NON LINEAR CLASSIFIERS

Neural Networks / Deep Learning
"Introduction"

Thamme Gowda "TG"
<tg@isi.edu> ತಮ್ಮೇಗೌಡ

Based on chapter 3 of Jacob Eisenstein's NLP book draft.

OVERVIEW

- → Logistic Regression
- → What's deep learning is offering
- → Matrix and Vectors as transformation functions
- → Feed forward Networks
- → Activation Functions
- → Back propagation

Computation Graph

LOGISTIC REGRESSION

→ Discriminative ; Probabilistic

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y')\right)}.$$

For each y — same θ — different featurizer

$\theta \in \mathbb{R}^d$, $f(x, y) \in \mathbb{R}^d$,

y is a discrete random variable

Two key components :

1. Dot product $\theta \cdot f(x, y)$
2. Softmax or Logistic function

VISUALISING DOT PRODUCT

Model computes $f(x,y).\theta \leftarrow$ parameters $\in \mathbb{R}^d$

$\uparrow$ feautures $\in \mathbb{R}^d$

say, $z = f(x,y)$

In general $z, \theta \in \mathbb{R}^d$

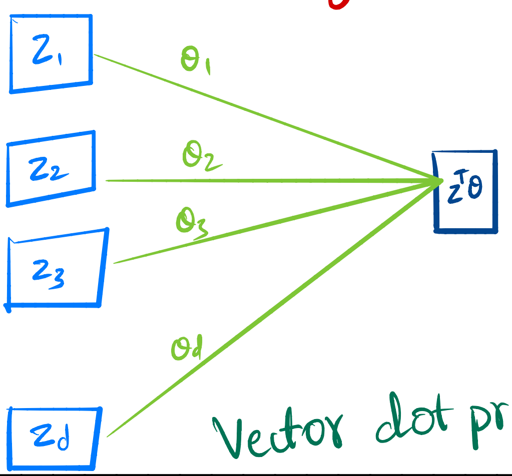
$$z.\theta = \theta.z = z^T\theta = \theta^T z = \sum_{i=1}^{d} z_i \theta_i$$

dotproduct: $\mathbb{R}^d \rightarrow \mathbb{R}$

a kind of transformation function

Weighted sum

Computation graph



Vector dot product

① Dot product is special case of matrix multiplication

$z^T\theta = 1\,\underbrace{[\,\,\,\,\,\,]}_{d}\; d$ $[1\times d][d\times 1] = [1\times 1]$

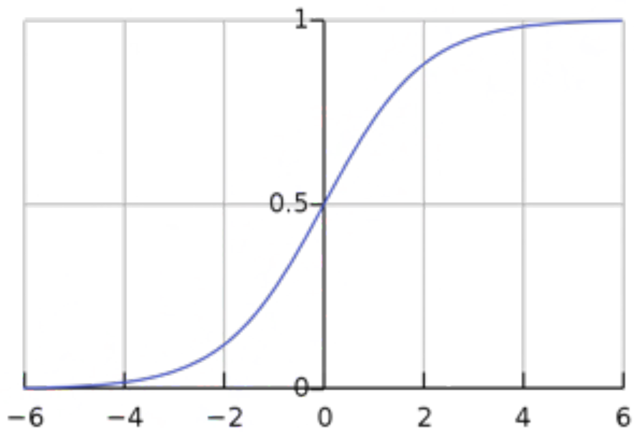
② Matrix multiplication is special case of Tensor product

$\Rightarrow$ Lets visualize Matrix multiplication

SIGMOID FUNCTION aka Std. Logistic

$\Rightarrow$ Maps $x \in \mathbb{R} \longrightarrow$ probability scale

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



Good for Binary classes

For Multiclass ?

SOFTMAX FUNCTION

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

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

SOFTMAX

```
>>> b = np.array([25, 25, 25, 25])
>>> b / b.sum()
array([0.25, 0.25, 0.25, 0.25])
>>> be = np.exp(b)
>>> be / be.sum()
array([0.25, 0.25, 0.25, 0.25])
>>>
>>> a = np.array([25, 25, 24, 26])
>>> a / a.sum()
array([0.25, 0.25, 0.24, 0.26])
>>> ae = np.exp(a)
>>> ae / ae.sum()
array([0.2 , 0.2 , 0.07, 0.53])
```

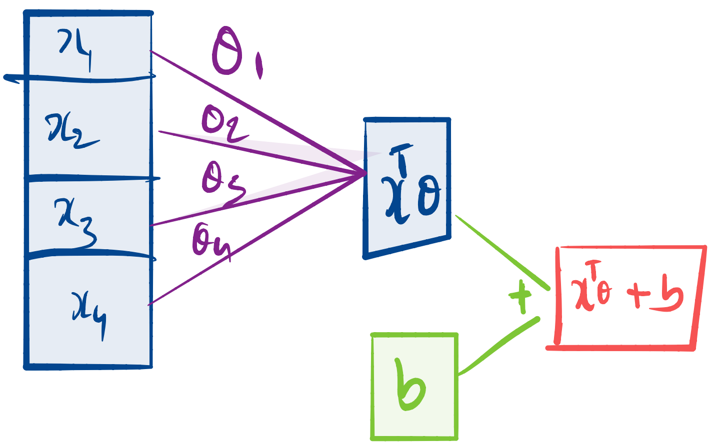
Preference Amplification

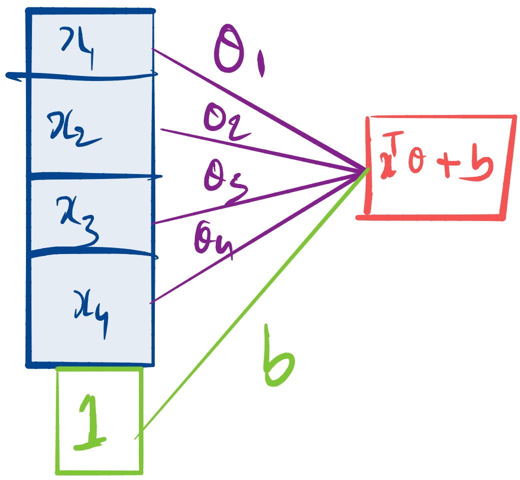
Huge punishment or reward

BIAS TERM

$$y = x^T\theta + b$$

$$y = [x; 1]^T [\theta; b]$$





$\Rightarrow$ Don't be surprised if bias term is dropped from proofs

LOGISTIC REGRESSION - Parameter Estimation

Maximum Likelihood of parameter θ on a dataset $(x^{1:N}, y^{1:N})$ $\Leftrightarrow$ IID assumption

Maximize $p(y^{1:N} | x^{1:N}; \theta) = \prod_{i=1}^{N} p(y^{(i)} | x^{(i)}; \theta)$ $\leftarrow$ Numerical instability

$\log p(y^{1:N} | x^{1:N}; \theta) = \sum_{i=1}^{N} \log p(y^{(i)} | x^{(i)}; \theta)$ $\leftarrow$ Stable

$\rightarrow$ Minimize Negative Log Likelihood $= - \sum_{i=1}^{N} \log p(y^{(i)} | x^{(i)}; \theta)$

Remember,

$$p(y^{(i)} | x^{(i)}) = \frac{e^{f(x^{(i)}, y^{(i)}) \cdot \theta}}{\sum_{y' \in Y} e^{f(x^{(i)}, y') \cdot \theta}}$$

FEATURE ENGINEERING vs POWERFUL MODELING

$$\hat{y} = \underset{y \in Y}{\operatorname{argmax}} \; p(y | x; \theta)$$

$$x \rightarrow \begin{matrix} f(x, y=y_1) \cdot \theta \\ f(x, y=y_2) \cdot \theta \\ f(x, y=y_k) \cdot \theta \end{matrix} \Rightarrow \text{Softmax} \Rightarrow \text{Argmax} \Rightarrow \hat{y}$$

aka feauture engineering

$\rightarrow$ Too much effort in building $f(x, y)$ $\forall y \in Y$

↳ Model is simple: dot product + softmax 

$\rightarrow$ good $f(x, y)$ is time consuming, requires domain expertise

$\Rightarrow$ Would you like to spend less time in feauture engineering but more time in designing models beyond dot product?

$\Rightarrow$ WELCOME TO DEEP LEARNING.

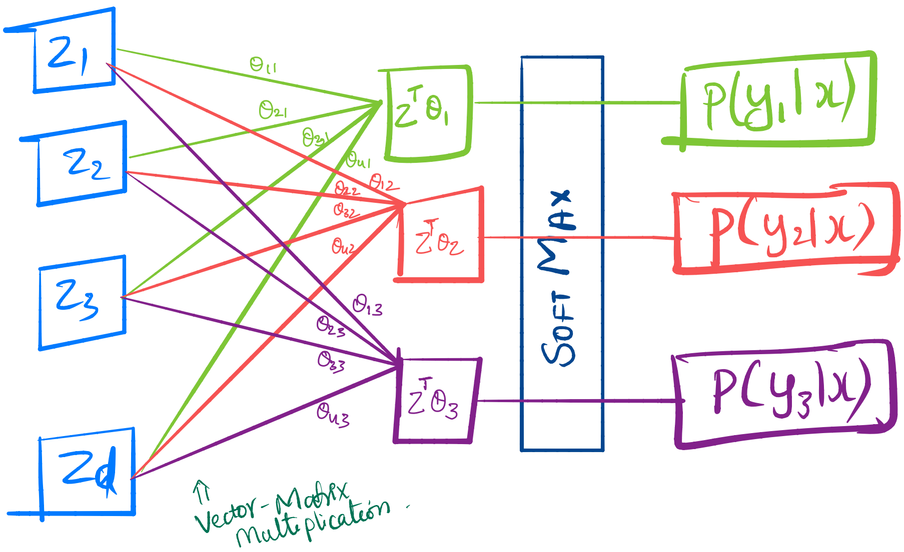
LOGISTIC REGRESSION (2.0)

→ Previous model had same θ for all $y \in Y$, — but different $z = f(x, y)$

→ Let's have same $f(x)$ but different θ for each Y.

→ Let's place all those θs in a matrix.

For feauture $\mathbb{R}^d$ and classes k, $\theta \in \mathbb{R}^{d \times k}$



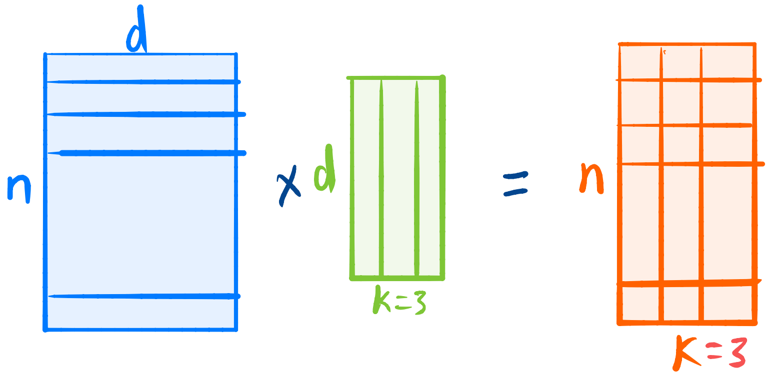
z_1	z_2	z_3	z_4

θ_{11}	θ_{12}	θ_{13}
θ_{21}	θ_{22}	θ_{23}
θ_{31}	θ_{32}	θ_{33}
θ_{41}	θ_{42}	θ_{43}

VISUALISING MATRIX MULTIPLICATION

$X \in \mathbb{R}^{n \times d}$; $W \in \mathbb{R}^{d \times k}$; $XW \in \mathbb{R}^{n \times k}$

GPUs are great for fast matrix multiplications.



n examples in minibatch

d dimension of model "hidden dimension"

k classes or hidden dimension

Look at rows in X ⇒ n examples with $\mathbb{R}^d$ representation

Look at columns in W ⇒ k $\mathbb{R}^d$ vectors for dot product.

$W \in \mathbb{R}^{d \times k}$: $\mathbb{R}^d \to \mathbb{R}^k$: Transformation operation "or function"

WHY DEEP LEARNING

→ Go beyond Linear ⟷ More power to models

→ No need for extensive feature engineering

↳ Models can learn useful features

→ Optimize end-to-end in a single network.

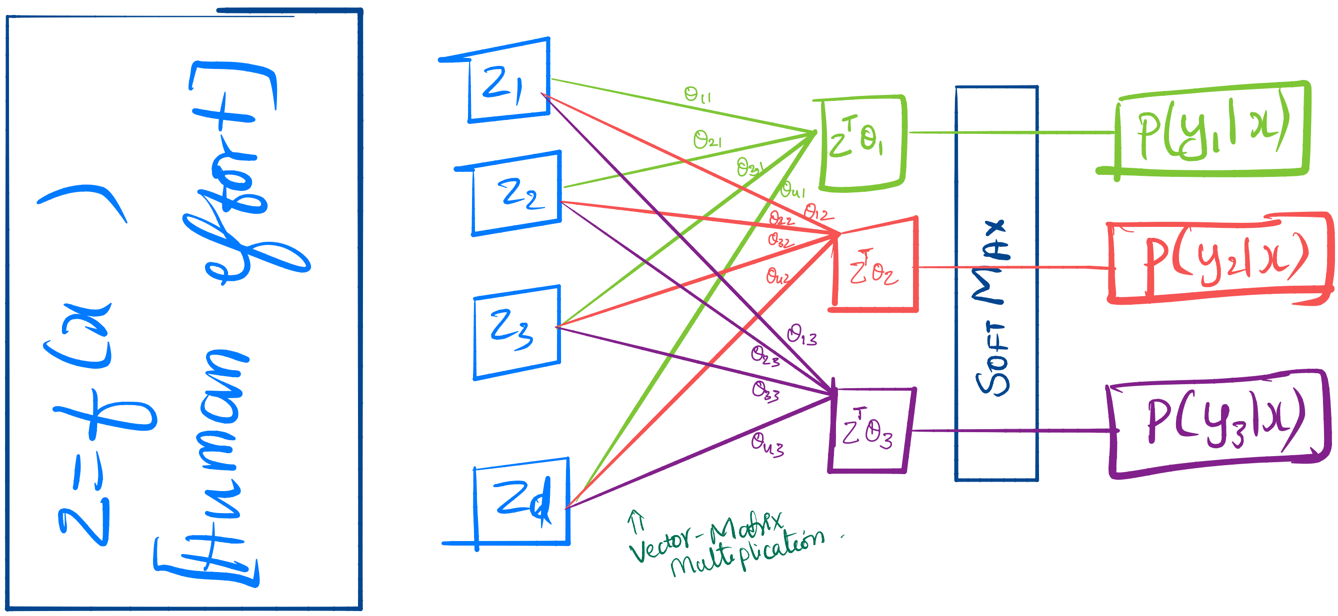
↳ NLP ~~is~~ was "pipeline" of subtasks

BUT...

* Computationally expensive → Look at GPUs and TPUs
* Lots of parameters → need lots of data → We may have it already / We may be able to get it. / That's a good research Q'
  ↳ Need efficient estimation → That's a good research Q'
* Not readily Interpretable ⇒ Hmm! 😬

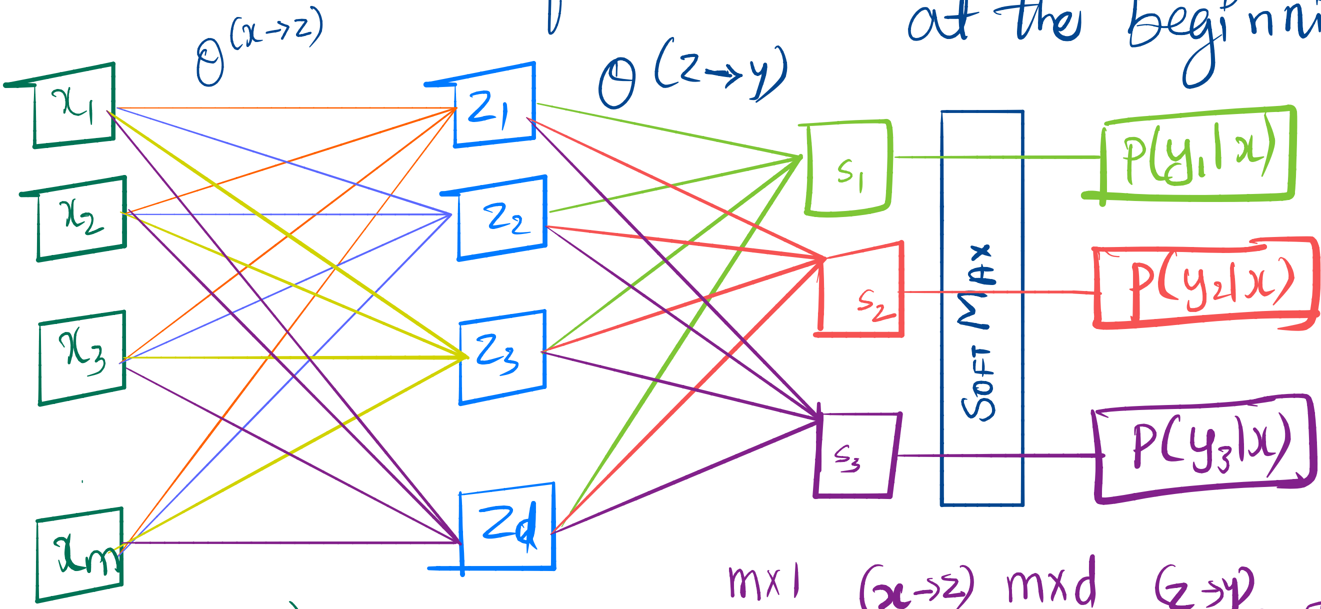
LOGISTIC REGRESSION → FEED FORWARD NN

Goal: Learn the features → Add another layer at the beginning



LOGISTIC REGRESSION → FEED FORWARD NN

Goal: Learn the features → Add another layer at the beginning



$s = (x^T \theta^{(x\to z)})\,\theta^{(z\to y)}$

$\hat{y} = \text{argmax Softmax}(s)$

$x \in \mathbb{R}^{m\times 1}$ $\quad \theta^{(x\to z)} \in \mathbb{R}^{m\times d}$ $\quad \theta^{(z\to y)} \in \mathbb{R}^{d\times k}$

What iff $W \in \mathbb{R}^{m\times k}$ such that $W = \theta^{(x\to z)}\theta^{(z\to y)}$

FEED FORWARD NETWORKS

$$S = (x^T \theta^{(x\to z)})\,\theta^{(z\to y)} = x^T(\theta^{(x\to z)}\theta^{(z\to y)}) = x^T W$$

Two or more linear transformations ⟺ One Layer

Non Linear functions → Violate the above
→ Extra power

$s = \left(A\left(x^T \theta^{(x\to z)}\right)\right)\theta^{(z\to y)}$ | $x \in \mathbb{R}^{m\times 1}$; $\theta^{(x\to z)} \in \mathbb{R}^{m\times d}$, $\theta^{(z\to y)} \in \mathbb{R}^{d\times k}$

'A' is some elementwise non-linear function

'A' is commonly called as ACTIVATION FUNCTION

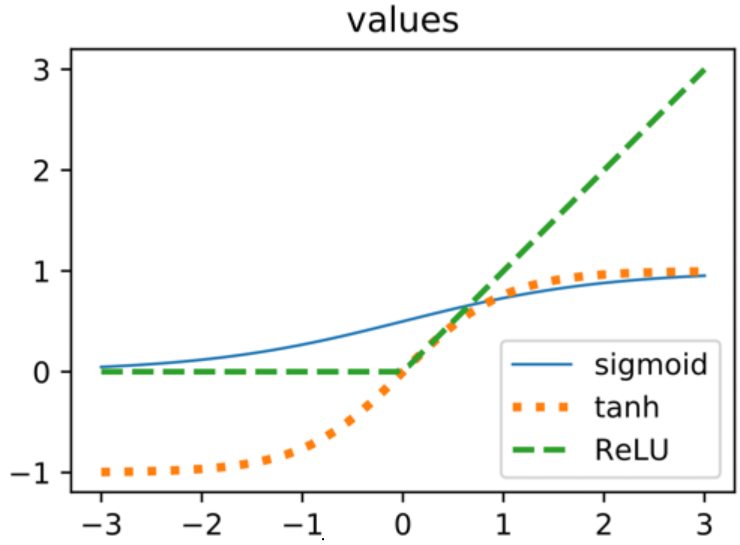
Example: Sigmoid, Tanh, ReLU, GELU ....

ACTIVATION FUNCTIONS

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

$$\operatorname{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\operatorname{ReLU}(x) = x^{+} = \max(0, x)$$

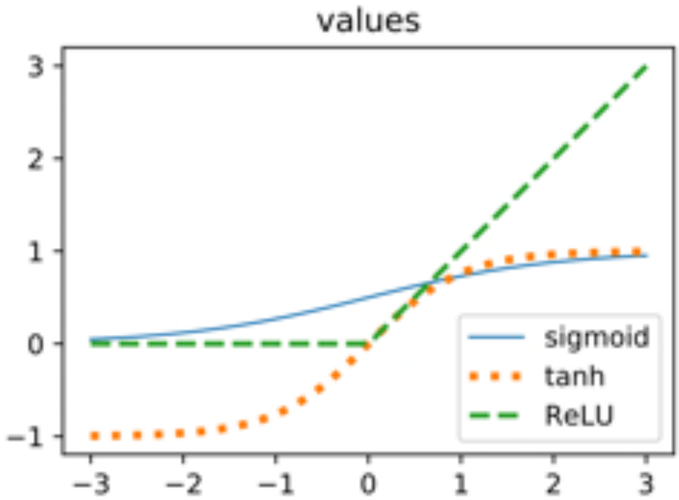


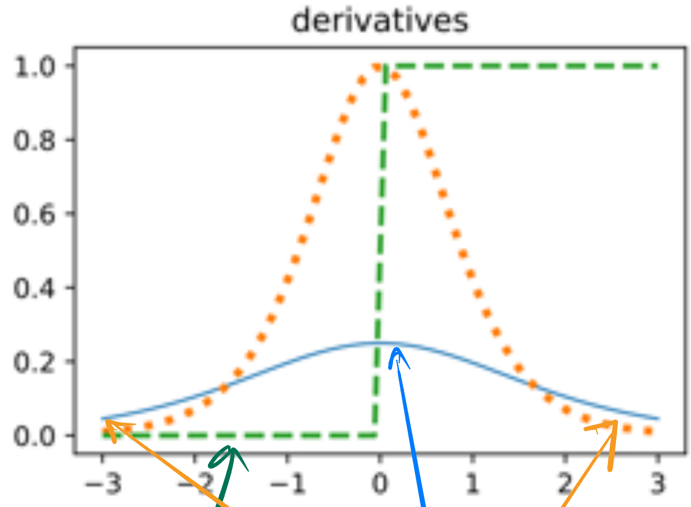
Consider these :-

→ Computational needs for calculating $f(x)$ and $f'(x)$

→ Vanishing Gradients problem

→ Dead neurons





deadzone

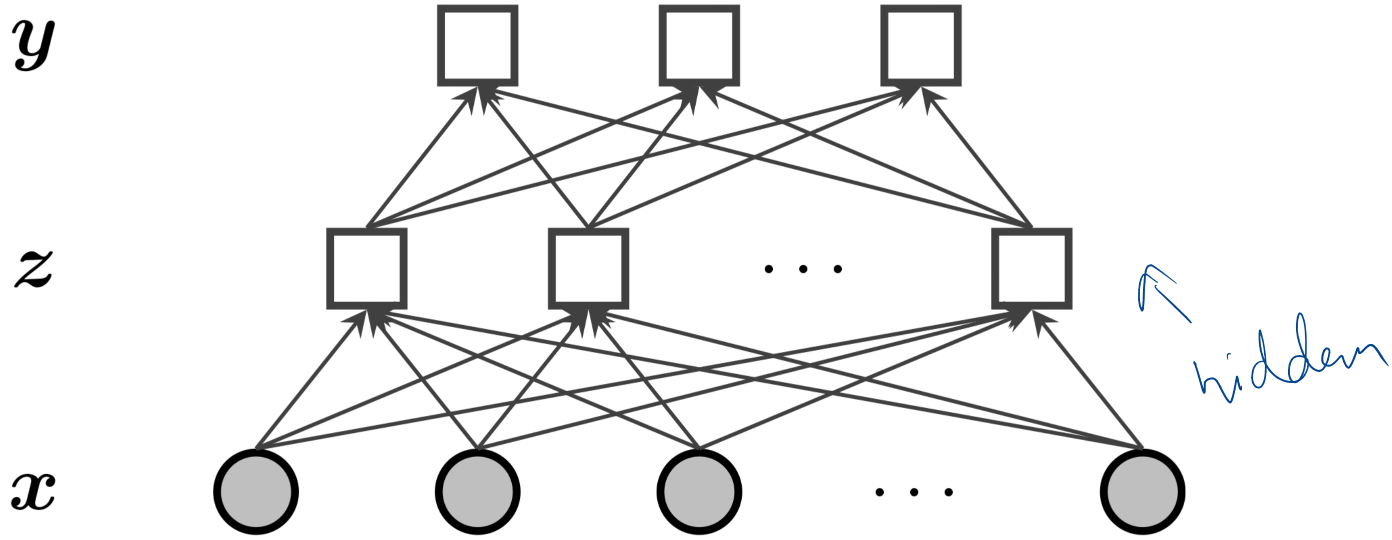
Problems with
- Sigmoid
- Tanh
- ReLU

Too small

Many multiplications yield underflow

FEED FORWARD NETWORKS

From Textbook

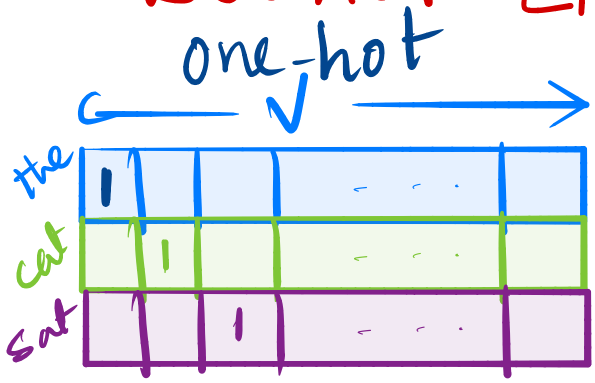


$$\mathrm{p}(\boldsymbol{z} \mid \boldsymbol{x}; \boldsymbol{\Theta}^{(x \to z)}) = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$$

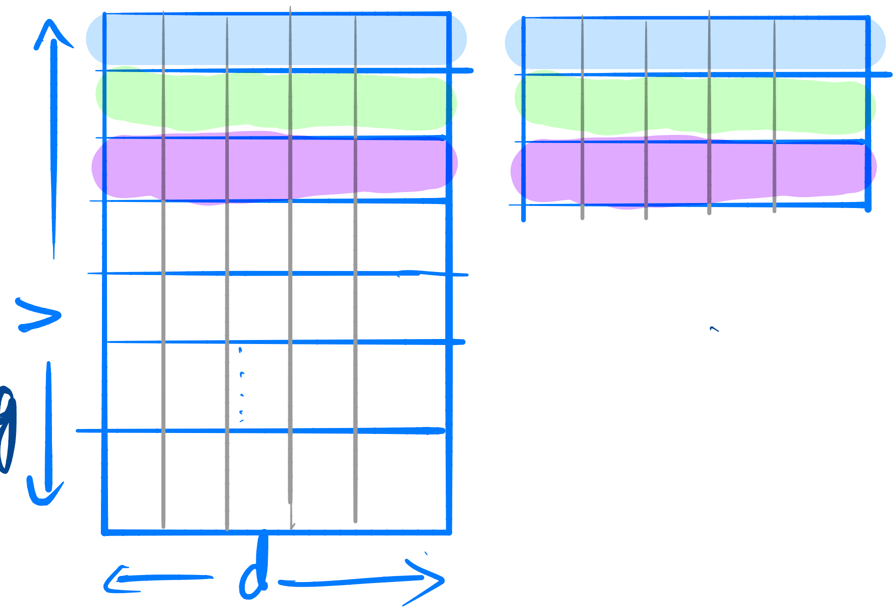
$$\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{z}; \boldsymbol{\Theta}^{(z \to y)}, \boldsymbol{b}) = \mathrm{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b}),$$

LOOKUP LAYER

WORD EMBEDDINGS



Embedding Matrix



Efficiency:

⇒ Instead of multiplying matrix of one-hot vectors, lookup vectors for words directly

PARAMETER ESTIMATION

Use an optimizer to approximate model's parameters that $\underbrace{\text{best fit}}_{\text{Loss function}}$ a dataset

Training data

Optimizer → batch, online ⇢ mini-batch

LOSS FUNCTION

Conditional Log Likelihood

$$-\mathcal{L} = -\sum_{i=1}^{N} \log \mathrm{p}(y^{(i)} \mid \boldsymbol{x}^{(i)}; \Theta).$$

close to 1

close to 0

n alternatively as follows:

$$\tilde{y}_j \triangleq \Pr(y = j \mid \boldsymbol{x}^{(i)}; \Theta)$$

$$-\mathcal{L} = -\sum_{i=1}^{N} \boldsymbol{e}_{y^{(i)}} \cdot \log \tilde{\boldsymbol{y}}$$

one-hot vector

CALCULUS REVISION [DIFFERENTIAL ONLY!]

Constant	$\frac{dc}{dx} = 0$	$\frac{d}{dx} a^x = a^x \log_e a$
power	$\frac{d}{dx} x^n = n x^{n-1}$	$\frac{d}{dx} \log_a x = \frac{1}{x \log_e a}$ $\Rightarrow$ If $a = e$, $\log_e e = 1$

Sum or Difference: $\frac{d}{dx}[f(x) \pm g(x)] = f'(x) \pm g'(x)$

Product Rule: $\frac{d}{dx}[f(x)\, g(x)] = f(x)\, g'(x) + g(x)\, f'(x)$

↳ Similarly Quotient Rule

Chain Rule: $\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$

Back propagation is an efficient algorithm to compute derivatives of such complex functions.

PARTIAL DERIVATIVES

$f(x)$ ← function of single variable ⇒ Total ⇒ $\frac{d}{dx} f(x)$

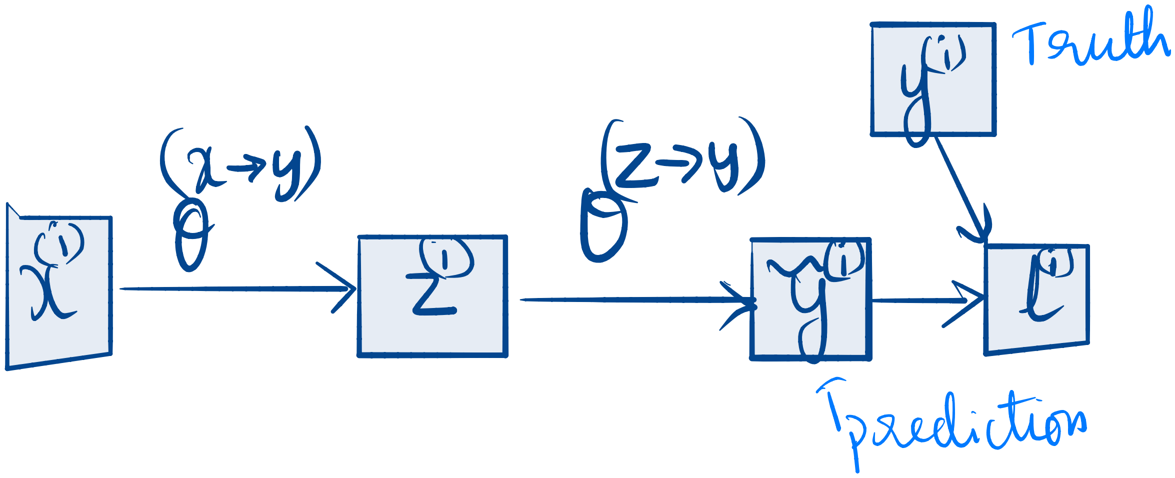
$f(x,y)$ ← function of two or more vars ⇒ Partial → $\frac{\partial}{\partial x} f(x,y)$, $\frac{\partial}{\partial y} f(x,y)$

→ NOTE: x and y are variables in calculus
Here we have parameters in Tensors

→ $z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$ $\quad \nabla_z$

See: http://cs231n.stanford.edu/vecDerivs.pdf

GRADIENT UPDATE RULE



$$\boldsymbol{\theta}_k^{(z\to y)} \leftarrow \boldsymbol{\theta}_k^{(z\to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)}$$

$$\boldsymbol{\theta}_n^{(x\to z)} \leftarrow \boldsymbol{\theta}_n^{(x\to z)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_n^{(x\to z)}} \ell^{(i)}$$

$$z^{(i)} = f\left(x^{(i)};\ \theta^{(x\to z)}\right)$$

$$\hat{y}^{(i)} = g\left(z^{(i)};\ \theta^{(z\to y)}\right)$$

$$L = \ell\left(\hat{y}^{(i)}, y^{(i)}\right)$$

FORWARD

BACKWARD

$$\frac{\partial L}{\partial \theta^{(x\to z)}}$$

$$\frac{\partial L}{\partial \theta^{(z\to y)}}$$

How to calculate $\dfrac{\partial L}{\partial \theta^{(x\to z)}}$ Chain Rule

Back propagation is an efficient algorithm

BACKPROPAGATION

Algorithm 6 General backpropagation algorithm. In the computation graph G, every node contains a function f_t and a set of parent nodes $\boldsymbol{\pi}_t$; the inputs to the graph are $\boldsymbol{x}^{(i)}$.

procedure BACKPROP($G = \{f_t, \boldsymbol{\pi}_t\}_{t=1}^{T}\}, \boldsymbol{x}^{(i)}$)
  $v_{t(n)} \leftarrow x_n^{(i)}$ for all n and associated computation nodes $t(n)$.
  for $t \in$ TOPOLOGICALSORT(G) **do** ▷ Forward pass: compute value at each node
    if $|\pi_t| > 0$ **then**
      $v_t \leftarrow f_t(v_{\pi_{t,1}}, v_{\pi_{t,2}}, \ldots, v_{\pi_{t,N_t}})$
  $g_{\text{objective}} = 1$ ▷ Backward pass: compute gradients at each node
  for $t \in$ REVERSE(TOPOLOGICALSORT(G)) **do**
    $g_t \leftarrow \sum_{t':t \in \boldsymbol{\pi}_{t'}} g_{t'} \times \nabla_{v_t} v_{t'}$ ▷ Sum over all t' that are children of t, propagating the gradient $g_{t'}$, scaled by the local gradient $\nabla_{v_t} v_{t'}$
  return $\{g_1, g_2, \ldots, g_T\}$

Demo:

https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

Further Reading:

Yes you should understand backprop by Andrej Karpathy

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

REGULARIZATION AND DROPOUT

→ All models overfit, non-linear models are no exception

→ Regularization → avoid overfitting

$$L = \sum_{i=1}^{N} l^{(i)} + \lambda \|\theta\|_2^2$$

→ Dropout : Randomly set some nodes to zero.