Julia ve Knet ile Derin Öğrenmeye Giriş

Deniz Yuret

8 Eylül 2016

ODTÜ Yapay Öğrenme ve Bilgi İşlemede Yeni Teknikler Yaz Okulu

Özet

- Julia
- Yapay öğrenme
- Knet8 ve örnek modeller
- Diğer yaklaşımlar

5 dakikada Julia

Neden Julia?

C diline kıyasla test süreleri (küçük daha iyi, C = 1.0).
http://julialang.org/benchmarks/

Neden Julia?

- Hız
- Dizi operasyonlarını kolay ifade edebilme: A*B
- Multi-core CPU ve GPU desteği
- Dil özellikleri: full multiple dispatch, güçlü type sistemi, LISP’i andıran meta-programlama.

Doğrusal cebir

```
julia> a
3x3 Array{Int64,2}:
 10  40  70
 20  50  80
 30  60  90

julia> b
3-elt Array{Int64,1}:
 1
 2
 3
```

```
julia> a * b
3-elt Array{Int64,1}:
 300
 360
 420

julia> b' * a
1x3 Array{Int64,2}:
 140  320  500
```

Tekli dizi işlemleri

```
julia> c
3x3 Array{Int64,2}:
 1  4  7
 2  5  8
 3  6  9
```

```
julia> log(c)
3x3 Array{Float64,2}:
 0.0       1.38629  1.94591
 0.693147  1.60944  2.07944
 1.09861   1.79176  2.19722

julia> exp(c)
3x3 Array{Float64,2}:
  2.71828   54.5982  1096.63
  7.38906  148.413   2980.96
 20.0855   403.429   8103.08
```

İkili dizi işlemleri

```
julia> a
3x3 Array{Int64,2}:
 10  40  70
 20  50  80
 30  60  90

julia> c
3x3 Array{Int64,2}:
 1  4  7
 2  5  8
 3  6  9
```

```
julia> a + c
3x3 Array{Int64,2}:
 11  44  77
 22  55  88
 33  66  99

julia> a * c
3x3 Array{Int64,2}:
 300  660  1020
 360  810  1260
 420  960  1500
```

Yayılan (broadcasting) dizi işlemleri

```
julia> a
3x3 Array{Int64,2}:
 10  40  70
 20  50  80
 30  60  90

julia> b
3-elt Array{Int64,1}:
 1
 2
 3
```

```
julia> a + b
ERROR:
DimensionMismatch

julia> a .+ b
3x3 Array{Int64,2}:
 11  41  71
 22  52  82
 33  63  93
```

Dizi indirgeme

```
julia> c
3x3 Array{Int64,2}:
 1  4  7
 2  5  8
 3  6  9
```

```
julia> sum(c)
45

julia> sum(c,1)
1x3 Array{Int64,2}:
 6  15  24

julia> sum(c,2)
3x1 Array{Int64,2}:
 12
 15
 18
```

Dizi indisleme

```
julia> a
3x3 Array{Int64,2}:
 10  40  70
 20  50  80
 30  60  90
```

```
julia> a[1,2]
40

julia> a[5]
50

julia> a[1:2,2:3]
2x2 Array{Int64,2}:
 40  70
 50  80

julia> a[1,:]
1x3 Array{Int64,2}:
 10  40  70
```

Dizi bitiştirme

```
julia> a
3x3 Array{Int64,2}:
 10  40  70
 20  50  80
 30  60  90

julia> b
3-elt Array{Int64,1}:
 1
 2
 3
```

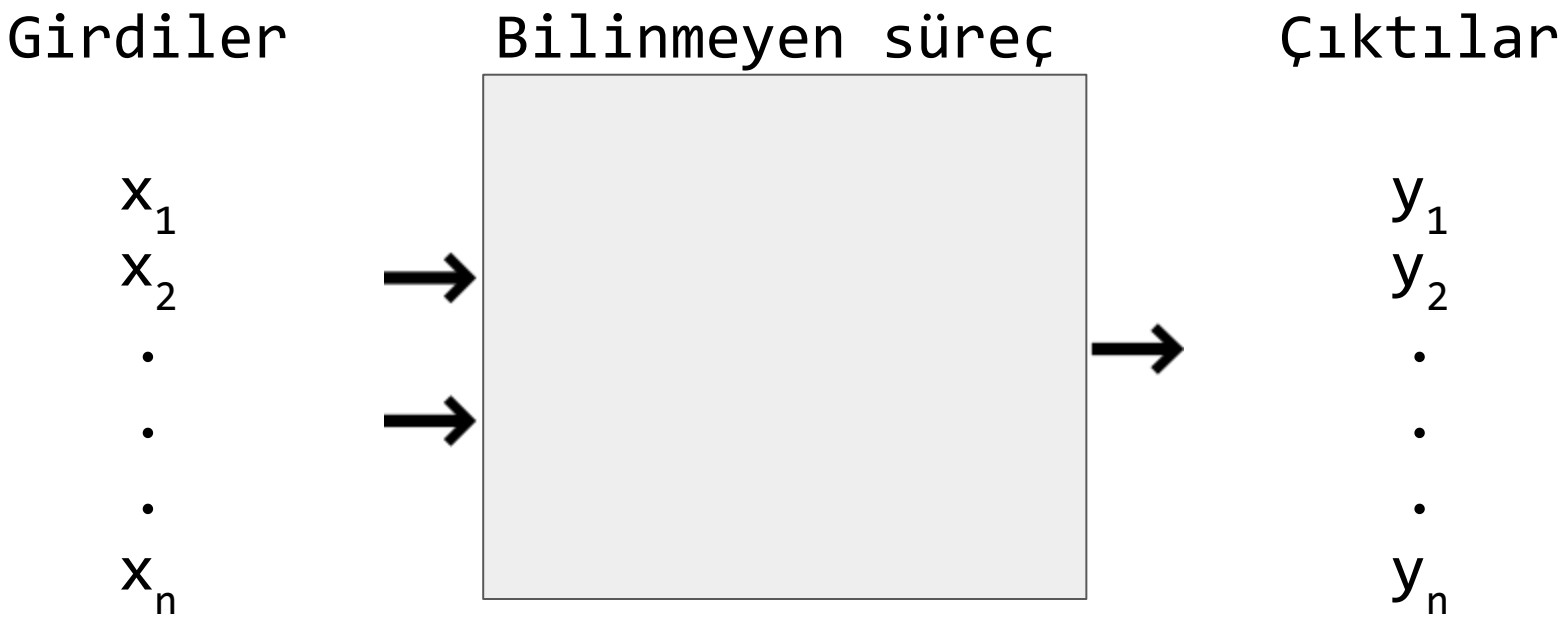
```
julia> [a b]
3x4 Array{Int64,2}:
 10  40  70  1
 20  50  80  2
 30  60  90  3

julia> [a;b']
4x3 Array{Int64,2}:
 10  40  70
 20  50  80
 30  60  90
  1   2   3
```

Fonksiyonlar

```
function loss(w, x, ygold)
    ypred = w * x
    ydiff = ypred - ygold
    sqerr = ydiff .^ 2
    qloss = sum(sqerr)
    return qloss
end

# veya kısa tanım ile:
loss(w,x,y)=sum((w*x-y).^2)
```

5 slaytta yapay öğrenme

Yapay öğrenme: gözlemleme

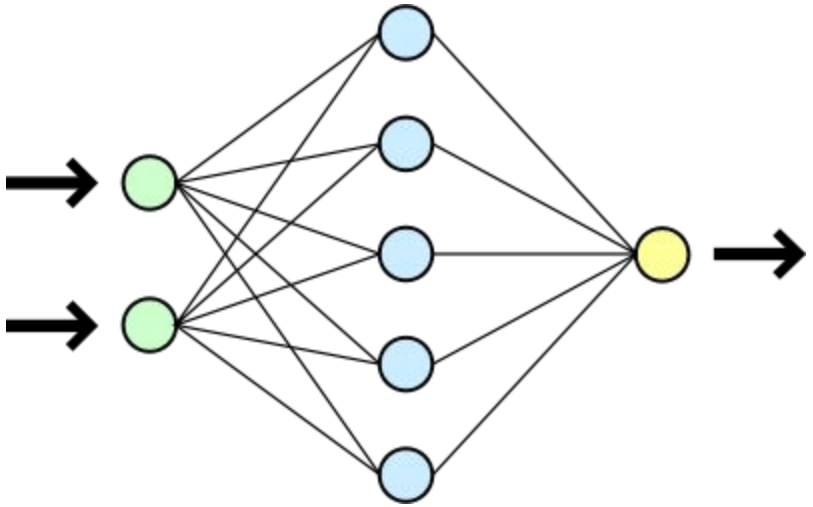


Yapay öğrenme: modelleme

Girdiler

x_1
x_2
.
.
.
x_n

Türevlenebilir
Program



Çıktılar

$\hat{y}_1$
$\hat{y}_2$
.
.
.
$\hat{y}_n$

Yapay öğrenme: hata (loss) fonksiyonu

```
function loss(w, x, ygold)
    ypred = predict(w, x)
    ydiff = ypred - ygold
    sqerr = ydiff .^ 2
    qloss = sum(sqerr)
end
```

Yapay öğrenme: optimizasyon döngüsü

- w parametreleri başlangıçta rastgele seçilir
- loss(w,x,y) => w ile yapılan tahmindeki hata
- gfun(w,x,y) => loss'un w parametrelerine göre türevi
- data = [(x1,y1),(x2,y2),...]: eğitim verisi
- SGD(w,data,loss) => hatayı azaltan w parametreleri bulur

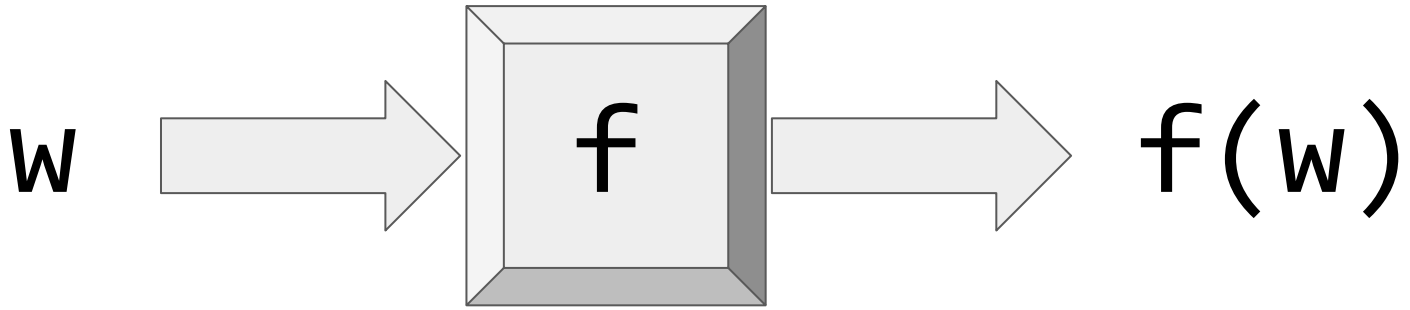
```
function SGD(w, data, loss)
    gfun = grad(loss)
    for (x,y) in data
        g = gfun(w, x, y)
        w = w - g * learningRate
    end
    return w
end
```

Yapay öğrenmede görev bölümü

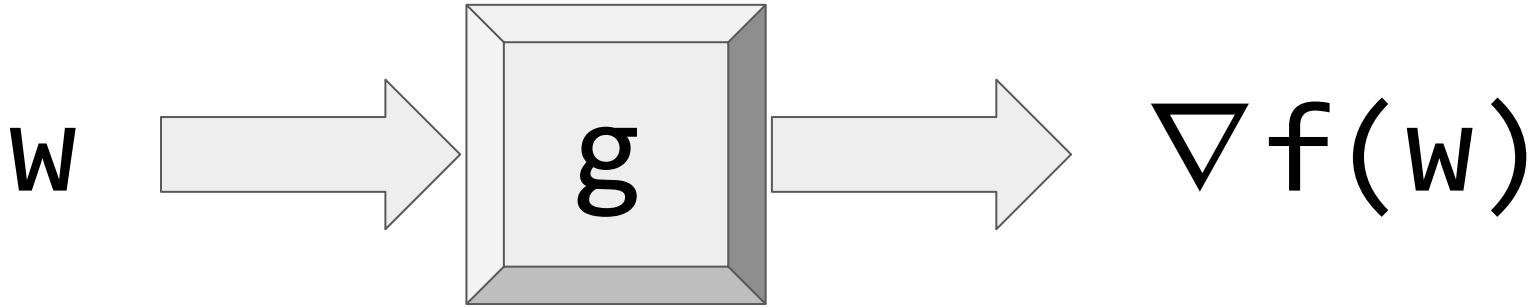
Programcı veriyi toplar, modelini, parametrelerini ve hata fonksiyonunu tanımlar.

Yapay öğrenme sistemi (Knet, TensorFlow vs.) parametreleri hata fonksiyonunun türevini kullanarak optimize eder.

1 slaytta Knet8




```
g = grad(f)
```



grad(f): Tüm Julia(!) için türev fonksiyonu

```
julia> sin(pi/6)      => 0.5
julia> cos(pi/6)      => 0.8660
julia> using Knet
julia> f(x)=sin(x)
julia> f(pi/6)        => 0.5
julia> f1 = grad(f)
julia> f1(pi/6)       => 0.8660
julia> f2 = grad(f1)
julia> f2(pi/6)       => -0.5
julia> f3 = grad(f2)
julia> f3(pi/6)       => -0.8660
```

+ GPU desteği

5 örnek model

Doğrusal regresyon: program

```
predict(w,x) = w[1]*x .+ w[2]

loss(w,x,y) = (sum((y-predict(w,x)).^2) / size(x,2))

lossgradient = grad(loss)
```

Doğrusal regresyon: eğitim

```
function train(w, data; lr=.1, epochs=20)
    for epoch=1:epochs
        for (x,y) in data
            g = lossgradient(w, x, y)
            for i in 1:length(w)
                w[i] -= lr * g[i]
            end
        end
    end
    return w
end
```

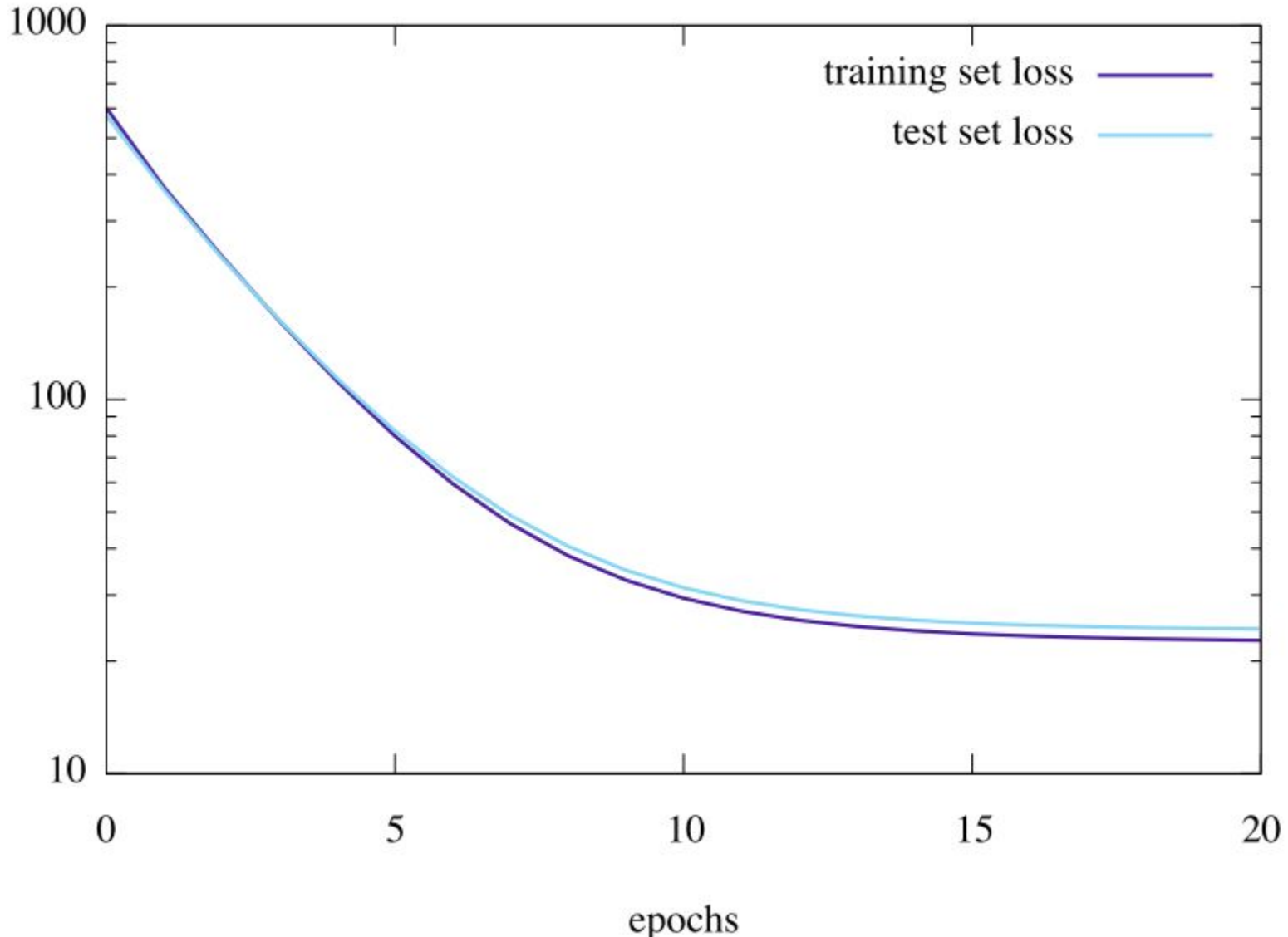
Doğrusal regresyon: data

- Örnek: UCI yapay öğrenme veri havuzundan “housing.data”
- 506 mahalle için Boston emlak verileri.
- Girdiler: suç oranı, averaj oda sayısı, vs. toplam 13 nitelik.
- Çıktı: ortalama ev değeri.

Doğrusal regresyon: eğitim

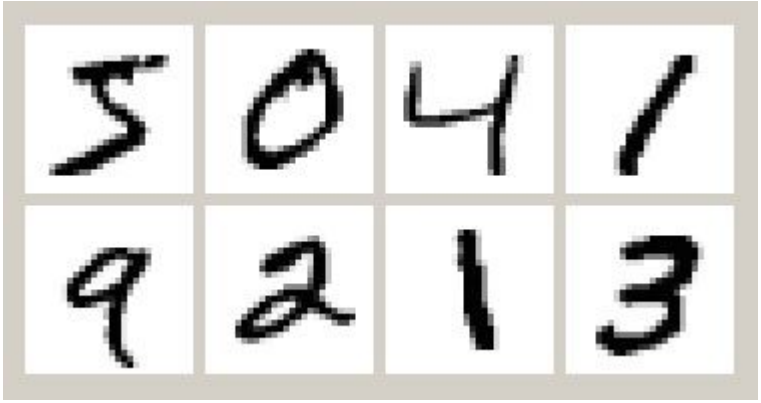
```
[Knet/examples]$ julia housing.jl
# housing.jl (c) Deniz Yuret, 2016. Linear regression model for
# the Housing dataset from the UCI Machine Learning Repository.

opts=(:seed,-1)(:epochs,20)(:lr,0.1)(:atype,"Array")(:fast,false)

size(data) = (14,506)

(:epoch,0,:trn,601.6809326430163,:tst,574.1631980915728)
(:epoch,1,:trn,369.1969797775598,:tst,362.5340494097422)
(:epoch,2,:trn,241.53609243589833,:tst,239.26355906515374)
...
(:epoch,18,:trn,22.9400510062471,:tst,24.566666584413355)
(:epoch,19,:trn,22.815344401770275,:tst,24.457955035865993)
(:epoch,20,:trn,22.717508849652422,:tst,24.37935078764147)

  2.095076 seconds (686.43 k allocations: 30.067 MB, 0.70% gc time)
```

Doğrusal regresyon: öğrenme eğrisi



Softmax regresyon: data

- Örnek veritabanı: MNIST
- Girdi: 28x28 elyazısı rakam, 60000 örnek
- Çıktı: Rakamların 0..9 kategorisi



Softmax regresyon: program

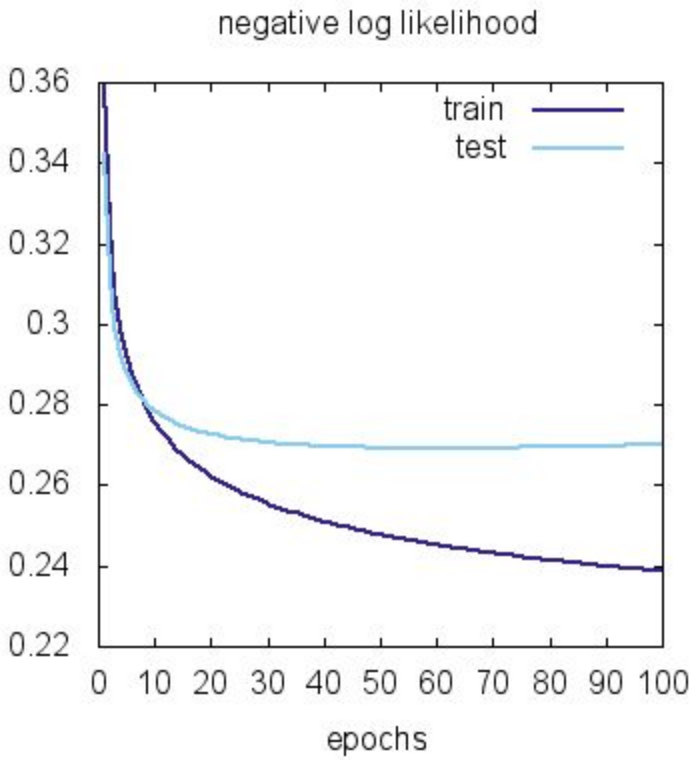
```
predict(w,x) = w[1]*x .+ w[2]    # doğrusal ile aynı

function loss(w,x,ygold)
    ypred = predict(w,x)
    ynorm = ypred .- log(sum(exp(ypred),1))
    -sum(ygold .* ynorm) / size(ygold,2)
end

lossgradient = grad(loss)        # doğrusal ile aynı

function train()...              # doğrusal ile aynı
```

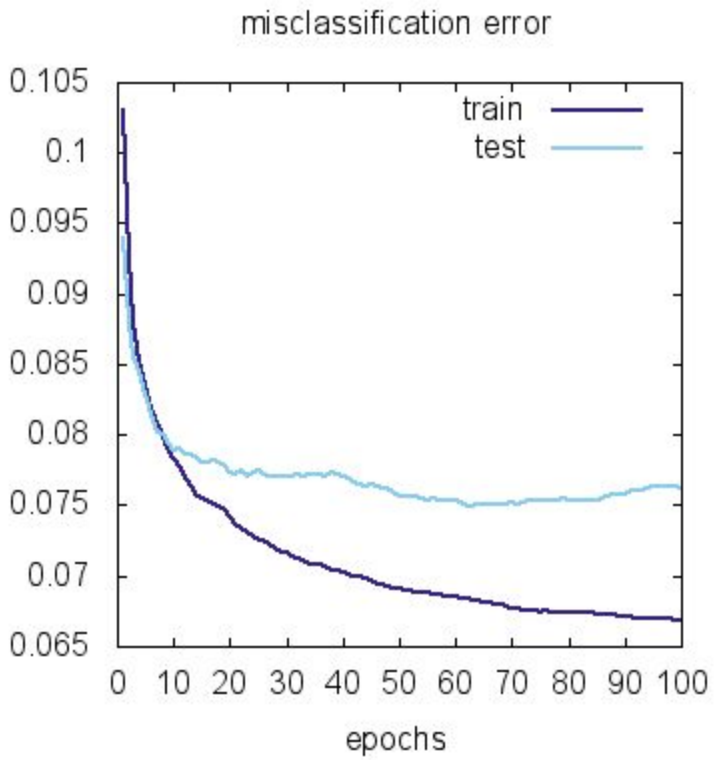
Cross entropy loss = $-\sum p_i \ln \hat{p}_i$

Softmax regresyon: eğitim

```
[Knet/examples]$ julia mnist.jl
# Handwritten digit recognition problem from
# http://yann.lecun.com/exdb/mnist.

INFO: Loading MNIST...
opts=(:seed,-1)(:batchsize,100)(:hidden,[])
(:epochs,10)(:lr,0.5)(:winit,0.1)(:fast,false)

(:epoch,0,:trn,0.08575,:tst,0.0807)
(:epoch,1,:trn,0.8991666666666667,:tst,0.9036)
 ...
(:epoch,9,:trn,0.9187666666666666,:tst,0.9154)
(:epoch,10,:trn,0.91945,:tst,0.9154)
```

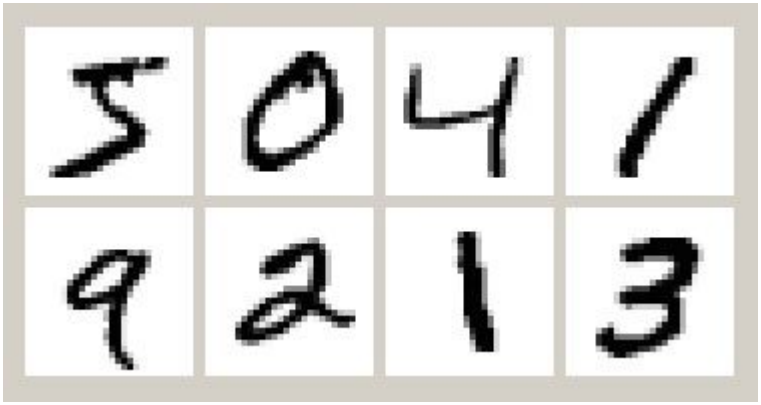
Softmax regresyon: öğrenme eğrisi



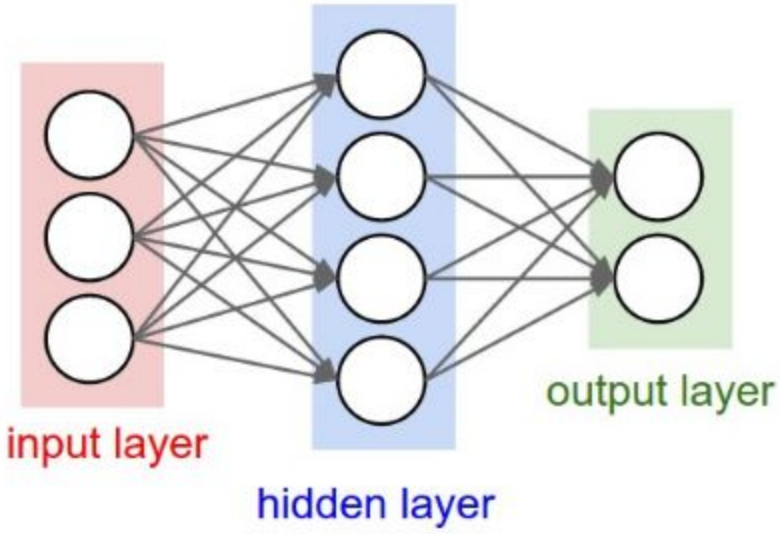


Çok katmanlı yapay sinir ağı: data

- Örnek veritabanı: MNIST
- Girdi: 28x28 elyazısı rakam, 60000 örnek
- Çıktı: Rakamların 0..9 kategorisi



Çok katmanlı yapay sinir ağı: model



http://cs231n.github.io/neural-networks-1/

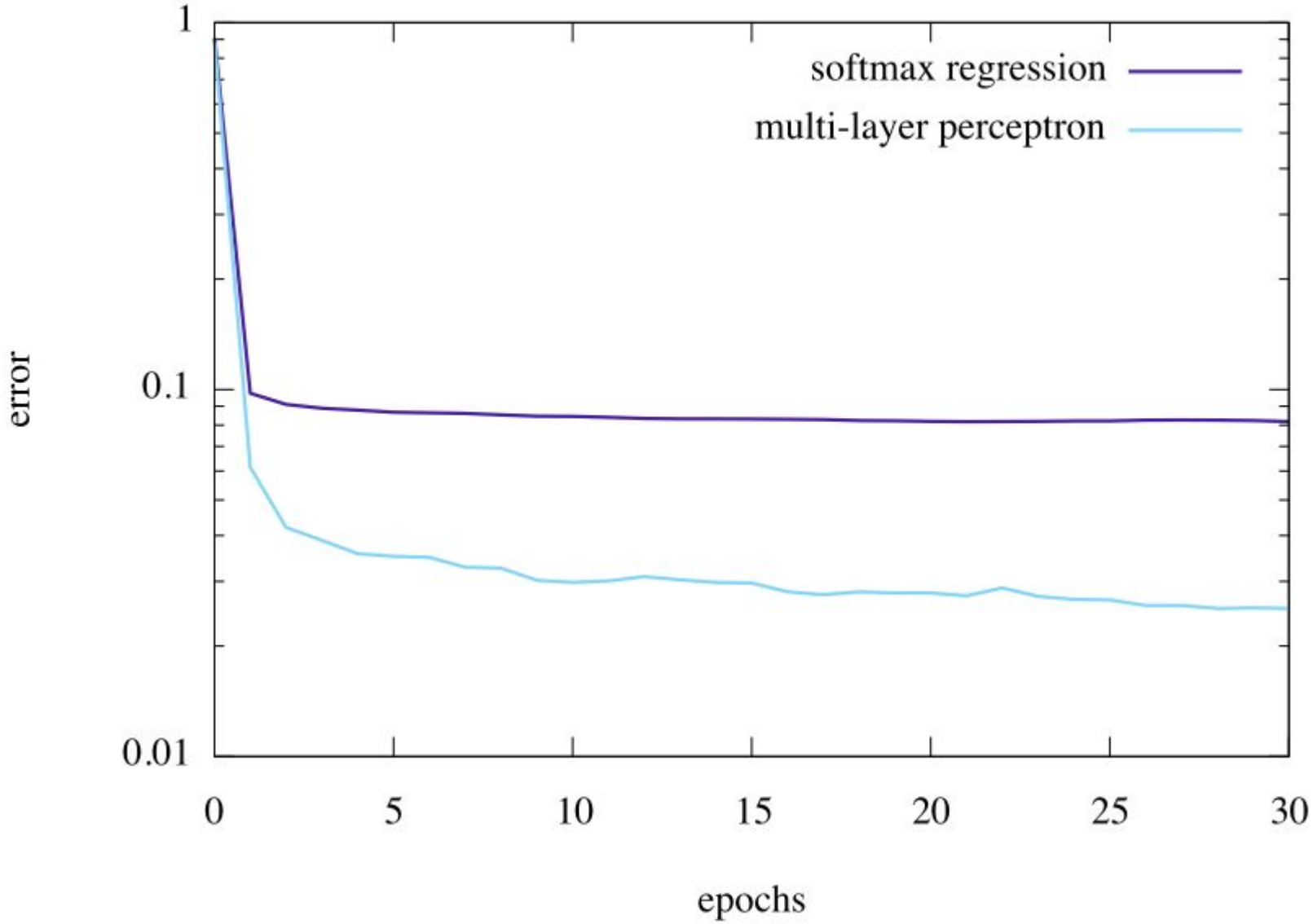
Çok katmanlı yapay sinir ağı: program

```
function predict(w,x)
    for i=1:2:length(w)-2
        x = max(0, w[i]*x .+ w[i+1])
    end
    return w[end-1]*x .+ w[end]
end

function loss(w,x,ygold)...     # softmax ile aynı

lossgradient = grad(loss)       # softmax ile aynı

function train()...             # softmax ile aynı
```

Çok katmanlı yapay sinir ağı: eğitim

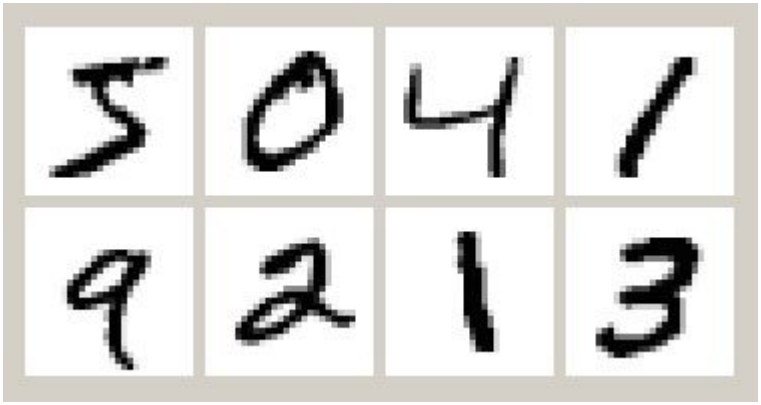
```
[Knet/examples]$ julia mnist.jl --hidden 64
# Handwritten digit recognition problem from
# http://yann.lecun.com/exdb/mnist.

INFO: Loading MNIST...
opts=(:seed,-1)(:batchsize,100)(:hidden,[])
(:epochs,10)(:lr,0.5)(:winit,0.1)(:fast,false)

(:epoch,0,:trn,0.1094,:tst,0.1071)
(:epoch,1,:trn,0.9435,:tst,0.9424)
 ...
(:epoch,9,:trn,0.9863,:tst,0.9728)
(:epoch,10,:trn,0.9875,:tst,0.9724)
```

Çok katmanlı yapay sinir ağı: öğrenme eğrisi



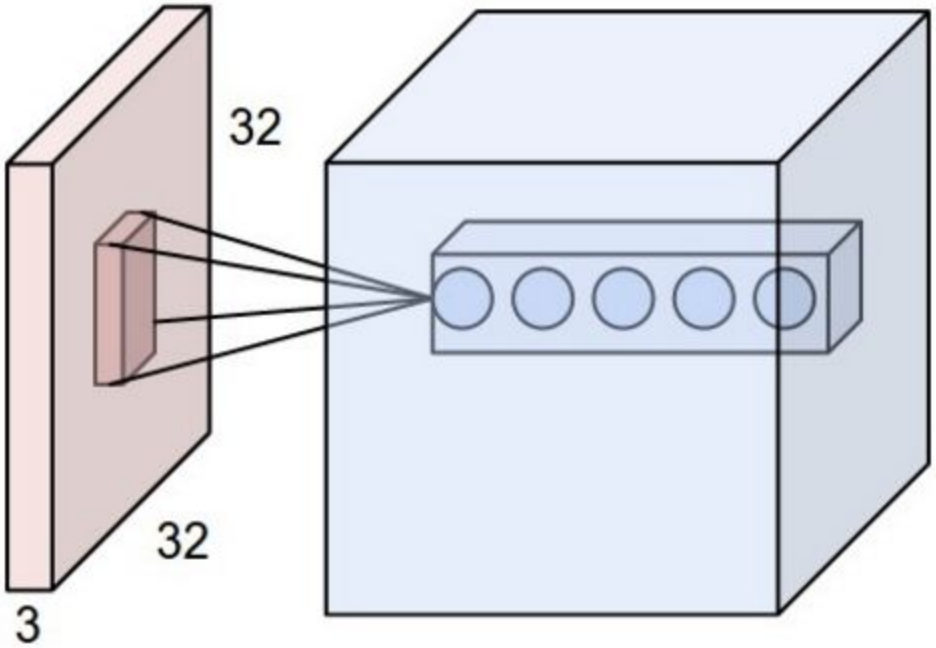
Evrişimsel (convolutional) sinir ağı: data

- Örnek veritabanı: MNIST
- Girdi: 28x28 elyazısı rakam, 60000 örnek
- Çıktı: Rakamların 0..9 kategorisi



(*) Convolutional: katlamalı ya da evrişimsel olarak da ifade edilmektedir.

Evrişimsel (convolutional) sinir ağı: model



http://cs231n.github.io/convolutional-networks/

Evrişimsel (convolutional) sinir ağı: program

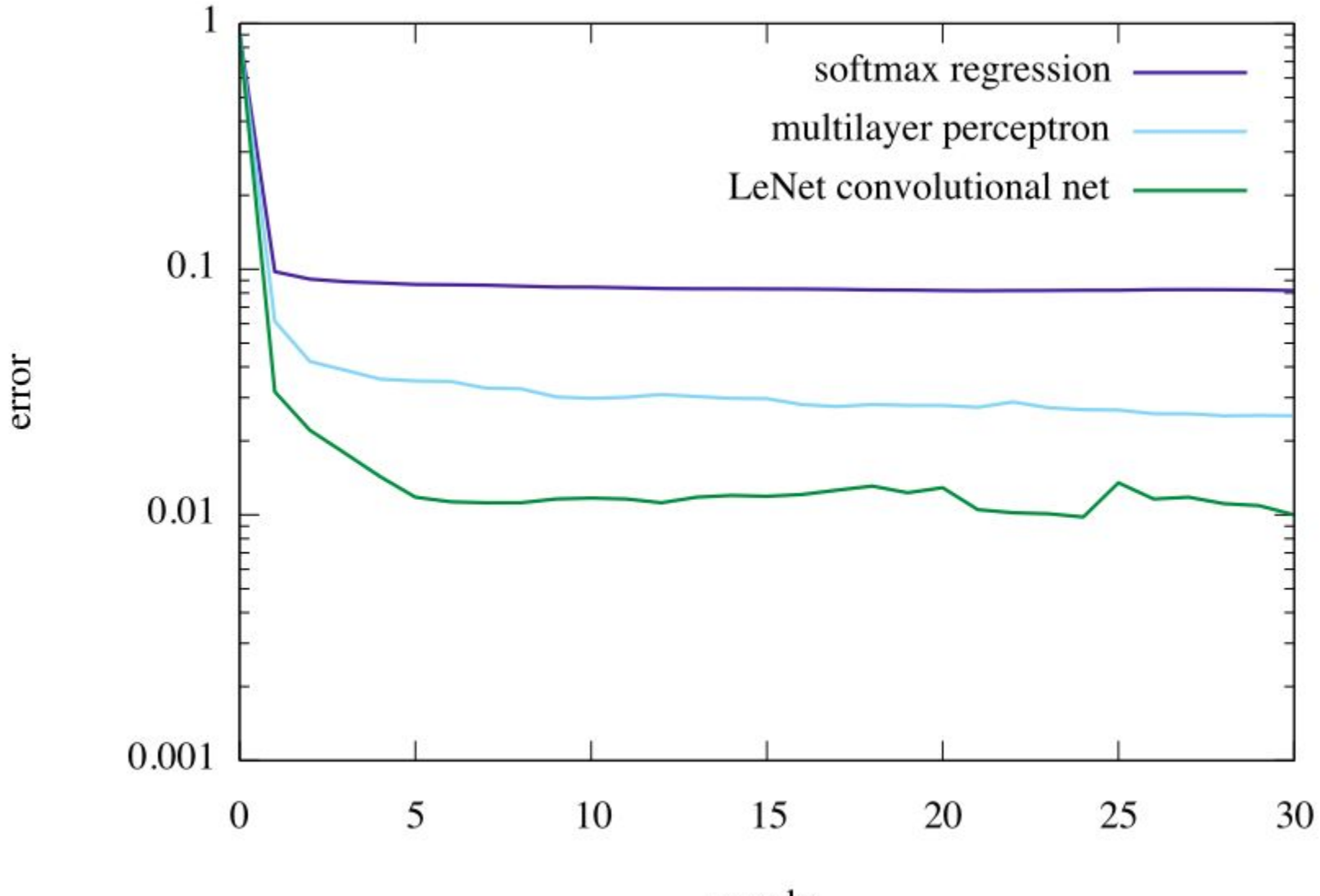
```
function predict(w,x)          # LeNet modeli
    x1 = pool(max(0, conv(w[1],x0) .+ w[2]))
    x2 = pool(max(0, conv(w[3],x1) .+ w[4]))
    x3 = max(0, w[5]*x2 .+ w[6])
    x4 = w[7]*x3 .+ w[8]
end

function loss(w,x,ygold)...    # softmax ile aynı

lossgradient = grad(loss)      # softmax ile aynı

function train()...            # softmax ile aynı
```

Evrişimsel (convolutional) sinir ağı: eğitim

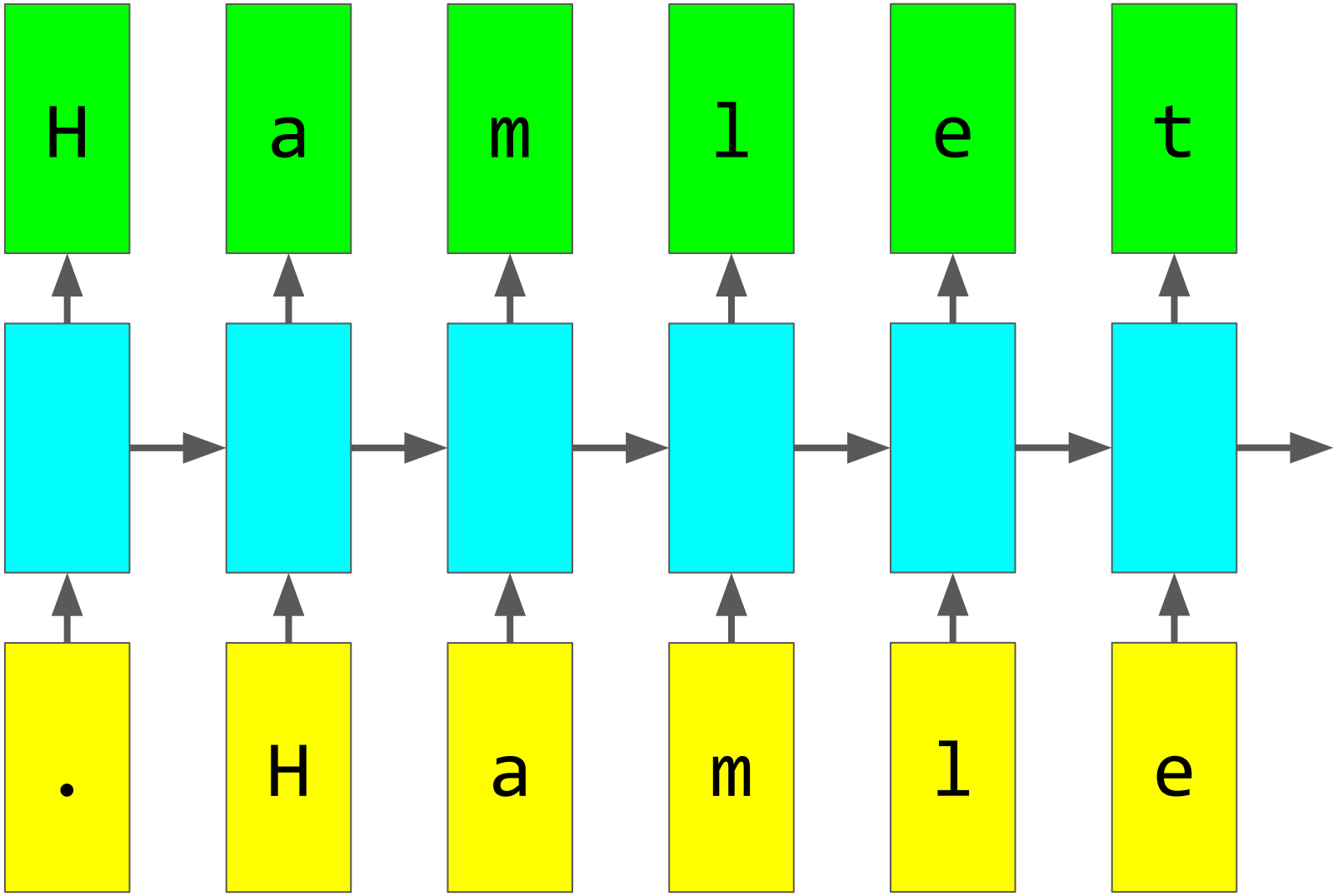
```
[Knet7/examples]$ julia mnist4d.jl
INFO: Loading MNIST
INFO: Testing lenet (convolutional net) on MNIST
("epochs"=>100,"lr"=>0.1,"seed"=>42,"gcheck"=>0,
"batchsize"=>100)

(1,0.9656,0.9683,...
(2,0.9774,0.9779,...
 ...
(9,0.9950,0.9884,...
(10,0.9957,0.9883,...
```

(*) Knet8 conv uygulaması henüz tamamlanmadığı için bu deney Knet7 ile yapıldı.

Evrişimsel (convolutional) sinir ağı: öğrenme eğrisi

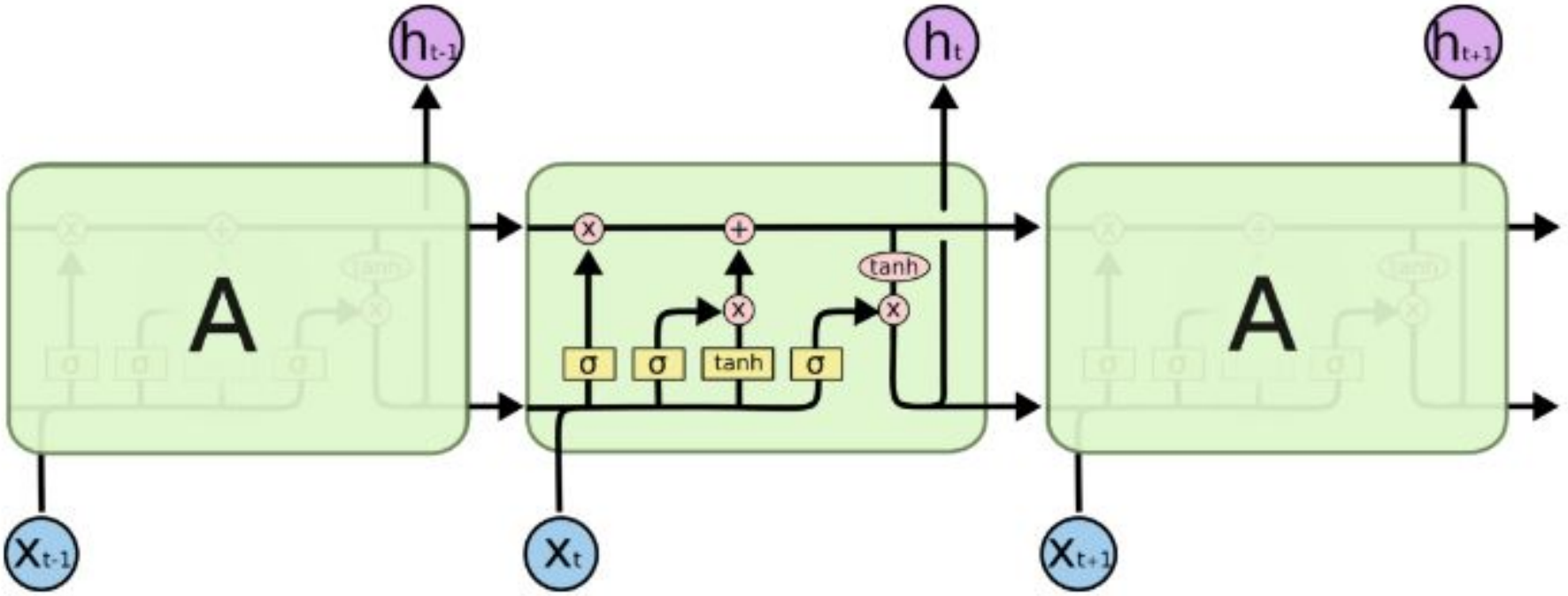


Özyinelemeli (recurrent) sinir ağı: model



(*) Recurrent: tekrarlayan ya da özyinelemeli olarak çevrilir.

Özyinelemeli (recurrent) sinir ağı: model



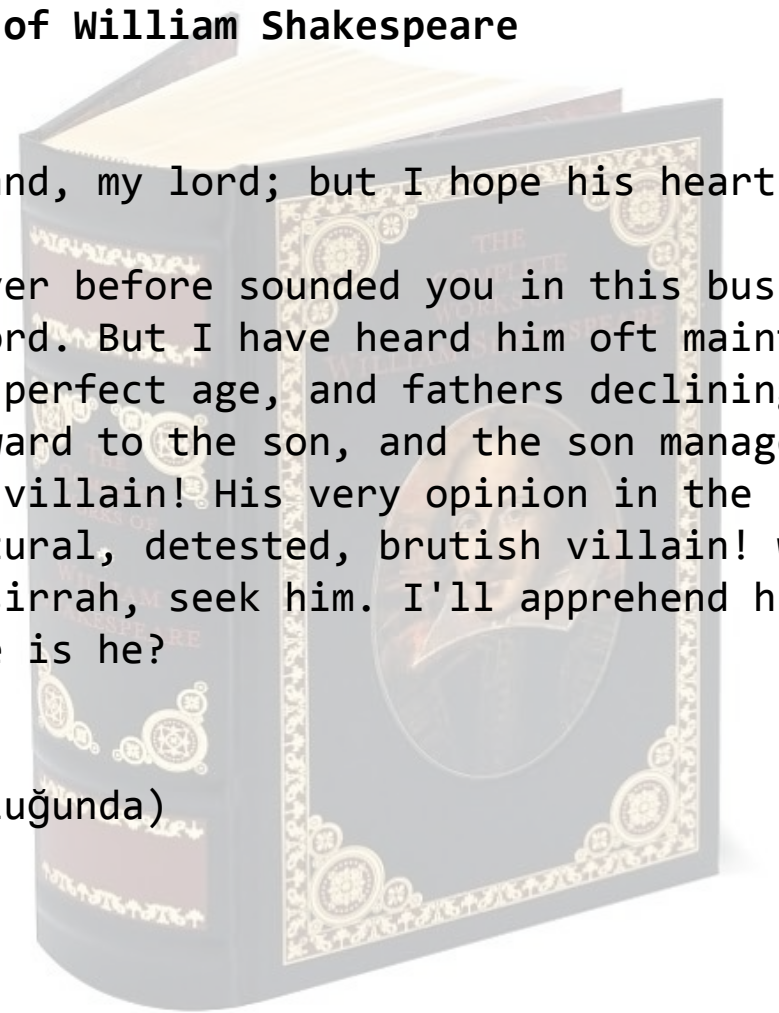
Long short term memory (LSTM) modülleri

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Özyinelemeli (recurrent) sinir ağı: program

```
function lstm(w, input, hidden, cell)
    x       = vcat(input, hidden)
    ingate  = sigm(w[:W_ingate]  * x .+ w[:b_ingate])
    forget  = sigm(w[:W_forget]  * x .+ w[:b_forget])
    outgate = sigm(w[:W_outgate] * x .+ w[:b_outgate])
    change  = tanh(w[:W_change]  * x .+ w[:b_change])
    cell    = cell .* forget + ingate .* change
    hidden  = outgate .* tanh(cell)
    return hidden, cell
end
```

Özyinelemeli (recurrent) sinir ağı: data



The Complete Works of William Shakespeare

```
  ...
  Edm. It is his hand, my lord; but I hope his heart is not in the
     contents.
  Glou. Hath he never before sounded you in this business?
  Edm. Never, my lord. But I have heard him oft maintain it to be fit
     that, sons at perfect age, and fathers declining, the father
     should be as ward to the son, and the son manage his revenue.
  Glou. O villain, villain! His very opinion in the letter! Abhorred
     villain! Unnatural, detested, brutish villain! worse than
     brutish! Go, sirrah, seek him. I'll apprehend him. Abominable
     villain! Where is he?
  ...
```

(5589887 harf uzunluğunda)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Özyinelemeli (recurrent) sinir ağı: eğitim

```
[Knet7/examples]$ julia charlm.jl --data 100.txt
--epochs 100
(:lr=>1.0,:dropout=>0.0,:embedding=>256,:gclip=>5
.0,:hidden=>256,:epochs=>100,:nlayer=>1,:decay=>0
.9,:seqlength=>100,:seed=>42,:batchsize=>128)
INFO: Chars read: (5589917,)
INFO: 92 unique chars
(epoch,lr,loss)
(1,1.0,2.2447511306910757)
(2,1.0,1.5556333172749894)
(3,1.0,1.3716149988793005)
(4,1.0,1.288365624960702)
(5,1.0,1.2409912395974114)
```

(*) Knet8 rnn gpu desteği henüz test edilmediği için bu deney Knet7 ile yapıldı.

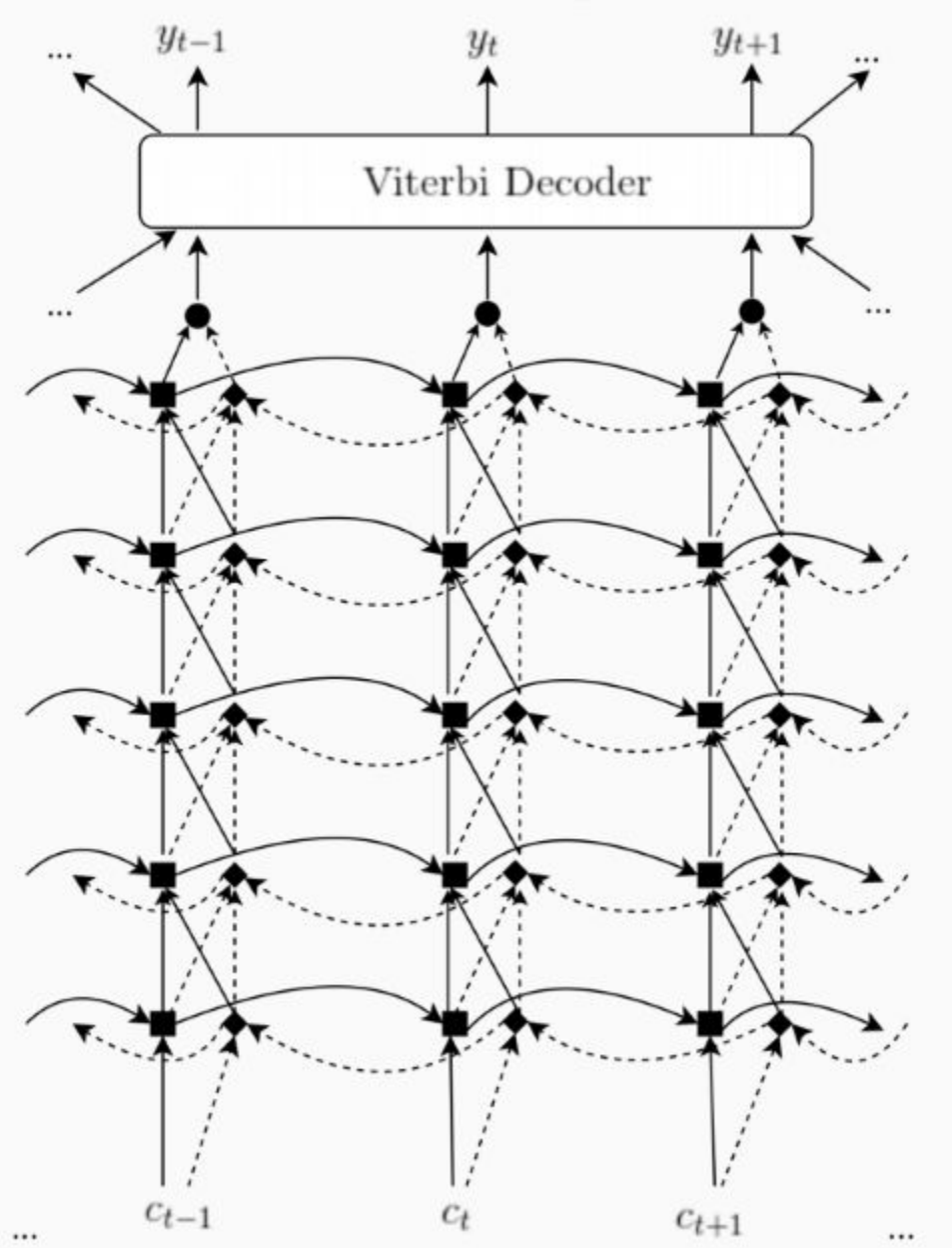
Özyinelemeli (recurrent) sinir ağı: çıktı

```
LUCETTA. Welcome, getzing a knot. There is as I thought you aim
  Cack to Corioli.
MACBETH. So it were timen'd nobility and prayers after God'.
FIRST SOLDIER. O, that, a tailor, cold.
DIANA. Good Master Anne Warwick!
SECOND WARD. Hold, almost proverb as one worth ne'er;
  And do I above thee confer to look his dead;
  I'll know that you are ood'd with memines;
  The name of Cupid wiltwite tears will hold
  As so I fled; and purgut not brightens,
  Their forves and speed as with these terms of Ely
  Whose picture is not dignitories of which,
  Their than disgrace to him she is.
...
```

Aynı modelle caz besteleri...

LSTM Music.aiff
by
Mustafa Ömer Gül

Daha karmaşık modeller



Onur Kuru (2016). Character-level tagging. MS Thesis. Koç University.

5 Türev Tekniği
(Neden Knet8?)

Türev programlama teknikleri

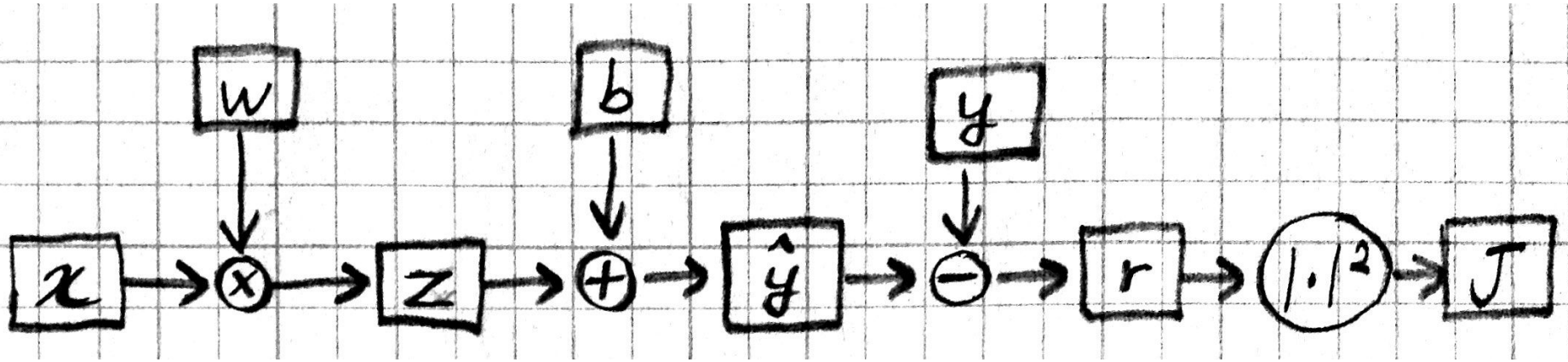
- Elle programlama
- Sayısal türev: $f'(x) \approx \frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon}$
- Sembolik türev: Maxima, Mathematica, Maple
- Türev derleyici: Theano, TensorFlow, **Knet7**
- Otomatik türev: autograd, **Knet8**

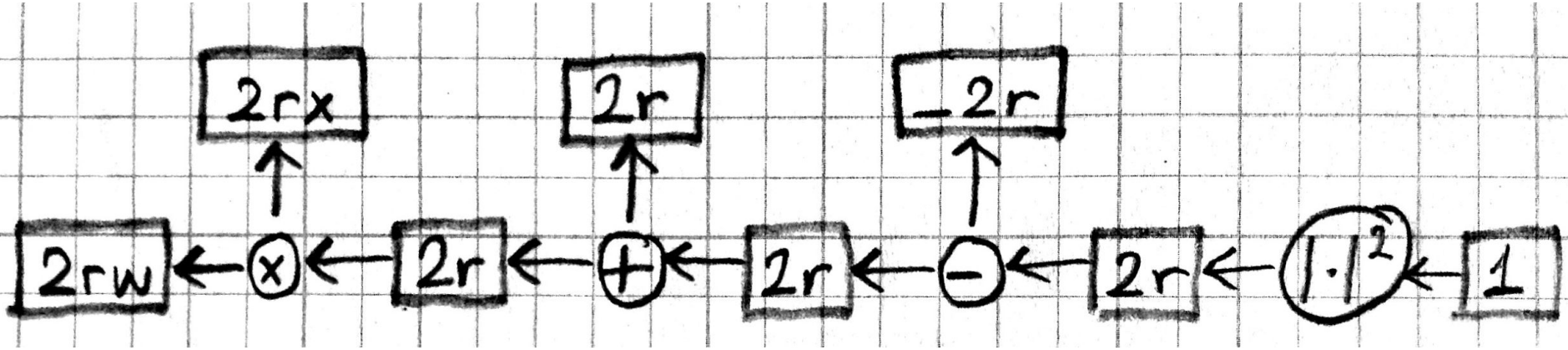
(*) Atılım Güneş Baydin et al. (2015) Automatic differentiation in machine learning: a survey

Elle programlama

Doğrusal regresyon örneği

$$loss(w,b,x,y) = |(wx+b) - y|^2$$





Model ve hata fonksiyonu

```
function loss(w, x, ygold)
    ypred = w * x
    ydiff = ypred - ygold
    sqerr = ydiff .^ 2
    qloss = sum(sqerr)
end
```

Hata fonksiyonunun türevi

```
function grad(w, x, ygold)
    ypred = w * x
    ydiff = ypred - ygold
    sqerr = ydiff .^ 2
    qloss = sum(sqerr)

    d_qloss = 1.0
    d_sqerr = d_qloss .* ones(sqerr)
    d_ydiff = d_sqerr .* (2 * ydiff)
    d_ypred = d_ydiff
    d_w     = d_ypred *  x'
end
```

Elle programlamanın dezavantajları

- Fazla emek ve dikkat gerektirir.

Sayısal türev

Sayısal türev

$$f'(x) \approx \frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon}$$

Dezavantajları:

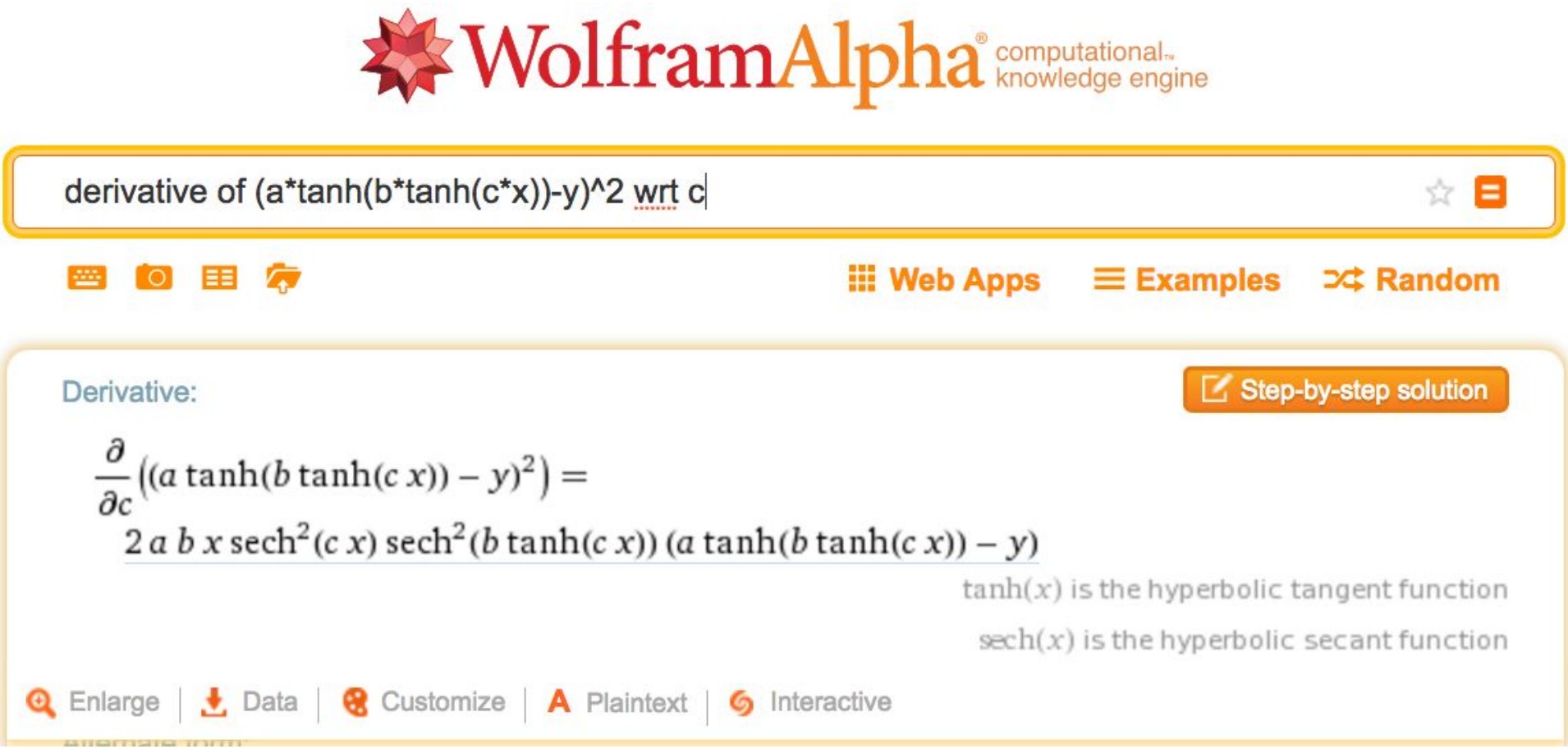
- Eğer *x* N boyutlu ise bu eşitliği N kere çözmek gerekir.

- Sayısal türevin ideal adım boyunu tespit etmek zor hata oranı yüksektir.

Sembolik türev (Mathematica, Maple)

Sayısal değil cebirsel ifadeler

Basit iki katmanlı bir model



Sembolik türevin dezavantajları

- Fonksiyonu kapalı formda ifade edebilmek gerekir (döngüsüz, koşulsuz).
- Komplex fonksiyonların türev ifadeleri üstsel (exponential) olarak büyüyebilir, verimli hesaba olanak vermeyebilir:

$$l_{n+1} = 4l_n(1 - l_n),\ l_1 = x$$

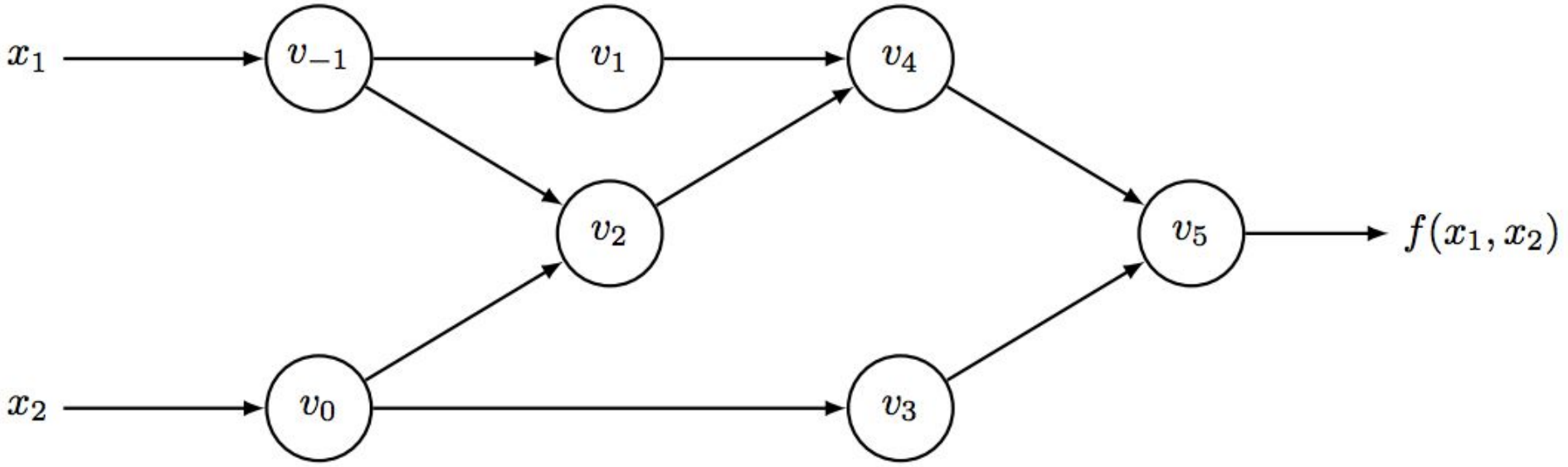
$$\frac{d}{dx} l_4 = \begin{array}{l} 128x(1-x)(-8+16x)(1- \\ 2x)^2(1-8x+8x^2)+64(1- \\ x)(1-2x)^2(1-8x+8x^2)^2- \\ 64x(1-2x)^2(1-8x+8x^2)^2- \\ 256x(1-x)(1-2x)(1-8x+ \\ 8x^2)^2 \end{array}$$

Türev derleyiciler (Knet7,Theano,Torch,Tensorflow…)

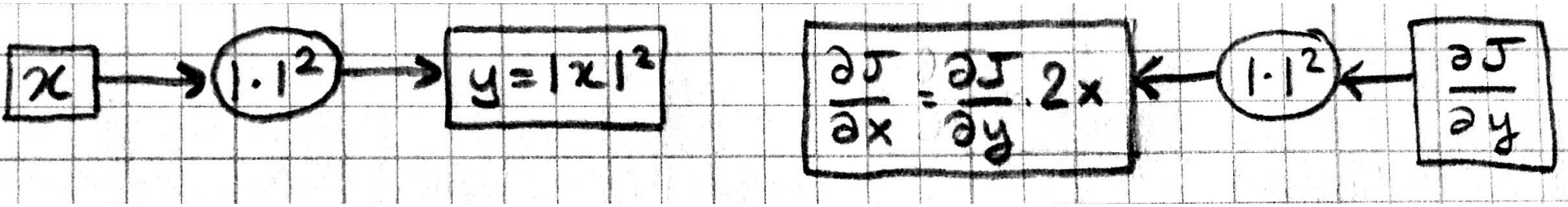
Fonksiyonun hesaplama grafiği oluşturulur

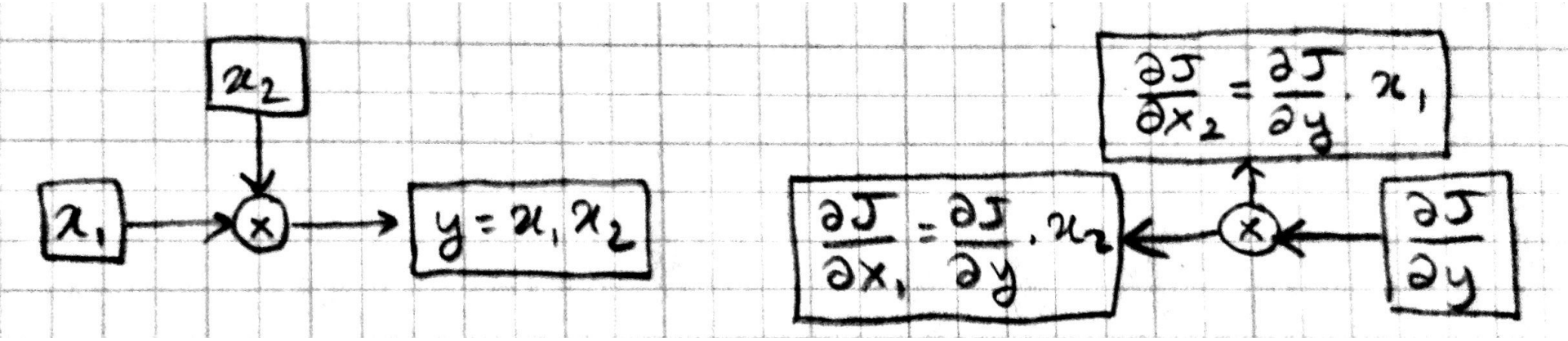
$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$

için hesaplama grafiği:



Sadece temel işlemlerin sembolik türevleri tanımlanır





Geriye doğru her değişkenin sonuca göre türevi numerik olarak hesaplanır

Forward Evaluation Trace

$$
\begin{array}{lll}
v_{-1} = x_1 & = 2 \\
v_0 = x_2 & = 5 \\
\hline
v_1 = \ln v_{-1} & = \ln 2 \\
v_2 = v_{-1} \times v_0 & = 2 \times 5 \\
v_3 = \sin v_0 & = \sin 5 \\
v_4 = v_1 + v_2 & = 0.693 + 10 \\
v_5 = v_4 - v_3 & = 10.693 + 0.959 \\
\hline
y = v_5 & = 11.652
\end{array}
$$

Reverse Adjoint Trace

$$
\begin{array}{lll}
\bar{x}_1 = \bar{v}_{-1} & & = \mathbf{5.5} \\
\bar{x}_2 = \bar{v}_0 & & = \mathbf{1.716} \\
\hline
\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}} & = \bar{v}_{-1} + \bar{v}_1 / v_{-1} & = 5.5 \\
\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0} & = \bar{v}_0 + \bar{v}_2 \times v_{-1} & = 1.716 \\
\bar{v}_{-1} = \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}} & = \bar{v}_2 \times v_0 & = 5 \\
\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} & = \bar{v}_3 \times \cos v_0 & = -0.284 \\
\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2} & = \bar{v}_4 \times 1 & = 1 \\
\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1} & = \bar{v}_4 \times 1 & = 1 \\
\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} & = \bar{v}_5 \times (-1) & = -1 \\
\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4} & = \bar{v}_5 \times 1 & = 1 \\
\hline
\bar{v}_5 = \bar{y} & & = 1
\end{array}
$$

Knet7 örnek hesaplama grafiği

```
@knet function f(x1,x2)
    return relu(x1) + x1.*x2 - soft(x2)
end
```

```
julia> compile(:f)
1 Knet.Input()
2 Knet.Input()
3 Knet.Relu(1,)
4 Knet.Mul(1,2)
5 Knet.Add(3,4)
6 Knet.Soft(2,)
7 Knet.Axpb(6,)
8 Knet.Add(5,7)
```

Knet7 örnek temel işlem türevleri

```
function back(::Sigm,y,dy,dx)
    for i = 1:length(dx)
        dx[i] = dy[i]*y[i]*(1-y[i])
    end
end

function back(::Relu,y,dy,dx)
    for i = 1:length(dx)
        dx[i] = dy[i] * (y[i] > 0)
    end
end
```

Türev derleyicilerin dezavantajları

- Modeller derlenebilmeleri için yeni ve sınırlı bir mini programlama dilinde ifade edilir.
- Bu mini dillerin dizi indisleme, yardımcı fonksiyon, ve kontrol akışı işlemleri (döngü, koşul) sınırlıdır. (Ör. Theano “scan”).
- Model bir kere derlendiği için işlem akışı girdilerin yapısına göre esneklik gösteremez.

Maclaurin, D. et al. (2015). "Autograd: Effortless Gradients in Numpy."

Otomatik türev (Knet8)

Tüm türevlenebilir programlara destek

```
function f(x)
    s = 0
    for i = 1:length(x)
        if x[i] <= 0
            s += exp(x[i])
        else
            s += log(x[i])
        end
    end
    return s
end
```

```
julia> x = randn(3)'
 0.88  -2.52  1.55

julia> f(x)
0.3949414650701943

julia> 1./x
 1.13  -0.39  0.64

julia> exp(x)
 2.41  0.08  4.71

julia> g=grad(f); g(x)
 1.13  0.08  0.64
```

Otomatik türev grad(f) nasıl çalışır

- Kullanıcının fonksiyonu f çalışırken temel işlemler kaydedilir.
- Program sonlandığında ortaya çıkan hesaplama grafiği üzerinden türev alınır.
- Girdiler işlenmeden herhangi bir derleme söz konusu olmadığı için model girdilere bağlı olarak esnek davranabilir.

Peki bu programı çok yavaşlatmaz mı?

işlem	süre
a1 = w1 * x	0.56
a2 = a1 .+ b1	0.59
a3 = max(0,a2)	0.62
a4 = w2 * a3	0.75
a5 = a4 .+ b2	0.78
a6 = a5 - y	0.82
a7 = a6 .^ 2	0.85
a8 = sum(a7)	1.06
kaydetme	1.18
türevler	2.10

Otomatik türevin avantajları

Model tanımlar ve eğitirken dilin tüm özellikleri kullanılabilir:

- döngüler
- koşullu ifadeler
- yardımcı fonksiyonlar
- özyinelemeli fonksiyonlar (recursion)
- fonksiyon üreten fonksiyonlar (closures)

github.com/denizyuret/Knet.jl
knet.rtfd.org
knet-users mailing list

Özet

Bu derste derin öğrenme modellerini tanımlamak, eğitmek ve değerlendirmek için gerekli adımlar örnekleriyle sunulacaktır. Amaç, sadece temel programlama bilgisine sahip bir katılımcının en kısa zamanda kendi problemleri için gerekli derin öğrenme modellerini geliştirebilir ve eğitebilir hale gelmesidir. Örnekler Julia [1] dilinde yazdığım Knet.jl [2] derin öğrenme paketi üzerinden verilecektir. Julia, açık kaynak kodlu, yüksek performanslı ve yüksek seviyede programlanabilen bir dil olarak derin öğrenmeye giriş için uygun bir platform oluşturmaktadır. Knet.jl derin öğrenme modellerinin program fonksiyonlarına yakın bir dille tanımlanmasına izin vermekte, alt fonksiyonlar kullanarak kompleks modellerin kolaylıkla geliştirilmesine olanak sağlamakta, model eğitimi için gerekli türev ve güncelleştirme işlemlerini otomatikleştirerek kullanıcının model yapısına konsantre olmasını kolaylaştırmaktadır. Ders basit regresyon ve sınıflandırma modelleri ile başlayıp, çok katmanlı (multi-layer), katlamalı (evrişimsel,convolutional) ve tekrarlayan (özyinelemeli,recurrent) sinir ağları ile devam edecek, bu modellerle emlak fiyat tahmini, el yazısı tanıma ve verilen bir stilde metin üretme gibi problemler için uygulamalar geliştirilecektir.