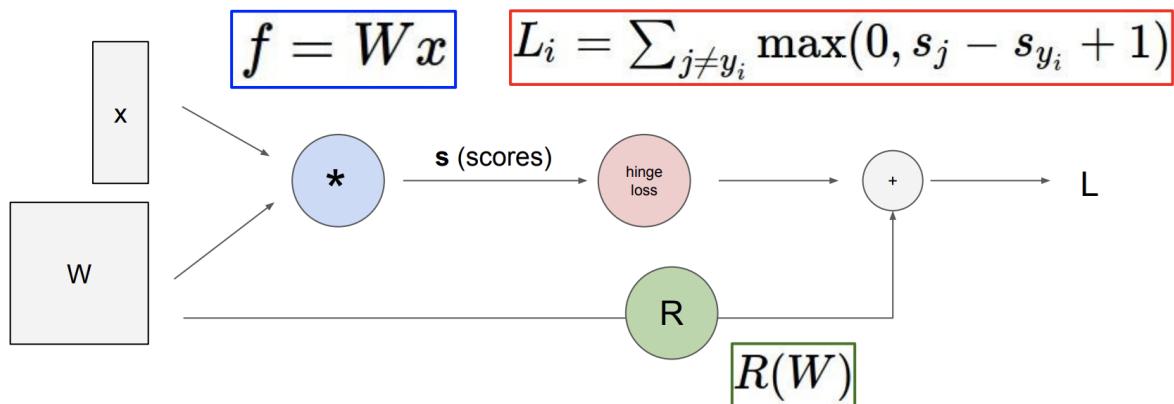


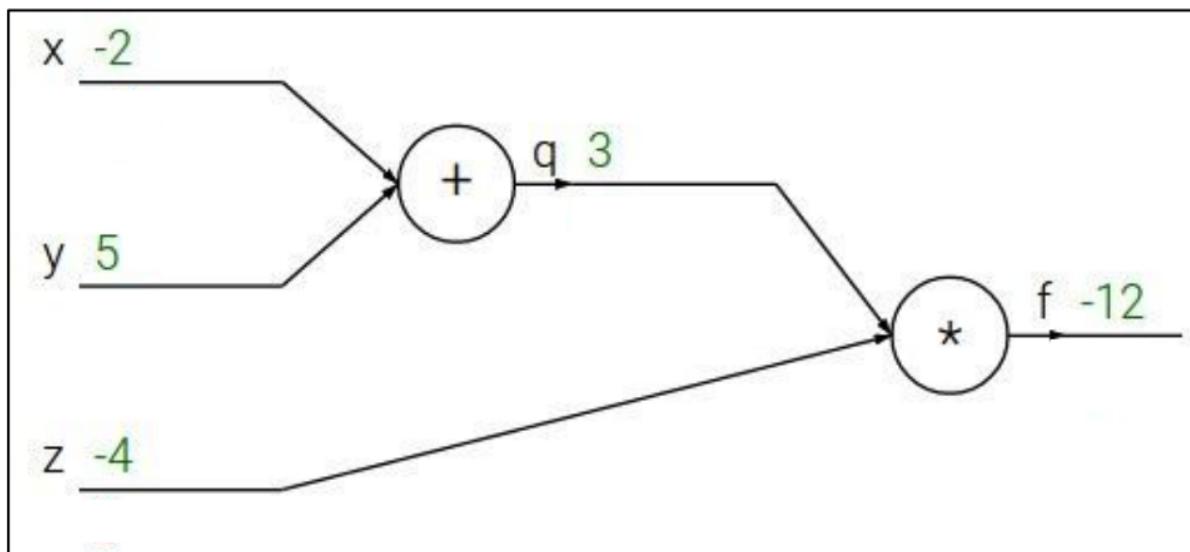
Lecture 4 | Introduction to Neural Networks

앞단원에서 우리는 SVM에 대해 배웠다. SVM에서 활용하는 연산들을 그래프로 나타낸 것이 바로 아래의 그림이다. 또, 우리의 현재 목적은 gradient를 구하고 이를 이용하여 최적의 W 를 구하는 것이다.

이번 단원에서는 복잡한 연산들을 그래프로 나타내고, 그 상황에서 어떻게 하면 gradient를 쉽게 구할 수 있을지 알아볼 것이다.



우선 간단한 그래프로 예를 들어보자. 아래는 $f(x, y, z) = (x + y)z$ 를 그래프로 나타낸 것이다.



여기서 우리가 궁금해하는 것은 $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial f}{\partial z}$ 이다. 즉, x,y,z에 대한 f의 변화량을 알고 싶은 것이다.

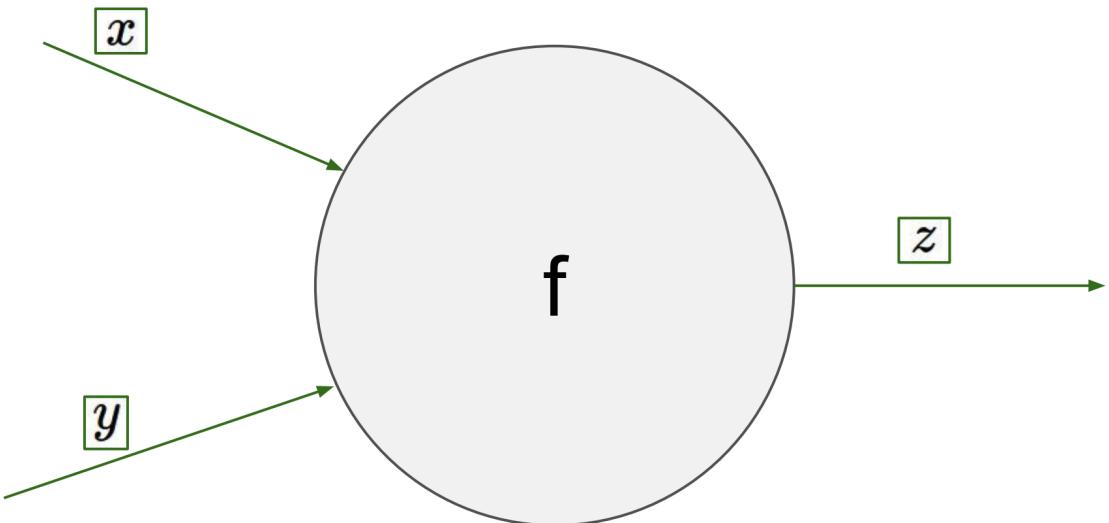
우선 $q = x + y$ 라는 식에서 $\frac{\partial q}{\partial x} = 1$, $\frac{\partial q}{\partial y} = 1$ 이라는 식을 얻을 수 있다.

또, $f = qz$ 라는 식에서 $\frac{\partial f}{\partial q} = z$, $\frac{\partial f}{\partial z} = q$ 라는 식을 얻을 수 있다.

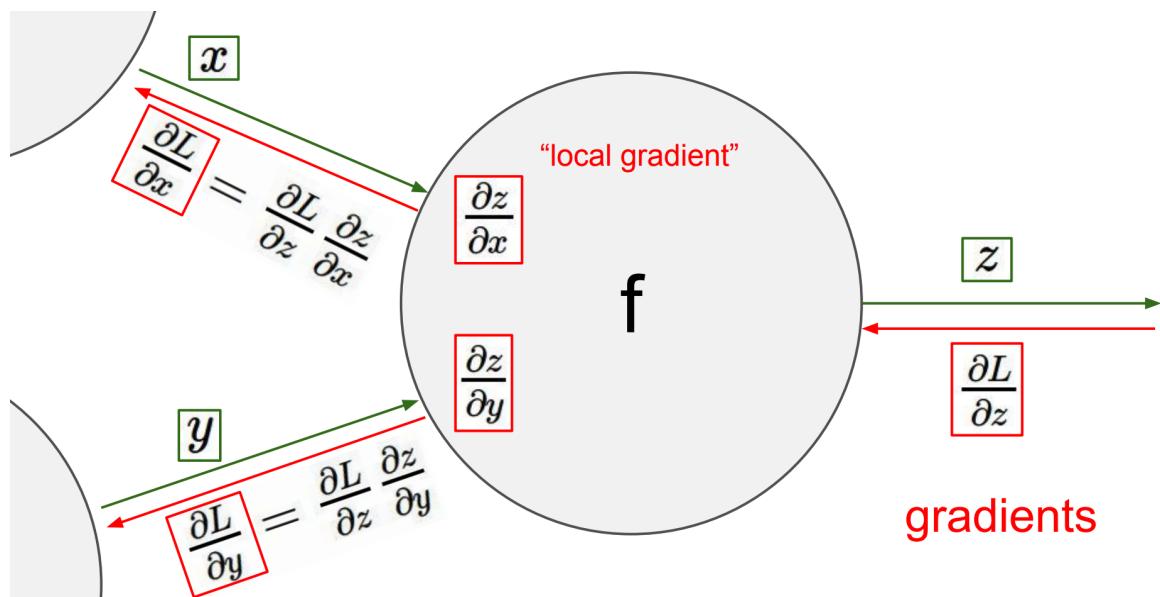
여기서 체인룰을 이용하면, 우리는 구하고자 하는 값들을 구할 수 있다.

$$\text{Chain rule : } \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

결과적으로 $\frac{\partial f}{\partial x} = -4$, $\frac{\partial f}{\partial y} = -4$, $\frac{\partial f}{\partial z} = 3$ 라는 결과를 얻을 수 있다.



위의 그림과 같은 상황이 있다고 하자. x 와 y 가 함수 f 의 input으로 들어가고 z 가 output으로 나온다. 그리고 z 에서 우리는 L 을 얻게 된다. 그런 상황에서, $\frac{\partial L}{\partial z}$ 를 구할 수 있고, 우리는 앞에서 한 것과 동일하게 chain rule을 이용하여 $\frac{\partial L}{\partial x}$, $\frac{\partial L}{\partial y}$ 를 얻어낼 수 있다.



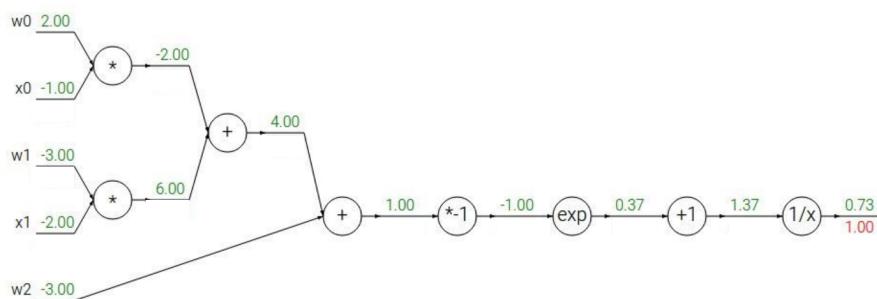
이처럼 전체 결과는 아니지만 중간 노드에서 우리가 연산할 수 있는 gradient들을 모두 local gradient라고 한다. 이러한 노드들이 여러개 붙어있다면, 우리는 각각의 변수에 대한 gradient를 앞선 노드의 gradient와, local gradient를 통해, 체인룰을 이용하여 연산할 수 있다.

다른 예시를 들어보자.

아래와 같은 다소 복잡한 식이 있다고 하자. 최종적으로 우리가 구하고자 하는 것은 w_0, x_0, w_1, x_1, w_2 에 대한 gradient이다. 이것을 바로 구하는 것은 어렵기 때문에 앞에서 한 것처럼 여러 단계로 나누고, 제일 앞의 gradient부터 구해가려고 한다.

Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$f(x) = e^x$	\rightarrow	$\frac{df}{dx} = e^x$	$f(x) = \frac{1}{x}$	\rightarrow	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	\rightarrow	$\frac{df}{dx} = a$	$f_c(x) = c + x$	\rightarrow	$\frac{df}{dx} = 1$

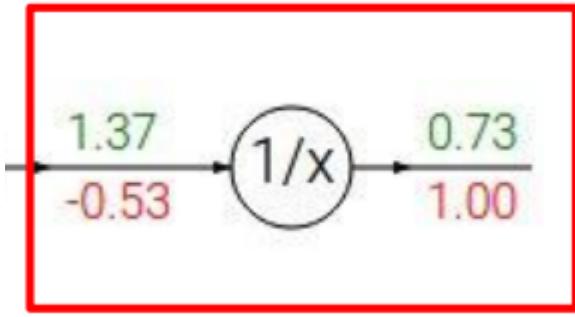
우선 가장 마지막 변수에 대한 gradient는 1이라고 하겠다.

이제 그 앞부분을 보자.

$\frac{df}{dx} = -\frac{1}{x^2}$ 이기 때문에 우리는 끝에서 두번째 변수에 대한 gradient를 구할 수 있다.

앞에서 언급한 chain rule을 이용하면 gradient는 $(\frac{-1}{1.37^2})(1.00) = -0.53$ 임을 알 수 있다.

$$(\text{local gradient} = -\frac{1}{x^2} = \frac{-1}{1.37^2})$$



이와 동일하게 계속 반복하면 우리는 초기 변수에 대한 gradient를 구할 수 있다.

위에서 우리는 연산을 쪼갤 수 있는 최대한 쪼개어 각각의 노드로 표현하였다. 하지만 굳이 이렇게 쪼개지 않고, 보다 더 복잡한 연산으로 쪼개는 것도 가능하다.

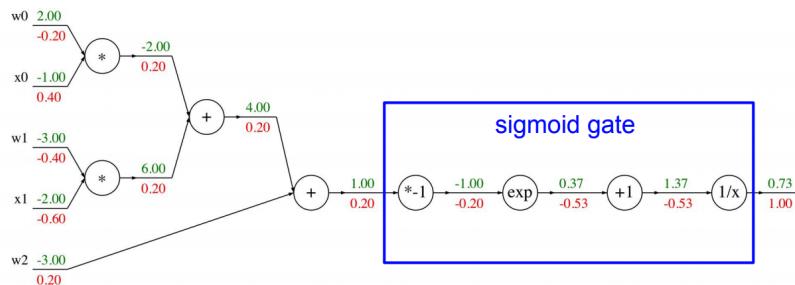
위에서 4개의 연산으로 쪼갠 부분을 조금 복잡하게 하여 $\sigma(x) = \frac{1}{1+e^{-x}}$ 로 표현할 수 있다.

그러면 이제 local gradient를 구하는 과정에서 위 함수를 미분하기만 하면 된다.

즉, 우리가 local gradient를 구할 수 있는 한에서는, 즉 미분하는데 어려움이 없을 정도라면 조금 복잡하게 쪼개어도 아무런 상관이 없다.

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \quad \sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{sigmoid function}$$

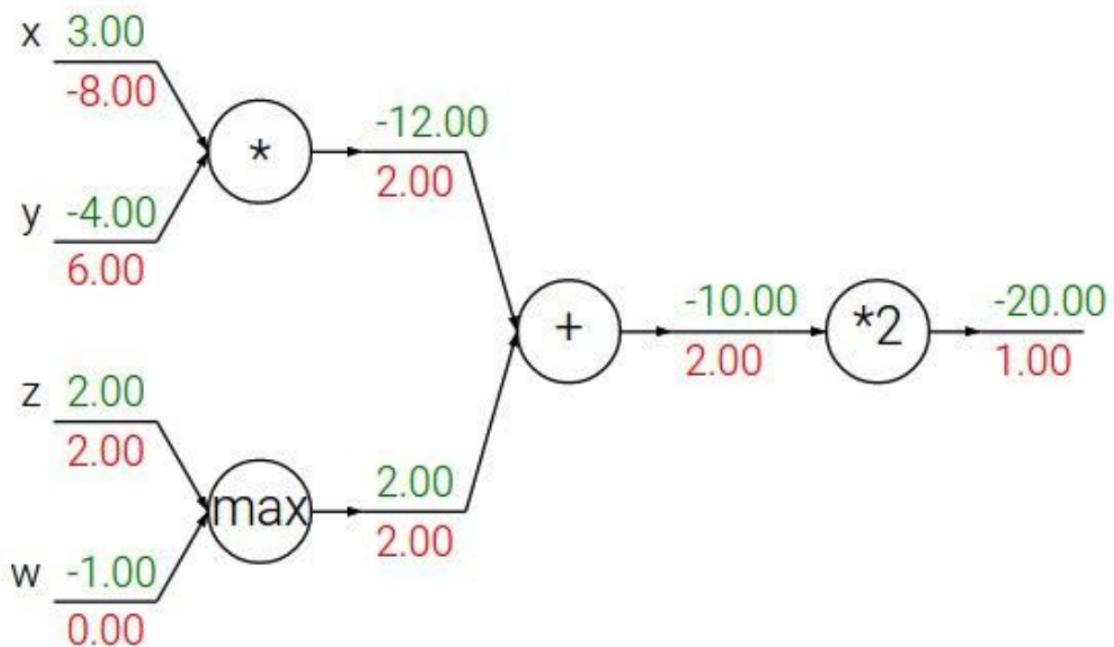
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left(\frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$



덧셈, 곱셈 등으로 이루어진 함수들은 이제 계산이 가능하겠지만 조금 더 알아야하는 부분이 있다.

바로 max함수이다. max함수에서는 어떻게 해야 할까?

둘중 큰 값은 앞의 gradient를 그대로 옮겨 받고, 작은 값은 gradient가 0이 될 것이다.

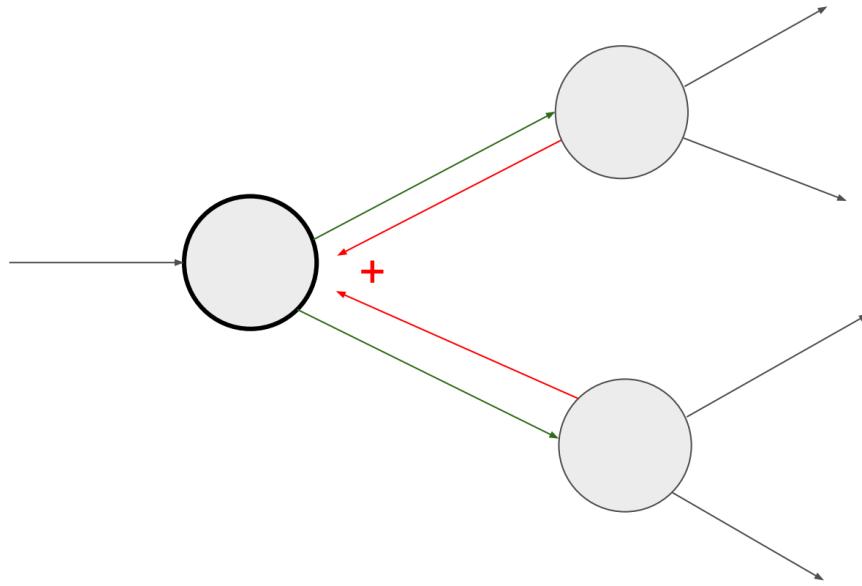


위 그림에서 알 수 있는 사실은 +, *, max gate의 의미이다.

add gate는 gradient를 그대로 전달하며, max gate는 둘중 하나로만 전달하며 mul gate는 함께 연산하는 값과 앞의 gradient를 곱한 값이다. 즉 우리는 각각을 gradient distributor, gradient router, gradient switcher라고 부른다.

- add gate : gradient distributor
- max gate : gradient router
- mul gate : gradient switcher

만약 아래 그림처럼 하나의 노드가 2개의 노드로 퍼져 나가는 상황은 어떨까?



이런 상황에서는 양쪽에서 연산한 두 gradient를 더하면 된다. 하나의 변수가 두번 존재하는 경우, 각각의 형태에서의 gradient의 합이 곧 그 변수에 대한 gradient인 것은 자명하다. "전체에 얼마나 영향을 주는지"라고 생각하면 이해하기 쉬울 것이다.

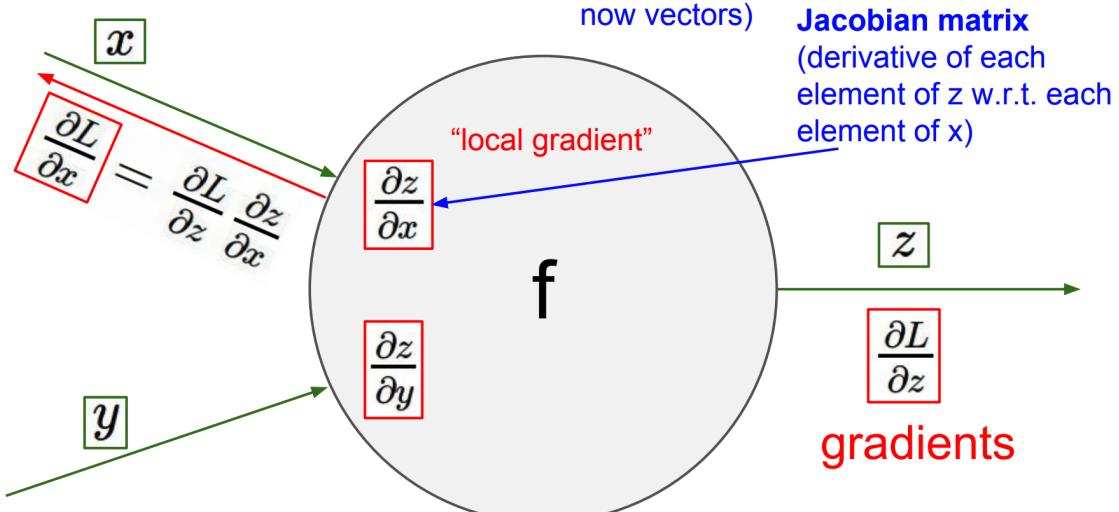
이를 식으로 표현하면 다음과 같다.

$$\frac{\partial f}{\partial x} = \sum_i \frac{\partial f}{\partial q_i} \frac{\partial q_i}{\partial x}$$

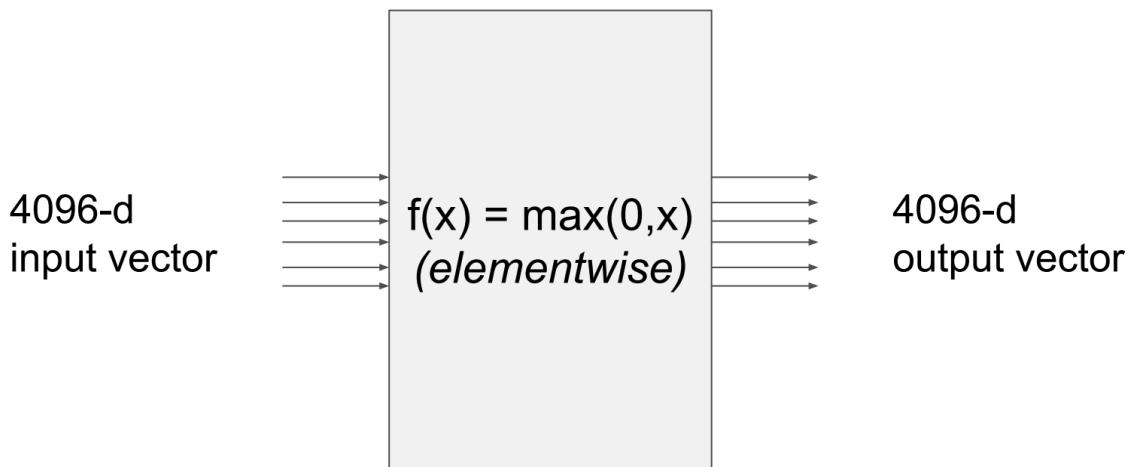
앞에서 했던 것과 동일한데, 이제는 각각이 vector의 형태로 주어지는 경우에 대해 살펴볼 것이다.

그러면 gradient가 jacobian matrix 형태로 될 것이다.

Gradients for vectorized code



아래의 그림처럼 input이 4096 크기의 벡터이고 output도 4096 크기의 벡터인 상황을 살펴보자.



▼ Q1 : 이런 상황에서 Jacobian matrix의 크기는 어떻게 될까?

A1 : 4096 * 4096 이다.

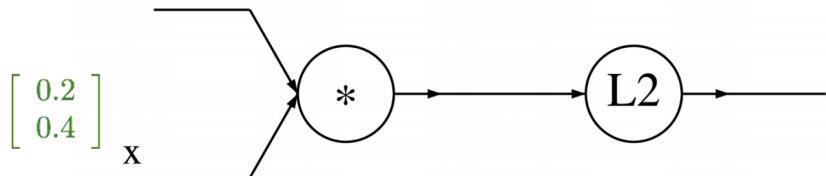
실제로 우리가 사용할 때에는 여러개(ex 100개)를 한번에 테스트 하곤 한다. 이런 경우, Jacobian matrix의 크기는 409600 * 409600이 될 것이다.

▼ Q2 : Jacobian matrix는 어떤 형태일까?

그러면 vector 형태로 된 예시를 살펴보자.

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^n (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} W$$

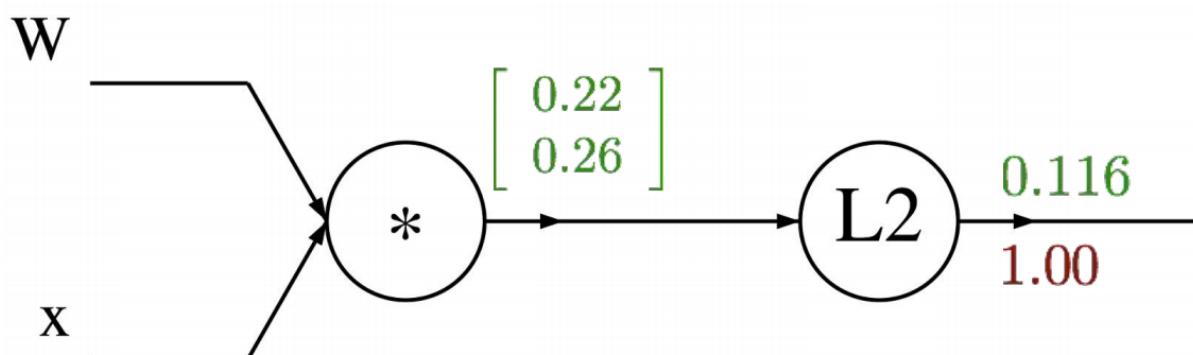


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$x = n \times 1$ 행렬, W 는 $n \times n$ 행렬의 형태이다. 그러면 이를 곱해서 나온 행렬은 $n \times 1$ 형태일 것이다.

각각 연산해주고 나면 최종 결과는 0.116이 나온다. 여기서도 마지막에 대한 gradient는 1이라고 하자.



이제 local gradient와 다음 노드의 gradient를 이용하여 현재 노드의 gradient를 구해야 한다. 그런데, 우리가 구해야 하는 것은 아까와 다르게 벡터가 된다. 그렇지만 이는 별 차이가 없다. 그냥 각각의 index별로 연산을 진행해주면 된다.

$\frac{\partial f}{\partial q_i} = 2q_i$ 이므로 $\nabla_q f = 2q$ 이다.

이를 이용하여 벡터의 각각의 항들에 대해 gradient를 계산해주면

$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$ 를 얻을 수 있다.

여기서 알 수 있는 사실은, vector의 크기와 gradient의 크기는 동일하며, 각각의 요소에 대한 gradient는 동일 위치에 있는 값이라는 것을 알 수 있다.

한 스텝 더 뒤로 가보자. 여기서 구해야 하는 것은 W 와 x 의 gradient이다.

우선, local gradient 먼저 구해보자.

$\frac{\partial q_k}{\partial W_{i,j}} = x_j$ 이다. q 가 어떻게 생긴지 보면 이해가 갈 것이다.

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

이제 체인룰을 이용하면 W 의 gradient를 구할 수 있다.

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(x_j) \\ &= 2q_i x_j \end{aligned}$$

이를 간단하게 표현하면 $\nabla_W f = 2q \cdot x^T$ 이다.

그러면 W 의 gradient는 $\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix}$ 이 된다.

동일하게 x 의 gradient도 구해보자.

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\begin{aligned} \frac{\partial f}{\partial x_i} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} \\ &= \sum_k 2q_k W_{k,i} \end{aligned}$$

이를 간단하게 표현하면 $\nabla_x f = 2W^T \cdot q$

그러면 x의 gradient는 $\begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$ 이 된다.

이곳에 가보면 여러가지를 구현해 둔 코드를 볼 수 있다.

Example: Caffe layers

Branch: master	cafe / src / caffe / layers /	Create new file	Upload files	Find file	History
shellhamer committed on GitHub Merge pull request #4639 from BlGene/load_hdf5_fix	Latest commit e687e71 21 days ago				
..					
absval_layer.cpp	dismantle layer headers	a year ago			
absval_layer.cu	dismantle layer headers	a year ago			
accuracy_layer.cpp	dismantle layer headers	a year ago			
argmax_layer.cpp	dismantle layer headers	a year ago			
base_conv_layer.cpp	enable dilated deconvolution	a year ago			
base_data_layer.cpp	Using default from proto for prefetch	3 months ago			
base_data_layer.cu	Switched multi-GPU to NCCL	3 months ago			
batch_norm_layer.cpp	Add missing spaces besides equal signs in batch_norm_layer.cpp	4 months ago			
batch_norm_layer.cu	dismantle layer headers	a year ago			
batch_reindex_layer.cpp	dismantle layer headers	a year ago			
batch_reindex_layer.cu	dismantle layer headers	a year ago			
bias_layer.cpp	Remove incorrect cast of gemm int arg to Dtype in BiasLayer	a year ago			
bias_layer.cu	Separation and generalization of ChannelwiseAffineLayer into BiasLayer	a year ago			
bnll_layer.cpp	dismantle layer headers	a year ago			
bnll_layer.cu	dismantle layer headers	a year ago			
concat_layer.cpp	dismantle layer headers	a year ago			
concat_layer.cu	dismantle layer headers	a year ago			
contrastive_loss_layer.cpp	dismantle layer headers	a year ago			
contrastive_loss_layer.cu	dismantle layer headers	a year ago			
conv_layer.cpp	add support for 2D dilated convolution	a year ago			
conv_layer.cu	dismantle layer headers	a year ago			
crop_layer.cpp	remove redundant operations in Crop layer (#5138)	2 months ago			
crop_layer.cu	remove redundant operations in Crop layer (#5138)	2 months ago			
cudnn_conv_layer.cpp	dismantle layer headers	a year ago			
cudnn_conv_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
cudnn_conv_layer.h	dismantle layer headers	11 months ago			
cudnn_icn_layer.cpp	dismantle layer headers	a year ago			
cudnn_icn_layer.cu	dismantle layer headers	a year ago			
cudnn_lrn_layer.cpp	dismantle layer headers	a year ago			
cudnn_lrn_layer.cu	dismantle layer headers	a year ago			
cudnn_pooling_layer.cpp	dismantle layer headers	a year ago			
cudnn_pooling_layer.cu	dismantle layer headers	a year ago			
cudnn_relu_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
cudnn_relu_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
cudnn_sigmoid_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
cudnn_sigmoid_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
cudnn_softmax_layer.cpp	dismantle layer headers	a year ago			
cudnn_softmax_layer.cu	dismantle layer headers	a year ago			
cudnn_tanh_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
cudnn_tanh_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago			
deconv_layer.cpp	Switched multi-GPU to NCCL	3 months ago			
deconv_layer.cu	enable dilated deconvolution	a year ago			
dropout_layer.cpp	supporting N-D Blobs in Dropout layer Reshape	a year ago			
dropout_layer.cu	dismantle layer headers	a year ago			
dummy_data_layer.cpp	dismantle layer headers	a year ago			
eltwise_layer.cpp	dismantle layer headers	a year ago			
eltwise_layer.cu	dismantle layer headers	a year ago			
elu_layer.cpp	ELU layer with basic tests	a year ago			
elu_layer.cu	ELU layer with basic tests	a year ago			
embed_layer.cpp	dismantle layer headers	a year ago			
embed_layer.cu	dismantle layer headers	a year ago			
euclidean_loss_layer.cpp	dismantle layer headers	a year ago			
euclidean_loss_layer.cu	dismantle layer headers	a year ago			
exp_layer.cpp	Solving issue with exp layer with base e	a year ago			
exp_layer.cu	dismantle layer headers	a year ago			

Caffe is licensed under [BSD 2-Clause](#)

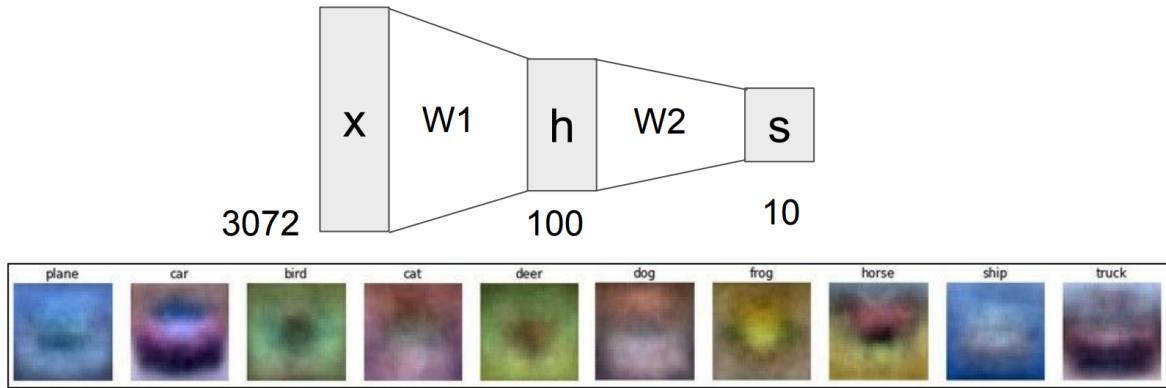
Neural Network

앞에서 점수를 매길 때 사용한 함수는 $f = Wx$ 이다. 하지만 이제는 분류하는 과정에 단계를 추가하여 2-layer Neural Network를 사용할 것이다. 여기서 사용하는 함수는 $f = W_2 \max(0, W_1 x)$ 이다.

즉, 분류하는 과정에 단계를 만들 것임을 의미한다.

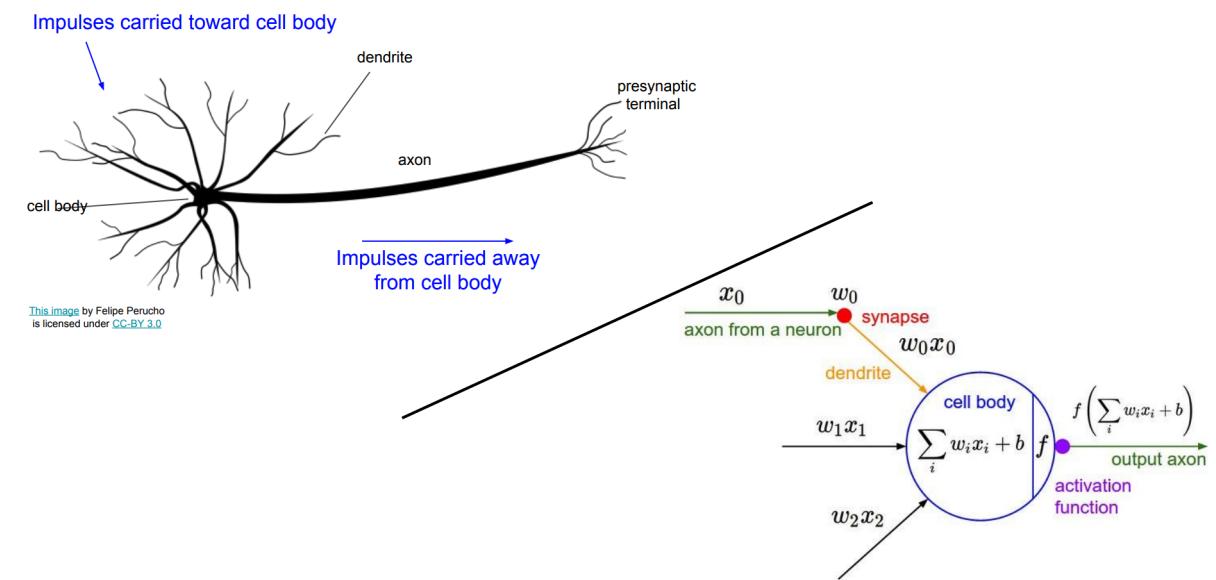
예를 들어 얼굴이 왼쪽에 있는 말 사진과 오른쪽에 있는 말 사진이 training data에 들어 있고, 입력으로 얼굴이 왼쪽에 있는 말 사진이 주어졌다고 하자. 그러면 우선 W_1 을 거치면서 각각에 대한 점수를 매긴 후, W_2 에서 같은 라벨들에 대한 점수의 총합을 구하여 그 것을 최종 점수로 할 수 있다.

아래의 그림처럼 W_1 을 먼저 거친 후, W_2 를 거쳐서 최종 결과를 얻을 것이다. 최종적으로 결정할 수 있는 라벨이 10개라고 할 때, 동일한 종류에서 다른 형태들이 많이 존재한다. 그렇기 때문에 W_1 에서는 100개의 라벨을 만들어 각각에 대한 데이터를 구하고, 이제 결과적으로 동일한 라벨들이 존재하기 때문에 이는 W_2 에서 처리해주는 것이다.



혹은 단계를 더 추가하여 3-layer Neural Network를 사용할 수도 있다.

실제 뉴런과의 비교



실제 뉴런과 우리가 위에서 본 구조를 비교해보면 유사함을 알 수 있다.

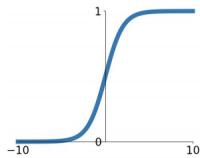
자극이 오는 것이 input이고, 이를 해석하여 cell body로 가는 것이 W를 거치는 것이다. 그리고 자극이 전달되는 것이 최종 활성 함수를 거치는 것으로 보면된다. 위 그림을 보면 이해가 쉬울 것이다.

하지만, 실제 뉴런과 우리가 만드는 뉴런 네트워크는 많이 다르다. 그렇기 때문에 우리가 만드는 신경망을 실제 신경망이라고 판단하면 안된다.

다양한 활성함수들

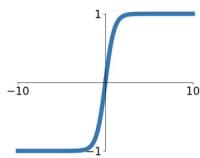
Sigmoid

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



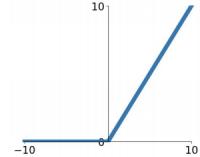
tanh

$$\tanh(x)$$



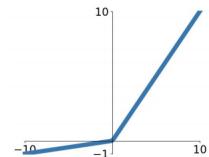
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

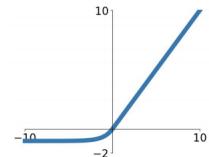


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



아래의 그림은 앞에서 언급한 2-layer Neural Network 와 3-layer Neural Network의 형태이다. input layer와 output layer 이외의 layer를 우리는 hidden layer라고 부르고, hidden layer의 개수로 네트워크의 형태를 말하기도 한다. 또, 아래의 그림처럼 모든 node가 연결되어 있는 형태를 Fully-connected layers라고 한다.

