Yöntem ile Sıfırdan Dil Modeli

ÖNDER ÖRNEK

Yöntem ile Sıfırdan Dil Modeli

BİRİNCİ BASKI
2026

Yöntem ile Sıfırdan Dil Modeli
Önder Örnek

Telif hakkı © 2026 Önder Örnek.

Bu eser Creative Commons Atıf 4.0 Uluslararası Lisansı (CC BY 4.0) ile lisanslanmıştır. Yazarın adı belirtilerek, lisansa bağlantı verilerek ve yapılan değişiklikler gösterilerek ticari amaçlar dahil her ortamda ve biçimde kopyalanabilir, dağıtılabilir ve uyarlanabilir. Lisans metni:
`https://creativecommons.org/licenses/by/4.0/deed.tr`

Eserdeki örnek programlar MIT lisansıyla kullanılabilir. Yöntem yazılımı da MIT lisanslıdır.

Birinci baskı: Kasım 2026

Dizgi: Metin TeX Gyre Pagella, program kodları DejaVu Sans Mono yazı tipiyle, XeLaTeX kullanılarak dizilmiştir.

İçindekiler

Önsöz vii

1 Dil modeli nedir? 1
Metin üretmek bir döngüdür 1
Model nasıl öğrenir? 2
Bu kitabın yolu 2

2 Metni sayılara çevirmek 5
Karakter düzeyinde simgeleştirme 5
Daha büyük parçalar: bayt çifti kodlaması 8

3 Sayarak dil modeli 9
Çıktıyı okumak 12
Kayıp ve şaşkınlık 13

4 Sinir ağıyla ikili model 15
Dört adım 15
Çıktıyı okumak 18
Neden ÇaprazEntropi'ye logit veriyoruz? 18

5 Bağlamı büyütmek 19
Modelin yapısı 19
İki yeni alışkanlık 20
Çıktıyı okumak: ezberleme 23

6 Öz-dikkat 25
Adım 1: geçmişin ortalaması 25

Adım 2: sorgu, anahtar, değer . . . 26
Adım 3: konum bilgisi . . . 26
Çıktıyı okumak . . . 29

7 Transformer bloğu . . . 31
Çok kafalı dikkat . . . 31
İleri besleme ağı . . . 32
Artık bağlantı . . . 32
Katman normalleştirme . . . 33
Kitabın modeli: KucukGPT birimi . . . 33
Parçaları sınamak . . . 37
Çıktıyı okumak . . . 40

8 Küçük GPT'yi eğitmek . . . 41
Çıktıyı okumak . . . 46

9 Metin üretmek . . . 47
Üç yöntem . . . 47
Çıktıyı okumak . . . 52

10 Küçükten büyüğe . . . 53
Parametreleri saymak . . . 53
Çıktıyı okumak . . . 55
Ön eğitim, ince ayar ve sohbet modelleri . . . 56
Bu kitaptan sonra . . . 56

A Kavram sözlüğü . . . 59

B Sık yapılan hatalar . . . 63

C Alıştırma ipuçları . . . 65

Önsöz

ChatGPT, Claude ve Gemini gibi büyük dil modelleri (LLM) dışarıdan bakınca sihir gibi görünür. Oysa hepsinin çekirdeğinde tek bir basit soru vardır: *şimdiye kadar yazılanlara bakılırsa, sıradaki parça ne olabilir?* Bu kitap o soruyu adım adım, sıfırdan, kendi yazacağınız programlarla yanıtlar.

Kitabın sonunda elinizde çalışan küçük bir GPT olacaktır. Bu GPT, 178 Türk atasözü ve deyimiyle eğitilip yeni "atasözleri" üretir. Büyük modellerden milyonlarca kez küçüktür ama aynı mimariye sahiptir: karakter gömmeleri, konum gömmeleri, çok kafalı öz-dikkat, ileri besleme ağları, artık bağlantılar ve katman normalleştirme. Aradaki fark mimaride değil, ölçektedir.

Kime yönelik? Yöntem'in temellerini (değişkenler, döngüler, fonksiyonlar, listeler, sınıflar) bilen okurlar için yazılmıştır. *Yöntem ile Programlamaya Giriş* kitabının 12. ve 13. bölümlerindeki tensörler ve otomatik türev konularını okumuş olmak işinizi kolaylaştırır ama şart değildir. Matematik olarak toplama, çarpma, ortalama ve logaritmanın ne olduğunu bilmek yeterlidir.

Kitap nasıl ilerliyor? Her bölüm bir öncekinin modelini biraz daha akıllı hale getirir:

Bölüm	Model	Neye bakıyor?	Doğrulama kaybı
3	Sayarak ikili model	1 önceki karakter	--- (yalnızca sayma)
4	Sinir ağıyla ikili model	1 önceki karakter	--- (sayma ile aynı sonuç)
5	Bağlamlı MLP	3 önceki karakter	2,40
8	Küçük GPT	16 önceki karakter	8. bölümde

Nasıl okuyabilirsiniz? Programları sırayla çalıştırmanızı öneririz. Hepsi `llm_kitabi/` klasöründedir ve şu komutla çalışır:

```
node js/dist/cli.js llm_kitabi/03_ikili_sayma.y
```

Programlar `Tohum(...)` ile rastgeleliği sabitler; bu yüzden kendi bilgisayarınızda da kitaptaki çıktıların aynısını görmelisiniz. Her bölümün sonundaki alıştırmalar, modelin bir ayarını değiştirip sonucunu gözlemenizi ister. Bir modeli anlamanın en iyi yolu onu kurcalamaktır.

Kitapta kullanılan üç birim de aynı klasördedir: `Atasozleri` (eğitim verisi), `DilAraclari` (2. ve 3. bölümde yazdığımız küçük yardımcılar) ve `KucukGPT` (6. ve 7. bölümde kurduğumuz model).

BÖLÜM

1

Dil modeli nedir?

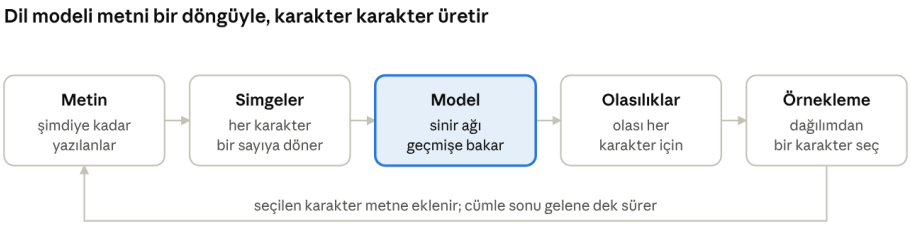
Bir dil modeli, bir metnin devamını tahmin eden bir programdır. “damlaya damlaya göl ol” yazıldığını gören bir model, sıradaki harfin büyük olasılıkla “u” olduğunu bilir. Ama tek bir cevap vermez; her olası devama bir olasılık verir. “u” için yüksek, “a” için düşük, “ş” için neredeyse sıfır gibi.

Bu tanım basit görünür ama çok güçlüdür. Bir metnin devamını iyi tahmin edebilmek için model dilin kurallarını (hangi harfler yan yana gelir, ekler nasıl sıralanır) öğrenmek zorundadır. Yeterince büyük bir modelde buna dünya bilgisi ve akıl yürütme de eklenir. “Türkiye’nin başkenti” diye başlayan bir cümleyi iyi tamamlamak için Ankara’yı bilmek gerekir.

Metin üretmek bir döngüdür

Model yalnızca bir sonraki parçayı tahmin eder. Uzun bir metin üretmek için bu tahmini döngüyle tekrarlarız.

Her turda model, yazılanları okur ve bir olasılık dağılımı verir. Biz bu dağılımdan bir karakter çekeriz, metne ekleriz ve yeniden başlarız. ChatGPT’nin cevabı



Şekil 1.1. *Dil modeli metni bir döngüyle, karakter karakter üretir*

ekranda parça parça belirirken olan da tam olarak budur. Tek fark, büyük modellerin karakter yerine "simge" denen sözcük parçalarıyla çalışmasıdır (2. bölüm).

Model nasıl öğrenir?

Modeli eğitmek için etiketli veriye gerek yoktur; metnin kendisi yeter. "damlaya damlaya göl olur" cümlesi, kendi içinde onlarca soru–cevap çifti barındırır:

Model şunu görür	Şunu tahmin etmeli
d	a
da	m
dam	l
...	...
damlaya damlaya göl olu	r

Eğitim sırasında modelin her tahmini doğru cevapla karşılaştırılır. Hatanın ölçüsüne **kayıp** denir: model doğru harfe ne kadar düşük olasılık verdiyse kayıp o kadar büyüktür. Kayıp, olasılığın eksi logaritmasıdır:

$$\text{kayıp} = -\ln P(\text{doğru karakter})$$

Model doğru harfe 1 olasılık verirse kayıp 0 olur. 0,5 verirse kayıp 0,69, 0,01 verirse 4,6 olur. Eğitim, milyonlarca küçük adımda modelin ayarlarını (ağırlıklarını) bu kaybı azaltacak yöne kaydırmaktır.

Bu kitabın yolu

Modeli dört aşamada büyüteceğiz. Önce yalnızca bir önceki harfe bakan ve sayarak çalışan bir model kuracağız (3. bölüm). Ardından aynı modeli sinir ağıyla

yapacağız (4. bölüm). Sonra birkaç önceki harfe bakan bir sinir ağı yazacağız (5. bölüm). En sonda da tüm geçmişe akıllıca bakabilen Transformer'ı kuracağız (6–8. bölümler).

Alıştırma. "öfke ile kalkan zararla oturur" atasözünden kaç eğitim örneği çıkar? (Cümlenin sonunu da tahmin edilecek bir "karakter" sayarsanız cevap, karakter sayısı + 1'dir.)

BÖLÜM

2

Metni sayılara çevirmek

Sinir ağları harflerle değil sayılarla çalışır. Bu yüzden ilk iş metni bir sayı dizisine çevirmektir. Bu işe **simgeleştirme** (tokenization), metnin parçalarına da **simge** (token) denir.

Karakter düzeyinde simgeleştirme

En basit yol her karaktere bir numara vermektir. Veri kümemizdeki 178 atasözünde 28 farklı harf ve boşluk geçiyor. Bunlara bir de nokta ekliyoruz. Nokta, "atasözü burada bitti" ya da "yeni atasözü başlıyor" anlamına gelen özel bir simgedir. Böylece sözlüğümüz 30 simgeden oluşur.

İki işlem birbirinin tersidir: **kodlama** metni sayılara, **çözme** sayıları metne çevirir.

```
program Simgeleştirme
{ Bir dil modeli harfleri değil sayıları işler. Bu program:
```

```
  1) Veri kümesindeki tüm karakterlerden bir sözlük kurar (karakter → numara).
  2) Bir cümleyi sayılara çevirir (kodlama) ve geri çevirir (çözme).
  3) Bayt çifti kodlamasının (BPE) fikrini gösterir: en sık yan yana gelen
     iki simgeyi birleştirip yeni bir simge yapar, bunu birkaç kez tekrarlar. }
kullan Atasozleri

değişken
  veri: metin dizisi
  harfler: metin dizisi
  numara: sözlük
  kodlar: tamsayı dizisi
  simgeler: metin dizisi
  çiftSay: sözlük
  enÇift, çift: metin
  i, tur, karakterSayısı: tamsayı
  yeni: metin dizisi

fonksiyon Kodla(s: metin): tamsayı dizisi
başla
  sonuç := Eşle(s.Karakterler, fonksiyon(c: metin) => numara[c])
bitir

fonksiyon Çöz(k: tamsayı dizisi): metin
başla
  sonuç := Birleştir(Eşle(k, fonksiyon(n: tamsayı) => harfler[n]), '')
bitir

başla
  veri := Atasozleri.Hepsi
  { Sözlük: '.' cümle sonu (ve başı) işaretidir; ardından görülen tüm karakterler,
      ↪ sıralı }
  harfler := ['.'] + Sırala(Benzersiz(Birleştir(veri, '').Karakterler))
  numara := [:]
  0'dan Uzunluk(harfler) - 1'e kadar her i için numara[harfler[i]] := i
  karakterSayısı := Toplam(Eşle(veri, fonksiyon(a: metin) => Uzunluk(a) + 1))
  yaz(Biçimle('{} atasözü, toplam {} karakter', Uzunluk(veri), karakterSayısı), 1)
  yaz(Biçimle('Sözlük boyutu: {}', Uzunluk(harfler)), 1)
  yaz('Sözlük: ' + Birleştir(harfler, ''), 2)

  kodlar := Kodla('damlaya damlaya göl olur')
  yaz('Kodlanmış : ' + kodlar, 1)
  yaz('Çözülmüş  : ' + Çöz(kodlar), 2)

  { BPE: başlangıçta her karakter bir simgedir }
  simgeler := Birleştir(Eşle(Dilim(veri, 0, 40), fonksiyon(a: metin) => a + '.'),
      ↪ '').Karakterler
  yaz(Biçimle('BPE başlangıcı: {} simge', Uzunluk(simgeler)), 1)
  1'den 6'ya kadar her tur için
  başla
```

```
    çiftSay := [:]
    0'dan Uzunluk(simgeler) - 2'ye kadar her i için
    başla
      çift := simgeler[i] + '|' + simgeler[i + 1]
      eğer çiftSay.İçerir(çift) ise çiftSay[çift] := çiftSay[çift] + 1 değilse
          ↪ çiftSay[çift] := 1
    bitir
    enÇift := Sırala(çiftSay.Anahtarlar, fonksiyon(k: metin) => -çiftSay[k])[0]
    { en sık çifti tek simgeye birleştir }
    yeni := []
    i := 0
    i < Uzunluk(simgeler) iken
    başla
      eğer i < Uzunluk(simgeler) - 1 ve simgeler[i] + '|' + simgeler[i + 1] = enÇift
          ↪ ise
      başla
        yeni.Ekle(simgeler[i] + simgeler[i + 1]); i := i + 2
      bitir
      değilse
      başla
        yeni.Ekle(simgeler[i]); i := i + 1
      bitir
    bitir
    simgeler := yeni
    yaz(Biçimle('  {}. birleştirme: "{}" ({} kez)  →  {} simge', tur,
      enÇift.Değiştir('|', ''), çiftSay[enÇift], Uzunluk(simgeler)), 1)
  bitir
  yaz('Simgelere bölünmüş ilk satırlar: ' + Birleştir(Dilim(simgeler, 0, 45), '|'),
      ↪ 1)
bitir.
```

Programın çıktısı şöyledir:

```
178 atasözü, toplam 5282 karakter
Sözlük boyutu: 30
Sözlük: . abcdefghiklmnoprstuvyzçöüğış

Kodlanmış : [5, 2, 13, 12, 2, 22, 2, 1, 5, 2, 13, 12, 2, 22, 2, 1, 8, 25, 12, 1, 15, 12,
    ↪ 20, 17]
Çözülmüş  : damlaya damlaya göl olur

BPE başlangıcı: 1140 simge
  1. birleştirme: "n " (43 kez)  →  1097 simge
  2. birleştirme: "r." (24 kez)  →  1073 simge
  3. birleştirme: "an " (19 kez)  →  1054 simge
  4. birleştirme: "de" (15 kez)  →  1039 simge
  5. birleştirme: "i " (14 kez)  →  1025 simge
```

```
  6. birleştirme: "ma" (13 kez)  →  1012 simge
Simgelere bölünmüş ilk satırlar: d|a|m|l|a|y|a| |d|a|m|l|a|y|a| |g|ö|l|
    ↪  |o|l|u|r.|a|c|e|l|e| |i|ş|e| |ş|e|y|t|an |k|a|r|ı|ş|ı
```

Sözlükteki ikinci simge boşluk karakteridir; bu yüzden çıktıda "." ile "a" arasında bir boşluk görünür. Kodlanmış dizide sıkça geçen 1 sayısı, boşluğun numarasıdır.

Daha büyük parçalar: bayt çifti kodlaması

Karakter düzeyinde çalışmanın bir bedeli vardır: model "an" ya da "ler" gibi çok sık görülen parçaları her seferinde harf harf kurmak zorundadır. Büyük dil modelleri bu yüzden **bayt çifti kodlaması** (BPE, byte pair encoding) kullanır. Fikir şudur: metinde en sık yan yana gelen iki simge bulunur ve bunlar yeni, tek bir simgeyle değiştirilir. Bu işlem binlerce kez tekrarlanır.

Programın ikinci yarısı bu birleştirmeyi ilk 40 atasözü üzerinde altı kez yapıyor. İlk birleştirme "n" ile boşluğu birleştiriyor, çünkü Türkçede pek çok sözcük "n" ile biter ("şeytan", "yaşken", "akıldan"). Üçüncü turda "a" bu yeni simgeye eklenerek "an" oluşuyor. Altı birleştirmeden sonra aynı metin 1140 yerine 1012 simgeyle yazılabiliyor.

Gerçek modellerde sözlük on binlerce, hatta yüz binlerce simgeden oluşur. Sık sözcükler tek simgedir; nadir sözcükler birkaç parçaya bölünür. Türkçe gibi eklemeli dillerde "gidemeyeceklermiş" gibi uzun bir sözcük genellikle birden çok simgeye ayrılır.

Bu kitapta karakter düzeyinde kalacağız. Sözlük küçük olduğu için modelimiz de küçük kalır; üstelik kayıp ve olasılıkları harf harf görmek öğrenmeyi kolaylaştırır. Bir dil modelinin geri kalanı simgelerin karakter mi, sözcük parçası mı olduğunu umursamaz; hepsi yalnızca birer numaradır.

Alıştırma. BPE döngüsünü 6 yerine 30 kez çalıştırınız. Hangi Türkçe ekler ve sözcükler tek simge haline geliyor?

BÖLÜM

3

Sayarak dil modeli

İlk dil modelimiz hiç eğitilmeyecek; yalnızca sayacak. Fikir şudur: bir sonraki karakter yalnızca bir önceki karaktere bağlıymış gibi davranalım. "ş" harfinden sonra neler geldiğini veri kümesinde sayarsak, "ş"den sonra ne geleceğinin olasılıklarını da bulmuş oluruz. İki karakterlik bu pencereye **ikili** (bigram) model denir.

$$P(\text{sonraki} = b \mid \text{şimdiki} = a) = \frac{\text{say}(a, b)}{\sum_x \text{say}(a, x)}$$

Her atasözünü noktayla başlatıp noktayla bitiriyoruz. Böylece "nokta → d" çifti "atasözleri d harfiyle ne sıklıkla başlar?" sorusunu, "r → nokta" çifti de "r ile biten bir sözcükten sonra atasözü ne sıklıkla biter?" sorusunu yanıtlar.

```
program SayarakDilModeli
{ İkili (bigram) dil modeli: bir sonraki karakterin olasılığı yalnızca bir önceki
  karaktere bağlıdır. Model eğitilmez, sayılır:
     P(sonraki = b | şimdiki = a) = say(a, b) / say(a, herhangi)
  Her atasözünün başına ve sonuna '.' koyarız; böylece model nerede başlayıp
```

```
  nerede bitireceğini de öğrenir. Sonra modelden yeni "atasözleri" örnekleriz
  ve modelin veriye ne kadar şaşırdığını ölçeriz. }
kullan Atasozleri, Matematik

değişken
  veri, harfler: metin dizisi
  numara: sözlük
  say: tamsayı dizisi dizisi
  P: ondalık dizisi dizisi
  a: metin
  i, k, n, öncekiNo, sonrakiNo: tamsayı
  satırToplamı: tamsayı
  logToplam: ondalık
  sözcük, kar: metin
  sıra: tamsayı dizisi

fonksiyon Örnekle(olasılıklar: ondalık dizisi): tamsayı
{ Olasılık dağılımından bir numara çeker: [0, 1) arasında bir sayı tutup
  olasılıkları sırayla toplar, toplam o sayıyı geçtiği yerde durur. }
değişken r, birikim: ondalık; i: tamsayı
başla
  r := Rastgele(1000000) / 1000000
  birikim := 0
  0'dan Uzunluk(olasılıklar) - 1'e kadar her i için
  başla
    birikim := birikim + olasılıklar[i]
    eğer r < birikim ise dön i
  bitir
  dön Uzunluk(olasılıklar) - 1
bitir

başla
  Tohum(42)
  veri := Atasozleri.Hepsi
  harfler := ['.'] + Sırala(Benzersiz(Birleştir(veri, '').Karakterler))
  n := Uzunluk(harfler)
  numara := [:]
  0'dan n - 1'e kadar her i için numara[harfler[i]] := i

  { 1) Say: say[i][j] = i. karakterden sonra j. karakterin kaç kez geldiği }
  say := Eşle(Aralık(n), fonksiyon(x: tamsayı) => Eşle(Aralık(n), fonksiyon(z:
      ↪  tamsayı) => 0))
  veri içindeki her a için
  başla
    öncekiNo := 0
    (a + '.').Karakterler içindeki her kar için
    başla
      sonrakiNo := numara[kar]
      say[öncekiNo][sonrakiNo] := say[öncekiNo][sonrakiNo] + 1
```

```
    öncekiNo := sonrakiNo
  bitir
bitir

{ 2) Olasılığa çevir. Her sayıma 1 eklemek (düzleştirme) hiç görülmemiş
     bir çiftin olasılığının sıfır olmasını engeller. }
P := []
0'dan n - 1'e kadar her i için
başla
  satırToplamı := Toplam(say[i]) + n
  P.Ekle(Eşle(say[i], fonksiyon(c: tamsayı) => (c + 1) / satırToplamı))
bitir

yaz('Atasözleri en sık şu harflerle başlar:', 1)
sıra := Sırala(Aralık(n), fonksiyon(x: tamsayı) => -P[0][x])
0'dan 4'e kadar her k için
  yaz(Biçimle('  "{}"  olasılık {:.3}', harfler[sıra[k]], P[0][sıra[k]]), 1)
yaz('"ş" harfinden sonra en sık gelenler:', 1)
i := numara['ş']
sıra := Sırala(Aralık(n), fonksiyon(x: tamsayı) => -P[i][x])
0'dan 4'e kadar her k için
  yaz(Biçimle('  "{}"  olasılık {:.3}', harfler[sıra[k]], P[i][sıra[k]]), 1)

{ 3) Örnekle: '.' ile başla, bir sonraki karakteri çek, '.' gelene kadar sürdür }
yaz('', 1); yaz('Modelin ürettiği "atasözleri":', 1)
1'den 8'e kadar her k için
başla
  sözcük := ''; i := 0
  tekrarla
    i := Örnekle(P[i])
    eğer i <> 0 ise sözcük := sözcük + harfler[i]
  i = 0 olana dek
  yaz('  ' + sözcük, 1)
bitir

{ 4) Değerlendirme: ortalama negatif log-olabilirlik (kayıp) ve şaşkınlık }
logToplam := 0; k := 0
veri içindeki her a için
başla
  öncekiNo := 0
  (a + '.').Karakterler içindeki her kar için
  başla
    sonrakiNo := numara[kar]
    logToplam := logToplam - Ln(P[öncekiNo][sonrakiNo])
    k := k + 1
    öncekiNo := sonrakiNo
  bitir
bitir
yaz('', 1)
```

```
  yaz(Biçimle('Ortalama kayıp (karakter başına): {:.4}', logToplam / k), 1)
  yaz(Biçimle('Şaşkınlık (perplexity): {:.2}   (rastgele tahmin: {})',
      ↪ Üstel(logToplam / k), n), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Atasözleri en sık şu harflerle başlar:
  "a"  olasılık 0.106
  "b"  olasılık 0.091
  "d"  olasılık 0.087
  "k"  olasılık 0.077
  "s"  olasılık 0.062
"ş" harfinden sonra en sık gelenler:
  " "  olasılık 0.164
  "e"  olasılık 0.119
  "ı"  olasılık 0.119
  "a"  olasılık 0.090
  "i"  olasılık 0.067

Modelin ürettiği "atasözleri":
  p ayekacu suşl ol a dene mavminaker yı
  uz
  siladesi bi hz olz
  tin çi mazhoğıllat asese ydückerun hectatıri or
  ktademaşmin shzuz
  dü yema dışlır domağıraccaktlkayolır hacin vağadokuter
  giraparyı
  suboğrmayolale k

Ortalama kayıp (karakter başına): 2.4175
Şaşkınlık (perplexity): 11.22   (rastgele tahmin: 30)
```

Çıktıyı okumak

Sayılar veri hakkında gerçek şeyler söylüyor. Atasözlerinin %10'dan fazlası "a" ile başlıyor ("acele", "ağaç", "akıl", "armut"). "ş" harfinden sonra en sık boşluk geliyor, çünkü "taş", "iş", "ateş" gibi sözcükler ş ile biter.

Üretilen "atasözleri" ise neredeyse anlamsız. Yine de bir şeyler öğrenilmiş: sözcük uzunlukları makul, ünlüler ile ünsüzler sırayla geliyor, "ol", "bi", "ya" gibi Türkçe heceler görünüyor. Model yalnızca bir önceki harfi bildiği için bir sözcüğün ortasında olduğunu, bir cümlenin neresinde bulunduğunu hatırlayamıyor.

Kayıp ve şaşkınlık

Bir dil modelinin ne kadar iyi olduğunu tek sayıyla ölçebiliriz. Verideki her karakter için modelin o karaktere verdiği olasılığın eksi logaritmasını alıp ortalarız. Bu, 1. bölümde tanıdığımız **kayıptır**.

$$\text{kayıp} = -\frac{1}{N}\sum_{i=1}^{N} \ln P(x_i \mid x_{i-1}), \qquad \text{şaşkınlık} = e^{\text{kayıp}}$$

Kayıp, sezgisel bir sözcükle de anlatılabilir: **şaşkınlık** (perplexity). Şaşkınlığı 11,2 olan bir model, her adımda 11 eşit olasılıklı seçenek arasından seçim yapıyormuş gibi kararsızdır. Hiçbir şey bilmeyen bir model 30 karakterin hepsine eşit olasılık verirdi; şaşkınlığı 30, kaybı $\ln 30 \approx 3{,}40$ olurdu. Bu sayıyı akılda tutmakta yarar vardır: eğitilmemiş her modelimiz bu civarda başlayacaktır.

Program, sayımlara 1 ekleyerek **düzleştirme** yapıyor. Bu olmasaydı veride hiç görülmemiş bir çiftin olasılığı 0 olurdu ve log 0 tanımsız olduğu için kayıp sonsuza giderdi. Düzleştirmenin küçük bir bedeli vardır: düzleştirmesiz sayımın kaybı 2,341 iken düzleştirilmiş modelinki 2,418'dir.

Alıştırma. Programı üçlü (trigram) modele çeviriniz: olasılığı son *iki* karaktere göre hesaplayınız. Tablonuz kaç satırlık olur? Kayıp nasıl değişiyor ve üretilen metin daha Türkçe görünüyor mu?

BÖLÜM

4

Sinir ağıyla ikili model

Saymak küçük bir pencere için işe yarar ama pencere büyüdükçe imkânsızlaşır. Son 10 karaktere bakan bir model için 30^{10}, yani yaklaşık 590 trilyon satırlık bir tablo gerekirdi ve bu satırların neredeyse hepsi veride hiç görülmemiş olurdu. Çıkış yolu, tabloyu doldurmak yerine olasılıkları **hesaplayan** bir fonksiyon öğrenmektir: bir sinir ağı.

Bu bölümde aynı ikili modeli sinir ağıyla kurup sayarak bulduğumuz sonuca ulaşıp ulaşamadığımıza bakacağız. Bu, sonraki bölümlerde kullanacağımız tüm parçaların en sade halidir.

Dört adım

1. **Tek-sıcak kodlama.** Karakter numarası 3 ise, 30 sayılık bir vektörün 3. yeri 1, geri kalanı 0 olur. `BirSicak` fonksiyonu bunu yapar.
2. **Logitler.** Bu vektörü 30×30'luk bir ağırlık matrisi W ile çarparız. Sonuç 30 sayıdır; her biri, o karakterin sırada gelme ihtimali için ham bir puandır. Bu

puanlara **logit** denir. Logitler eksi de olabilir, toplamları 1 olmak zorunda değildir.
3. **Softmaks.** Logitleri olasılığa çevirir: her birinin üstelini alır, toplama böler.
4. **Çapraz entropi.** Doğru karaktere verilen olasılığın eksi logaritmasıdır; 3. bölümdeki kayıpla aynı şeydir.

$$P(b \mid a) = \frac{e^{W_{ab}}}{\sum_x e^{W_{ax}}}$$

Bu formülün sürpriz bir sonucu vardır. Tek-sıcak bir vektörü W ile çarpmak, W'nin bir satırını seçmekten başka bir şey değildir. Yani ağ, her karakter için bir satır sayı öğreniyor. Bu satıra o karakterin **gömmesi** (embedding) denir. Büyük dil modellerinin ilk katmanı da tam olarak budur: her simge için bir gömme satırı olan dev bir tablo.

```
program SinirAğıylaİkiliModel
{ Aynı ikili modeli bu kez sinir ağıyla kuruyoruz. Saymak yerine bir ağırlık
  matrisi W öğreniriz:
     girdi   : önceki karakterin tek-sıcak vektörü x (30 sayı, biri 1, kalanı 0)
     logit   : x @ W        (30 puan, her olası sonraki karakter için bir tane)
     olasılık: softmaks(logit)
  Kayıp çapraz entropidir: doğru sonraki karaktere verilen olasılığın -log'u.
  x tek-sıcak olduğu için x @ W, W'nin bir satırını seçer; bu satıra o karakterin
  "gömmesi" (embedding) denir. Büyük dil modellerinin ilk katmanı tam olarak budur. }
kullan Atasozleri, DilAraclari, YZ

değişken
  veri, harfler: metin dizisi
  numara: sözlük
  girdiler, hedefler: tamsayı dizisi
  a, kar: metin
  i, j, n, önceki, adım: tamsayı
  X, W, kayıp, P: tensör
  opt: herhangi
  sözcük: metin

başla
  Tohum(42)
  veri := Atasozleri.Hepsi
  harfler := DilAraclari.Harfler
  n := Uzunluk(harfler)
  numara := DilAraclari.Numaralar(harfler)

  { Eğitim çiftleri: (önceki karakter, sonraki karakter) }
  girdiler := []; hedefler := []
  veri içindeki her a için
  başla
    önceki := 0
```

```
    (a + '.').Karakterler içindeki her kar için
    başla
      girdiler.Ekle(önceki); hedefler.Ekle(numara[kar]); önceki := numara[kar]
    bitir
  bitir
  yaz(Biçimle('{} eğitim çifti', Uzunluk(girdiler)), 1)

  X := BirSıcak(girdiler, n)                  { 5282 × 30 }
  W := RastgeleTensör(n, n, 0.1, doğru)       { öğrenilecek 900 ağırlık }
  opt := Adam([W], 0.1)
  1'den 300'e kadar her adım için
  başla
    opt.SıfırlaGrad
    kayıp := ÇaprazEntropi(X @ W, hedefler)
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 50 = 0 ise
      yaz(Biçimle('Adım {:3}: kayıp {:.4}', adım, kayıp.Değer), 1)
  bitir

  yaz('', 1); yaz('Sinir ağının ürettiği "atasözleri":', 1)
  1'den 8'e kadar her j için
  başla
    sözcük := ''; i := 0
    tekrarla
      P := Softmaks(BirSıcak([i], n) @ W)       { 1 × 30 olasılık }
      i := DilAraclari.Örnekle(P.Satır(0).Veri)
      eğer i <> 0 ise sözcük := sözcük + harfler[i]
    i = 0 olana dek
    yaz('  ' + sözcük, 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
5282 eğitim çifti
Adım   1: kayıp 3.4135
Adım  50: kayıp 2.3569
Adım 100: kayıp 2.3475
Adım 150: kayıp 2.3449
Adım 200: kayıp 2.3436
Adım 250: kayıp 2.3429
Adım 300: kayıp 2.3424

Sinir ağının ürettiği "atasözleri":
  n ayekacu suşi n a baker
  k sttin klazü
  tükolde dör
  an hvar
```

```
yaven çi mazarüşen sa paye yağ hanevademaserırgere ktademaşmen seyvu dü yema
    ↪ dışiştakomağır bacayliltlezan dancayağadi yoll inden muş
sugöz umir ban k
sır
itanı yolürar
```

Çıktıyı okumak

Kayıp 3,41'den başlıyor; bu, 3. bölümde hesapladığımız "hiçbir şey bilmeyen model" değerine ($\ln 30 \approx 3{,}40$) çok yakındır. Ağırlıklar küçük ve rastgele olduğu için model başta tüm karakterlere hemen hemen eşit olasılık veriyor.

300 adımda kayıp 2,3424'e iniyor. Düzleştirmesiz saymanın kaybı 2,3408 idi. Sinir ağı neredeyse aynı yere vardı ve vardığı yer bir rastlantı değildir: ikili bir model için en iyi olasılıklar, sayımdan çıkan oranlardır. Gradyan inişi bu oranları kendi başına buldu. Üretilen metnin 3. bölümdekine bu kadar benzemesinin nedeni de budur.

Değişen şey yöntemdir. Artık elimizde kayıp fonksiyonunu otomatik olarak küçülten bir düzenek vardır: `kayıp.GeriYay` her ağırlığın kayba etkisini (gradyanını) hesaplıyor, `opt.Adım` da ağırlıkları kaybı azaltacak yöne biraz kaydırıyor. Bu düzenek ağ ne kadar karmaşık olursa olsun aynı kalacak. Kitabın geri kalanında yalnızca `X @ W` kısmını daha akıllı bir fonksiyonla değiştireceğiz.

Neden ÇaprazEntropi'ye logit veriyoruz?

Dikkat ederseniz eğitimde `ÇaprazEntropi(X @ W, hedefler)` yazıyoruz; araya `Softmaks` koymuyoruz. `ÇaprazEntropi` softmaksı kendisi uygular ve bunu sayısal olarak daha güvenli bir yolla yapar. Metin üretirken ise olasılıkları kendimiz istediğimiz için `Softmaks`'ı elle çağırıyoruz.

Alıştırma. Eğitimden sonra "ş" harfinin satırına softmaks uygulayınız: `Softmaks(BirSıcak([numara['ş']], n) @ W)`. En büyük beş olasılığı 3. bölümdeki "ş harfinden sonra en sık gelenler" listesiyle karşılaştırınız.

BÖLÜM

5

Bağlamı büyütmek

İkili modelin en büyük zayıflığı unutkanlığıdır: "ol" yazdıktan sonra "u" mu "a" mı geleceğini bilmek için yalnızca "l" harfine bakar. Bu bölümde modeli son **üç** karaktere bakacak şekilde genişletiyoruz. Kuracağımız model, Yoshua Bengio ve arkadaşlarının 2003'te yayımladığı sinirsel dil modelinin küçük bir halidir.

Modelin yapısı

1. Son üç karakterin her biri, 4. bölümde tanıdığımız gömme tablosundan (burada `C`) 10 sayılık bir vektöre çevrilir. Tablo üç konum için ortaktır: "a" harfi nerede görülürse görülsün aynı gömmeyi alır.
2. Üç gömme 96 nöronlu bir gizli katmana girer. Her konumun kendi ağırlık matrisi vardır (`W1`, `W2`, `W3`); böylece model "son harf" ile "üç önceki harf" arasındaki farkı bilir.
3. Gizli katmanın çıktısı, 30 logit üreten bir çıkış katmanından geçer.

$$h = \tanh(e_1 W_1 + e_2 W_2 + e_3 W_3 + b), \qquad \text{logit} = h\,W_{ç} + b_{ç}$$

Makalelerde genellikle üç gömme yan yana konup 30 sayılık tek bir vektör yapılır ve tek bir büyük matrisle çarpılır. Bizim yazdığımız biçim matematiksel olarak aynıdır; büyük matrisi üç parçaya bölmüş olduk.

İki yeni alışkanlık

Eğitim ve doğrulama ayrımı. Bir modelin gerçekten öğrenip öğrenmediğini, hiç görmediği veride sınamadan bilemeyiz. Atasözlerinin ilk %90'ını eğitim, son %10'unu (18 atasözü) doğrulama için ayırıyoruz. Model doğrulama atasözlerini eğitim boyunca hiç görmez.

Mini-yığınlar. 4. bölümde her adımda tüm veriyi kullanıyorduk. Burada her adımda rastgele seçilmiş 64 örnekle çalışıyoruz. Gradyan biraz gürültülü olur ama adımlar çok daha ucuzlar. Büyük modeller de böyle eğitilir; tüm internet verisini tek adımda işlemek mümkün değildir.

```
program BağlamlıDilModeli
{ İkili model yalnızca bir önceki karaktere bakıyordu. Bu model son BAĞLAM (3)
  karaktere bakar (Bengio ve arkadaşlarının 2003'teki modelinin küçük bir hali):
     1) Her karakter, öğrenilen C tablosundan 10 sayılık bir gömmeye çevrilir.
     2) Üç gömme gizli katmana girer: h = tanh(e1 @ W1 + e2 @ W2 + e3 @ W3 + b)
        (üç gömmeyi yan yana koyup tek bir büyük matrisle çarpmakla aynı şeydir).
     3) Çıktı katmanı 30 logit üretir: logit = h @ Wç + bç
  Veriyi eğitim (%90) ve doğrulama (%10) diye ayırırız; doğrulama kaybı, modelin
  ezberleyip ezberlemediğini gösterir. Her adımda rastgele 64 örnekten oluşan bir
  yığın (mini-batch) kullanırız. }
kullan Atasozleri, DilAraclari, YZ

sabit
  BAĞLAM: tamsayı = 3
  GÖMME: tamsayı = 10
  GİZLİ: tamsayı = 96
  YIĞIN: tamsayı = 64

değişken
  veri, harfler: metin dizisi
  numara: sözlük
  bağlamlar: liste                  { her örnek: son 3 karakterin numarası }
  hedefler: tamsayı dizisi
  eğitimSayısı, n, i, k, adım: tamsayı
  C, W1, W2, W3, b, Wç, bç, kayıp: tensör
  opt: herhangi
  seçilen: tamsayı dizisi
  pencere: tamsayı dizisi
  kar, sözcük: metin
```

```
fonksiyon Logitler(bağ: liste): tensör
{ bağ: bağlam listesi (her biri 3 numara) → her bağlam için 30 logit }
değişken e1, e2, e3, h: tensör
başla
  e1 := BirSıcak(Eşle(bağ, fonksiyon(p: liste) => p[0]), Uzunluk(harfler)) @ C
  e2 := BirSıcak(Eşle(bağ, fonksiyon(p: liste) => p[1]), Uzunluk(harfler)) @ C
  e3 := BirSıcak(Eşle(bağ, fonksiyon(p: liste) => p[2]), Uzunluk(harfler)) @ C
  h := (e1 @ W1 + e2 @ W2 + e3 @ W3 + b).Tanh
  dön h @ Wç + bç
bitir

fonksiyon OrtalamaKayıp(ilk, son: tamsayı): ondalık
{ [ilk, son) aralığındaki örneklerin ortalama kaybı (eğitim yapmadan) }
başla
  dön ÇaprazEntropi(Logitler(Dilim(bağlamlar, ilk, son)), Dilim(hedefler, ilk,
      ↪ son)).Değer
bitir

başla
  Tohum(7)
  veri := Atasozleri.Hepsi
  harfler := DilAraclari.Harfler
  n := Uzunluk(harfler)
  numara := DilAraclari.Numaralar(harfler)

  { Örnekleri çıkar. Atasözlerinin ilk %90'ı eğitim, son %10'u doğrulama. }
  bağlamlar := []; hedefler := []
  0'dan Uzunluk(veri) - 1'e kadar her i için
  başla
    eğer i = Uzunluk(veri) * 9 böl 10 ise eğitimSayısı := Uzunluk(hedefler)
    pencere := [0, 0, 0]
    (veri[i] + '.').Karakterler içindeki her kar için
    başla
      bağlamlar.Ekle(pencere); hedefler.Ekle(numara[kar])
      pencere := [pencere[1], pencere[2], numara[kar]]
    bitir
  bitir
  yaz(Biçimle('{} eğitim, {} doğrulama örneği', eğitimSayısı, Uzunluk(hedefler) -
      ↪ eğitimSayısı), 1)

  C  := RastgeleTensör(n, GÖMME, 1.0, doğru)
  W1 := RastgeleTensör(GÖMME, GİZLİ, 0.2, doğru)
  W2 := RastgeleTensör(GÖMME, GİZLİ, 0.2, doğru)
  W3 := RastgeleTensör(GÖMME, GİZLİ, 0.2, doğru)
  b  := Sıfırlar(GİZLİ, boş, doğru)
  Wç := RastgeleTensör(GİZLİ, n, 0.1, doğru)
  bç := Sıfırlar(n, boş, doğru)
  opt := Adam([C, W1, W2, W3, b, Wç, bç], 0.01)
```

```
  yaz(Biçimle('Parametre sayısı: {}', n * GÖMME + 3 * GÖMME * GİZLİ + GİZLİ + GİZLİ *
      ↪ n + n), 1)

  1'den 3000'e kadar her adım için
  başla
    seçilen := Eşle(Aralık(YIĞIN), fonksiyon(x: tamsayı) => Rastgele(eğitimSayısı))
    opt.SıfırlaGrad
    kayıp := ÇaprazEntropi(Logitler(Eşle(seçilen, fonksiyon(x: tamsayı) =>
        ↪ bağlamlar[x])),
                           Eşle(seçilen, fonksiyon(x: tamsayı) => hedefler[x]))
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 500 = 0 ise
      yaz(Biçimle('Adım {:4}: eğitim kaybı {:.4}   doğrulama kaybı {:.4}', adım,
        OrtalamaKayıp(0, eğitimSayısı), OrtalamaKayıp(eğitimSayısı,
            ↪ Uzunluk(hedefler))), 1)
  bitir

  yaz('', 1); yaz('Modelin ürettiği "atasözleri":', 1)
  1'den 8'e kadar her k için
  başla
    sözcük := ''; pencere := [0, 0, 0]
    tekrarla
      i := DilAraclari.Örnekle(Softmaks(Logitler([pencere])).Satır(0).Veri)
      eğer i <> 0 ise sözcük := sözcük + harfler[i]
      pencere := [pencere[1], pencere[2], i]
    i = 0 olana dek
    yaz('  ' + sözcük, 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
4724 eğitim, 558 doğrulama örneği
Parametre sayısı: 6186
Adım    1: eğitim kaybı 3.4243   doğrulama kaybı 3.4153
Adım  500: eğitim kaybı 1.8190   doğrulama kaybı 2.4013
Adım 1000: eğitim kaybı 1.5370   doğrulama kaybı 2.4427
Adım 1500: eğitim kaybı 1.4076   doğrulama kaybı 2.5698
Adım 2000: eğitim kaybı 1.3662   doğrulama kaybı 2.7467
Adım 2500: eğitim kaybı 1.2681   doğrulama kaybı 2.7659
Adım 3000: eğitim kaybı 1.2563   doğrulama kaybı 2.8837

Modelin ürettiği "atasözleri":
  kın gönül sıvamahme verkendi söylemek açtırı yoğurttur
  ayama kırk dibin olar
  akıllüyarı söyleme ne insan gele kaz geler yoğuğa döne zotuz
  davul ne gülüm üzüm üzüm uçun elin konu kokla san ne aştan ne anlı düşen kargayır gün
      ↪ işler ölür
```

```
gülünen olur
gülüşen olur
akılsan düşmanı yürür
kur düşen insan da da düşma meyen olur
```

Çıktıyı okumak: ezberleme

Üretilen metin çok daha Türkçe görünüyor. "gönül", "söylemek", "kırk", "insan", "düşmanı yürür" gibi gerçek sözcükler ve "gülüşen olur" gibi atasözü tadında dizeler belirdi. Üç karakterlik bağlam, sözcük ve hece düzeyinde büyük fark yaratıyor.

Ama kayıp sütunları önemli bir uyarı veriyor. Eğitim kaybı 3000 adım boyunca düzenli olarak düşüyor (1,82 → 1,26). Doğrulama kaybı ise en iyi değerine (2,40) 500. adımda ulaşıyor, sonra **yükseliyor** ve 2,88'e çıkıyor. Model, eğitim atasözlerini giderek daha iyi ezberliyor ama hiç görmediği atasözlerinde giderek kötüleşiyor. Bu duruma **aşırı uyum** (overfitting) denir.

Nedeni basit: 6186 parametreli bir model, 4724 örneği neredeyse tek tek ezberleyebilecek kadar büyüktür. Üretilen metinde "kırk", "davul", "üzüm üzüm" gibi eğitim verisinden birebir parçalar görmeniz bunun işaretidir. Büyük dil modelleri bu sorunu öncelikle devasa veriyle çözer: modelin ezberleyemeyeceği kadar çok metin. Bizim verimiz küçük olduğu için 8. bölümde iki teknik kullanacağız. **Erken durdurma**, doğrulama kaybı en düşükken modeli kaydetmektir. **Bırakma** (dropout) ise eğitim sırasında ağın bir kısmını rastgele susturmaktır.

Bir karşılaştırma notu: 3. ve 4. bölümlerdeki kayıplar tüm veri üzerinde, yani modelin gördüğü veride ölçülmüştü. Bu bölümden itibaren asıl ölçütümüz doğrulama kaybıdır.

Alıştırma. `GİZLİ` sabitini 96'dan 24'e indiriniz ve programı yeniden çalıştırınız. Doğrulama kaybının en düşük değeri ve yükselmeye başladığı adım nasıl değişiyor? Ardından adım sayısını 500'e indirip üretilen metne bakınız.

BÖLÜM

6

Öz-dikkat

Bir önceki bölümdeki model son üç karaktere bakıyordu. Bağlamı 30 karaktere çıkarmak istesek ne olur? Her konum için ayrı bir ağırlık matrisi gerekir. Parametre sayısı bağlamla birlikte büyür. Daha kötüsü, model "8 karakter önceki harf" ile "9 karakter önceki harf" için ayrı ayrı kurallar öğrenmek zorunda kalır.

2017'de yayımlanan "Attention Is All You Need" makalesi farklı bir yol önerdi: her karakter geçmişteki *tüm* karakterlere baksın, ama hangisine ne kadar bakacağına kendisi karar versin. Bu düzeneğe **öz-dikkat** (self-attention) denir. Bugünkü tüm büyük dil modellerinin kalbi budur.

Adım 1: geçmişin ortalaması

En basit fikirle başlayalım. Her karakter, kendisi dahil önceki tüm karakterlerin vektörlerinin ortalamasını alsın. Böylece geçmişin bir özetine sahip olur. Bunu bir matris çarpımıyla yapabiliriz: alt üçgen bir **ağırlık matrisi** ile çarpmak, her satırda geçmişin ağırlıklı toplamını verir.

Ağırlıkları doğrudan yazmak yerine bir hile kullanırız. Önce her çift için bir **puan** hesaplarız. Gelecekteki karakterlerin puanına çok büyük bir eksi sayı ekleriz; bu işleme **nedensel maske** denir. Sonra softmaks uygularız. Softmaks, eksi sonsuza yakın puanları sıfır olasılığa, kalanları toplamı 1 olan ağırlıklara çevirir. Tüm puanlar eşitse sonuç tam olarak ortalamadır.

Maske neden gerekli? Eğitimde model bir pencerenin tüm konumları için aynı anda tahmin yapar. Maske olmasaydı 3. konumdaki karakter, tahmin etmesi gereken 4. karakteri görebilirdi; model kopya çekmeyi öğrenir, metin üretirken de işe yaramazdı.

Adım 2: sorgu, anahtar, değer

Eşit ağırlık kötü bir fikirdir: "gönül" sözcüğündeki "l", kendinden önceki "ü"ye "g"den daha çok bakmak isteyebilir. Öz-dikkatte her karakter üç vektör üretir:

- **Sorgu** (q): "Ben ne arıyorum?"
- **Anahtar** (k): "Bende ne var?"
- **Değer** (v): "Bana bakan olursa ona ne aktarırım?"

Bir karakterin sorgusu ile ötekinin anahtarının nokta çarpımı puanı verir: ikisi ne kadar uyuşuyorsa puan o kadar yüksektir. Puanlar maskelenip softmakstan geçirilir ve ağırlıklar, değerlerin ağırlıklı ortalamasını almak için kullanılır. Üç vektör de karakterin gömmesinden öğrenilen üç matrisle (Wq, Wk, Wv) elde edilir.

$$\text{Dikkat}(Q, K, V) = \text{softmaks}\left(\frac{QK^\top}{\sqrt{d}} + M\right) V$$

Buradaki $\sqrt{\text{d}}$ bölümü (d: sorgu vektörünün uzunluğu) puanların çok büyümesini önler. Puanlar çok büyük olursa softmaks tüm ağırlığı tek bir karaktere verir ve gradyanlar neredeyse sıfırlanır.

Adım 3: konum bilgisi

Öz-dikkatin şaşırtıcı bir özelliği vardır: sırayı bilmez. Ağırlıklı ortalama, toplanan şeylerin sırasına bağlı değildir. "elma" ile "lema"nın son harfi aynı karakterlere bakar ve aynı sonucu üretir. Çözüm, her konuma öğrenilen bir **konum gömmesi** vermek ve bunu karakter gömmesine eklemektir. Böylece "1. konumdaki e" ile "2. konumdaki e" farklı vektörler olur.

Program bu üç adımı sırasıyla gösteriyor. Ağırlıklar henüz eğitilmedi; bu bölümde yalnızca hesabın nasıl işlediğini görüyoruz.

```
program ÖzDikkat
{ Öz-dikkat, her karakterin kendinden önceki karakterlerden bilgi toplamasını sağlar.
```

```
  A) En basit hali: geçmişin ortalamasını almak. Nedensel maske, bir karakterin
     GELECEKTEKİ karakterleri görmesini engeller.
  B) Gerçek öz-dikkat: ortalama eşit ağırlıklı değildir. Her karakter bir sorgu (q),
     bir anahtar (k) ve bir değer (v) üretir. q·k büyükse o karaktere çok bakılır.
        ağırlıklar = softmaks(Q Kᵀ / √d + maske),   çıktı = ağırlıklar @ V
  C) Dikkat sırayı bilmez: aynı karakterler farklı sırayla gelse de sonuç aynıdır.
     Konum gömmesi eklemek bu sorunu çözer. }
kullan Atasozleri, DilAraclari, YZ, Matematik

değişken
  x, M, A, Q, K, V, E, Wq, Wk, Wv, Konum, girdi: tensör
  harfler: metin dizisi
  numara: sözlük

fonksiyon NedenselMaske(n: tamsayı): tensör
değişken i, j: tamsayı; satırlar: liste; s: ondalık dizisi
başla
  satırlar := []
  0'dan n - 1'e kadar her i için
  başla
    s := []
    0'dan n - 1'e kadar her j için eğer j < = i ise s.Ekle(0.0) değilse
        ↪  s.Ekle(-1000000000.0)
    satırlar.Ekle(s)
  bitir
  dön Tensör(satırlar)
bitir

prosedür MatrisYaz(başlık: metin; mat: tensör)
değişken i, j: tamsayı; s: metin
başla
  yaz(başlık, 1)
  0'dan mat.Şekil[0] - 1'e kadar her i için
  başla
    s := '   '
    0'dan mat.Şekil[1] - 1'e kadar her j için s := s + Biçimle('{:7.2}', mat[i, j])
    yaz(s, 1)
  bitir
bitir

fonksiyon SonSatırınÇıktısı(sözcük: metin; konumlu: mantıksal): tensör
{ Sözcüğün karakterlerini gömer, öz-dikkati uygular, son karakterin çıktısını
    ↪  döndürür. }
değişken G, Ağ: tensör; n: tamsayı
başla
  n := Uzunluk(sözcük)
  G := BirSıcak(DilAraclari.Kodla(sözcük, numara), Uzunluk(harfler)) @ E
  eğer konumlu ise G := G + BirSıcak(Aralık(n), 8) @ Konum
  Ağ := Softmaks((G @ Wq) @ (G @ Wk).T / Kök(4) + NedenselMaske(n))
```

```
  dön (Ağ @ (G @ Wv)).Satır(n - 1)
bitir

başla
  { A) Geçmişin ortalaması }
  x := Tensör([[1.0, 0.0], [3.0, 2.0], [5.0, 4.0], [7.0, 2.0]])
  M := NedenselMaske(4)
  A := Softmaks(Sıfırlar(4, 4) + M)          { tüm puanlar eşit → eşit ağırlık }
  MatrisYaz('A) Maskeli eşit ağırlıklar (satır: bakan, sütun: bakılan):', A)
  MatrisYaz('   ağırlıklar @ x  =  her satırda, o ana kadarki satırların
      ↪  ortalaması:', A @ x)

  { B) Sorgu, anahtar, değer }
  Tohum(3)
  harfler := DilAraclari.Harfler
  numara := DilAraclari.Numaralar(harfler)
  E := RastgeleTensör(Uzunluk(harfler), 8, 1.0)      { 8 boyutlu karakter gömmeleri }
  Wq := RastgeleTensör(8, 4, 0.5)
  Wk := RastgeleTensör(8, 4, 0.5)
  Wv := RastgeleTensör(8, 4, 0.5)
  girdi := BirSıcak(DilAraclari.Kodla('gönül', numara), Uzunluk(harfler)) @ E
  Q := girdi @ Wq; K := girdi @ Wk; V := girdi @ Wv
  A := Softmaks(Q @ K.T / Kök(4) + NedenselMaske(5))
  yaz('', 1)
  MatrisYaz('B) "gönül" için dikkat ağırlıkları (rastgele, henüz eğitilmemiş Wq,
      ↪ Wk):', A)
  yaz('   Satır toplamları: ' + Eşle(Aralık(5), fonksiyon(r: tamsayı) =>
      ↪ Yuvarla(A.Satır(r).Toplam.Değer, 3)), 1)
  yaz('   Çıktının şekli: ' + (A @ V).Şekil + '  (5 karakter, her biri için 4 sayı)',
      ↪ 1)

  { C) Konum bilgisi }
  Konum := RastgeleTensör(8, 8, 1.0)
  yaz('', 1)
  yaz('C) Son karakterin ("a") çıktısı:', 1)
  yaz('   konumsuz  "elma": ' + Eşle(SonSatırınÇıktısı('elma', yanlış).Veri,
      ↪ fonksiyon(z: ondalık) => Yuvarla(z, 3)), 1)
  yaz('   konumsuz  "lema": ' + Eşle(SonSatırınÇıktısı('lema', yanlış).Veri,
      ↪ fonksiyon(z: ondalık) => Yuvarla(z, 3)), 1)
  yaz('   konumlu   "elma": ' + Eşle(SonSatırınÇıktısı('elma', doğru).Veri,
      ↪ fonksiyon(z: ondalık) => Yuvarla(z, 3)), 1)
  yaz('   konumlu   "lema": ' + Eşle(SonSatırınÇıktısı('lema', doğru).Veri,
      ↪ fonksiyon(z: ondalık) => Yuvarla(z, 3)), 1)
bitir.
```

Programın çıktısı şöyledir:

```
A) Maskeli eşit ağırlıklar (satır: bakan, sütun: bakılan):
      1.00   0.00   0.00   0.00
      0.50   0.50   0.00   0.00
      0.33   0.33   0.33   0.00
      0.25   0.25   0.25   0.25
   ağırlıklar @ x  =  her satırda, o ana kadarki satırların ortalaması:
      1.00   0.00
      2.00   1.00
      3.00   2.00
      4.00   2.00

B) "gönül" için dikkat ağırlıkları (rastgele, henüz eğitilmemiş Wq, Wk):
      1.00   0.00   0.00   0.00   0.00
      0.76   0.24   0.00   0.00   0.00
      0.23   0.30   0.47   0.00   0.00
      0.41   0.14   0.32   0.13   0.00
      0.01   0.02   0.03   0.51   0.43
   Satır toplamları: [1.0, 1.0, 1.0, 1.0, 1.0]
   Çıktının şekli: [5, 4]  (5 karakter, her biri için 4 sayı)

C) Son karakterin ("a") çıktısı:
   konumsuz  "elma": [-0.273, -0.001, -0.047, 1.101]
   konumsuz  "lema": [-0.273, -0.001, -0.047, 1.101]
   konumlu   "elma": [0.153, 2.455, -0.951, 0.739]
   konumlu   "lema": [0.536, -1.385, 0.089, 0.55]
```

Çıktıyı okumak

A bölümündeki matris tam beklendiği gibidir: köşegenin üstü sıfır (geleceğe bakılmıyor) ve her satırda ağırlıklar eşittir. Çarpımın ikinci satırı $(2, 1)$, ilk iki satırın ortalamasıdır: $(1 + 3) \,/\, 2 = 2$ ve $(0 + 2) \,/\, 2 = 1$.

B bölümünde ağırlıklar artık eşit değildir. Son satırda "l", ağırlığının %51'ini "ü"ye, %43'ünü kendisine veriyor; "g"ye neredeyse hiç bakmıyor. Bu ağırlıklar rastgele Wq ve Wk'den geldiği için şimdilik anlamsızdır. Eğitim, bu matrisleri faydalı bakış desenleri üretecek biçimde ayarlayacak. Her satırın toplamı yine 1'dir.

C bölümü konum sorununu gösteriyor. Konum gömmesi yokken "elma" ve "lema"nın son harfi birebir aynı çıktıyı üretiyor; model iki sözcüğü ayırt edemez. Konum gömmesi eklenince çıktılar farklılaşıyor.

Alıştırma. B bölümünde puanları $\sqrt{4}$'e bölmeyi kaldırınız ve Wq ile Wk'nin ölçeğini 0,5'ten 2,0'a çıkarınız. Ağırlık matrisi nasıl değişiyor? Neden çoğu satırda tek bir 1,00 görüyorsunuz?

BÖLÜM

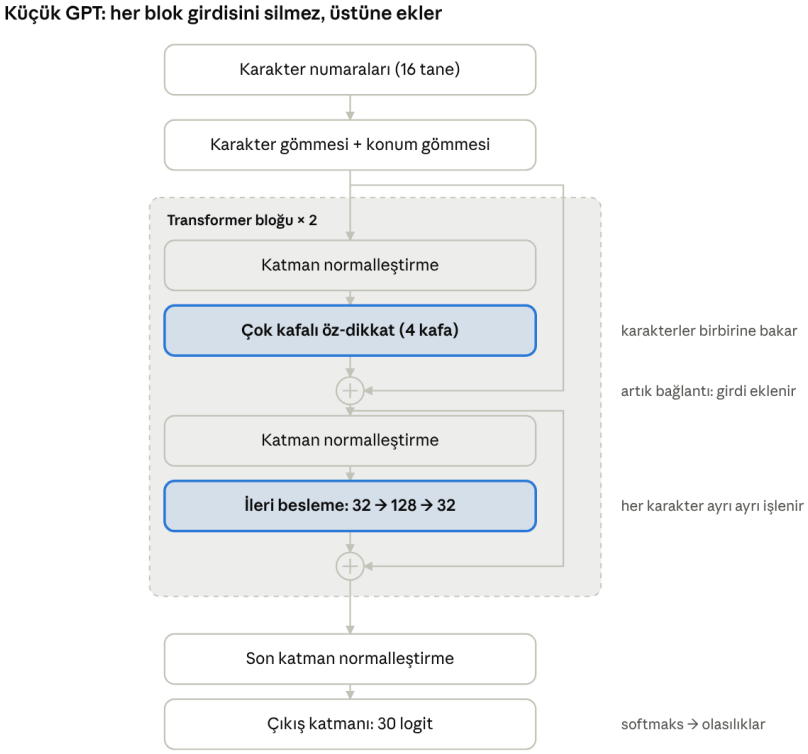
7

Transformer bloğu

Öz-dikkat tek başına bir dil modeli değildir. Onu gerçek bir modele çeviren dört parça daha vardır. Bu parçalar bir **Transformer bloğu** oluşturur ve bloklar üst üste dizilir. Bizim modelimizde iki blok vardır; büyük modellerde ise onlarca, hatta yüzden fazla blok bulunur.

Çok kafalı dikkat

Tek bir dikkat "kafası" bir tür bakış deseni öğrenebilir: örneğin "bir önceki harfe bak". Ama bir karakterin aynı anda birkaç şeye bakması gerekebilir: bir önceki harfe, sözcüğün başına, bir önceki sözcüğün son ünlüsüne. Çözüm, birkaç küçük kafayı yan yana çalıştırmaktır. Modelimizde 32 boyutu 4 kafaya bölüyoruz; her kafanın sorgu, anahtar ve değer vektörleri 8 boyutludur. Kafaların çıktıları birer çıkış matrisiyle (`Wo`) yeniden 32 boyuta taşınıp toplanır.



Şekil 7.1. *Küçük GPT'nin katmanları: iki Transformer bloğu*

İleri besleme ağı

Dikkat, karakterler arasında bilgi taşır ama bu bilgiyi işlemez; yalnızca ağırlıklı ortalama alır. Her dikkat katmanından sonra her karakter, kendi vektörünü küçük bir sinir ağından geçirir: 32 sayıyı 128'e genişletir, ReLU uygular, yeniden 32'ye indirir. Bu ağ her karakter için aynıdır ve karakterler arasında bilgi alışverişi yapmaz. Bazen şöyle özetlenir: dikkat "iletişim", ileri besleme ise "düşünme"dir.

Artık bağlantı

Her alt katmanın sonucu, girdisinin **üstüne eklenir**: `x := x + Dikkat(x)`. Buna **artık bağlantı** (residual connection) denir. İki yararı vardır. Birincisi, bir katman hiçbir şey öğrenmese bile girdiyi bozmaz; en kötü ihtimalle sıfır ekler. İkincisi, gradyan eğitim sırasında toplama işlemlerinin üzerinden kısalmadan geriye akar; derin modellerin eğitilebilmesinin sırrı büyük ölçüde budur.

Katman normalleştirme

Vektörlerdeki sayılar katmandan katmana büyüyüp küçülebilir; bu da eğitimi dengesizleştirir. **Katman normalleştirme**, her karakterin vektörünü ortalaması 0, standart sapması 1 olacak şekilde ölçekler. Ardından öğrenilen iki vektörle (g ve b) yeniden ölçekleyip kaydırır. Bizim modelimizde (GPT-2'de olduğu gibi) normalleştirme her alt katmanın *önünde* durur.

$$\text{KatmanNorm}(x) = g \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + b$$

Kitabın modeli: KucukGPT birimi

Anlattığımız her şey `KucukGPT` biriminde, `GPT` adında bir sınıfta toplanıyor. Birim ayrıca 8. bölümde kullanacağımız iki yardımcıyı da içeriyor: **bırakma** (`Bırak`) ile ağırlıkları dosyaya kaydedip geri yükleyen `Kaydet` ve `Yükle`.

Verimlilik için bir hile kullanıyoruz. Bir yığındaki 8 pencereyi ayrı ayrı işlemek yerine alt alta koyup 128 satırlık tek bir matris yapıyoruz. Maske, her karakterin yalnızca *kendi* penceresindeki önceki karakterlere bakmasını sağlıyor. Böylece pencereler birbirine karışmıyor.

```
birim KucukGPT
{ Kitabın küçük GPT'si: karakter düzeyinde, yalnızca kod çözücüden (decoder-only)
  oluşan bir Transformer. 6. ve 7. bölümde parça parça kurduğumuz her şey burada
  tek bir sınıfta toplanır.

  Yığın (batch) düzeni: B dizinin her biri t karakter uzunluğundadır. Hepsini alt
      ↪ alta
  koyup (B·t) × D boyutlu tek bir matrisle çalışırız. Dikkat maskesi, bir karakterin
  yalnızca KENDİ dizisindeki ve KENDİSİNDEN ÖNCEKİ karakterlere bakmasına izin verir.
      ↪ }
kullan YZ, Matematik, Sistem

arayüz

tip
  GPT = sınıf
    sözlükBoyu, bağlam, boyut, kafaSayısı, katmanSayısı, kafaBoyu: tamsayı
    Gömme, Konum, Wç, bç, sonG, sonB: tensör   { karakter ve konum gömmeleri, çıkış
        ↪ katmanı, son normalleştirme }
    Wq, Wk, Wv, Wo: liste                { [katman * kafaSayısı + kafa] → tensör }
    g1, b1, g2, b2: liste                { katman normalleştirme parametreleri, katman
        ↪ başına }
    İ1, İb1, İ2, İb2: liste              { ileri besleme ağı, katman başına }
    maskeler: sözlük                     { hazırlanmış maskeler, yeniden kurmamak için
        ↪ }
```

```
    bırakma: ondalık                  { eğitimde rastgele sıfırlanan değerlerin
        ↪ oranı (dropout) }
    eğitimde: mantıksal               { doğruysa bırakma uygulanır }
    kurucu Oluştur(sB, bU, bD, kH, kL: tamsayı)
    fonksiyon Parametreler: liste
    fonksiyon ParametreAdedi: tamsayı
    fonksiyon İleri(numaralar: tamsayı dizisi; diziSayısı, uzunluk: tamsayı): tensör
    fonksiyon Bırak(x: tensör): tensör
    prosedür Kaydet(yol: metin)
    prosedür Yükle(yol: metin)
  bitir

fonksiyon KatmanNorm(x, g, b: tensör): tensör
fonksiyon Maske(diziSayısı, uzunluk: tamsayı): tensör

uygulama

fonksiyon KatmanNorm(x, g, b: tensör): tensör
{ Her satırı (her karakterin vektörünü) ortalaması 0, sapması 1 olacak biçimde
  ölçekler; sonra öğrenilen g ile çarpar, b ile kaydırır. }
değişken satır: tamsayı; m, xm, s: tensör
başla
  satır := x.Şekil[0]
  m := x.Ortalama(1).Yeniden(satır, 1)
  xm := x - m
  s := ((xm ** 2).Ortalama(1) + 0.00001).Kök.Yeniden(satır, 1)
  dön xm / s * g + b
bitir

fonksiyon Maske(diziSayısı, uzunluk: tamsayı): tensör
{ Nedensel maske: i. satır j. sütuna bakabiliyorsa 0, bakamıyorsa çok büyük bir
  negatif sayı. Softmaks bu negatif puanları sıfır olasılığa çevirir. }
değişken n, i, j: tamsayı; satırlar: liste; satır1: ondalık dizisi
başla
  n := diziSayısı * uzunluk
  satırlar := []
  0'dan n - 1'e kadar her i için
  başla
    satır1 := []
    0'dan n - 1'e kadar her j için
      eğer (i böl uzunluk = j böl uzunluk) ve j < = i ise satır1.Ekle(0.0) değilse
          ↪ satır1.Ekle(-1000000000.0)
    satırlar.Ekle(satır1)
  bitir
  dön Tensör(satırlar)
bitir

kurucu GPT.Oluştur(sB, bU, bD, kH, kL: tamsayı)
değişken l, h: tamsayı; ölçek: ondalık
```

```
başla
  sözlükBoyu := sB; bağlam := bU; boyut := bD; kafaSayısı := kH; katmanSayısı := kL
  kafaBoyu := boyut böl kafaSayısı
  ölçek := 1 / Kök(boyut)
  Gömme := RastgeleTensör(sözlükBoyu, boyut, 0.5, doğru)
  Konum := RastgeleTensör(bağlam, boyut, 0.1, doğru)
  Wq := []; Wk := []; Wv := []; Wo := []
  g1 := []; b1 := []; g2 := []; b2 := []
  İ1 := []; İb1 := []; İ2 := []; İb2 := []
  0'dan katmanSayısı - 1'e kadar her l için
  başla
    0'dan kafaSayısı - 1'e kadar her h için
    başla
      Wq.Ekle(RastgeleTensör(boyut, kafaBoyu, ölçek, doğru))
      Wk.Ekle(RastgeleTensör(boyut, kafaBoyu, ölçek, doğru))
      Wv.Ekle(RastgeleTensör(boyut, kafaBoyu, ölçek, doğru))
      Wo.Ekle(RastgeleTensör(kafaBoyu, boyut, ölçek / Kök(2 * katmanSayısı), doğru))
    bitir
    g1.Ekle(Birler(boyut, boş, doğru)); b1.Ekle(Sıfırlar(boyut, boş, doğru))
    g2.Ekle(Birler(boyut, boş, doğru)); b2.Ekle(Sıfırlar(boyut, boş, doğru))
    İ1.Ekle(RastgeleTensör(boyut, 4 * boyut, ölçek, doğru)); İb1.Ekle(Sıfırlar(4 *
        ↪ boyut, boş, doğru))
    İ2.Ekle(RastgeleTensör(4 * boyut, boyut, 0.5 / Kök(4 * boyut * 2 * katmanSayısı),
        ↪ doğru)); İb2.Ekle(Sıfırlar(boyut, boş, doğru))
  bitir
  maskeler := [:]
  bırakma := 0.0
  eğitimde := yanlış
  sonG := Birler(boyut, boş, doğru); sonB := Sıfırlar(boyut, boş, doğru)
  Wç := RastgeleTensör(boyut, sözlükBoyu, 0.1, doğru); bç := Sıfırlar(sözlükBoyu,
      ↪ boş, doğru)
bitir

fonksiyon GPT.Parametreler: liste
başla
  dön [Gömme, Konum] + Wq + Wk + Wv + Wo + g1 + b1 + g2 + b2 + İ1 + İb1 + İ2 + İb2 +
      ↪ [sonG, sonB, Wç, bç]
bitir

fonksiyon GPT.ParametreAdedi: tamsayı
değişken p: tensör
başla
  sonuç := 0
  Parametreler içindeki her p için
    sonuç := sonuç + İndirge(p.Şekil, fonksiyon(a, c: tamsayı) => a * c, 1)
bitir

fonksiyon GPT.İleri(numaralar: tamsayı dizisi; diziSayısı, uzunluk: tamsayı): tensör
```

```
{ numaralar: diziSayısı × uzunluk karakter, alt alta. Dönüş: her konum için
    ↪ sözlükBoyu logit. }
değişken x, n1, dikkat, k, q, v, ağırlık, m: tensör; l, h, i: tamsayı
başla
  { maske her boyut için bir kez kurulur, sonra hazır olan kullanılır }
  eğer değil maskeler.İçerir(Biçimle('{}x{}', diziSayısı, uzunluk)) ise
    maskeler[Biçimle('{}x{}', diziSayısı, uzunluk)] := Maske(diziSayısı, uzunluk)
  m := maskeler[Biçimle('{}x{}', diziSayısı, uzunluk)]
  { karakter gömmesi + konum gömmesi }
  x := BirSıcak(numaralar, sözlükBoyu) @ Gömme +
       BirSıcak(Eşle(Aralık(diziSayısı * uzunluk), fonksiyon(j: tamsayı) => j mod
           ↪ uzunluk), bağlam) @ Konum
  x := Bırak(x)
  0'dan katmanSayısı - 1'e kadar her l için
  başla
    { 1) çok kafalı öz-dikkat, artık bağlantıyla }
    n1 := KatmanNorm(x, g1[l], b1[l])
    dikkat := boş
    0'dan kafaSayısı - 1'e kadar her h için
    başla
      i := l * kafaSayısı + h
      q := n1 @ Wq[i]; k := n1 @ Wk[i]; v := n1 @ Wv[i]
      ağırlık := Softmaks(q @ k.T / Kök(kafaBoyu) + m)
      eğer dikkat = boş ise dikkat := ağırlık @ v @ Wo[i] değilse dikkat := dikkat +
          ↪ ağırlık @ v @ Wo[i]
    bitir
    x := x + Bırak(dikkat)
    { 2) ileri besleme ağı, artık bağlantıyla }
    n1 := KatmanNorm(x, g2[l], b2[l])
    x := x + Bırak((n1 @ İ1[l] + İb1[l]).ReLU @ İ2[l] + İb2[l])
  bitir
  dön KatmanNorm(x, sonG, sonB) @ Wç + bç
bitir

fonksiyon GPT.Bırak(x: tensör): tensör
{ Bırakma (dropout): eğitim sırasında değerlerin rastgele bir kısmını sıfırlar,
    ↪ kalanları
  1 / (1 - oran) ile büyütür. Model tek bir yola güvenmemeyi öğrenir; ezberleme
      ↪ azalır. }
değişken r, c, i, j: tamsayı; satırlar: liste; s: ondalık dizisi; kalan: ondalık
başla
  eğer değil eğitimde veya bırakma < = 0 ise dön x
  r := x.Şekil[0]; c := x.Şekil[1]; kalan := 1 / (1 - bırakma)
  satırlar := []
  0'dan r - 1'e kadar her i için
  başla
    s := []
    0'dan c - 1'e kadar her j için eğer Rastgele < bırakma ise s.Ekle(0.0) değilse
        ↪ s.Ekle(kalan)
```

```
    satırlar.Ekle(s)
  bitir
  dön x * Tensör(satırlar)
bitir

prosedür GPT.Kaydet(yol: metin)
{ Her parametre bir satır: sayılar boşlukla ayrılmış }
değişken p: tensör; satırlar: metin dizisi; adet: tamsayı
başla
  satırlar := []
  Parametreler içindeki her p için
  başla
    adet := İndirge(p.Şekil, fonksiyon(a, c: tamsayı) => a * c, 1)
    satırlar.Ekle(Birleştir(Eşle(p.Yeniden(adet).Veri, fonksiyon(z: ondalık) =>
        ↪ Metne(z)), ' '))
  bitir
  Sistem.DosyaYaz(yol, Birleştir(satırlar, Karakter(10)))
bitir

prosedür GPT.Yükle(yol: metin)
değişken satırlar, sayılar: metin dizisi; p: tensör; i, j, sütun: tamsayı; ps: liste
başla
  satırlar := Sistem.DosyaOku(yol).Ayır(Karakter(10))
  ps := Parametreler
  0'dan Uzunluk(ps) - 1'e kadar her i için
  başla
    p := ps[i]
    sayılar := satırlar[i].Ayır(' ')
    eğer Uzunluk(p.Şekil) = 1 ise
      0'dan Uzunluk(sayılar) - 1'e kadar her j için p[j] := Ondalığa(sayılar[j])
    değilse
    başla
      sütun := p.Şekil[1]
      0'dan Uzunluk(sayılar) - 1'e kadar her j için p[j böl sütun, j mod sütun] :=
          ↪ Ondalığa(sayılar[j])
    bitir
  bitir
bitir

başla
bitir.
```

Parçaları sınamak

Büyük bir modeli eğitmeden önce parçalarını sınamak iyi bir alışkanlıktır. Aşağıdaki program üç şey yapıyor. Katman normalleştirmenin gerçekten ortalama 0, sapma 1 verdiğini gösteriyor. Parametrelerin nerede olduğunu elle sayarak `ParametreAdedi` ile karşılaştırıyor. Son olarak modelin tek bir küçük yığını ezber-

leyebildiğini doğruluyor. Bu son sınama çok değerlidir: model iki pencereyi bile ezberleyemiyorsa kodda bir hata vardır ve uzun bir eğitime başlamak boşuna olur.

```
program TransformerBloğu
{ Bir Transformer bloğu dört parçadan oluşur:
     n = KatmanNorm(x);  x = x + ÇokKafalıDikkat(n)   (karakterler birbirine bakar)
     n = KatmanNorm(x);  x = x + İleriBesleme(n)      (her karakter kendi başına
         ↪ işlenir)
  "x = x + ..." biçimine artık bağlantı denir: her blok girdiyi silmez, üstüne ekler.
  Bu program parçaları KucukGPT biriminden alıp tek tek sınar:
  A) Katman normalleştirme ne yapar?
  B) Küçük GPT'nin parametreleri nerede?
  C) Bir hata ayıklama alışkanlığı: model tek bir yığını ezberleyebiliyor mu? }
kullan Atasozleri, DilAraclari, KucukGPT, YZ

değişken
  x, y, kayıp: tensör
  model: GPT
  opt: herhangi
  harfler: metin dizisi
  numara: sözlük
  akış, g, h: tamsayı dizisi
  i, adım, D, gömme, dikkat, ileri, norm, çıkış: tamsayı

başla
  Tohum(5)
  { A) Katman normalleştirme }
  x := Tensör([[2.0, 4.0, 6.0, 8.0], [100.0, 101.0, 99.0, 100.0], [-3.0, 0.0, 0.0,
      ↪ 3.0]])
  y := KucukGPT.KatmanNorm(x, Birler(4), Sıfırlar(4))
  yaz('A) Katman normalleştirme öncesi ve sonrası:', 1)
  0'dan 2'ye kadar her i için
    yaz(Biçimle('   {}  →  {}   (ortalama {:.2}, sapma {:.2})', x.Satır(i).Veri,
      Eşle(y.Satır(i).Veri, fonksiyon(z: ondalık) => Yuvarla(z, 2)),
      y.Satır(i).Ortalama.Değer, ((y.Satır(i) ** 2).Ortalama.Değer) ** 0.5), 1)

  { B) Parametreler }
  harfler := DilAraclari.Harfler
  numara := DilAraclari.Numaralar(harfler)
  model := GPT.Oluştur(Uzunluk(harfler), 16, 32, 4, 2)
  D := 32
  gömme := Uzunluk(harfler) * D + 16 * D
  dikkat := 2 * (4 * D * D)
  ileri := 2 * (D * 4 * D + 4 * D + 4 * D * D + D)
  norm := 2 * (4 * D) + 2 * D
  çıkış := D * Uzunluk(harfler) + Uzunluk(harfler)
  yaz('', 1); yaz('B) Küçük GPT (2 katman, 4 kafa, 32 boyut, 16 karakter bağlam):',
      ↪ 1)
  yaz(Biçimle('   Karakter ve konum gömmeleri : {:6}', gömme), 1)
```

```
  yaz(Biçimle('   Dikkat (Q, K, V, çıkış)     : {:6}', dikkat), 1)
  yaz(Biçimle('   İleri besleme ağları        : {:6}', ileri), 1)
  yaz(Biçimle('   Katman normalleştirmeler    : {:6}', norm), 1)
  yaz(Biçimle('   Çıkış katmanı               : {:6}', çıkış), 1)
  yaz(Biçimle('   Toplam                      : {:6}   (model.ParametreAdedi = {})',
    gömme + dikkat + ileri + norm + çıkış, model.ParametreAdedi), 1)

  { C) Tek yığını ezberleme sınaması }
  akış := DilAraclari.Kodla('.damlaya damlaya göl olur.acele işe şeytan karışır.',
      ↪ numara)
  g := Dilim(akış, 0, 16) + Dilim(akış, 20, 36)
  h := Dilim(akış, 1, 17) + Dilim(akış, 21, 37)
  opt := Adam(model.Parametreler, 0.01)
  yaz('', 1); yaz('C) İki pencerelik tek bir yığını ezberleme:', 1)
  1'den 150'ye kadar her adım için
  başla
    opt.SıfırlaGrad
    kayıp := ÇaprazEntropi(model.İleri(g, 2, 16), h)
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 30 = 0 ise yaz(Biçimle('   Adım {:3}: kayıp {:.4}',
        ↪ adım, kayıp.Değer), 1)
  bitir
bitir.
```

Programın çıktısı şöyledir:

```
A) Katman normalleştirme öncesi ve sonrası:
   [2.0, 4.0, 6.0, 8.0]  →  [-1.34, -0.45, 0.45, 1.34]   (ortalama 0.00, sapma 1.00)
   [100.0, 101.0, 99.0, 100.0]  →  [0.0, 1.41, -1.41, 0.0]   (ortalama 0.00, sapma 1.00)
   [-3.0, 0.0, 0.0, 3.0]  →  [-1.41, 0.0, 0.0, 1.41]   (ortalama 0.00, sapma 1.00)

B) Küçük GPT (2 katman, 4 kafa, 32 boyut, 16 karakter bağlam):
   Karakter ve konum gömmeleri :   1472
   Dikkat (Q, K, V, çıkış)     :   8192
   İleri besleme ağları        :  16704
   Katman normalleştirmeler    :    320
   Çıkış katmanı               :    990
   Toplam                      :  27678   (model.ParametreAdedi = 27678)

C) İki pencerelik tek bir yığını ezberleme:
   Adım   1: kayıp 3.4682
   Adım  30: kayıp 0.0085
   Adım  60: kayıp 0.0020
   Adım  90: kayıp 0.0014
   Adım 120: kayıp 0.0011
   Adım 150: kayıp 0.0008
```

Çıktıyı okumak

Normalleştirme, çok farklı ölçeklerdeki üç satırı (2–8, 99–101, −3–3) aynı ölçeğe getiriyor. Burada dikkat çeken nokta şudur: ikinci satırdaki 100 civarındaki sayıların büyüklüğü kayboluyor, yalnızca aralarındaki fark kalıyor.

Parametre dökümü ilginç bir gerçeği gösteriyor: modelin yıldızı dikkat olsa da parametrelerin yarıdan fazlası (16 704 / 27 678) ileri besleme ağlarındadır. Büyük modellerde de oran benzerdir.

Ezberleme sınamasında kayıp 3,47'den 30 adımda 0,01'in altına iniyor. Model, iki pencere için her konumda doğru harfi neredeyse kesinlikle tahmin ediyor. Kod doğru çalışıyor; artık gerçek eğitime geçebiliriz.

Alıştırma. `GPT.İleri` içinde iki artık bağlantıyı kaldırınız (`x := x + Bırak(dikkat)` yerine `x := Bırak(dikkat)` yazınız, ileri beslemede de aynısını yapınız) ve ezberleme sınamasını yeniden çalıştırınız. Kayıp ne kadar hızlı düşüyor?

BÖLÜM

8

Küçük GPT'yi eğitmek

Artık tüm parçalar hazırdır. Bu bölümde küçük GPT'yi atasözleriyle eğitiyoruz. Program 5. bölümdeki eğitim döngüsüne dört yenilik ekliyor.

Akış ve pencereler. Eğitim atasözleri, aralarına nokta konularak tek uzun bir metne (akış) dönüştürülür. Her adımda bu metinden rastgele 8 pencere alınır; her pencere 16 karakterdir. Hedef, pencerenin bir karakter kaydırılmışıdır. Model bir pencerede 16 konumun hepsi için aynı anda tahmin yapar; maske sayesinde her konum yalnızca kendinden öncekileri görür. Böylece tek bir yığından 128 eğitim örneği çıkar.

Bırakma. 5. bölümde MLP'nin veriyi ezberlediğini görmüştük. Burada `model.bırakma := 0.2` ile eğitim sırasında gömmelerin, dikkat ve ileri besleme çıktılarının %20'sini her adımda rastgele sıfırlıyoruz. Model tek bir yola güvenemediği için daha genel kurallar öğrenir. Değerlendirmede ve metin üretirken bırakma kapalıdır (`model.eğitimde := yanlış`).

Öğrenme oranı takvimi. Eğitimin başında büyük adımlar hızlı ilerletir; sona doğru küçük adımlar ince ayar yapar. Öğrenme oranını bir kosinüs eğrisiyle 0,003'ten sıfıra indiriyoruz. Büyük modellerin çoğu benzer bir takvimle eğitilir.

Erken durdurma. Her 250 adımda eğitim ve doğrulama kaybı ölçülür. Doğrulama kaybı o ana kadarki en iyi değerse ağırlıklar `kucuk_gpt.agirlik` dosyasına kaydedilir. Eğitim bitince en iyi ağırlıklar geri yüklenir. Model sonradan ezberlemeye başlasa bile elimizde en iyi hali kalır.

```
program KüçükGPTEğitimi
{ Kitabın asıl modeli: 2 katmanlı, 4 kafalı, 32 boyutlu, 16 karakterlik bağlama
  bakan küçük bir GPT. Atasözlerinin %90'ıyla eğitilir, %10'uyla sınanır.
  Eğitim verisi, atasözlerinin araya '.' konularak art arda dizildiği uzun bir
  metindir. Her adımda bu metinden rastgele 8 pencere seçeriz; her pencerede model,
  her konumda bir sonraki karakteri tahmin etmeye çalışır.
  Eğitimden sonra ağırlıklar 'kucuk_gpt.agirlik' dosyasına kaydedilir;
  9. bölümün programı bu dosyayı kullanır. }
kullan Atasozleri, DilAraclari, KucukGPT, YZ, Sistem, Matematik

sabit
  BAĞLAM: tamsayı = 16
  YIĞINBOYU: tamsayı = 8
  ADIMSAYISI: tamsayı = 3000
  ENYÜKSEKORAN: ondalık = 0.003

değişken
  veri, harfler: metin dizisi
  numara: sözlük
  eğitim, doğrulama: tamsayı dizisi
  model: GPT
  opt: herhangi
  kayıp: tensör
  adım, i, k, ayrım: tamsayı
  süre: ondalık
  sözcük: metin
  bağ: tamsayı dizisi
  P: tensör
  yığın: liste
  enİyi, eğitimKaybı, doğrulamaKaybı: ondalık
  işaret: metin

fonksiyon Pencereler(akış: tamsayı dizisi; adet: tamsayı): liste
{ Akıştan rastgele 'adet' pencere: [girdiler, hedefler]. Hedef, girdinin bir
    ↪ kaydırılmışı. }
değişken s, b: tamsayı; g, h: tamsayı dizisi
başla
  g := []; h := []
  1'den adet'e kadar her b için
  başla
    s := Rastgele(Uzunluk(akış) - BAĞLAM - 1)
    g := g + Dilim(akış, s, s + BAĞLAM)
    h := h + Dilim(akış, s + 1, s + BAĞLAM + 1)
  bitir
```

```
  dön [g, h]
bitir

fonksiyon OrtalamaKayıp(akış: tamsayı dizisi): ondalık
{ Akışın ilk 512 karakterini üst üste binmeyen pencerelerle tarar ve ortalama kaybı
  verir. (Tüm akışı taramak daha doğru ama yavaştır; tahmin için bu kadarı yeter.) }
değişken s, adet: tamsayı; g, h: tamsayı dizisi; top: ondalık
başla
  top := 0; adet := 0; s := 0
  s + BAĞLAM + 1 < = Min(Uzunluk(akış), 512) iken
  başla
    g := Dilim(akış, s, s + BAĞLAM); h := Dilim(akış, s + 1, s + BAĞLAM + 1)
    top := top + ÇaprazEntropi(model.İleri(g, 1, BAĞLAM), h).Değer
    adet := adet + 1; s := s + BAĞLAM
  bitir
  dön top / adet
bitir

fonksiyon Akış(parçalar: metin dizisi; numara: sözlük): tamsayı dizisi
başla
  dön DilAraclari.Kodla('.' + Birleştir(parçalar, '.') + '.', numara)
bitir

başla
  Tohum(2026)
  veri := Atasozleri.Hepsi
  harfler := DilAraclari.Harfler
  numara := DilAraclari.Numaralar(harfler)
  ayrım := Uzunluk(veri) * 9 böl 10
  eğitim := Akış(Dilim(veri, 0, ayrım), numara)
  doğrulama := Akış(Dilim(veri, ayrım, Uzunluk(veri)), numara)
  yaz(Biçimle('Eğitim akışı {} karakter, doğrulama akışı {} karakter',
      ↪ Uzunluk(eğitim), Uzunluk(doğrulama)), 1)

  model := GPT.Oluştur(Uzunluk(harfler), BAĞLAM, 32, 4, 2)
  model.bırakma := 0.2
  yaz(Biçimle('Model: 2 katman, 4 kafa, 32 boyut, {} parametre',
      ↪ model.ParametreAdedi), 1)
  opt := Adam(model.Parametreler, ENYÜKSEKORAN)
  süre := Sistem.Zaman

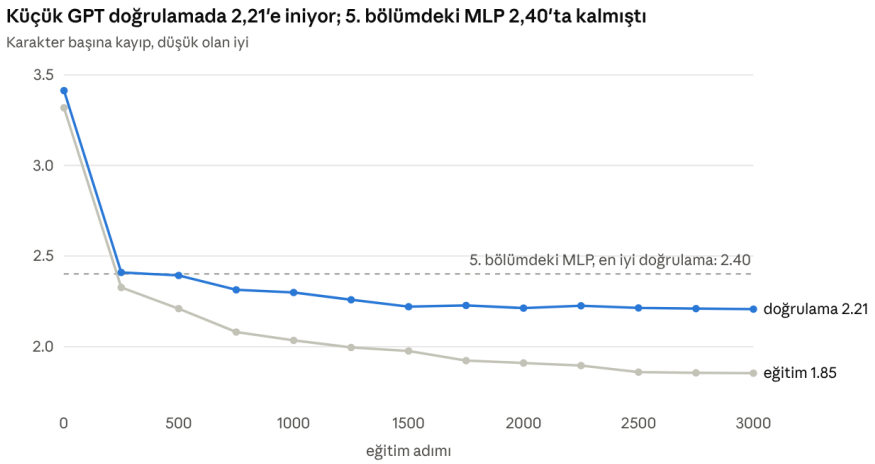
  enİyi := 1000.0
  1'den ADIMSAYISI'na kadar her adım için
  başla
    yığın := Pencereler(eğitim, YIĞINBOYU)
    { öğrenme oranı takvimi: kosinüs eğrisiyle en yüksek orandan sıfıra doğru azalır
        ↪ }
    opt.oran := ENYÜKSEKORAN * 0.5 * (1 + Cos(Matematik.pi * adım / ADIMSAYISI))
    model.eğitimde := doğru
```

```
    opt.SıfırlaGrad
    kayıp := ÇaprazEntropi(model.İleri(yığın[0], YIĞINBOYU, BAĞLAM), yığın[1])
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 250 = 0 ise
    başla
      { değerlendirirken bırakma kapalı olmalı }
      model.eğitimde := yanlış
      eğitimKaybı := OrtalamaKayıp(eğitim)
      doğrulamaKaybı := OrtalamaKayıp(doğrulama)
      işaret := ''
      eğer doğrulamaKaybı < enİyi ise
      başla
        { erken durdurma: en iyi doğrulama kaybındaki ağırlıkları sakla }
        enİyi := doğrulamaKaybı; işaret := '  ← en iyi, kaydedildi'
        model.Kaydet('kucuk_gpt.agirlik')
      bitir
      yaz(Biçimle('Adım {:4}: eğitim kaybı {:.3}   doğrulama kaybı {:.3}{}', adım,
        eğitimKaybı, doğrulamaKaybı, işaret), 1)
    bitir
  bitir
  yaz(Biçimle('Eğitim süresi: {:.0} saniye', Sistem.Zaman - süre), 1)
  model.Yükle('kucuk_gpt.agirlik')
  model.eğitimde := yanlış

  yaz('', 1); yaz('Küçük GPT modelinin ürettiği "atasözleri" (yeni olanlar * ile
      ↪ işaretli):', 1)
  1'den 12'ye kadar her k için
  başla
    sözcük := ''; bağ := [0]
    tekrarla
      P := Softmaks(model.İleri(bağ, 1, Uzunluk(bağ)))
      i := DilAraclari.Örnekle(P.Satır(Uzunluk(bağ) - 1).Veri)
      eğer i <> 0 ise sözcük := sözcük + harfler[i]
      bağ := bağ + [i]
      eğer Uzunluk(bağ) > BAĞLAM ise bağ := Dilim(bağ, 1, Uzunluk(bağ))
    i = 0 veya Uzunluk(sözcük) > 80 olana dek
    eğer veri.İçerir(sözcük) ise yaz('    ' + sözcük, 1) değilse yaz('  * ' + sözcük,
        ↪ 1)
  bitir
bitir.
```

Programın çıktısı şöyledir (eğitim süresi bilgisayarınıza göre değişir):

```
Eğitim akışı 4725 karakter, doğrulama akışı 559 karakter
Model: 2 katman, 4 kafa, 32 boyut, 27678 parametre
Adım    1: eğitim kaybı 3.318   doğrulama kaybı 3.413  ← en iyi, kaydedildi
Adım  250: eğitim kaybı 2.327   doğrulama kaybı 2.410  ← en iyi, kaydedildi
Adım  500: eğitim kaybı 2.210   doğrulama kaybı 2.393  ← en iyi, kaydedildi
```



Şekil 8.1. *Küçük GPT'nin eğitimi: her 250 adımda ölçülen eğitim ve doğrulama kaybı*

```
Adım  750: eğitim kaybı 2.081   doğrulama kaybı 2.314  ← en iyi, kaydedildi
Adım 1000: eğitim kaybı 2.035   doğrulama kaybı 2.299  ← en iyi, kaydedildi
Adım 1250: eğitim kaybı 1.996   doğrulama kaybı 2.259  ← en iyi, kaydedildi
Adım 1500: eğitim kaybı 1.976   doğrulama kaybı 2.221  ← en iyi, kaydedildi
Adım 1750: eğitim kaybı 1.923   doğrulama kaybı 2.228
Adım 2000: eğitim kaybı 1.910   doğrulama kaybı 2.213  ← en iyi, kaydedildi
Adım 2250: eğitim kaybı 1.896   doğrulama kaybı 2.226
Adım 2500: eğitim kaybı 1.860   doğrulama kaybı 2.214
Adım 2750: eğitim kaybı 1.856   doğrulama kaybı 2.210  ← en iyi, kaydedildi
Adım 3000: eğitim kaybı 1.854   doğrulama kaybı 2.207  ← en iyi, kaydedildi
Eğitim süresi: 576 saniye

Küçük GPT modelinin ürettiği "atasözleri" (yeni olanlar * ile işaretli):
  * kükünsen günül seyi yacar
  * dağıp daşmun dirindene billen tıktır
  * ayıcız her oğrolar sönürünü zeçen beli dör kerdimi domnuna yesin yünü eve karder
  * ağen yalak vırık
  * gelilecece ayığı öy verit kişerin anlt biş gele olstmaz
  * haraktttılda hirlin teden herdeğri firini göle konuşmazsun çoran balanır yır biğil
  * işi söden idemin bil binden sötüğrüne tavardar
  * surt yeyokt a çtalut
  * götürdür
  * ağla ayırı kersen mur
  * yoğşta yolur kidi güş kakmana sende yosut dit keçtind kadır
  * sere terendi yok çundi bülür ker içinde hapatamapa
```

Çıktıyı okumak

Doğrulama kaybı eğitim boyunca düşmeye devam ediyor ve 2,207'de bitiyor. Bu, 5. bölümdeki MLP'nin en iyi değerinden (2,40) belirgin biçimde iyi. Eğitim ve doğrulama kayıpları arasındaki fark da daha küçük: 1,85'e karşı 2,21. MLP'de bu fark 1,26'ya karşı 2,88'e kadar açılmıştı. Bırakma ve daha uzun bağlam birlikte işe yarıyor.

Üretilen metne bakalım. On iki örneğin hepsi yeni (yıldızlı): model hiçbir atasözünü birebir kopyalamıyor. "göle", "götürdür", "içinde", "konuşmaz" gibi gerçek sözcükler ve "-ar", "-maz", "-dır" gibi Türkçe ekler doğru yerlerde görünüyor. Ama cümleler henüz anlamlı değildir.

Bu, dürüst bir sonuçtur ve dil modelleri hakkında önemli bir şey öğretir. Modelimizin gördüğü tüm metin 4725 karakterdir, yani bu sayfanın birkaç paragrafı kadardır. Bir çocuk bile konuşmayı öğrenirken milyonlarca sözcük duyar. Mimari doğru, eğitim düzgün işliyor; eksik olan veri ve ölçektir. 10. bölümde bu farkı sayılarla göreceğiz. 9. bölümde ise aynı modelden metni nasıl seçtiğimizin sonucu ne kadar değiştirdiğini inceleyeceğiz.

Değerlendirme hakkında bir not: program kaybı hız için akışların yalnızca ilk 512 karakteri üzerinde ölçüyor. Bu, eğilimi görmek için yeterlidir ama kesin bir ölçüm değildir.

Alıştırma. `model.bırakma := 0.2` satırını silip programı yeniden çalıştırınız. Doğrulama kaybı hangi adımda en düşük değerine ulaşıyor ve ondan sonra ne oluyor? Ardından `Atasozleri.y` dosyasına kendi bildiğiniz atasözlerini ekleyiniz; veri büyüdükçe doğrulama kaybı nasıl değişiyor?

BÖLÜM

9

Metin üretmek

Eğitilmiş bir dil modeli metin yazmaz; her adımda bir olasılık dağılımı verir. Bu dağılımdan bir karakteri nasıl seçtiğimiz ise çıkan metnin karakterini belirler. Aynı model, seçim yöntemine göre sıkıcı ve tekrarcı da olabilir, yaratıcı ve dağınık da. Sohbet uygulamalarındaki "sıcaklık" ayarı da tam olarak bu bölümün konusudur.

Üç yöntem

Açgözlü seçim. Her adımda en olası karakteri alırız. Rastgelelik yoktur; aynı başlangıç her zaman aynı metni verir. Kulağa en mantıklı yöntem gibi gelir ama genellikle kötü sonuç verir. Model kendi ürettiği metne bakarak karar verdiği için kolayca bir döngüye girer.

Sıcaklık. Örneklemeden önce logitleri bir T sayısına böleriz. $T < 1$ iken büyük logitler daha da öne çıkar; model daha emin ve tutucu davranır. $T > 1$ iken dağılım düzleşir; düşük olasılıklı karakterler de seçilmeye başlar. $T = 1$ modelin öğrendiği dağılımın kendisidir. T sıfıra yaklaştıkça açgözlü seçime yaklaşır.

$$P_T(i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

Top-k. Yalnızca en olası k karakter arasından seçeriz; geri kalanların olasılığını sıfırlarız. Model bazen düşük ama sıfır olmayan bir olasılıkla saçma bir karakter seçer ve metin oradan sonra raydan çıkar. Top-k bu kuyruğu keser. Büyük modellerde benzer bir yöntem olan top-p (çekirdek örnekleme) da yaygındır: k sabit sayı yerine, olasılıkları toplamı p'yi geçene kadar en olasıdan başlayarak alınır.

Program, 8. bölümde kaydedilen ağırlıkları yükler. Dosya yoksa modeli 8. bölümle birebir aynı biçimde yeniden eğitir. Aynı tohumla başlayıp rastgele sayıları aynı sırayla çektiği için aynı ağırlıklara ulaşır. Bu, oyun alanında programlar arasında dosya taşınamadığı için gereklidir.

```
program MetinÜretmek
{ Eğitilmiş bir dil modeli her adımda bir OLASILIK DAĞILIMI verir. Bir sonraki
  karakteri bu dağılımdan nasıl seçtiğimiz, üretilen metnin karakterini belirler:
    açgözlü   : her zaman en olası karakter (hep aynı metin, çoğu zaman tekrar)
    sıcaklık  : logitler T'ye bölünür; T < 1 dağılımı sivriltir, T > 1 düzleştirir
    top-k     : yalnızca en olası k karakter arasından seçilir
  Program 8. bölümde kaydedilen ağırlıkları yükler. Dosya yoksa (örneğin oyun
  alanında) modeli 8. bölümle aynı ayarlarla yeniden eğitir; bu yaklaşık 10 dakika
      ↪ sürer. }
kullan Atasozleri, DilAraclari, KucukGPT, YZ, Sistem, Matematik

sabit
  BAĞLAM: tamsayı = 16
  DOSYA: metin = 'kucuk_gpt.agirlik'

değişken
  harfler: metin dizisi
  numara: sözlük
  model: GPT
  P: ondalık dizisi
  sıra: tamsayı dizisi
  baş: metin
  i, k: tamsayı
  T: ondalık

fonksiyon SonrakiDağılım(bağ: tamsayı dizisi; sıcaklık: ondalık; enİyiK: tamsayı):
    ↪ ondalık dizisi
{ Son karakterden sonraki karakterin olasılıkları; sıcaklık ve top-k uygulanmış. }
değişken logit: ondalık dizisi; eşik: ondalık
başla
  logit := model.İleri(bağ, 1, Uzunluk(bağ)).Satır(Uzunluk(bağ) - 1).Veri
  logit := Eşle(logit, fonksiyon(z: ondalık) => z / sıcaklık)
  eğer enİyiK > 0 ise
  başla
```

```
    eşik := Sırala(logit, fonksiyon(z: ondalık) => -z)[enİyiK - 1]
    logit := Eşle(logit, fonksiyon(z: ondalık) => eğer z >= eşik ise z değilse
        ↪ -1000000000.0)
  bitir
  dön Softmaks(Tensör([logit])).Satır(0).Veri
bitir

fonksiyon Üret(başlangıç: metin; sıcaklık: ondalık; enİyiK: tamsayı; açgözlü:
    ↪ mantıksal): metin
değişken bağ: tamsayı dizisi; j: tamsayı; dağ: ondalık dizisi
başla
  sonuç := başlangıç
  bağ := [0] + DilAraclari.Kodla(başlangıç, numara)
  tekrarla
    eğer Uzunluk(bağ) > BAĞLAM ise bağ := Dilim(bağ, Uzunluk(bağ) - BAĞLAM,
        ↪ Uzunluk(bağ))
    dağ := SonrakiDağılım(bağ, sıcaklık, enİyiK)
    eğer açgözlü ise j := ArgMaks(dağ) değilse j := DilAraclari.Örnekle(dağ)
    eğer j <> 0 ise sonuç := sonuç + harfler[j]
    bağ := bağ + [j]
  j = 0 veya Uzunluk(sonuç) > 70 olana dek
bitir

prosedür YenidenEğit
{ 8. bölümle birebir aynı eğitim: aynı tohum, veri, bırakma, öğrenme oranı takvimi ve
  erken durdurma. Rastgele sayılar aynı sırayla çekildiği için aynı ağırlıklar çıkar.
    ↪ }
değişken veri: metin dizisi; eğitim, doğrulama, g, h: tamsayı dizisi
         opt: herhangi; adım, b, s, ayrım: tamsayı; kayıp: tensör; enİyi, dk: ondalık
başla
  yaz('Ağırlık dosyası bulunamadı; model 8. bölümdeki gibi yeniden eğitiliyor
      ↪ (yaklaşık 10 dakika)...', 1)
  Tohum(2026)
  veri := Atasozleri.Hepsi
  ayrım := Uzunluk(veri) * 9 böl 10
  eğitim := DilAraclari.Kodla('.' + Birleştir(Dilim(veri, 0, ayrım), '.') + '.',
      ↪ numara)
  doğrulama := DilAraclari.Kodla('.' + Birleştir(Dilim(veri, ayrım, Uzunluk(veri)),
      ↪ '.') + '.', numara)
  model := GPT.Oluştur(Uzunluk(harfler), BAĞLAM, 32, 4, 2)
  model.bırakma := 0.2
  opt := Adam(model.Parametreler, 0.003)
  enİyi := 1000.0
  1'den 3000'e kadar her adım için
  başla
    g := []; h := []
    1'den 8'e kadar her b için
    başla
      s := Rastgele(Uzunluk(eğitim) - BAĞLAM - 1)
```

```
      g := g + Dilim(eğitim, s, s + BAĞLAM); h := h + Dilim(eğitim, s + 1, s + BAĞLAM
          ↪ + 1)
    bitir
    opt.oran := 0.003 * 0.5 * (1 + Cos(Matematik.pi * adım / 3000))
    model.eğitimde := doğru
    opt.SıfırlaGrad
    kayıp := ÇaprazEntropi(model.İleri(g, 8, BAĞLAM), h)
    kayıp.GeriYay
    opt.Adım
    eğer adım = 1 veya adım mod 250 = 0 ise
    başla
      model.eğitimde := yanlış
      dk := 0; s := 0; b := 0
      s + BAĞLAM + 1 < = Min(Uzunluk(doğrulama), 512) iken
      başla
        dk := dk + ÇaprazEntropi(model.İleri(Dilim(doğrulama, s, s + BAĞLAM), 1,
            ↪ BAĞLAM),
                                 Dilim(doğrulama, s + 1, s + BAĞLAM + 1)).Değer
        b := b + 1; s := s + BAĞLAM
      bitir
      eğer dk / b < enİyi ise başla enİyi := dk / b; model.Kaydet(DOSYA) bitir
    bitir
  bitir
  model.Yükle(DOSYA)
  model.eğitimde := yanlış
bitir

başla
  harfler := DilAraclari.Harfler
  numara := DilAraclari.Numaralar(harfler)
  eğer Sistem.DosyaVarMı(DOSYA) ise
  başla
    model := GPT.Oluştur(Uzunluk(harfler), BAĞLAM, 32, 4, 2)
    model.Yükle(DOSYA)
  bitir
  değilse YenidenEğit

  yaz('"damlaya damlaya g" yazıldıktan sonra modelin tahmini:', 1)
  P := SonrakiDağılım([0] + DilAraclari.Kodla('damlaya damlaya g', numara), 1.0, 0)
  sıra := Sırala(Aralık(Uzunluk(P)), fonksiyon(z: tamsayı) => -P[z])
  0'dan 4'e kadar her k için yaz(Biçimle('   "{}"  {:.3}', harfler[sıra[k]],
      ↪ P[sıra[k]]), 1)

  yaz('', 1); yaz('Açgözlü seçim (her çalıştırmada aynı sonuç):', 1)
  ['', 'damlaya', 'sabır', 'dost', 'göz'] içindeki her baş için
    yaz('   ' + Üret(baş, 1.0, 0, doğru), 1)

  Tohum(9)
  [0.5, 1.0, 1.5] içindeki her T için
```

```
  başla
    yaz('', 1); yaz(Biçimle('Sıcaklık {}:', T), 1)
    1'den 4'e kadar her i için yaz('   ' + Üret('', T, 0, yanlış), 1)
  bitir

  yaz('', 1); yaz('Top-k (k = 3, sıcaklık 1.0):', 1)
  1'den 4'e kadar her i için yaz('   ' + Üret('', 1.0, 3, yanlış), 1)
bitir.
```

Programın çıktısı şöyledir:

```
"damlaya damlaya g" yazıldıktan sonra modelin tahmini:
   "a"  0.220
   "e"  0.199
   "ı"  0.188
   "ü"  0.130
   "ö"  0.114

Açgözlü seçim (her çalıştırmada aynı sonuç):
   alan yaran yerin derin derin derin derin derin derin derin derin derin
   damlayan yerin derin derin derin derin derin derin derin derin derin de
   sabır der derin derin derin derin derin derin derin derin derin derin d
   dostun derin derin derin derin derin derin derin derin derin derin deri
   gözün alan alarar

Sıcaklık 0.5:
   dolun ye derin dişin olur güşmaz
   akarıyı yolur
   aşan der yayır
   bili biş yelinde alarır

Sıcaklık 1.0:
   kakır yülm z biş kasanı kaparı sok övük düzüş tayı bırdığa pedinin meyi
   ağacıyık dilver
   esol
   gelemeyn dolanl apırsın sörmeden binekince dermik

Sıcaklık 1.5:
   aşıgcı dok
   acilmama
   yelanca butaya koktuçt oçuknununz kebıden düzüderlcs dererin aktloşan
   ısarlın götürüsaz ork ult

Top-k (k = 3, sıcaklık 1.0):
   alın deren dolunun düşmez daranı yeri deliş
   alan kere yokmaz
   atla kaş dürün karan dalır
   alılan deri delir
```

Çıktıyı okumak

İlk tablo modelin ne bildiğini gösteriyor. "damlaya damlaya g" yazıldıktan sonra model, g'den sonra bir ünlü geleceğinden emindir: ilk beş tahminin hepsi ünlü. Ama doğru cevap olan "ö"yü ancak beşinci sıraya koyuyor. Bu atasözü eğitim verisinde olduğu halde model onu ezberlememiş; bırakma ve erken durdurmanın istediğimiz etkisi budur. Model, gördüğü metinlerden hecelerin ve eklerin genel kurallarını çıkarmış.

Açgözlü seçim bir döngüye takılıyor. Beş başlangıcın dördü kısa sürede "derin derin derin…" tekrarına düşüyor. Model "derin" yazdıktan sonra en olası devamın yine "derin" olduğunu düşünüyor; açgözlü seçim de bu döngüden hiç çıkamıyor. Büyük dil modellerinde de aynı sorun görülür; bu yüzden sohbet uygulamaları nadiren açgözlü seçim kullanır.

Sıcaklık bir ayar düğmesidir. Sıcaklık 0,5'te üretilen metin kısa, düzenli ve Türkçeye en yakın olanı: "olur", "akarıyı", "bili" gibi sözcükler ve "-maz", "-ır" ekleri. Sıcaklık 1,5'te sözcükler dağılıyor; "düzüderlcs" gibi Türkçede mümkün olmayan harf dizileri belirliyor.

Top-k sınırlı bir özgürlük tanıyor. Top-k ile sıcaklık 1,0'da metin çeşitli kalıyor ama saçma harf dizileri kayboluyor. "alın deren dolunun düşmez" gerçek bir atasözü olmasa da kulağa bir atasözü gibi geliyor.

Uygulamada bu ayarlar birlikte kullanılır. Bir kod yazma asistanı düşük sıcaklık ister, çünkü doğru ve tutarlı olmak yaratıcı olmaktan önemlidir. Şiir yazdıran biri ise sıcaklığı yükseltebilir.

Alıştırma. Programa top-p örneklemeyi ekleyiniz: `SonrakiDağılım`'a bir `p` parametresi veriniz; olasılıkları büyükten küçüğe sıralayıp toplamı p'yi geçene kadar olanları tutunuz, kalanları sıfırlayıp yeniden normalleştiriniz. $p = 0{,}9$ ile üretilen metni top-k ile karşılaştırınız.

BÖLÜM

10

Küçükten büyüğe

Küçük GPT'miz ile günümüzün sohbet modelleri aynı aileden gelir. İkisi de simgeleri sayılara çevirir, gömmelere dönüştürür, üst üste yığılmış Transformer bloklarından geçirir ve bir sonraki simgenin olasılığını hesaplar. Bu son bölümde aradaki farkı üç başlıkta inceliyoruz: **ölçek**, **veri** ve **eğitimin aşamaları**.

Parametreleri saymak

Bir Transformer'ın parametre sayısı birkaç sayıdan hesaplanır: sözlük boyu V, bağlam uzunluğu T, model boyutu D ve katman sayısı L. Parametrelerin büyük kısmı katmanlardaki matrislerdedir ve D'nin karesiyle büyür: her katmanda dikkat için yaklaşık $4 \cdot D^2$, ileri besleme için $8 \cdot D^2$ parametre vardır.

$$N \approx L\,(12D^2 + 13D) + VD + TD + 2D$$

Eğitimin ne kadar hesap gerektirdiği için de yaygın bir kestirim vardır: işlenen her simge başına yaklaşık 6·N kayan noktalı işlem. Bunun 2·N'si ileri hesaptan,

4·N'si geri yayılımdan gelir. Program bu iki formülü GPT-2 ailesine ve kendi modelimize uygular.

```
program Ölçekleme
{ Küçük GPT'mizle büyük modeller arasındaki fark mimaride değil, sayılardadır.
  Bu program GPT-2 ailesinin parametre sayılarını aynı formülle hesaplar ve
  eğitim için gereken hesap miktarını kabaca tahmin eder.

  Bir GPT-2 katmanının parametreleri (D: boyut):
     dikkat         : 4·D·D + 4·D        (Q, K, V ve çıkış matrisleri, sapmalarıyla)
     ileri besleme  : 8·D·D + 5·D        (D → 4D → D, sapmalarıyla)
     normalleştirme : 4·D                (iki katman normalleştirme)
  Bunlara karakter/simge gömmesi (V·D), konum gömmesi (T·D) ve son normalleştirme
      ↪ (2·D)
  eklenir. GPT-2, çıkış katmanında gömme matrisini yeniden kullanır; bu yüzden ayrı
  bir çıkış matrisi yoktur.
  Eğitim hesabı için yaygın bir kestirim: işlenen her simge başına 6·N kayan nokta
      ↪ işlemi
  (N: parametre sayısı; 2·N ileri, 4·N geri yayılım). }

kullan Matematik

değişken
  adlar: metin dizisi
  boyutlar, katmanlar: tamsayı dizisi
  i: tamsayı
  N, simge: ondalık

fonksiyon ParametreSayısı(V, T, D, L: tamsayı): ondalık
başla
  dön L * (12.0 * D * D + 13.0 * D) + V * D + T * D + 2.0 * D
bitir

fonksiyon Bilimsel(x: ondalık): metin
{ 63770112000 → '6.4 × 10^10' }
değişken üs: tamsayı
başla
  üs := Kes(Ln(x) / Ln(10))
  dön Biçimle('{:.1} × 10^{}', x / (10.0 ** üs), üs)
bitir

fonksiyon Sola(s: metin; n: tamsayı): metin
başla
  sonuç := s
  Uzunluk(sonuç) < n iken sonuç := sonuç + ' '
bitir

başla
  adlar := ['GPT-2 küçük', 'GPT-2 orta', 'GPT-2 büyük', 'GPT-2 XL']
```

```
  boyutlar := [768, 1024, 1280, 1600]
  katmanlar := [12, 24, 36, 48]
  yaz('Model           boyut  katman   parametre', 1)
  0'dan 3'e kadar her i için
    yaz(Sola(adlar[i], 14) + Biçimle('  {:5}  {:6}  {:10.1} milyon', boyutlar[i],
        ↪  katmanlar[i],
      ParametreSayısı(50257, 1024, boyutlar[i], katmanlar[i]) / 1000000), 1)

  yaz('', 1)
  N := 27678
  simge := 3000.0 * 8 * 16
  yaz(Biçimle('Küçük GPT: {:.0} parametre, eğitimde {:.0} karakter işlendi', N,
      ↪  simge), 1)
  yaz('  eğitim hesabı ≈ 6 · N · simge = ' + Bilimsel(6 * N * simge) + ' işlem', 1)
  N := ParametreSayısı(50257, 1024, 768, 12)
  simge := 20 * N
  yaz('GPT-2 küçük, parametre başına 20 simgeyle eğitilseydi:', 1)
  yaz('  ' + Bilimsel(simge) + ' simge, ' + Bilimsel(6 * N * simge) + ' işlem', 1)
  yaz(Biçimle('  Küçük GPT modelinden {:.0} kat fazla hesap', (6 * N * simge) / (6 *
      ↪  27678 * 3000.0 * 8 * 16)), 1)
bitir.
```

Programın çıktısı şöyledir:

```
Model           boyut  katman   parametre
GPT-2 küçük       768      12       124.4 milyon
GPT-2 orta       1024      24       354.8 milyon
GPT-2 büyük      1280      36       774.0 milyon
GPT-2 XL         1600      48      1557.6 milyon

Küçük GPT: 27678 parametre, eğitimde 384000 karakter işlendi
  eğitim hesabı ≈ 6 · N · simge = 6.4 × 10^10 işlem
GPT-2 küçük, parametre başına 20 simgeyle eğitilseydi:
  2.5 × 10^9 simge, 1.9 × 10^18 işlem
  Küçük GPT modelinden 29139543 kat fazla hesap
```

Çıktıyı okumak

Formül gerçeği tutturuyor. OpenAI'ın 2019'da yayımladığı GPT-2 modelleri 124 milyon, 355 milyon, 774 milyon ve 1,5 milyar parametreyle bilinir. Bizim tek satırlık formülümüz bu sayıları neredeyse birebir veriyor. Demek ki "büyük model" dediğimiz şey, Küçük GPT ile aynı tarifin daha büyük D ve L değerleriyle uygulanmasıdır. Küçük GPT'de D = 32 ve L = 2 idi; GPT-2 küçükte ise D = 768 ve L = 12'dir.

Hesap farkı akıl almaz boyuttadır. Küçük GPT'nin eğitimi yaklaşık 6,4 × 10^{10} işlem sürdü; bir dizüstü bilgisayar bunu saniyeler içinde yapar. 2022'de DeepMind'ın "Chinchilla" çalışması, belirli bir hesap bütçesiyle en iyi sonucu

almak için parametre başına kabaca 20 simgeyle eğitmeyi önerdi. Bu kurala uyulsaydı GPT-2 küçük bile bizim modelimizden yaklaşık 29 milyon kat fazla hesap isterdi. Günümüzün öncü modelleri ise yüzlerce milyar parametreye ve trilyonlarca simgeye çıkar. Bu ölçekte eğitim binlerce grafik işlemcisinin haftalarca birlikte çalışmasıyla yapılır.

Ölçekleme yasaları. 2020'de Kaplan ve arkadaşlarının yayımladığı bir çalışma, dil modellerinin kaybının parametre sayısı, veri miktarı ve hesapla çok düzenli biçimde azaldığını gösterdi: kayıp bu üçünün üslü bir fonksiyonu gibi davranıyor. Bu bulgu, "model ve veriyi büyütürsek daha iyi olur" sezgisini ölçülebilir bir tahmine dönüştürdü ve son yılların büyüme yarışının temelini attı. 8. bölümde gördüğümüz "model ezberliyor ama genelleyemiyor" sorunu da bu çerçevede anlaşılır: 27 bin parametreye karşı yalnızca 4,7 bin karakterlik veri, Chinchilla oranının çok gerisindedir.

Ön eğitim, ince ayar ve sohbet modelleri

Bir sohbet asistanı tek bir eğitimle oluşmaz. Genellikle üç aşama vardır.

1. **Ön eğitim.** Model, internetten, kitaplardan ve kod depolarından derlenmiş çok büyük bir metin yığını üzerinde, tam olarak bu kitaptaki görevle eğitilir: bir sonraki simgeyi tahmin etmek. Hesabın büyük çoğunluğu bu aşamada harcanır. Sonuç, dili ve dünyaya dair pek çok bilgiyi öğrenmiş ama yalnızca "metni sürdürmeyi" bilen bir modeldir. Bir soru sorarsanız cevap vermek yerine başka sorular yazmayı sürdürebilir.
2. **Denetimli ince ayar.** Ardından model, insanların hazırladığı çok daha küçük bir "soru → iyi cevap" örnekleri kümesiyle eğitilmeye devam eder. Kayıp aynıdır, mimari aynıdır; değişen yalnızca veridir. Model böylece bir konuşmanın biçimini öğrenir: kullanıcı sorar, asistan cevap verir.
3. **Tercihlerden öğrenme.** Son aşamada model, aynı soruya verdiği farklı cevapların hangisinin daha yararlı, dürüst ve zararsız olduğuna dair geri bildirimle eğitilir. Bu geri bildirim insanlardan ("insan geri bildirimiyle pekiştirmeli öğrenme", RLHF) ya da yazılı ilkelere göre değerlendirme yapan başka modellerden gelebilir. Bu aşama, modelin neyi nasıl söylediğini şekillendirir.

Küçük GPT'ye ikinci aşamayı uygulamak için yeni bir şey öğrenmeniz gerekmez. Aşağıdaki alıştırma bunu yaptırıyor.

Bu kitaptan sonra

- **Simge düzeyine geçebilirsiniz.** Modelimiz harf harf çalıştı. 2. bölümdeki BPE simgeleştiricisini Küçük GPT'ye bağlarsanız her adımda daha fazla metin görür ve aynı bağlam uzunluğuyla daha uzağı hatırlar.

- **Daha fazla veri toplayabilirsiniz.** Bu kitabın en önemli dersi, aynı mimarinin daha fazla veriyle çok daha iyi sonuç verdiğidir. Telif sorunu olmayan Türkçe metinler (eski edebiyat eserleri, kendi yazılarınız) iyi bir başlangıçtır.
- **Büyük kütüphanelere geçebilirsiniz.** Gerçek ölçekte çalışmak için grafik işlemcisinde çalışan PyTorch gibi kütüphaneler gerekir. Bu kitaptaki her kavramın orada bir karşılığı vardır: `Yoğun` katman `nn.Linear`, katman normalleştirme `nn.LayerNorm`, `kayıp.GeriYay` ise `loss.backward()` olur. Andrej Karpathy'nin açık kaynaklı nanoGPT projesi, bu kitaptaki modelin büyük ölçekli bir karşılığını birkaç yüz satırda gösterir.
- **Okumayı sürdürebilirsiniz.** Transformer'ı tanıtan 2017 tarihli "Attention Is All You Need" makalesi, bu kitabı bitirdikten sonra beklenenden çok daha anlaşılır gelecektir.

Alıştırma. Küçük GPT'yi ince ayarlayınız. 8. bölümde kaydedilen ağırlıkları yükleyiniz. `S: öfke ile kalkan? C: zararla oturur.` biçiminde, her biri bir atasözünün başını soru, sonunu cevap yapan on kadar satır hazırlayınız. Bu satırlarla, 8. bölümdekinin onda biri öğrenme oranıyla birkaç yüz adım daha eğitiniz. Sonra modele `S: damlaya damlaya? C:` yazıp devamını ürettiriniz. İnce ayardan önceki ve sonraki çıktıları karşılaştırınız: model cevabı her zaman doğru bilmese de konuşmanın biçimini, yani "C:"den sonra kısa bir cevap yazıp durmayı hızla öğrenecektir.

EK

A

Kavram sözlüğü

Terim	İngilizcesi	Kısa açıklama	Bölüm
Dil modeli	language model	Bir metnin ardından gelecek simgenin olasılık dağılımını hesaplayan model	1
Simge	token	Modelin işlediği en küçük birim: bir harf, bir hece ya da bir sözcük parçası	2
Simgeleştirme	tokenization	Metni simgelere, simgeleri sayılara çevirme	2

Terim	İngilizcesi	Kısa açıklama	Bölüm
BPE	byte pair encoding	En sık yan yana gelen iki simgeyi tekrar tekrar birleştirerek sözlük kurma yöntemi	2
İkili model	bigram model	Yalnızca bir önceki simgeye bakan dil modeli	3, 4
Kayıp	loss	Modelin doğru simgeye verdiği olasılığın negatif logaritmasının ortalaması	1, 3
Şaşkınlık	perplexity	e^x (x: kayıp); modelin her adımda kaç seçenek arasında kararsız kaldığı	3
Softmaks	softmax	Ham puanları toplamı 1 olan olasılıklara çeviren fonksiyon	4
Çapraz entropi	cross-entropy	Sınıflandırma kaybı; softmaksı kendisi uygular	4
Gömme	embedding	Bir simgeyi temsil eden, öğrenilen sayı vektörü	4, 5
Bağlam penceresi	context window	Modelin bir tahmin yaparken bakabildiği önceki simge sayısı	5
Aşırı uyum	overfitting	Modelin eğitim verisini ezberleyip yeni veride kötüleşmesi	5, 8
Öz-dikkat	self-attention	Her simgenin önceki simgelere ne kadar bakacağına kendisinin karar verdiği işlem	6

Terim	İngilizcesi	Kısa açıklama	Bölüm
Sorgu, anahtar, değer	query, key, value	Dikkatte ``ne arıyorum'', ``bende ne var'', ``ne aktarırım'' vektörleri	6
Nedensel maske	causal mask	Bir simgenin kendinden sonraki simgeleri görmesini engelleyen maske	6
Konum gömmesi	positional embedding	Simgenin dizideki sırasını modele bildiren vektör	6
Çok kafalı dikkat	multi-head attention	Farklı ilişkilere bakan birkaç dikkatin paralel çalışması	7
Artık bağlantı	residual connection	Bir alt katmanın çıktısını girdisinin üstüne ekleme	7
Katman normalleştirme	layer normalization	Her vektörü ortalaması 0, varyansı 1 olacak şekilde ölçekleme	7
Bırakma	dropout	Eğitimde değerlerin bir kısmını rastgele sıfırlayarak ezberlemeyi azaltma	7, 8
Erken durdurma	early stopping	Doğrulama kaybı en iyiyken ağırlıkları saklama	8
Sıcaklık	temperature	Örneklemeden önce puanları bölerek dağılımı sivrilten ya da yayan sayı	9
Top-k	top-k sampling	Yalnızca en olası k simge arasından örnekleme	9
Ön eğitim	pretraining	Büyük metin yığınında sonraki simgeyi tahmin ederek eğitim	10

Terim	İngilizcesi	Kısa açıklama	Bölüm
İnce ayar	fine-tuning	Eğitilmiş modeli küçük ve özel bir veriyle eğitmeyi sürdürme	10

EK

B

Sık yapılan hatalar

1. **Softmaks'ı iki kez uygulamak.** `ÇaprazEntropi` ham puanlar (logit) bekler ve softmaksı kendisi hesaplar. Araya bir `Softmaks` daha koymak programı bozmaz ama eğitimi sessizce bozar: kayıp yüksek kalır. Softmaks yalnızca örneklerken, olasılıkları görmek istediğinizde kullanılır.
2. **Maskeyi unutmak.** Nedensel maske olmadan model eğitimde geleceği görür, "sonraki harfi" bakarak kopyalar. Kayıp şaşırtıcı derecede hızlı düşer ama üretilen metin anlamsız olur, çünkü üretirken gelecek yoktur. Kayıp "fazla iyi" görünüyorsa önce maskeyi denetlemekte yarar vardır.
3. **Hedefi bir kaydırmayı unutmak.** Girdi `akış[s .. s+T]` ise hedef `akış[s+1 .. s+T+1]` olmalıdır. Hedefle girdi aynı olursa model yalnızca girdisini kopyalamayı öğrenir.
4. **Doğrulama verisi ayırmamak.** Yalnızca eğitim kaybına bakarsanız ezberlemeyi öğrenmeden ayırt edemezsiniz. 8. bölümdeki gibi verinin bir kısmını hiç eğitmeden saklamanız gerekir.

5. **Öğrenme oranını büyük seçmek.** Kayıp düşeceğine zıplıyor ya da çıktı "NaN" oluyorsa öğrenme oranını üçte bire indirebilirsiniz. Transformer'larda 0,001 çevresi güvenli bir başlangıçtır.
6. **Bırakmayı üretimde açık bırakmak.** Bırakma yalnızca eğitimde uygulanır. Metin üretmeden ya da doğrulama kaybı ölçmeden önce `model.eğitimde := yanlış` yapmanız gerekir; yoksa her çalıştırmada farklı ve daha kötü sonuç alırsınız.
7. **Bağlamdan uzun dizi vermek.** Konum gömmesi yalnızca `bağlam` kadar konum bilir. Üretirken metin uzadıkça modele yalnızca son `bağlam` karakteri vermeniz gerekir.
8. **Yöntem'e özgü: ad çakışmaları.** Yöntem büyük/küçük harf ayırmaz; `x` ile `X`, `w` ile `W` aynı addır. Matristen `W`, tek bir değerden `w` diye söz etmek isterseniz "zaten tanımlı" hatası alırsınız. `sonuç` da fonksiyonlarda dönüş değerinin adı olduğu için değişken adı olarak kullanılmamalıdır.

EK

C

Alıştırma ipuçları

- **Bölüm 1.** Boşluklar dahil karakterleri sayabilirsiniz. Her karakter, kendinden önceki metne bakılarak tahmin edilecek bir hedeftir; cümlenin bittiğini bildiren son simge de bir hedef sayılır.
- **Bölüm 2.** Birleştirmeler hep en sık yan yana gelen çifti seçtiği için önce Türkçede çok sık geçen ekler ve kısa sözcükler tek simgeye dönüşür. Döngü ilerledikçe sözlük büyür, metnin simge sayısı kısalır.
- **Bölüm 3.** Sayım sözlüğünün anahtarı iki harften oluşur (`a + b`); tabloda en fazla sözlük boyunun karesi kadar satır olabilir. Üçlülerin çoğu metinde hiç geçmeyeceği için düzleştirme burada daha da önemlidir.
- **Bölüm 4.** Sonucu 3. bölümdeki listeyle yan yana yazabilirsiniz. Eğitim yeterince uzun sürdüyse sıralamanın ve olasılıkların büyük ölçüde örtüşmesi gerekir; çünkü iki model aynı şeyi, bir önceki harfe göre sonraki harfin dağılımını öğrenir.
- **Bölüm 5.** Küçük bir gizli katman daha az şey ezberleyebilir. Doğrulama kaybının eğrisini iki çalıştırmada yan yana koyup hangi adımda yükselmeye başladığını karşılaştırabilirsiniz.

- **Bölüm 6.** Ölçek büyüdükçe puanların arasındaki farklar da büyür. Softmaks büyük farklarda en büyük puana neredeyse tüm ağırlığı verir. $\sqrt{d}$'ye bölmek puanları bu yüzden küçültür ve dikkatin birden fazla karaktere yayılabilmesini sağlar.
- **Bölüm 7.** Artık bağlantılar olmadan gradyan katmanlardan geçerken zayıflar. Kaybın ilk yüz adımdaki düşüş hızını iki durumda karşılaştırabilirsiniz.
- **Bölüm 8.** Bırakma olmadan eğitim kaybı daha düşük, doğrulama kaybı ise daha erken yükselmeye başlar. Veri büyüdükçe eğitim ve doğrulama kayıpları birbirine yaklaşır.
- **Bölüm 9.** Top-p'de aday sayısı sabit değildir: model eminse bir iki aday kalır, kararsızsa çok aday kalır. Olasılıkları sıralayıp birikimli toplamı p'yi aşana kadar olanları tutmanız yeterlidir.
- **Bölüm 10.** İnce ayar verisindeki her satırı ayrı bir dizi olarak verip diziyi `.` ile bitirebilirsiniz; model böylece cevabın sonunda durmayı da öğrenir. Öğrenme oranı çok büyük olursa model ön eğitimde öğrendiklerini unutur.