- Context'i ücretsiz ya da sınırsız saymayın -- dahil ettiğiniz her token ("Maliyet ve Gecikme Birimi Olarak Token'lar"a bakın), sert context window tavanına sayılmasının yanı sıra hem parada hem yanıt süresinde gerçek bir maliyete sahiptir.
- Bir sohbetin bir context window'a rahatça sığandan daha fazla bilgiyi kapsaması gerektiğinde, bir sistemin sessizce hangisi en eskiyse onu düşürmesine izin vermek yerine truncation, summarization ya da retrieval arasında bilinçli olarak karar verin ("Sınırlı Bir Context Window'u Yönetmek"e bakın).

- Token sayısının kelime ya da karakter sayısıyla aynı olmadığını unutmayın ("Token Tam Olarak Nedir?"a bakın) -- bir context window'a ne kadarının sığdığını tahmin ederken kelimeleri değil token'ları sayın.

Yaygın Hatalar

- **Bir modelin çok uzun bir sohbetin çok daha önceki bir yerinde söylenmiş bir şeyi "hatırladığını" varsaymak.** "Context Dolduğunda Ne Olur?"un açıkladığı gibi, içerik context window'dan çıktığında, modelin ona hiçbir erişimi yoktur -- geri dönülebilecek daha derin bir bellek yoktur.
- **Tüm dillerin ya da içerik türlerinin aynı metin için kabaca aynı sayıda token'a mal olduğunu varsaymak.** "Token Tam Olarak Nedir?"teki ipucunun belirttiği gibi, tokenization dile ve içerik türüne göre anlamlı biçimde değişir.
- **Daha büyük bir context window'u, nelerin dahil edildiğini düşünme ihtiyacını ortadan kaldıran bir şey olarak görmek.** Çok büyük bir context window'la bile, ilgisiz bilgiyi dahil etmek hâlâ token'lara mal olur (para ve gecikme, "Maliyet ve Gecikme Birimi Olarak Token'lar"a bakın) ve bir modelin bir istekte gerçekten önemli olan şeyden dikkatini dağıtabilir.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Bir **token,** bir LLM'in gerçekte okuduğu ve ürettiği birimdir -- genellikle bir kelimeye yakındır, ama her zaman değil -- **tokenization** denen bir süreçle üretilir.
- **Context window,** bir modelin bir kerede işleyebileceği maksimum token sayısıdır, talimatları, arka plan bilgisini, sohbet geçmişini ve yanıtın kendisini birleştirir.
- Context dolduğunda, daha eski içeriğin düşürülmesi, özetlenmesi ya da başka biçimde ele alınması gerekir -- modelin şu anda context'inde olanın ötesinde hiçbir belleği yoktur.
- Token'lar aynı zamanda LLM kullanımının ölçüldüğü ve faturalandırıldığı pratik birimdir, bu da context boyutunu yalnızca bir doğruluk değil, gerçek bir maliyet ve gecikme sorunu haline getirir.
- Truncation, summarization ve retrieval, sınırlı bir context window içinde çalışmak için üç yaygın stratejidir, her birinin farklı bir ödünleşimi vardır.

Cheat Sheet

- Token ≈ genellikle bir kelimeye yakındır, ama daha küçük olabilir (kelime parçası, noktalama, karakter).
- Context window = belirli bir model için sert token tavanı.
- Context dolu → truncate (en eskiyi düşür), summarize (sıkıştır), ya da retrieve (yalnızca ilgili olanı getir).
- Context'te daha fazla token = her zaman daha fazla maliyet + daha fazla gecikme.

Terimler Sözlüğü

- **Token:** bir LLM'in gerçekte okuduğu ve ürettiği birim, tokenization ile üretilir -- genellikle bir kelimeye yakındır, ama bazen daha küçüktür.
- **Tokenization:** metni bir modelin sabit kelime haznesini kullanarak token'lara bölme süreci.
- **Context window:** belirli bir modelin tek bir istekte işleyebileceği maksimum token sayısı, tüm context kaynaklarını birleştirir.
- **Truncation (kesme):** token sınırına yaklaşıldığında context'ten en eski içeriği düşürmek.
- **Summarization (bu bağlamda özetleme):** token tasarrufu için daha eski sohbet geçmişini daha kısa bir özetle değiştirmek.
- **Retrieval (getirme):** her şeyi context'te tutmak yerine, mevcut istek için gereken yalnızca belirli, ilgili bilgi parçalarını getirmek.