CS479 neural network

Austin Xia

April 10, 2022

Contents

List of Figures

List of Tables

1 Neurons

Definition 1.0.1 (neuron)
cell that send and receive electric signal to and from each other

soma is body of neuron cell, it generate signal.
signal travels along **axon**.
dentrites receives signal.

1.1 nueron membrane potential

Definition 1.1.1 (ion)
Ions are molecules or atoms in which number of elections (-) does not match number of protons(+), resulting in net charge.

Many Ions float around your cells. The cells's membrane, stops most ions from crossing.
However, ion channels embedded in the cell membrane can allow ions to pass

on membrane there is **sodium channel, potassium channel, sodium-potassium pump**
sodium channel lets in 1 Na,
potassium chaneel lets out 1 $K+$, sodium-potassium pump lets in $2K+$ and lets out $3Na+$
the Sodium-Potassium Pump cause net negative charge inside the cell. the voltage difference is called the **membrane potential**

$$\text{voltage} \equiv \text{membrane potential}$$

1.2 Action Potential

a Neuron Behavior. neuron produce a spike of electrical activity called **an action potential**. After collecting enough membrane potential, The electrical burst travels along neuron's **axon** to its **synapses**, where it passes signals to other neurons

1.3 Hodgkin-Huxley Model

a model of action potential (spike), based on the nonlinear relationship between membnrane potential (voltage) and opening/closing of $Na+/K+$ ion channels
let V be membrane potential. A neuron usually keeps a V of $-70mV$
The fraction of $K+$ channels that are open is $n(t)^4$
(somehow probability of one channel to be open is n, then probability all 4 channel open is n^4, 4 channels has to be opened simutaliously to open potassium channel)
the dynamic model is

$$\frac{dn}{dt} = \frac{1}{\tau_n(v)}(n_\infty(v) - n)$$

$n(t)$ converges to n_∞. Bigger τ leads to slower converge

the fraction of $Na+$ ion channels open is $(m(t))^3 h(t)$

where

$$\frac{dm}{dt} = \frac{1}{\tau_n(v)}(m_\infty(v) - m)$$
$$\frac{dh}{dt} = \frac{1}{\tau_n(v)}(h_\infty(v) - h)$$

m,h,n are functions of v

now an equation for V:

$$c\frac{dV}{dt} = J_{in} - g_L(V - V_L) - g_{Na}m^3h(V - V_{Na}) - g_K n^4(V - V_k)$$

J_{in} is input current (like from another neuron)
$g_L(V - V_L)$ leak current
V_{Na}, V_K are zero-current potential, at what voltage would K be balanced g is maxium conductance
$g_{Na}m^3h(V - V_{Na})$ is Na current (conductance times fraction), same for K

The HH model is greatly simplified

- a neuron is treated as a point in space (do not consider dentrites)
- conductance are approximated with formulas
- only consider $K+$, $Na+$ and generic leak currents (there are also Ca)

But to model a single spike takes many time steps of this 4-D system. However spikes are fairly generic, and it is thought the **presence** of a spike is more important than its specific shape

1.4 Leaky Integrate-and-Fire (LIF) Model

Definition 1.4.1 (sub-threshold membrane potential)
(informal) a voltage that after reaching it, a spike occurs

The LIF model only considers the **sub-threshold memberane potential** (voltage).

$$C\frac{dV}{dt} = J_{in} - g_L(V - V_L)$$

this is basically how the voltage will we change depending on how far we are from zero-potential(V_L) voltage by physics law,

$$\tau_m\frac{dV}{dt} = V_{in} - (V - V_L) \text{ for } V < V_{th}$$

τ_m is a time constant, determines how quickly things happen.
V_{th} is sub-threshold membrane potentail
change of variable:

$$v = \frac{V - V_L}{V_{th} - V_L}, v_{in} = \frac{V_{in}}{V_{th} - V_L}$$

then we have

$$\tau_m\frac{dv}{dt} = v_{in} - v$$

if $v_{in} = 0$, $v \to 0$
at threshold, $v = 1$
The model works by: integrating $v \sim t$ until v reaches 1
Then wait for t_{ref}, then start integrating again starting from $v = 0$

1.5 LIF Firing Rate

suppose we hold v_{in} constant, we call solve

$$\tau_m \frac{dv}{dt} = v_{in} - v$$

by

$$v(t) = v_{in}(1 - e^{-\frac{t}{\tau}})$$

so v will start from 0 and approach v_{in}

if v_{in} is larger than 1, a spike will occur some time

we can get the firing rate by time it takes for a spike to occur

It can be shown steady-state firing rate for a constant input v_{in} is

$$G(v_{in}) = \frac{1}{\tau_{ref} - \tau_m ln(1 - \frac{1}{v_{in}})} \text{ for } v_{in} > 1 \text{ , 0 otherwise}$$

the curve for steady-state firing rate is called **tuning curve**

1.6 sigmoid neron

activity of neuron is very low or zero when input is low. and it goes up and approaches some maximum as input increases.

this general behaviour can be represented by **activation functions or** $\sigma(z)$

a common $\sigma(z)$ is logistic function

Definition 1.6.1 (softmax)

$$SoftMax(\vec{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

2 how neuron interact

2.1 synapse

Definition 2.1.1 (synapse)
The junction where one neuron communicates with the next neuron is called a synapse

- a pre-synaptic action potential
- causes release of neural transmitter
- neural-transmitter binds to receptor on the post-synaptic neuron
- opening ion channels and changing membrane potential

The synaptic process takes time. some fast some slow.

using τ_s to represent the time constant that an action potential affects next neuron by synaptic process the current entering the post-synaptic neuron can be written as

$$h(t) = kt^n e^{\frac{-t}{\tau_s}} \text{ if } t \geq 0 \exists n \in Z_+, \text{ or 0 otherwise}$$

k is choosen so $\int h(t)dt = 1$
τ_s is how long it takes for spike to cause current
h(t) is called post-synaptic current (PSC) filter, or post-synaptic potential (PSP) filter

2.2 spike train

means multiple spikes, the spike train can be modelled as:

$$a(t) = \sum \delta(t - t_p)$$

Definition 2.2.1

$$\delta(t) = \infty \text{ if t=0, 0 otherwise}$$
$$\int \delta(t)dt = 1, \int f(t)\delta(T - t)dt = f(T)$$

a(t) is spike train, h(t) is PSC(post synaptic current) filter, s(t) filtered spike train, or PSC
$s(t) = (a * h)(t) = \sum_p h(t - t_p)$ sum of PSC, one for each spike
a spike train influence the post-synaptic neuron by:adding all PSC filters, one for each spike

2.3 connection weight

the current induced by action potential onto a particular post-synaptic neuron can vary based on

- numbersize of synapses
- amounttype of neurotransmitter
- numbertype of receiptor

we combine them into **connection weight**.
Thus total input to neuron is a weighted sum of filtered spike-trains w_{ca} is weight to c caused by a
neuron X has node x_1, x_2, neuron Y has y_1, y_2, y_3 then $W = \begin{bmatrix} w_{11}w_{12} \\ w_{21}w_{22} \\ w_{31}w_{32} \end{bmatrix}$ and we store neuron activities $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$
we compute input to nodes in Y using

$$\vec{z} = W\vec{x} + \vec{b}$$

b holds the biases for nodes(neurons) in Y
$\vec{y} = \sigma(\vec{z}) = \sigma(W\vec{x} + \vec{b})$
to elimilate the annoying $\vec{b}$ $W\vec{x} + \vec{b} = [W|\vec{b}] \begin{bmatrix} \vec{x} \\ - \\ 1 \end{bmatrix}$

2.4 implementing connections between spiking neurons

for simplicity when n=0

$$h(t) = \frac{1}{\tau_s} e^{\frac{-t}{\tau_s}}$$

is the solution to initial value problem

$$\tau_s \frac{ds}{dt} = -s, s(0) = \frac{1}{\tau_s}$$

question here
s is current
algorithum:

$$\tau_m \frac{dv_i}{dt} = s_i - v_i$$

$$\tau_s \frac{ds_i}{dt} = -s_i$$

if v_i reaches 1..

- start refractory period
- send spike along axon
- reset v to 0

if a spike arrives from neuron j, we

$$s_i \leftarrow s_i + \frac{w_{ij}}{\tau_s}$$

3 an overview

- supervised learning. The desired output is known so we can compute the error and use that error to adjust our network
- unsupervised learning. The output is not known, cannot be used to generate error signal, instead, find efficient representations for statistical structure in the input
- reinforcement learning. feedback is given, but less often. and the error signal is usually less specific

3.1 supervised learning

our neural network performs some mapping from input space to output space
we are given training data, with many examples of iput/target pairs.
Our task is to alter connection weights in network so our network mimics this mapping
our goal is bring output close to target as possible. For now we use scalar functions $L(y, t)$ cost function
2 common types of mappings encountered in supervised learning are: *regression* and *classification* the difference is result being continous/discrete
this goes down to optimization problem
Neural learning seeks

$$min_\theta E[L(f(x, \theta), t(x))]_{x \in data}$$

4 universal approximation theorem

Theorem 4.0.1
let σ be any continous sigmoidal function. Then finite sum of the form

$$G(x) = \sum_j \alpha(w_j x + \theta_j)$$

are dense in $C(I_n)$, in other words, given $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, G(x) of the above form, for which

$$|G(x) - f(x)| < \epsilon \text{ for all } x \in I_n[0,1]^n$$

a function is sigmoidal if

$$\sigma(x) = 1 \text{ as } x \to \infty \text{ ,and =0 as } x \to -\infty$$

like logistic function
the theorem:

$$\exists N, w_j, \theta_j, \alpha_j, \text{ for j=1...N such that } |G(x) - f(x)| < \epsilon$$

Heaviside step functino,

$$H(x) = lim_{w \to \infty} \sigma(w(x-b))$$

we can use 2 such function to create a peice . $P(x, b, \delta) = h(x, b) - H(H, b + \delta)$

5 loss function

cost function OR objective function OR loss function OR error function
suppose we are given $\{x_i, t_i\}_{i=1...N}$ for input x_i, network's output is $y_i = f(x_i, \theta)$
then MSE =

$$L(y,t) = |y-t|_2^2/2$$

$$E = \frac{1}{N} \sum_{i=1}^{N} L(y_i, t_i)$$

5.1 coess-entropy (bernoulli cross entropy)

$f(x;\theta) = y, y \in (0,1)$
suppose y is probablity that $x \to 1$ $y = P(x = 1|\theta) = f(x;\theta)$
we can treate y as bernoulli distribution
likelihood of our data sample given our model is

$$p(x \to t|\theta) = y^t(1-y)^{1-t}$$

$$L(y,t) = -(tlog(y) + (1-t)log(1-y))$$

the log-likelihood is basis of cross-entrophy loss function
expected cross entropy over dataset is

$$E = -E[t_i logy_i + (1-t_i)log(1-y_i)]_{data}$$

categorical cross-entropy (multinouli cross-entropy):
we have K classes, and input x
network output is $y = f(x;\theta)$
we interpret y_k as probability x is from class k
$y \in [0,1]^k$, eg, $y = [0.2, 0.3, 0.5]$ sum to 1
suppose we observe sample from class k, likelihood of that observation is

$$P(x \in C_k) = y_k$$

where $C_k = \{x|xisfromclassk\}$
note y is function of input x, if t = [0,0,0,1,0,0,0]
likelihood is

$$P(x \in C_k) = \prod y_k^{t_k}$$

loglikelihood is

$$-\sum t_k logy_k$$

loss function is known as categorical cross-entropy

$$L(y,t) = -\sum t_k logy_k$$

expected categorical cross-entropy for a dataset is

$$\frac{-1}{N}\sum_i \sum_k t_k^{(i)} logy_k^{(i)}$$

6 gradient descent learning

operation of our network can be written

$$y = f(x;\theta)$$

where θ is connection weights and biases

$$min_\theta E(\theta) where E(\theta) = E[L(f(x;\theta), t(x))]_{x \in data}$$

we can apply gradient descent to E using gradient

$$dE = [\frac{dE}{d\theta_0}, ... \frac{dE}{d\theta_p}]^T$$

approcimating the gradient numerically:
finite-difference approximation: for function $E(\theta)$, $\frac{dE}{d\theta} = \frac{E(\theta+d\theta)-E(\theta-d\theta)}{2d\theta}$

6.1 Error Backpropagation

Goal: find efficient method to compute gradients for gradient-descent optimization
consider network

$$\frac{dE}{dM_{41}} = \frac{dE}{dy_1}\frac{dy_1}{d\beta_1}\frac{d\beta_1}{dM_{41}}$$

since $p_1 = \sum_i h_i M_{ii} + b_1$, $\frac{d\beta_1}{dM_{41}} = h_4$

x input, h hidden nodes, y output, $Wx = \alpha \to h, Mh = \beta \to y$:
$x \in R^x, h \in R^H, y, t \in R^Y$,

$$\frac{dE}{d\alpha_1} = \frac{dE}{dh_1}\frac{dh_1}{d\alpha_1} = \left(\frac{dE}{d\beta_1}\frac{d\beta_1}{dh_1} + \frac{dE}{d\beta_2}\frac{d\beta_2}{dh_1}\right)\frac{dh_1}{d\alpha_1}$$

assume we know $\frac{dE}{d\beta}$ and $\frac{d\beta_i}{dh_1} = M_{1i}$
More generally:
$x \in R^x, h \in R^H, y, t \in R^Y, M \in R^{H*Y}$

$$\frac{dE}{d\alpha_i} = \frac{dh_i}{d\alpha_i}[\frac{dE}{d\beta_1}...\frac{dE}{d\beta_Y}] * [M_{i1}...M_{iY}] = \frac{dh_i}{d\alpha_i}[\frac{dE}{d\beta_1}...\frac{dE}{d\beta_Y}] * \begin{bmatrix} M_{i1} \\ ... \\ M_{iY} \end{bmatrix}$$

in matrix form

$$\delta_\alpha E = \frac{dh}{d\alpha} * (\delta_\beta E * M^T)$$

Theorem 6.1.1 (important stuff)

$$u = xW + a \qquad \nabla_x L = \nabla_u L W^T \tag{1}$$
$$h = \sigma(u) \qquad \nabla_u L = \nabla_h L \odot \sigma^{'}(u) \tag{2}$$
$$z = hM \qquad \nabla_h L = \nabla_z L * M^T \tag{3}$$
$$y = \sigma(z) \qquad \nabla_z L = \nabla_y L \odot \sigma^{'}(z) \tag{4}$$
$$L(y, t\nabla) \qquad \nabla_y L \tag{5}$$
$$\tag{6}$$

$$\nabla_{z^{(l)}} E = \sigma^{'}(z^{(l)}) \odot (\nabla_{z^{(l+1)}} E * (W^{(l)})^T)$$
$$\nabla_{W^{(l)}} E = [h^{(l)}]^T \nabla_{z^{(l+1)}} E$$

if $Y = H * W$

$$\nabla_H L = \nabla_y L * W^T$$
$$\nabla_W L = H^T * \nabla_Y L$$

7 combatting overfitting

7.1 regularization

weight decay:
adding a term to loss function that penalize for magnitude of weight

$$\tilde{E}(y, t\theta) = E(y, t, \theta) + \frac{1}{2}\lambda|\theta|_F^2$$

$$|\theta|_F = \sqrt{\sum \theta^2}$$

also change the update rule

$$\nabla_\theta \tilde{E} = \nabla_\theta E + \lambda \theta_i$$

7.2 data augumentation

add wider variety of samples

7.3 dropout

bizzare
ignore lots of (more than half) hidden nodes (some times on the last layer) and form output
activation of dropout layer is

$$d(h) = \frac{1}{1-\alpha}\{h\}_\alpha$$

$\{h\}_\alpha = 0$ with prob α, $= h$ with prob $1-\alpha$
for derivative:

$$h^{'} = d(h)$$

$$\nabla_h L = \nabla_{h'} L \frac{dh^{'}}{dh}$$

$\frac{dh^{'}}{dh} = 0$ if h was dropped or $\frac{1}{1-\alpha}$ if h not dropped

8 vanishing gradient problem

gradient gets smaller and smaller as you go deeper. when this happen, learning comes to a halt.
especially in the deep layers. This is called the vanishing gradient problem

9 stochasstic gradinet descent

choosing a batch

$$\{(t_{\delta_1, t_{\delta_1}})...\}$$

as a batch
we use estimate from this batch to update our weight, then choose subsequent batches from remaining samples
this is SGD
in each epoch, select a batch, update weight continue until finishing all batches

10 enhacing optimization

$$v \leftarrow \beta v + \nabla_W E$$

$$W \leftarrow W - \kappa v$$

11 autocoder

it learns the encoding automatically
latent representation or embedding space is vector representation of this coding layer
if we enforce $M = W^T$, we say the weights are tied

12 convolutional neural network(CNN)

Definition 12.0.1
continous domain: $f, g : R \to R$

$$(f * g)(x) = \int f(s)g(x-s)ds$$

discrete domain: $f, g \in R$

$$(f * g)_m = \sum f_n g_{m-n}$$

image f * conv kernel g = feature map / activation map
in 2D

$$(f * g)_{m,n} = \sum f_{ij} g_{m-i,n-j}$$

convolve a 32*32*3 image by a 5*5*3 filter w
for each small 5*5*3 chunk of image, we get a $w^t x + b$ (a 75-dimensional dot product)
we can use 6 filters to get 6 seperate maps.
we can use stride, so output size is $(N - F)/stride + 1$
we can pad the input with zero so ouput image is same size as orignal image (if padding = 1)

13 batch normalization

consider mini-batch of D samples $\{(x^{(1)}, t^{(1)}), (x^{(D)}, t^{(D)})\}$
and its output $\{h^{(1)}...h^{(D)}\}$
where $h^{(d)} = [h_1^{(d)}, ..., h_n^{(d)}]$
we will normalize the batch of output so its mean 0, variance 1
$\mu_i = \frac{1}{D} \sum_d h_i^{(d)}$

$$\hat{h_i}^{(d)} = \frac{h_i^{(d)} - \mu_i}{\sigma_i}$$

or

$$\hat{h_i}^{(d)} = \frac{h_i^{(d)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

14 hopefield

this is unsupervised.
a content-addressable memory (CAM) is system take part of a pattern, produce most likely match from memory

a network learns the pattern, and converges to closest pattern when shown a partial pattern
each node x_i are connected to each other
w_{ij} is connection strength from node i to node j, $w_{ij} = w_{ji}$
$x_j \in \{-1, 1\}$
say a binary string of length N,
each node want to change to -1 if $\sum x_i w_{ij} < b_j$, to 1 if $\sum x_i w_{ij} > b_j$ assume $b_j = 0$
and $w_{ij} > 0$ between nodes of same state, < 0 otherwise
to find W that works, $w_{ij} = \frac{1}{M} \sum_{s=1}^{M} x_i^{(s)} x_j^{(s)}$
$W = \frac{1}{M} \sum_s x^{(s)} (x^{(s)})^T - I$
each $x^{(s)}(x^{(s)})^T$ is a rank 1 matrix
For our network, assume W is symmetrical, $E = -\frac{1}{2} x^t W x - b^T x$, hopfield energy
gradient descent: $\frac{dE}{dW_{i,j}} = -\frac{1}{2} x_i x_j$
$\frac{dE}{db_j} = -x_j$
and
$\frac{dW}{dt} = k x_i x_j$ for $i \neq j$
$\frac{db_j}{dt} = k x_j$

15 variational autoencoders

goal: generate reasonable samples not in training set
we would like to sample the distribution p(x), the distribution of inputs
we generate samples by choosing elements from latent space
$z \sim p(z)$
then generate samples from latent representations
$p(x) = \int p_\theta(x|z) p(z) dz$
assume $p_\theta(x|z)$ gaussian, $\mu = d(z, \theta), \sigma$
then

$$-lnp_\theta(x|z) = \frac{1}{2\sigma^2} |X - d(z, \theta)|^2$$

given z, we can learn $d(z, \theta)$
we don't know how to sample $z \sim p(z)$
our generator is bad because we are choosing improbable z samples ($p(z) \approx 0$)
assume we can choose distribution of z in latent space, call it q(z)
Then $p(x) = E_{z \sim p}[p(x|z)] = \sum p(x|z) p(z) = \sum p(x|z) \frac{p(z)}{q(z)} q(z) = E_{z \sim q}[p(x|z) \frac{p(z)}{q(z)}]$
for this, negative log likelihood
$-lnp(x) = -E_q[lnp(x|z)] + KL(q(z)||p(z))$
let's choose $p(z) \sim N(0, I)$
(we are assuming latent distribution is N(0,1))
then our aim is to design q(z) so it is close to $N(0, I)$
we design our encoder, ask its output to be N(0,I)
MNIST digit encoder $\sigma\mu$, we push to $\mu = 0$, $\sigma = I$
These Gaussians are convenient because $KL(N(\mu, \sigma)||N(0, I)) = (\sigma^2 + \mu^2 - ln\sigma^2 - 1)$
$E[ln\ p(x|\hat{x})]$ where $z = \mu(x, \theta) + \epsilon\sigma(x, \theta), \epsilon \sim N(0, I)$

and $\hat{x} = d(z, \theta)$
The algorithum:

- start with θ
- encode x by $\mu(x, \theta)$, $\sigma(x, \theta)$ using neural network
- sample z by $z = \mu + \epsilon\sigma, \epsilon \sim N(0, I)$
- calculate KL loss $\frac{1}{2}(\sigma^2 + \mu^2 - ln\sigma^2 - 1)$
- decode $\hat{x}$ using $d(\mu(x, \theta) + \epsilon\sigma(x, \theta))$
- calculate reconstruction loss $L(x, \hat{x})$

both terms of our objective function are differentiable wrt θ

16 population coding

goal: to learn how to systematically encode, decode, transform data stored in activity of a population of neurons
we add a hardware abstraction layer between data and neural network

- choose a bunch of x, input=target=$x^{(p)}$
- learn decoding weights, $y^{(p)} = h(x^{(p)})d$

we want

$$d^* = argmin_d |Hd - x|_2^2$$

and can be computed with $d^* = (H^T H)^{-1} H^T x$
if H is singular, we do $U \sum V^T = H, H^T H$ will be nonsingular
$d = v\sigma^{-1} u^T x$
for gradient descent:

$$L = \frac{1}{2p} \sum (h^{(p)} d - t^{(p)})$$

$$\nabla_d L = \frac{1}{P} \sum (h^{(p)})^T (h^{(p)} d - t^{(p)}) = \frac{1}{p} H^T (Hd - T)$$

17 transformation

goal: to see how to use population coding to pass data between populations of neurons
we want B to encode the same value encoded in A,

- encode x in A. $a = \sigma(xE_A + \beta_A)$
- decode $\hat{x}$ from A. $\hat{x} = aD_A$
- re-encode $\hat{x}$ into B. $b = \sigma(\hat{x}E_B + \beta_B) = \sigma(aD_A E_B + \beta_B)$
- (optional) decode $\hat{\hat{x}}$ from B. $\hat{\hat{x}} = bD_B$

the reason we connect polulations of neurons is connection can perform transformations on the data
example: given x,y, we build a network computing x*y, $D_{xy} = argmin_D|AD - XY|$
low dimensional bottle-neck
even we can form a full connection matrix $W = DE$, W is low-rank.
if $D_{xy} \in R^{N*1}, E_B \in R^{1*M}$, this is rank 1
this saves computation time
aW takes NM flops, aDE takes $N + M$ flops

18 network dynamics

goal: to see how to build dynamic, recurrent networks using polulation coding methods

$$\tau_s \frac{ds}{dt} = -s - c(input)$$

synaptic time constant

$$\tau_m \frac{dv}{dt} = -v + \sigma(s)$$

membrane time constant
at equilibrium, for current $-s + c = 0$
at equilibrium, for activity $v = \sigma(s)$
variants:
if $\tau_m << \tau_s$: $\tau_s \frac{ds}{dt} = c - s$, $v = \sigma(s)$
if $\tau_m >> \tau_s$: $s = c$, $\tau_m \frac{dv}{dt} = -v + \sigma(s)$
if steady: $s = c$, $v = \sigma(s)$

19 recurrent networks

We want to build running memory into our network behaviour
soluation 1 we can design network so the entire sequence is put in all at once
problem: only fixed length sequence, no processing can occur until whole sequence is given **soluation 2** let network be recurrent
to train such network, we have bptt(backward propogation through time), this is unrolling through time

$$h^i = \sigma(x^i U + h^{i-1} W + b)$$
$$y^i = \sigma(h^i V + c)$$

loss function is

$$E(y^1, ...y^t, t^1, ..., t^t) = \sum L(y^i, t^i)\alpha_i$$

α_i is different weights on the outputs

$$\nabla_{z^k} E = \nabla_{z^k} L(y^k, t^k) = \nabla_{y^k} L(y^k, t^k) \otimes \sigma^{'}(z^i)$$

variables before k do not depend on variables after k
$\nabla_{h^t} E = \nabla_{h^k} \sum_{i=k}^{\tau} L(y^i, t^i) = \nabla_{h^k} E^k$
thus $\nabla_{h^t} E = \nabla_{z^t} E^\tau \frac{dz^t}{dh^t} = \nabla_{z^t} E^T V^t$
suppose we computed $\nabla_{h^{i+1}} E$
$\nabla_{h^i E} = \nabla_{h^i} L(y^i, t^i) + \nabla_{h_i} E^{i+1} = \nabla_{y^i} L(y^i, t^i) \otimes \sigma^{'}(z^i) V^T + (\nabla_{h^{i+1}} E \otimes \sigma^{'}(s^{i+1})) W^T$

20 long short-term memory

BSTT become instable from vanishing gradient $\frac{dE}{dh^i}$ will be multipled by connection weights and pass through derivative of activation function every step
forget gate $\sigma([h_{t-1}x_t]W_i + b_i)$
input $\tilde{c_t} = tanh([h_{t-1}x_t]W_c + b_c)$
$c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c_t}$

21 adversarial attack

it's easy to fool a classifier network into misclassfying an input
define ϵ ball (neighbourhood) of an input x as $B(x, \epsilon) = \{x' \in X || x' - x| \leq \epsilon\}$
we ask if there $x' \in B(x, \epsilon)$, argmax$(y_i) \neq t$, $y = f(x')$
this happen quite easily, called adversarial attack
we have black box attack and white box attack
blackbox attack estimate gradient by giving input and see its output
whitebox attack calculate $\nabla_x E = \nabla_z E(W^0)^T$
there is targetted attack and non-targetted attack
for untargetted

$$x = x + \kappa \nabla_x E(f, t)$$

for targetted

$$x = x - \kappa \nabla_x E(f, l) \text{ for } l \neq t(x)$$

FGSM:(fast gradient sign method) $x' = x + 0.01 sign(\nabla_x C(f(x), t))$

22 adversarial defense

- add adversarial traning: add samples
- trades (tradedoff-inspired adversarial defense via surrogate-loss minimization)

consider a network $f : x \to R$ and dataset (X,T) where T is -1 or 1
classification is correct if $f(x)T > 0$
classification loss can be written $R_{net}(f) = E[I\{f(x)T \leq 0\}]$
we add a term:
$R_{rob}(f) = E[I\{\exists x' \in B(x, \epsilon) | f(x')T \leq 0\}]$
we use a most curved function
$min_f E[\phi(f(x)T) + max_{x' \in B(x,\epsilon)} \phi(f(x)f(x'))]$
this tries to ensure x is ocrrectly classified and adds a penalty for models f that put x within ϵ of decision boundary
implementation:
for each gradient-descent step:

- run several steps of gradient ascent to find x'
- evaluate joint loss
- use gradinet of loss for gradent step

23 generative adversatial network(GAN)

real image → discriminator network
noise vector → generator network → fake images → discriminator network
loss of GAN has 2 parts:
$E(\theta, \phi) = E_{R,F}[L(D(x;\theta), t)]$, x is real or fake input, target is 1 if real, 0 if fake (better at detecting fake images)
$E_z[L(D(G(z;\phi)), 1)]$, 1 because target for fake input (better at making fake image)

24 Vector embeddings

$2 = [0, 0, 1, 0, 0...] \in \{0, 1\}^{10}$
for word cat $\sim v \in W$
$W = \{0, 1\}^{N_v}$
what if different words have similar meaning
scale of happiness: content, happy, elated, esctatic
predicting word pairs:
we get a lot of information from simple fact some words occur together
our network: input is one-hot word vector, output is probablity of each word's co-occurrence
our network performs: $y = f(v, \theta), v \in W$ and $y \in P^{N_v} = \{p \in R^{N_v} | \text{p is probablity vector, sums to 1}\}$
y_i equals probablity that v is nearby
the hidden-layer queezing forces a compressed representation, requiring similar words take on similar representations, this is called an **embedding**
word2vec uses additional tricks to speed up learning

- treats common phrases as one word like 'new york'
- randomly ignores common words (the, a)
- negative sampling: backprop only some of the negative cases, ignoring most zeros

similar inputs are mapped to similar locations. we see this in autoencoder
cosine angle is often used to measure distance between 2 vectors in the embedding space
king+man-queen=women

25 predictive coding

real brain limitation:

- synaptic updates can only be based on local information
- connection weights cannot be copied to other connections

some network implement backprop in biologically plausible way
predictive coding
in predictive coding, predictions or commands are sent one way through network, and errors/deviations are sent the other way
PC network there is forward prediction, and backward errors.

each node is split into 2 parts (error ϵ and value/state x).
input $\leftarrow \epsilon^{(1)} \rightleftarrows x^{(1)} \rightleftarrows \epsilon^{(2)} \rightleftarrows x^{(2)} \leftarrow Y$
μ^i is prediction being sent up to layer i
$\mu^i = \sigma(x^{i-1})M^{i-1} + \beta^i$
error node ϵ^i is defference between x^i and μ^i

$$\tau \frac{d\epsilon^i}{dt} = x^i - \mu^i - v^i \epsilon^i$$

at equilibrium: we get $\epsilon^i = \frac{x^i - \mu^i}{v^i}$, v^i is a constant
Goal for traning the PC network: given dataset $(X, Y), \theta = \{M^i, W^i\}$
$max_\theta p(y(x), \theta)$
bayesian chain:
$p(y(x), \theta) = p(Y(x)|\theta)p(\theta) = p(Y|\mu^n)p(x^{n-1}|\mu^{n-1}...p(\theta))$
consider $p(x^i|\mu^i)$, assume $x^i \sim N(\mu^i, v^i)$ normally distributed
$-lnp(x^i|\mu^i) = \frac{1}{2v^i}|x^i - \mu^i|^2$
$-lnp(Y(x), \theta) = \sum \frac{|x^i - \mu^i|^2}{2v^i}$
hopfield function: $F = \sum v^i |\epsilon^i|^2/2$
$\mu^i = \sigma(x^{i-1})\mu^{i-1}$
now we show the network activity acts to decrease hopefield energy
$\nabla x^l F$ note that x^l appears in ϵ^l and ϵ^{l+1}
$\epsilon^l = \frac{1}{v}(x^l - \mu^l)$
$\epsilon^{l+1} = \frac{1}{v^{l+1}}(x^{l+1} - \mu^{l+1}) = \frac{1}{v^{l+1}}(x^{l+1} - \sigma(x^l)M^l)$
$\nabla_{x^l} F = \epsilon^l - \sigma'(x^l) \otimes [\epsilon^{l+1}(M^l)^T]$
and gradient descent gives us

$$\tau \frac{dx^l}{dt} = \sigma'(x^l)\epsilon^{l+1}W^l - \epsilon^l$$

training: you clamp the input on both ends
$\beta = 0$ hold x and y and run network to equilibrium

$$\tau \frac{dx^l}{dt} = \sigma'(x^l)\epsilon^{l+1}W^l - \epsilon^l = 0$$

we get $\epsilon^l = \sigma'(x^l) \otimes [\epsilon^{l+1}(M^l)^T]$
link to backprop
start with top gradient Y
$\nabla_{\mu^n} F = v^n/2\nabla_{\mu^n}|\epsilon^n|^2 = -\epsilon^n$
but we also derived that, at equilibrium, $\epsilon^l = \sigma'(x^l) \otimes (\epsilon^{l+1}(\mu^l)^T)$
updating weights: consider $\nabla_{\mu^l} F = -\sigma(x^l) \otimes \epsilon^{l+1}$
likewise $\nabla_{W^l} F = -\epsilon(x^{l+1}) \otimes \sigma'(x^l)$
$\frac{dM^l}{dt} = \sigma(x^l) \otimes \epsilon^{l+1}$ $\frac{dW^l}{dt} = \epsilon^{l+1} \otimes \sigma'(x^l)$
the whole system is 4 differential equations
testing to run it, clamp input x and run the network till equilibrium, once at equilibrium, x^n is the output