CS224N: Stanford

Chris Manning:

Lecture1: Introduction and Word Vectors

→ An understanding of the effective modern methods for deep learning
Basics first, then key methods used in NLP:
Recurrent networks, attention, etc.

→ A big picture understanding of human languages and the difficulties in understanding and producing them.

→ An understanding of an ability to build systems for some of the the major problems in NLP
→ word meaning, dependency parsing, machine translation, question answering

'Language isn't a formal system. It's a glorious chaos.

Humans are better because of the language and human communication network

Manning vs Lecun (Humans are better)

Writing takes knowledge spatially throughout the word and temporally throught time

12

1) How do you represent the ~~word~~ meaning of a word?

Definition: meaning

→ "idea that is represented by a word, phrase, etc.

→ idea that a person wants to express by using words, signs, etc

→ idea that is expressed in a work of writing, art, etc.

Linguistic way of thinking of meaning:

signifier (symbol) ⇔ signified (idea of thing)

= denotational semantics

Common solutions to have usable meaning in computers

Use WordNet (nltk), a thesaurus containing lists of synonym sets and hypernyms

But that's not ~~g~~ terribly good ~~at~~ anything

Problem: can't adapt

Representing words as discrete symbols

Traditional NLP (NLP till 2012): regards words as discrete symbols → localist rep.
were represented as one-hot vectors
⇒ but words are infinite.
one possible solution to find the synonyms can be word similarity table

Distributional semantics: A word's meaning is given by the words that frequently appear close-by
⇒ when a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window)

Word vectors: We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

Word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

Word meaning as a neural word vector

(Yoshua Bengio's work)

Word2Vec: (Mikolov et. al. 2013)

Idea:

i) We have a large corpus of text

ii) Every word in a fixed vocabulary is represented by a vector

iii) Go through each position t in the text, which has a center word c and context ("outside") words o

iv) Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)

v) Keep adjusting the word vectors to maximise this probability $P(w_{t+j} / w_t)$

objection function (cost or loss)

For each position $t = 1, \dots, T$ predict context words within a window of fixed size m, give center word w_j

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m,\ j \neq 0} P(w_{t+j} / w_t; \theta)$$

θ: all variables to be optimized

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = \ominus \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m,\ j \neq 0} \log P(w_{t+j} | w_t; \theta)$$

$\ominus$: minimization; $\frac{1}{T}$: average

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

How to calculate $P(w_{t+L} | w_t, \theta)$

Answers: Two vector representations

- v_w when w is center word
- u_w when w is a context word

center c, context o

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$u_o^T v_c$ → compare similarity of o and c

$\sum_{w \in V}$ → normalize over entire vocab

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

The softmax function maps arbitrary values x_i to a probability distribution p_i

$\Rightarrow$ amplifies probability of largest x_i while still assigning some to smaller x

Training a model by optimizing parameters

$$\frac{\delta}{\delta v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{n} \exp(u_w^T v_c)}$$

$$= \frac{\delta}{\delta v_c} \log \exp(u_o^T v_c) - \frac{\delta}{\delta v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)$$

$$= \frac{\delta}{\delta v_c} u_o^T v_c$$

$$= u_o$$

negative word gensim visualization. There's some ambiguity with names.

$$\frac{d}{dv_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)$$

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T c)} \cdot \frac{\delta}{\delta v_c} \sum_{x=1}^{V} \frac{d}{dv_c} \exp(u_x^T v_c)$$

$$\sum_{x=1}^{V} \exp(u_x^T v_c) \frac{d}{dv_c} u_x^T v_c$$

$$\sum_{x=1}^{V} \exp(u_x^T v_c) u_x$$

$$\frac{\delta}{\delta v_0} \log P(O|c) = u_0 - \frac{\sum_{x=1}^{V} \exp(u_x^T v_c) u_x}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

$$= u_0 - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v)} u_x$$

actual context word → expected context word

$$= u_0 - \sum_{x=1}^{V} P(x|c) u_x$$

u_0: observed representation of context word

u_x: representation of each word x

$P(x|c)$: probability of it in current model

u → outside
v → center

$V \cdot V_4^T$

softmax($V \cdot V_4^T$) probs

gensim (word similarity packet) | LDA | negative similarity

Lecture 2 (Word vectors & Word Senses) 14/10/2020 (2)

The word vectors are represented as rows

We want a model that gives a reasonably high probability estimate to all words that occur in the context (that's why we are using softmax)

Optimization: Gradient Descent:

minimize $J(\theta)$

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

Stochastic Gradient Descent

Since $J(\theta)$ is very expensive to compute here

→ very bad idea for pretty much all nns

Solution: SGD update after each sample

Stochastic gradient with word vecs

We might only update the word vectors that actually appear.

Solution: Either we need ⊛ sparse matrix update operations to only update certain rows of full embedding matrices U and V, or we need to keep a hash of word vectors

- If you have millions of words and do distributed computing, it is important to not have to send gigantic updates around.

Why two vectors?

Easier optimization. Average both at the end

Two model variants:

1) Skip Gram (SG) => predict context words given center word
2) Continous Bag of ⊛Words⊛ (CBOW) => predict center word from (bag of) context words

⊛Naive Softmax

Skip-gram with Negsampling: Normalization factor is too computationally expensive

Idea of Negative sampling:

Train binary logistic regression for a p true pair (center word and word in its context) versus several noise pairs (center word paired with a random word) → then try to give them as lower prob as possible

Paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov 2013)

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$$

$$J_t(\theta) = \log\sigma(u_o^T v_c) + \sum_{i=1}^{k} E_{j\sim P(w)}\left[\log\sigma(-u_j^T v_c)\right]$$

→ We maximize the probability of two words co-occuring in the first log

$$J_{neg\text{-}sample}(o, v_c, U) = -\log\left(\sigma(u_o^T v_c)\right) - \sum_{k=1}^{K}\log\left(\sigma(-u_k^T v_c)\right)$$

We take k negative samples (using word probs)

- Maximize probability that real outside word appears and minimize prob that random words appear around center word

$$P(w) = \frac{U(w)^{3/4}}{Z}$$

($U(w)$: unigram dist; Z: normalization term)

makes less frequent word to be sampled more often

But why not capture co-occurence counts directly?

With a co-occurence matrix X

- 2 options: windows vs. full doc
- Window: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis"

Example: window base co-occurance matrix

→ Window length 1 (more common 5-10)

→ Symmetric (irrelevant whether left or right context)

Problems with simple co-occurance vecs

→ Increase the size with vocab

→ very high dimensional: requires a lot of storage

→ subsequent classificatim model have sparsity issues

→ models are less robust

Solution: Low dimensional vectors

Store "most" of the important infos in a fixed small number of dimensions: a dense vector

Usually 25-100 dimensions, similar to word2Vec

Reducing dimensionality

Method 1: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurence matrix X.

Factorizes X to $U\Sigma V^T$ where U and V are orthogonal

If we ~~or~~ take k values then, we ~~don~~ ignore the least frequent context vectors

$\hat{X}$ is the best rank approx to X, in terms of least squares

Problems: Expensive to compute for large matrices.

[Play around with it in homework]

Hacks to X (Rohde 2005)

Scaling the counts in the cells help a lot.

- Problems function word (the, he, has) are too frequent → syntax has too much impact.
  - → Some fixes $\min(X, t)$ $t \approx 100$
  - → ignore all

- Ramped windows that count closer words more
- Use pearson correlation instead of counts, then set negatives to 0

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Count-based vs direct prediction

Count-based

- LSA, HAL (Lund & Burgess)
- COALS, Hellinger-PCA (Rohde, Lebret & Collobert)

Pros
- Fast training
- Efficient usages of statistics

Cons
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

direct prediction

- Skip-gram/CBOW (Mikolov)
- NNLM, HLBL, RNN (Bengio, Huang, Mnih)

Cons
- Scales with corpus size
- Inefficient usage of stats

Pros
- Generate improve performance
- Can capture complex patterns beyond word similarity

Encoding meaning in vector differences [EMNLP2014]

(Pennington) [Combining the best of both worlds]

Crucial insight: Ratios of co-occurence probabilities can encode meaning components

Capturing ratios of co-occurence probs as linear meaning

$$w_i w_j = \log P(i/j)$$

$$w_x (w_a - w_b) = \log \frac{P(x/a)}{P(x/b)}$$

$$J = \sum_{i,j=1}^{V} f(x_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$ [GloVe]

- Fast training
- Scalable huge corpora
- Good performance even with small corpus and small vectors

Evaluate Word Vectors:

General Evaluation in NLP: Intrinsic vs Extrinsic

Intrinsic:

- Evaluation of a specific/intermediate subtask
- Fast to compute
- Helps to understand that system
- Not clear if really helpful unless correlation to real task is established.

Extrinsic:

- Evaluation on a real task
- Can take long time to compute accuracy
- Unclear if the subsystem is the problem or its interaction ~~is established~~ or other subsystems
- If replacing exactly one subsystem with another improves accuracy → Winning

Intrinsic word vector evaluation:

- Word vector analogies

  a:b :: c:?

  man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactical analogy questions.
- Discarding the input words from the search!
- Problem: what if the info is there but not linear.

For Glove

- Good dimension is ~300
- Asymmetric context (are not as good
- This might be different for downstream tasks
- Window size of 8 around each center word is good for GloVe vectors

Recent work: On dimensionality of Word Embedding [Neurs IPS 2018]

- Wikipedia is better than news text

- Word vector distances and their correlation with human judgment.

Example Wordsim353 (check out)

Cosine distance for similarity judging

Word senses and word sense ambiguity

-> Most words have lots of meanings.

- Does one vector capture all these meanings?

Improving Word Rep via Global Context (Huang et al 2012)

Idea: Cluster word windows around words, retain with each word assigned to multiple different clusters $bank_1$, $bank_2$

Linear Algebraic Structure of Word Senses with Applications to Polysemy (Arora, TACL 2018)

- Different senses of a word reside in a linear superposition (weighted sum). In standard word embeddings like word2vec

$$V_{pike} = \alpha_1 V_{pike_1} + \alpha_2 V_{pike_2} + \alpha_3 V_{pike_3}$$

$$\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3} \qquad f = \text{frequency}$$

- Because of ideas from sparse coding we can actually separate out the senses.

Extrinsic Word Vector Evaluation:

All subsequent tasks in this class

→ Good Word vectors would help in named entity recognition:

Assignment: Exploring Word Vectors

- Word Vectors and Word Embeddings are often used interchangably. The term "embedding" refers to the fact that we are encoding aspects of a word's meaning in a lower dimensional space

Problems with thesaurus-based meaning:

-> They have problems with recall
-> Many words are missing
-> Most phrases are missing
-> Some connection between senses are missing.
-> Doesn't work well for verbs and adjectives

(Reuters)
↑
business and financial news corpus

Singular value Decomposition:

PCA

Lecture 3 - Word Window Classification, Neural Nets

Classification setup:

$$\{x_i, y_i\}_{i=1}^{N}$$

→ x_i are inputs, e.g. words, sentences, documents (Dimension d)

→ y_i are labels (one of C classes) we try to predict, for example: → classes: sentiment, named entities, buy/sell

→ other words

→ multi word sentences

Traditional ML / Stats approach:

Assume x_i are fixed, train softmax/logistic regression weights $W \in \mathbb{R}^{C \times d}$ to determine a decision boundary ~~as in the picture~~

$$P(y/x) = \frac{\exp(W_{y \cdot} x)}{\sum_{i=1}^{C} \exp(W_{c \cdot} x)}$$

Softmax in two steps

1) find all f_c for $c = 1 \ldots C$

$$W_{y \cdot} x = \sum_{i=1}^{d} W_{y_i} x_i = f_y$$

2) $P(y/x) = \frac{\exp(f_c)}{\sum_{c=1}^{C} \exp(f_c)} = \text{softmax}(f_y)$

→ Our objective is to maximize the probability of the correct ~~call~~ class y.

→ Or we can minimize the negative log probability of that class

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

* NLL loss (Negative log-likelihood loss)

→ Cross entropy loss

True probability distribution p

computed probability distribution q

Cross entropy $H(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$

- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else
- Because of one-hot p, the only term left is the negative log prob of the true class.

Classification over a full dataset:

- Cross entropy loss function over full dataset

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

$$f_y = f_y(x) = W_y x = \sum_{j=1}^{d} W_{yj} x_j$$

$$f = Wx$$

Traditional ML Optimizations

$$\theta = \begin{bmatrix} w_{.1} \\ \vdots \\ w_{.d} \end{bmatrix} = W(:) \in \mathbb{R}^{Cd}$$

Sow we onld update the decision boundar

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla w_{.1} \\ \vdots \\ \nabla w_{.d} \end{bmatrix} \in R^{Cd}$$

Neural Networks Classifiers (Non-linearity)

- Can learn more complex functions and non-linear decision boundaries.

Classification difference with word vectors:

- Commonly in NLP deep learning:

→ We learn both W and word vectors x

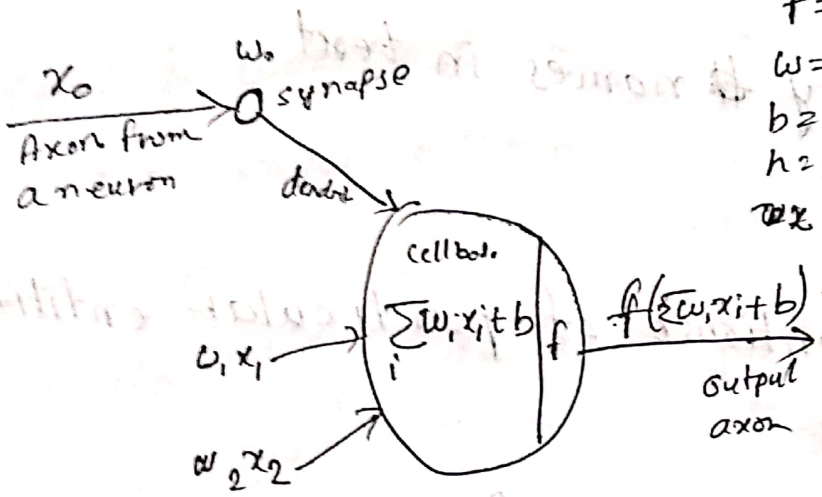
→ We learn both conventional params and representation

→ The word vectors re-represent one-hot vectors - move them around in an intermediate layer vector space - for easy classification with a (linear) softmax classifier via layer x = Le

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla w_{.1} \\ \vdots \\ \nabla w_d \\ \nabla x_{aardvark} \\ \vdots \\ \nabla x_{zebra} \end{bmatrix} \in \mathbb{R}^{C_d + V_d}$$

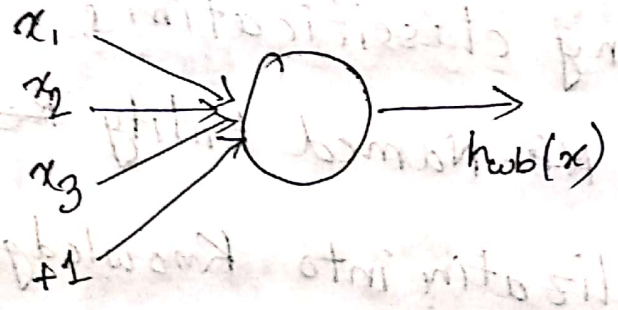
↑ very large

An artificial neuron



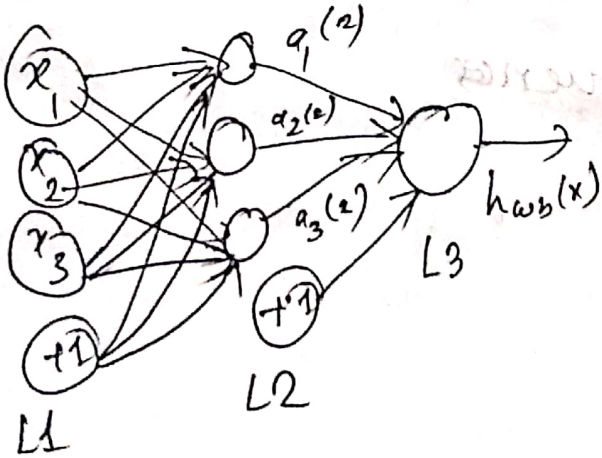
f = nonlinear activation
w = weights
b = bias
h = hidden
x = input

$$f(z) = \frac{1}{1+e^{-z}} \qquad h_{w,b}(x) = f(w^T x + b)$$



A neuron can be a binary logistic regression unit

A neural net = running several logistic regressions at the same time



It is the loss function that will ~~decide~~ direct what the intermediate hidden variables should be

Named Entity Recognition (NER):

- Find and classify names in text

Possible purposes:

- Tracking mentions of particular entities in document
- For question answering. Answers are usually named entities
- The same technique can be extended to other slot-filling classifications.

→ often followed by Named Entity Linking / Canonicalization into Knowledge Base

→ We predict Entities by classifying words in context and then extracting entities as word subsequences

Why is it hard?

- Hard to work out boundaries of entity
- Hard to know if something is an entity
- Hard to know class of unknown/novel entity
- ~~Entide~~ Entity class is ambiguous and depends on context.

Binary word Window classification

- In general, classifying single words is rarely done
- Interesting problems like ambiguity arise in context.

Example: Auto antonyms

- "To sanction" can mean 'to permit' or 'to punish'
- "To seed" can mean 'to place seeds' or 'to remove seeds'

Example: resolving of ambiguous named entities

- Paris → Paris, France vs. Paris Hinton vs. Paris, Texas
- Hathaway → Berkshire Hathaway vs Anne Hathaway

Window classification:

- Idea: classify a word in its context window of neighboring words.

- For example, Named Entity Classification of a word in context

- A simple way to classify a word in context might be to average the word vectors in a window and to classify the average vector
  - Problem: that would lose positional info

Softmax idea:

Train softmax classifier to classify a center word by taking concatenation of word vectors surrounding it in a window.

Resulting vector $x_{window} = x \in \mathbb{R}^{wd}$

w = windowsize * 2 + 1

Simplest window classifier

- with $x = x_{window}$ we can use the same softmax classifier as before

predicted model output probability $\hat{y}_y = p(y|x) = \frac{\exp(w_y x)}{\sum_{c=1}^{C} \exp(w_c x)}$

$f_{y_i} = w_y x$

- with cross entropy error as before

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

To update the word vectors, we need to calculate grads like we did previously

Binary classification with unnormalized scores

Collobert & Weston (2008, 2011) ICML 2018 Test of time

- Assume we want to classify whether the center word is a Location
- Similar to Word2Vec, we will go over all positions in a corpus. But this time, it will be supervised and only some positions should get a high score.

The positions that have an actual NER location in their centers are "true" positions and get a high score.

NN Feed forward Computation:

$$score(x) = U^T a \in \mathbb{R}$$

Compute a window's score with a 3-layer neural net:

$$s = score(\text{"museums in Paris are amazing"})$$

$$s = U^T f(Wx + b)$$

$$x \in \mathbb{R}^{20\times 1}, \quad W = \mathbb{R}^{8\times 20}, \quad U \in \mathbb{R}^{8\times 2}$$

Main intuition befor extra layer:

The middle layer learns non-linear interactions between input word vectors.

Alternative: Max-margin loss (no Softmax!)

- Idea for training objective: Make true window's score larger and corrapt window's score lower (until they are good enough)

Minimize ~~max~~ $J = \max(0, 1 - s + s_c)$

→ This is not differentiable but continuous

→ we can use sgd.

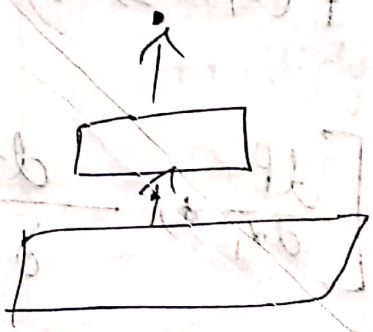
$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

Simple net for score

$s = u^T h$

$h = f(Wx + b)$

x = input



Let's find out $\frac{\delta s}{\delta b}$

- In practice we care about gradient of the loss.

Computing Gradients by Hand

- Review of multivariable derivatives
- Matrix calculus: fully vectorized gradients
  - Much faster and more useful than non-vectorized gradients
  - But doing a non-vectorized gradient can be good practice

Gradients:

$$f(x) = f(x_1, x_2, x_3)$$

$$\frac{\delta f}{\delta x} = \left[\frac{df}{dx_1} \; \frac{df}{\delta x_2} \; \cdots \; \frac{\delta f}{dx_n} \right]$$

Jacobian Matrix: Generalization of the Gradient

Given a function with m outputs and n inputs

$$f(x) = \left[f_1(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n) \right]$$

It's Jacobian is an $m \times n$ matrix of partial derivatives

$$\frac{df}{dx} = \begin{bmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_1}{dx_n} \\ \vdots & & \vdots \\ \frac{df_m}{dx_1} & \cdots & \frac{df_m}{dx_n} \end{bmatrix} \qquad \left(\frac{df}{dx}\right)_{ij} = \frac{df_i}{dx_j}$$

Chain Rule

single variable

multiply derivative

$$z = 3y$$

$$y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = (3)(2x)$$

$$= 6x$$

multivariable

multiply Jacobian

$$h = f(x)$$

$$z = Wx + b$$

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

Jacobian: Elementwise activation function

$h = f(z)$ what is $\frac{dh}{dz}$? $\quad h, z \in \mathbb{R}^n$

$h_j = f(z_i)$

$$\left(\frac{dh}{dz}\right)_{ij} = \frac{dh_i}{dz_j} = \frac{\delta}{\delta z_j} f(z_i)$$

$$= \begin{cases} f(z_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

regular 1-variable derivative

$$\frac{\delta h}{\delta z} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(f'(z))$$

$$\frac{\delta}{\delta x}(wx+b) = w$$

$$\frac{\delta}{\delta b}(wx+b) = I \quad \text{(identity matrix)}$$

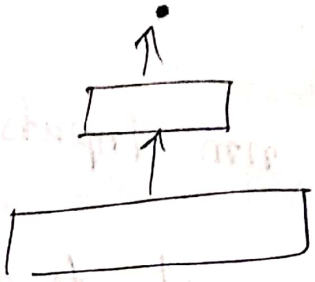
$$\frac{\delta}{\delta u}(u^T h) = h^T$$

$s = u^T h$

$h = f(Wx+b) \Rightarrow h = f(z)$

$z = Wx+b$

x (input)



$$\frac{\delta s}{\delta b} = \frac{\delta s}{\delta h} \frac{\delta h}{\delta z} \frac{\delta z}{\delta b}$$

$$= h^T \cdot \text{diag}(f'(z)) \cdot I$$

$$= h^T \circ f'(z)$$

$$\frac{\delta s}{\delta W} = \frac{\delta s}{\delta h} \frac{\delta h}{\delta z} \frac{\delta z}{\delta W} = \delta \frac{\partial z}{\partial W}$$

$$= h^T \text{diag}(f'(z))$$

$$\boxed{\delta = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} = h^T \circ f'(z)}$$

local error signal

$W \in \mathbb{R}^{n \times m}$

- 1 output nm inputs : 1 by nm Jacobian
  - Inconveniant to do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

Instead follow convention: shape of the gradient is shape of params

$\frac{\partial s}{\partial W}$ is n by m: $\begin{bmatrix} \frac{\partial s}{\partial w_{11}} & \cdots & \frac{\partial s}{\partial w_{1m}} \\ \vdots & & \vdots \\ \frac{\partial s}{\partial w_{n,1}} & \cdots & \frac{\partial s}{\partial w_{n,m}} \end{bmatrix}$

$\frac{\partial s}{\partial W} = \delta^T x^T$ [to get to appropriate shape]

δ: local error signal; x: local input signal

$\frac{\partial s}{\partial b} = h^T \circ f'(z)$ is a row vector

but according to shape convention this should be column vector since b is a column vector.

Disagreement between Jacobian form (which makes chain rule easy) and the shape convention (which makes implementing SGD easy)

- We expect answers to follow the shape convention
- But Jacobian form is useful for computing answers

So,

Two options:

1) Use Jacobian form as much as possible, reshape to follow the convention at the end.
2) Always follow the convention.
   [Look at dimensions to figure out when to transpose and/or reorder them]