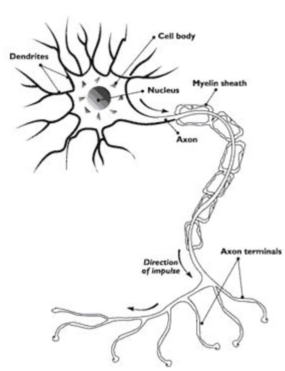
Deep Nets

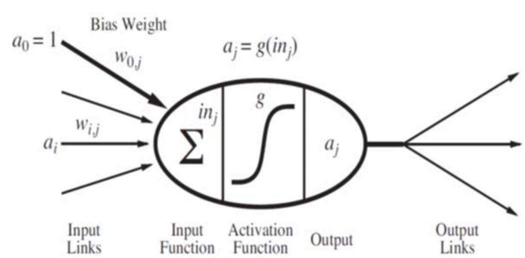
Professor Marie Roch

SAN DIEGO STATE UNIVERSITY

Unit – Basic building block



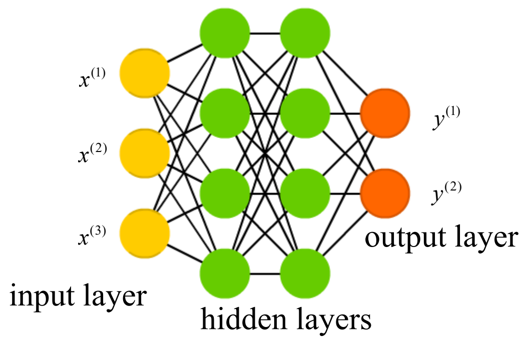
Neuron
National Institute on Drug Abuse



Loosely inspired model of a neuron
Fig. 18.19 Russell and Norvig

SAN DIEGO STATE UNIVERSITY

Feed forward network



$$f : \mathbb{R}^3 \to \mathbb{R}^2$$

Example

- Learn exclusive or: $x^{(1)} \oplus x^{(2)} = y$
- No need to generalize, only 4 possible inputs
- Select neuron model:

$$f(x|w, b) = x^T w + b$$

- and loss/cost function

$$\text{MSE } J(\theta) = \frac{1}{4} \sum_x (f^*(x) - f(x|w, b))^2$$

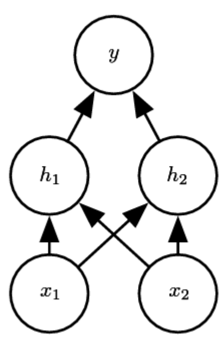
Learning xor

- We have a linear model and can use regression to fit it.
- Regression model:
  - w=0
  - b=.5
  - Predicts line y=.5 ☹

xor, take 2

heroesand.com/heroes-and-take-two

- Linear function won't work
- Try a feed forward network

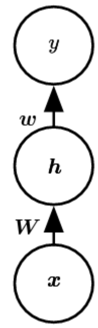


Goodfellow et al. Fig. 6.2

xor

- Alternate representation of network
- Each node is a vector

Goodfellow et al. Fig. 6.2



$$y = f^{(2)}(h \mid w, b)$$
$$= f^{(2)}(f^{(1)}(x \mid W, c) \mid w, b)$$

$$h = f^{(1)}(x \mid W, c)$$

Unit definition

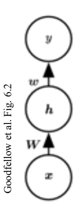
- Linear function, $w^T x$
- Nonlinear transformation of linear function Default is rectified linear unit (ReLU):

$$g(z) = \max(0, z)$$

- Using the compact vector notation, our fn is:

$$f(x \mid W, c, w, b) = w^T \max(0, W^T x + c\} + b$$

Goodfellow et al. Fig. 6.2



$$y = f^{(2)}(h \mid w, b)$$
$$= f^{(2)}(f^{(1)}(x \mid W, c) \mid w, b)$$

$$h = f^{(1)}(x \mid W, c)$$

Solution to xor

$$f(x \mid W,c,w,b) = w^T \max(0, W^T x + c\} + b$$

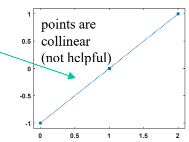
$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

We'll talk about how to get this solution later…

Data matrix $\begin{bmatrix} 0 & 0 & \vert & 0 \\ 0 & 1 & \vert & 1 \\ 1 & 0 & \vert & 1 \\ 1 & 1 & \vert & 0 \end{bmatrix}$ with columns X | Y

$$W^T x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} x \text{ applied row by row} \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

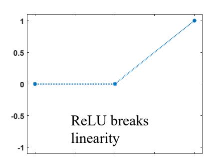
$$\text{adding bias } c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \text{ yields} \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



Solution to xor

$$f(x \mid W,c,w,b) = w^T \max(0, W^T x + c) + b$$

$$\max\left(0, \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}\right) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



$$w^T \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} + 0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

which is what we wanted: $\begin{bmatrix} 0 & 0 & \vert & 0 \\ 0 & 1 & \vert & 1 \\ 1 & 0 & \vert & 1 \\ 1 & 1 & \vert & 0 \end{bmatrix}$ with columns X | Y

Remember $w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$

Optimization

The linear problems we have seen are examples of *convex optimization*.

In theory, convex optimization always converges.

Nonlinear functions in neural nets usually lead to non-convex optimization functions.

Gradient-based learning

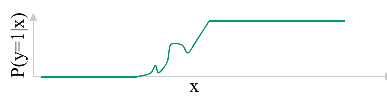
- Sensitive to initial parameters, typically use
  - small random values for weights
  - biases zero or small +values
- Convex loss functions can also be trained with gradient descent; useful for large data sets.

Gradient-based learning

- Be careful of saturation
  - Sometimes, loss functions locally flat.
  - Problematic for gradient descent

Example for Bernoulli (binary) output:

$$P(y=1\mid x)=\max\left(0,\min(1,w^T h+b)\right)$$



Learning conditional distributions

- Cross entropy cost function as previously seen:

$$J(\theta)=E_{x,y\sim\hat{p}_{data}}\left[-\log P(y\mid x,\theta)\right]$$

- Form depends on distribution. Our normal distribution model reduced to

$$J(\theta)=E_{x,y\sim\hat{p}_{data}}\left[-\log P(y\mid x,\theta)\right]=\frac{1}{m}\sum_{i=1}^{m}-\log P(y\mid x,\theta)+k$$

where k was a constant that could be ignored.

Learning conditional statistics

- Sometimes we want a statistic, e.g.
  - $E[Y \mid X = x]$
  - $E[(Y - \bar{y})^2 \mid X = x]$
- Cost function lets learning select a function

SAN DIEGO STATE UNIVERSITY

15

Learning conditional statistics

- Details require variational calculus
- Example fns
  - conditional mean

$$f^* = \arg\min_f E_{x,y \; p_{data}}[\|y - f(x)\|^2]$$

mean squared error

  - conditional median

$$f^* = \arg\min_f E_{x,y \; p_{data}}[\|y - f(x)\|_1]$$

mean absolute error

SAN DIEGO STATE UNIVERSITY

16

Note on cost functions

- Avoid cost functions that saturate
- When saturated (flat), gradient is small
- Common to use negative log likelihoods
- Examples of saturating functions
  - e^{-k} where k is large
  - Mean squared error
  - Mean absolute error

SAN DIEGO STATE UNIVERSITY

17

Output units

What type of outputs?

- Conditional probabilities
- Two class – output 0|1
- Multiple class – N outputs, one high

Most of what we discuss can also be applied to hidden units

SAN DIEGO STATE UNIVERSITY

18

Conditional outputs

- Produce mean of conditional normal

$$P(y \mid x) = n(y \mid \hat{y}, I)$$

mean targets
observed training y

variance can be learned too
be careful, must be + definite

- $\hat{y}$ produced as *linear* output, does not saturate

$$\hat{y} = W^T h + \mathrm{b}$$

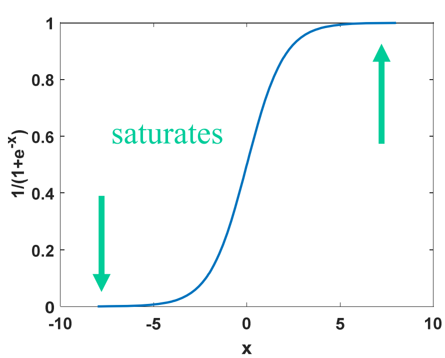
- Can be fed into normal pdf if $P(y|x)$ desired

SAN DIEGO STATE UNIVERSITY

Bernoulli output

- Predicts P(y=1|x)
- Common to use sigmoid activation function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



SAN DIEGO STATE UNIVERSITY

Bernoulli output

- Straight forward:

$$\hat{y} = \sigma(w^T h + b) \text{ or } z = w^T h + b, \hat{y} = \sigma(z)$$

- Problem:

We want to model P(y=1|x)

SAN DIEGO STATE UNIVERSITY

21

Bernoulli output

- Simplifying assumption:

$$\log \tilde{P}(y) = yz$$

implies: $\log \tilde{P}(y) = \begin{cases} z & y = 1 \\ 0 & y = 0 \end{cases}$

- Without log,

$$\tilde{P}(y) = \begin{cases} e^z & y = 1 \\ e^0 & y = 0 \end{cases}$$

Does this look like a probability distribution?

SAN DIEGO STATE UNIVERSITY

22

Statistician's bag of tricks

Solution:

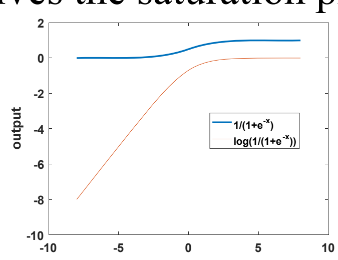
- Normalize by P(y=0|x) + P(y=1|x)

$$P(y) = \begin{cases} \dfrac{e^z}{e^0 + e^z} = \dfrac{e^z}{1+e^z} = \dfrac{1}{\dfrac{1+e^z}{e^z}} = \dfrac{1}{1+e^{-z}} & y = 1 \\[2ex] \dfrac{e^0}{e^0 + e^z} = \dfrac{1}{1+e^z} & y = 0 \end{cases}$$

- Can be simplified: $P(y) = \sigma((2y-1)z)$

Bernoulli output

- With MLE approach, we optimize –log P(y|x). Solves the saturation problem



- Sigmoid activation less common with non MLE approaches

Bernoulli output

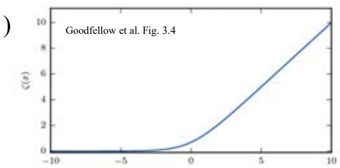
- Loss function with sigmoid

$$
\begin{aligned}
J(\theta) &= -\log P(y \mid x) \\
&= -\log \sigma((2y-1)z)
\end{aligned}
$$

- Sometimes written in terms of softplus

$\zeta(x) = \log(1+e^{x})$ Goodfellow et al. eq. (3.31)

$$
\begin{aligned}
J(\theta) &= -\log\left(\sigma\left((2y-1)z\right)\right) \\
&= -\log\left(\frac{1}{1+e^{-(2y-1)z}}\right) = -\log\left(1+e^{-(2y-1)z}\right)^{-1} \\
&= \log\left(1+e^{(1-2y)z}\right) = \zeta((1-2y)z)
\end{aligned}
$$



“gentler” version of max(0,x)

SAN DIEGO STATE UNIVERSITY

25

Categorical outputs

- Classification output, e.g. digit label
- Output node represents each category
- Can model as a multinoulli distribution

SAN DIEGO STATE UNIVERSITY

26

Multinoulli distributions

- Appropriate for k categories
- Parameter vector p
  - $\text{length}(p) = k-1, 1^T p \le 1$
  - $P(x=i) = \begin{cases} p_i & 0 \le i < k \\ 1-1^T p & i = k \end{cases}$
- Rarely worry about mean & variance as these do not mean much for categories

Categorical outputs

- Output layer is vector $\hat{y}$

$$\hat{y}_i = P(y=i \mid x)$$

- Requirements

$$\hat{y}_i \in [0,1]$$
$$1^T y = 1$$

- Use a similar approach as we used for Bernoulli distributions

Categorical outputs

- Input function as usual: $z = W^T h + b$
- Activation function is softmax:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Following the MLE approach, use log-likelihood to avoid saturation

$$\log \text{softmax}(z_i) = z_i - \log\left(\sum_j e^{z_j}\right)$$

SAN DIEGO STATE UNIVERSITY

Categorical outputs

- Softmax
  - When difference in z_i's large, drives towards a "one hot" solution
  - Numerical stability improved if we normalize:

$$\text{softmax}(z - \max_i z_i)$$

equivalent as

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} = \frac{e^c e^{z_i}}{e^c \sum_j e^{z_j}} = \frac{e^{z_i + c}}{\sum_j e^{z_j + c}} = \text{softmax}(z_i + c)$$

SAN DIEGO STATE UNIVERSITY

Categorical outputs

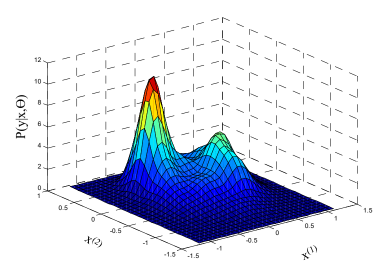
- Input layer to softmax can use identity function for activation and feed outputs into the softmax layer.

Other output types exist, but we will not cover them in detail for now.

Other output types

- Learning distributional parameters

- Learning multimodal distributions



Hidden units

- Output layer activation functions:
  - linear
  - sigmoid
  - softmax

  can be used in hidden units
- Sigmoid functions are commonly used, but there are other possibilities

San Diego State University

33

Hidden units

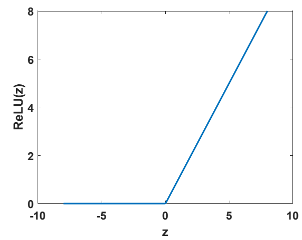
- Rare to have good theoretical justification for activation function type
- A number of the activation functions have near-linear properties, making optimization easier.

San Diego State University

34

Rectified linear units

- $g(z) = \max(0, z)$
- Effectively shuts off unit for negative inputs



- Saturation could be a problem…

  in practice, rarely so due to local minima
- Not differentiable at 0, implementations usually pick the left or right derivative

Rectified linear units

- Variants exist

  $$g(z_i, \alpha_i) = \max(0, z_i) + \min \alpha_i(0, z_i)$$

- Differ mainly in how $\alpha_i$is set.
- Example

  Absolute value rectification $\alpha_i = 1$

  Used to compensate for reverse polarization illumination

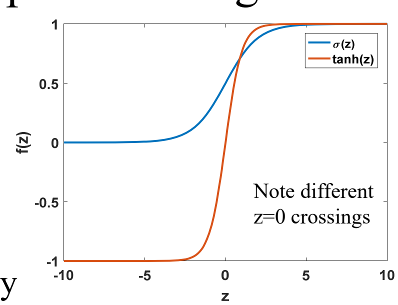
Maxout units

- Separate z's into *k* groups
- Propagates maximum z of each group to next layer

$$g_i(z) = \max_{j \in group(i)} z_j$$

- Learns piecewise linear function
- Reduces number of parameters to next layer
- Can help protect network against forgetting things it has learned

Logistic sigmoid & hyperbolic tangent



- Remember: sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

- tanh is similar
  - tanh(0)=0 more like identity
  - Relation: $\tanh(z) = 2\sigma(2z) - 1$

- Saturation a problem, avoid using in hidden layers if at all possible!

Other activation functions

- Goodfellow et al. discuss a couple other units, some of which are discouraged
- Many other activation functions will work
- So many so, that finding a new one isn't a big deal unless it fundamentally changes what you can solve.

Architecture

- Organized into layers
- Depth vs width
  - Deeper networks can have fewer parameters
  - Wide networks easier to optimize

Universal approximation

A network with at least

- One hidden layer *and*
- A nonlinear activation (e.g. σ, ReLU)

can model any continuous f on a bounded subset of a finite dimensional space

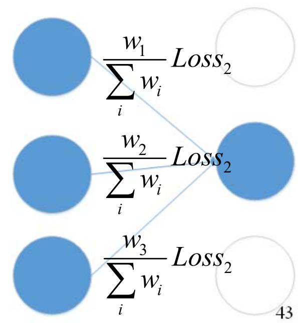
It does not state that we will find the network that models the function, merely that it exists.

Backpropagation

- Method for computing gradients in a computation network.
- Usually interested in gradients of the cost function with respect to model parameters.
- Only part of the training algorithm… Learning algorithms exploit the gradient to optimize the model.

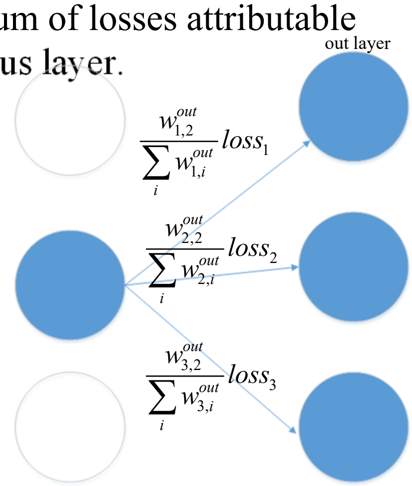
Backpropagation overview

- After applying the update to the output layer, there still exists loss
- We assign a portion of the loss to each of the input nodes based on their weight.
- This contribution is computed for each node of the current layer



Backpropagation overview

- Now we can look at the sum of losses attributable to each node in the previous layer.
- The sum of these provides us with a loss to minimize.
- Repeat recursively



$w^l_{j,k}$ layer l weight from node k to j

A more formal view

Computation represented by a simple graph language

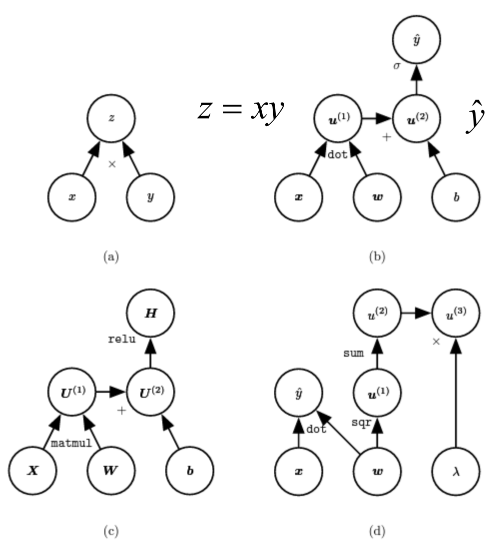
- Nodes: Variables (e.g. scalar, tensor)
- Directed edges: Simple functions that transform variables.
  - Tie edges together for multiple operands
  - Annotated with operation name unless obvious

SAN DIEGO STATE UNIVERSITY

45

Sample graphs

$z = xy$ $\quad \hat{y} = \sigma(w^T x + b)$



Goodfellow et al. Fig. 6.8

$H = relu(XW + B)$

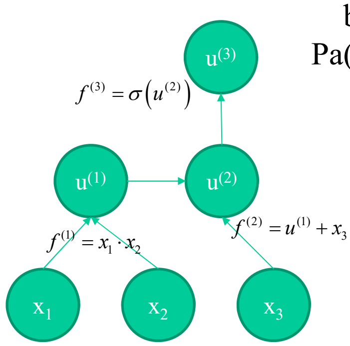
$\hat{y} = x^T w,$

$u^{(3)} = \lambda \sum_i w_i^2$

SAN DIEGO STATE UNIVERSITY

46

Forward graph & computation



G

Graph G arranged so that $u^{(i)}$ computed before $u^{(j)}$ when i<j.

Pa(node) = predecessor indices

```
for i = 1,2,…,n_i
  u^(i) = x_i
for i = n_(i+1),…,n
  𝔸(i)←{u^(j) | j ∈Pa(u^(i))}
  u^(i)←f^(i)(𝔸(i))
return u^(n)
```

Goodfellow et al. Alg. 6.1

Calculus chain rule

Animal House
©1978 Universal Pictures

Recall the chain rule

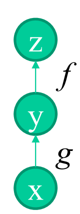
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

generalize to the following functions

$$g:\mathbb{R}^m \to \mathbb{R}^n, f:\mathbb{R}^m \to \mathbb{R}^1$$

and let $y = g(x)$, $z = f(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

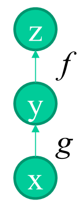


$z \in \mathbb{R}^1$

$y \in \mathbb{R}^n$

$x \in \mathbb{R}^m$

Chain rule



$z \in \mathbb{R}^1$

$y \in \mathbb{R}^n$

$x \in \mathbb{R}^m$

Remember $x \in \mathbb{R}^m$ $and\, y \in \mathbb{R}^n$

Let's put this in matrix notation

Construct a Jacobian matrix

$$\left[\frac{\partial y}{\partial x}\right] = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_M} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & & \frac{\partial y_2}{\partial x_M} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} & & \frac{\partial y_1}{\partial x_M} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_N}{\partial x_1} & \frac{\partial y_N}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_M} \end{bmatrix}$$

SAN DIEGO STATE UNIVERSITY

49

Chain rule

and

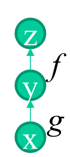
$$\nabla_y z = \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \frac{\partial z}{\partial y_2} \\ \vdots \\ \frac{\partial z}{\partial y_N} \end{bmatrix}$$

then

$$\nabla_x z = \left[\frac{\partial y}{\partial x}\right]^T \nabla_y z = \underbrace{\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_M} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & & \frac{\partial y_2}{\partial x_M} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} & & \frac{\partial y_1}{\partial x_M} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_N}{\partial x_1} & \frac{\partial y_N}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_M} \end{bmatrix}^T}_{(N\times M)^T} \underbrace{\begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \frac{\partial z}{\partial y_2} \\ \vdots \\ \frac{\partial z}{\partial y_n} \end{bmatrix}}_{N\times 1}$$

Jacobian
gradient

We can apply this to tensors by extending the dimensions



$z \in \mathbb{R}^1$

$y \in \mathbb{R}^n$

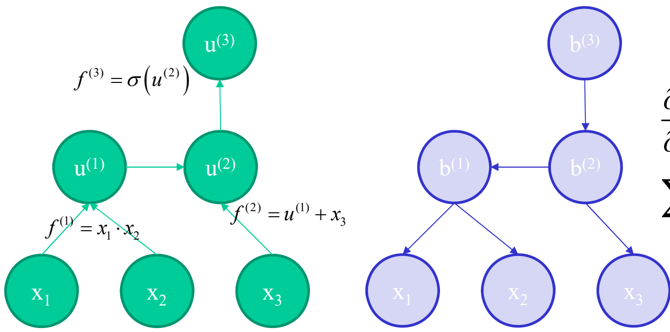
$x \in \mathbb{R}^m$

SAN DIEGO STATE UNIVERSITY

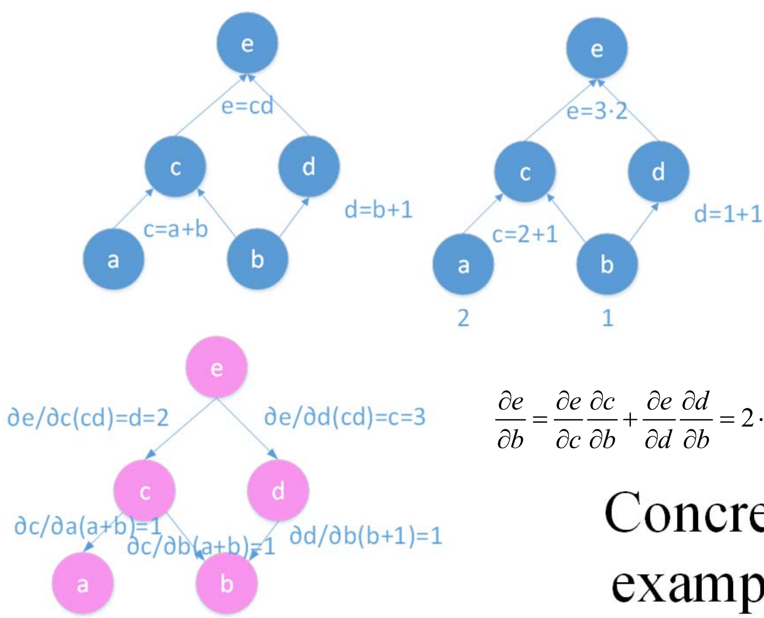
50

Backpropagation

- Enhance computation graph G with new subgraph B
- One-to-one correspondence with $u^{(i)}$ nodes



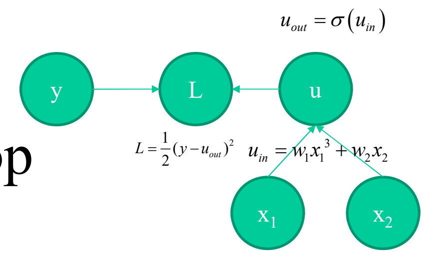
$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j\in Pa\left(u^{(i)}\right)} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$$



$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 2\cdot 1 + 3\cdot 1 = 5$$

Concrete example

Example based on Christopher Olah's blog post



Activation fn example for backprop

partial derivatives

$$\frac{\partial L}{\partial u_{out}} = \frac{2}{2}(y - u_{out})(-1) = u_{out} - y$$

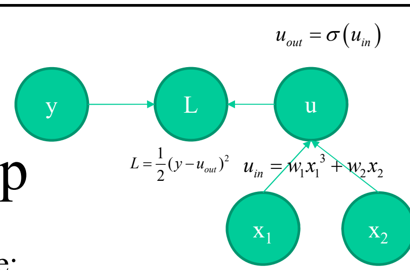
$$\frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in}))$$

$$\frac{\partial u_{in}}{\partial u_{x_1}} = w_1 3 x_1^2$$

$$\frac{\partial u_{in}}{\partial u_{x_2}} = w_2$$

$$\frac{\partial u_{in}}{\partial u_{w_1}} = x_1^3$$

$$\frac{\partial u_{in}}{\partial u_{w_2}} = x_2$$



Activation fn example for backprop

To update w_1 we use the chain rule:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = (u_{out} - y)\sigma(u_{in})(1 - \sigma(u_{in}))x_1^3$$

from previous slide

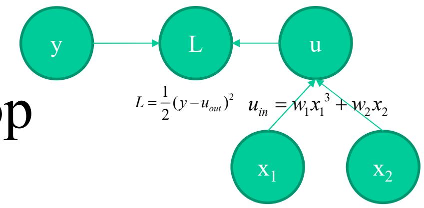
$$\frac{\partial L}{\partial u_{out}} = u_{out} - y$$

$$\frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in}))$$

$$\frac{\partial u_{in}}{\partial u_{w_1}} = x_1^3$$

$u_{out} = \sigma(u_{in})$

Activation fn example for backprop



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = (u_{out} - y)\sigma(u_{in})(1 - \sigma(u_{in}))x_1^3$$

Concrete example

$$y = 0, w = \begin{bmatrix} .02 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

implies

$$u_{in} = w_1 x^3 + w_2 x_2 = .02 \cdot 3^3 + .01 \cdot 5 = .59$$

$$u_{out} = \frac{1}{1 + e^{-u_{in}}} = \frac{1}{1 + e^{-.23}} = .6434$$

$$L = \frac{1}{2}(y - u_{out})^2 = .5 \cdot (0 - .6434)^2 = .2070$$

$$\frac{\partial L}{\partial u_{out}} = y - u_{out} = .6434 - 0 = .6434$$

$$\frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in})) = \sigma(.59)(1 - \sigma(.59))$$

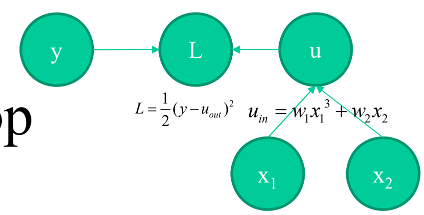
$$= \frac{1}{1 + e^{-.59}}\left(1 - \frac{1}{1 + e^{-.59}}\right) = .6434(1 - .6434) = .2294$$

$$\frac{\partial u_{in}}{\partial u_{w_1}} = x_1^3 = 3^3 = 27$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = .6434 \cdot .2294 \cdot 27 = 3.9851$$

$u_{out} = \sigma(u_{in})$

Activation fn example for backprop



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = (u_{out} - y)\sigma(u_{in})(1 - \sigma(u_{in}))x_1^3$$

Suppose we have a learning rate ε=*.01*

$$w_1 = w_1 - \epsilon \frac{\partial L}{\partial w_1} = .02 - .01 \cdot 3.9851 = -0.0599$$

Update of w_2 is left as an exercise, but loss with only w_1 changed:

$$y = 0, w = \begin{bmatrix} -.06 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

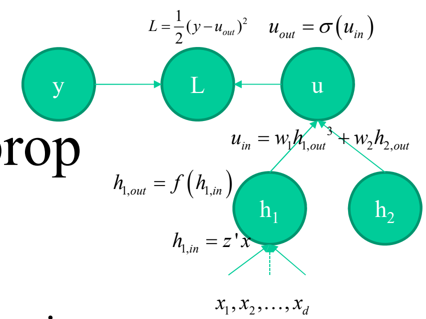
implies

$$u_{in} = w_1 x^3 + w_2 x_2 = -.06 \cdot 3^3 + .01 \cdot 5 = -1.57$$

$$u_{out} = \frac{1}{1 + e^{-u_{in}}} = \frac{1}{1 + e^{-(-1.57)}} = .1722$$

$$L = \frac{1}{2}(y - u_{out})^2 = .5 \cdot (0 - .1722)^2 = .0148 \text{ old L=.2070}$$

Activation fn example for backprop

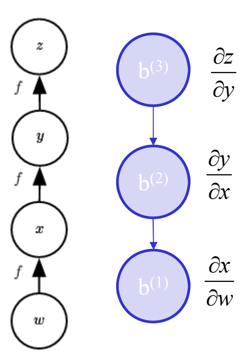


Example did now show previous layers, but they can be handled in the same way. Suppose we wanted to adapt weight vector z leading into h_1:

$$\frac{\partial L}{\partial z_i}=\frac{\partial L}{\partial u_{out}}\frac{\partial u_{out}}{\partial u_{in}}\frac{\partial u_{in}}{\partial u_{h_{1,out}}}\frac{\partial u_{h_{1,out}}}{\partial u_{h_{1,in}}}\frac{\partial u_{h_{1,in}}}{\partial u_{h_{1,z_i}}}$$

Implementing derivative computation

Simple example



$$\frac{\partial z}{\partial w}=\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

Two possible implementations memory vs time trade-off

$$\frac{\partial z}{\partial w}=f'(y)f'(x)f'(w)$$ (better) or

$$\frac{\partial z}{\partial w}=f'(f(f(w)))f'(f(w))f'(w)$$

Simplified backprop w/scalars

$\nabla[\mathrm{u}^{(\mathrm{n})}] = 1$

`for j = n-1,…,2,1`

$$\nabla[u^{(j)}] = \sum_{i:j\in Pa(u^{(i)})} \nabla\,[u^{(i)}]\,\frac{\partial u^{(i)}}{\partial u^{(j)}}$$

`return gradient table` ∇

We use activations from minibatch to compute concrete values.

Recall $\dfrac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j\in Pa(u^{(i)})} \dfrac{\partial u^{(n)}}{\partial u^{(i)}} \dfrac{\partial u^{(i)}}{\partial u^{(j)}}$

$\nabla[u^{(i)}] = \dfrac{\partial u^{(n)}}{\partial u^{(i)}}$

San Diego State University

59