lect 10 CNN

Topic : Image Classification

Classify a image to → cat, dog, plane ⋮

Machine Learning alg & Data-driven approach

1. Nearest Neighbour Classifier. (Naive)

Training Step: Simply Remember all images

Test Step : Find which training image is the closest to the test sample and report its label. or which "k" samples are similar and vote.

Distance Metric

L1 Distance $|A-A'| = \sum_{i,j} |A_{ij} - A'_{ij}|$

L2 Distance $\|A-A'\|_2 = \sqrt{\sum_{i,j} (A_{ij} - A'_{ij})^2}$

L-p norm $\|A-A'\|_p = \sqrt[p]{\sum (A_{ij} - A'_{ij})^p}$ $\quad p \geq 0$

Remember

Don't deal with test data as training!

It's cheating!

But you can split training data to { training part / check part

and do Cross Check

E.g Split Training to 3 parts. Set 1, Set 2. Set 3.

① Train Set 1 & 2 ⟶ Test on Set 3

② Train Set 1 & 3 ⟶ Test on Set 2

③ Train Set 2 & 3 ⟶ Test on Set 1

choose the best model

For Nearest Neighbour Classifier

Training Complexity O(1) only storing, or O(N)

Testing Complexity O(NM) N: Training, M Testing

large.

2. Linear Classifier.

① Define a score function

$$f(x_i, W, b) = Wx_i + b$$

x_i : Serialized Image say 1000×1

W ; Weight matrix say 10×1000 for 10 target labels

b . Bias say 10×1

Score is a 10×1 matrix corresponding to 10 labels.

$$Score = [W]x_i + b$$

↖ ↗

Matched Filter or, projection

of course, we could make use of Homogenious representation to get rid of "b", then

$$Score = [W, b] \cdot \begin{bmatrix} x_i \\ 1 \end{bmatrix}$$

② Loss function

loss function is to find out how "unhappy" we are to the result.

$$l(X, W, Y)$$

- X: training images
- W: Weight
- Y: label/value

Commonly Use Loss Function.

1. Softmax

$$y_i: \quad \underbrace{f(x_i, W)}_{\text{model.}} = \underbrace{[f_0 \cdots f_k]^T}_{\text{Score for (k+1) labels.}}$$

$$\text{Happyness} = \frac{1}{\Gamma} e^{F}, \quad \Gamma \text{ is a normalization term}$$

$$\Gamma = \sum_{i=0}^{k} e^{F_i}$$

So, the output for "f_i" is $\left(-\log\left(\frac{1}{\Gamma} e^{f_{y_i}}\right)\right)$ ← Unhappyness

2. Multiple SVM loss

$$L = \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

Once the output is separable, we are happy!

E.g. if $y_i = 0$. $f_i = [10, -2, 3]$

$= [10, 9, 9]$

$= [10, -100, -100]$

→ $L = 0$

↑

They are equally good

Summary.

Softmax

$$L = -\ln \frac{e^{f_{y_i}}}{T} \qquad T = \sum_{j=0}^{k} e^{f_j}$$

SVM

$$L = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + 1)$$

Now, target turns to find a good W, b w.r.t L

Gradient Descent

Numerical Computation for Gradient

$$\nabla_x f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

Slow!

Analytical Gradient by calculous.

But use Numerical Gradient to check.

Batch Gradient Descent

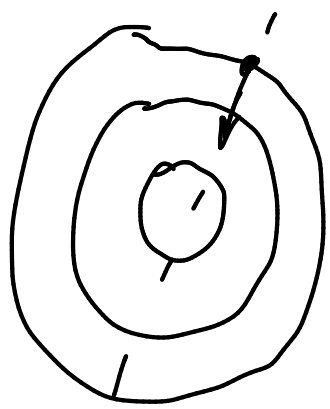
Compute Gradient based on all samples

Mini-Batch Gradient Descent

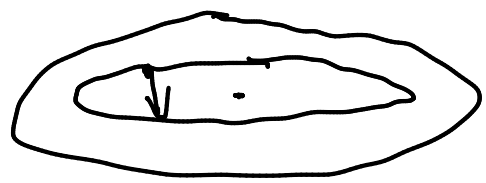
Select a small group of samples.

Problems for GD.

if the function is not of a shape of a circle.
the performance maybe poor



One solution



- gradient direction
is not the best direction

Momentum Update

$val = val * \alpha - step * gradient$

Another.

Newton's Method, steepest update

Back Propagation

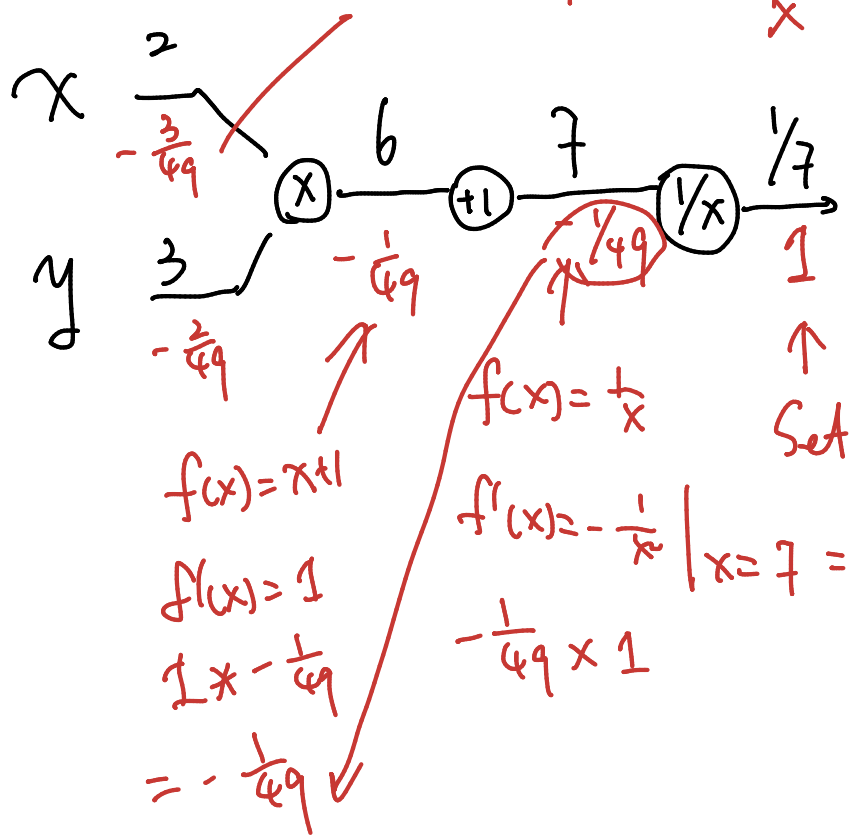
Suppose:

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

$$= q \cdot z \qquad q = x+y$$

$$\frac{\partial}{\partial x} f = \frac{\partial}{\partial x}(x+y)z = \frac{\partial}{\partial q}(q \cdot z) \cdot \frac{\partial}{\partial x} q$$

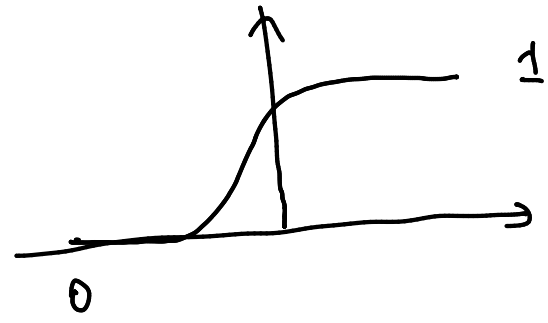
Suppose



Local Gradient can be computed when do Forward Process. Which is something like $\frac{\partial}{\partial x} f(x) = -x^2+2$, only need x for back propagation.

Sigmod function

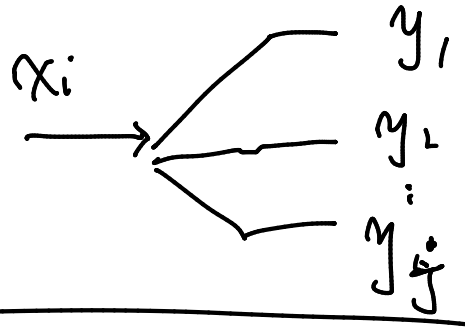
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



$$\frac{\partial}{\partial x}\sigma(x) = (1-\sigma(x))\,\sigma(x)$$

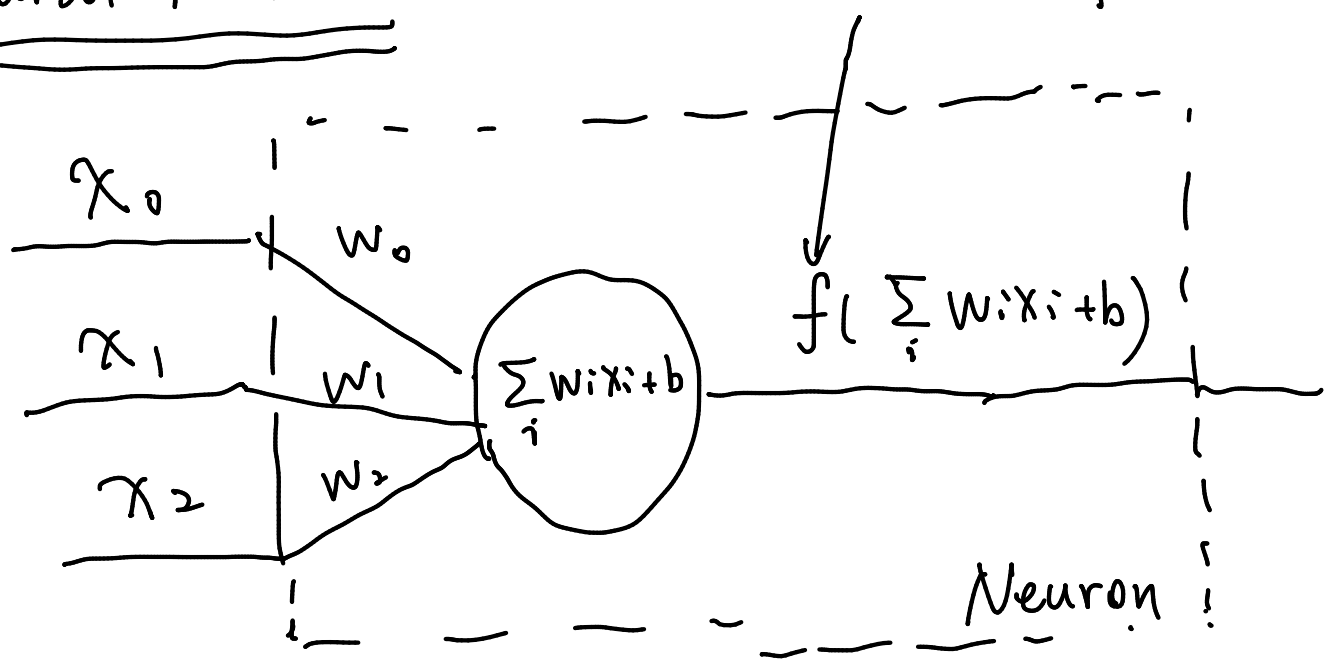
Suppose We have multiple inputs. for

$$\frac{\partial}{\partial x_i} = \sum_j \frac{\partial}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$



Neural Network

activation function. say sigmod $\frac{1}{1+e^{-x}}$



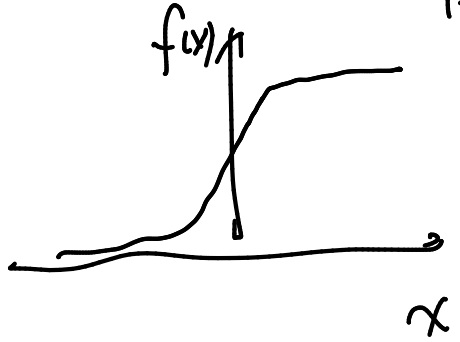
A single Neuron can be treated as a binary linear classifier.

Neuron Model is highly simplified!

Activation Functions.

① Sigmod

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



Derived from biological discoveries

Problem: Gradient at large/small part is almost 0! Needs very high precision! Also, training rate after propagation is also small!

② ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

- Won't saturate
- Easy to compute
- No need to worry about Gradient Saturation, traing is fast. (x6 comp to sigmod)

Problem

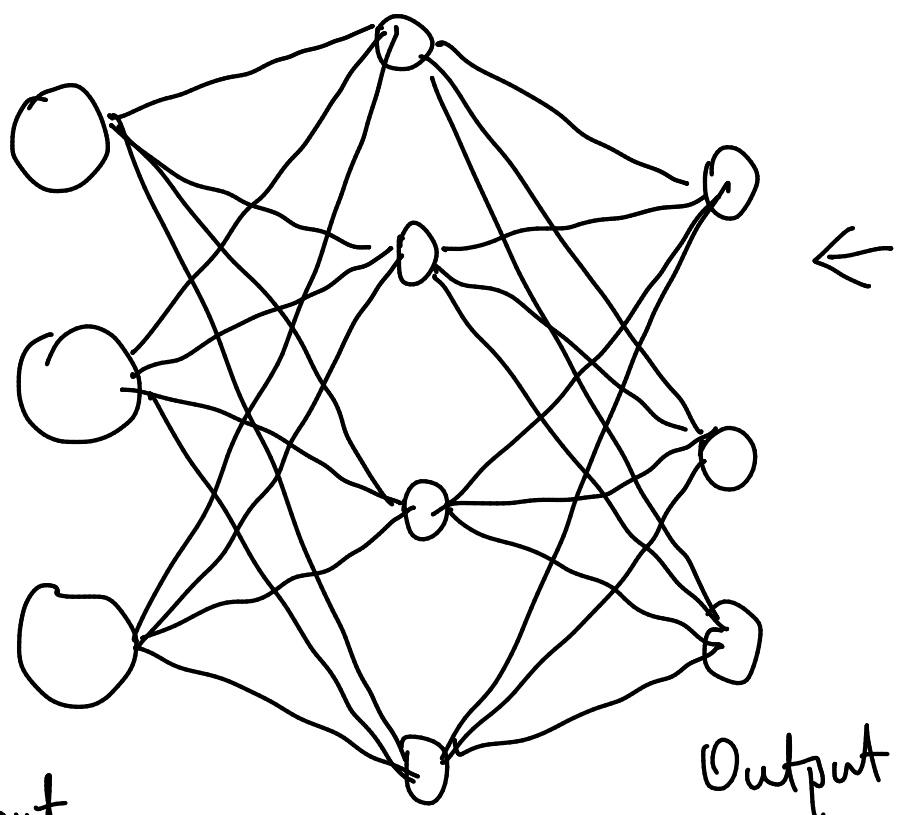
Gradient at $x=0$ is ambiguous

For neurons which starts with negative input for activation function, They will never be updated! Called Dead Neuron, because gradient is 0!

Summary:

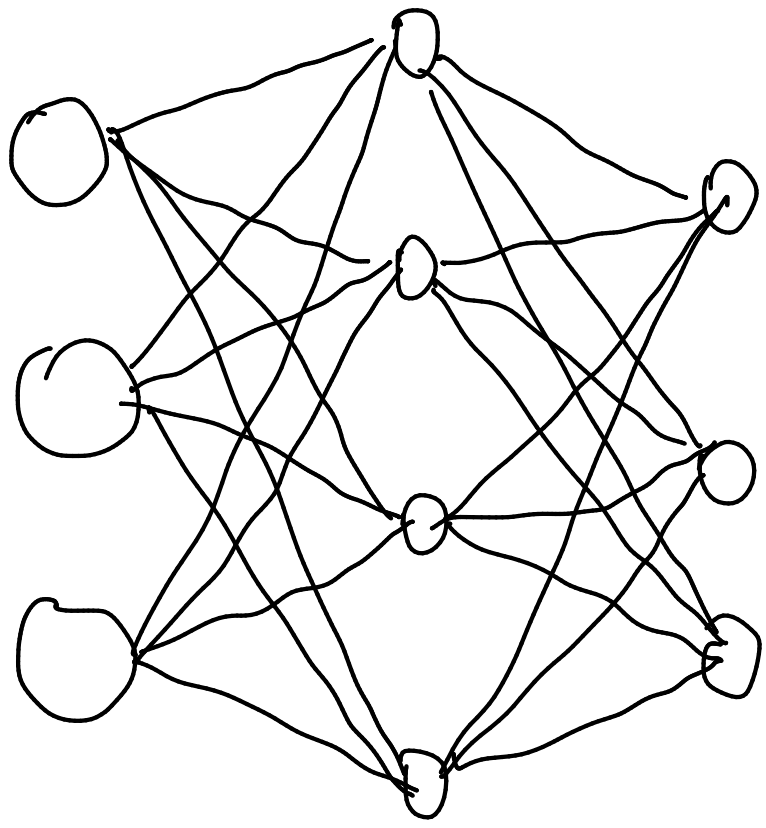
* Avoid Sigmod
* Use ReLU, but be careful with learning rate.

Architectures



← Fully Connected Layer.

1- hidden- layer Neural Net
or 2- layer Neural Net



There are
hidden output
$4+3=7$ Neurons

$3\times4+3\times4=24$ Weights

$24+7=31$ Biases

Modern NN $\sim 10e6$ Neurons

Human Vision $\sim 5e8$ Neurons

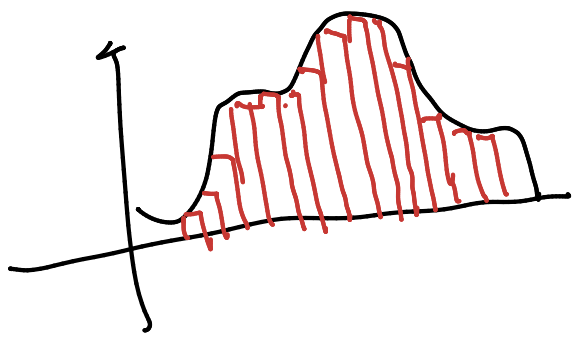
What kind of function can a NN simulate?

By Taylor Expansion. we can represent arbitrary func.

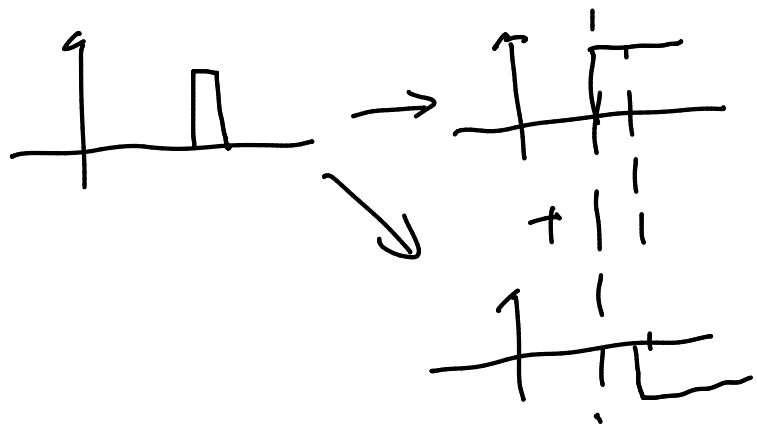
If we have <u>unlimited neurons</u> !

What we can do?.

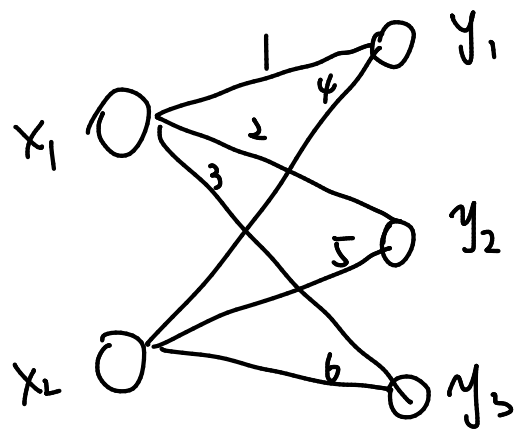
Quantization! Let Neuron Network to approx descrete function.



It's easy to use two neurons to simulate



Weight of a Layer can be expressed as a matrix.



$$Y = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Train Neural Network

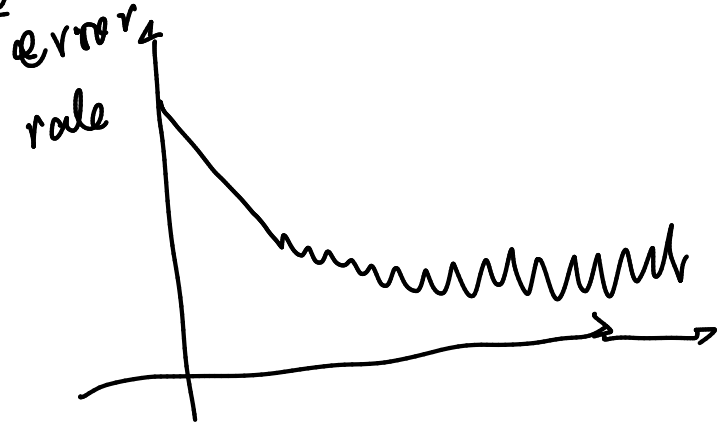
$$W^+ = W - \eta \frac{\partial}{\partial W_i} \sum L(f(x_i; W), y_i)$$

Learning rate & updates

* SGD + momentum > SGD
* Momentum = 0.9 is great
* Decrease Learning Rate over time.

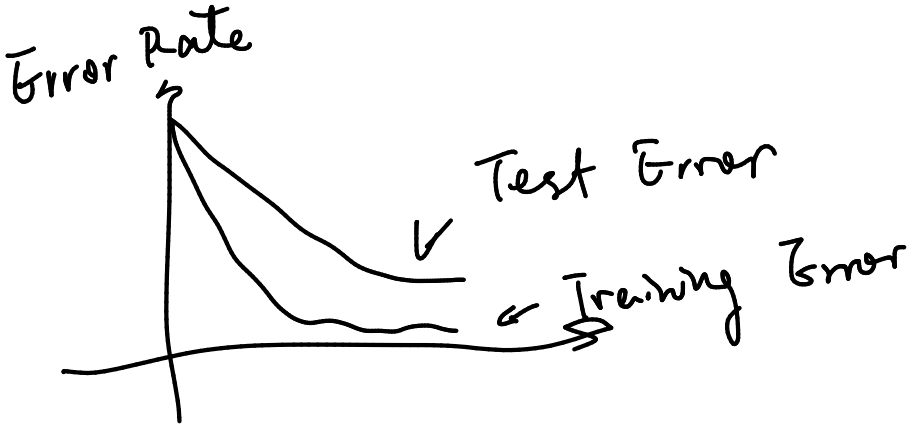
Training Accuracy

@



Fluctuation because of noise, because we are <u>not</u> doing Batch Gradient Descent, but Mini-batch GD.

② Overfit

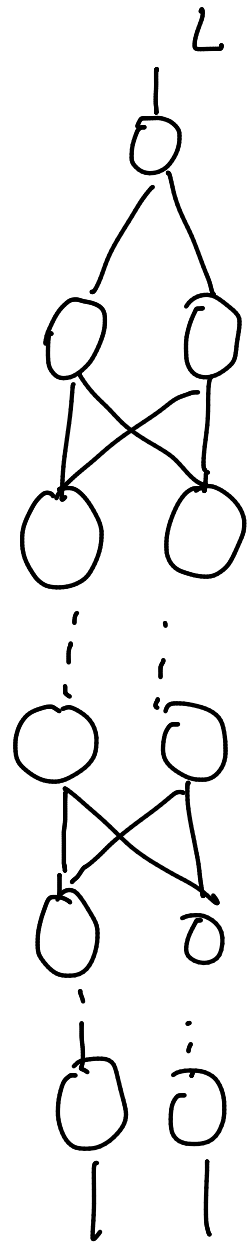


For Network

Target

$$\frac{\partial}{\partial W_i} L = \frac{\partial L}{\partial x_i} \cdot \frac{\partial x_i}{\partial W_i}$$

$$= \sum_{j=1}^{N} \frac{\partial L}{\partial x_i^{(j)}} \cdot \frac{\partial x_i^{(j)}}{\partial W_i}$$



① The reason that we use summation is that function in current layer is linear.

if $f(x) = h(x) + g(x)$

$$\frac{\partial}{\partial x} f(x) = \frac{\partial}{\partial x} h(x) + \frac{\partial}{\partial x} g(x)$$

② We can use matrix multiplication to simply the expression

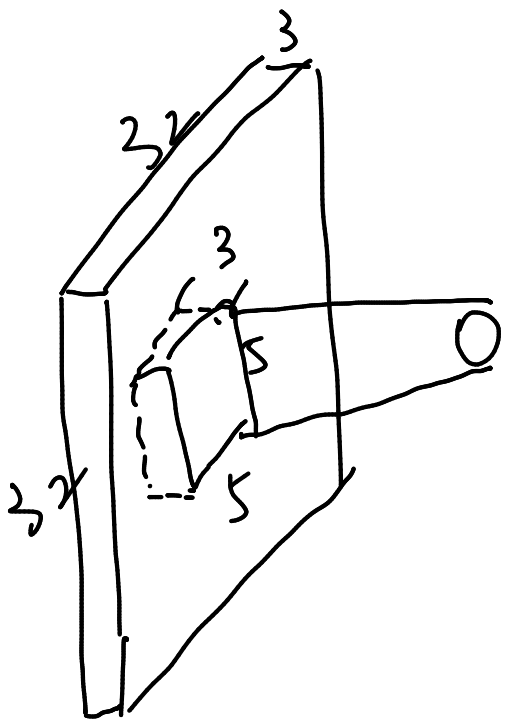
$$\text{Local Gradient} = \begin{bmatrix} \frac{\partial x_i^{(1)}}{\partial W_{i1}} & \frac{\partial x_i^{(1)}}{\partial W_{i2}} & \cdots \\ \vdots & & \\ \frac{\partial x_i^{(N)}}{\partial W_{i1}} & \frac{\partial x_i^{(N)}}{\partial W_{i2}} & \end{bmatrix}$$

$$\text{Accumulate Gradient} = \begin{bmatrix} \frac{\partial L}{\partial x_i^{(1)}} & \cdots & \frac{\partial L}{\partial x_i^{(N)}} \end{bmatrix}^T$$

CNN

Neurons at each layer is more! Say 32×32×3

for a Image location, each neuron is connected to a 5×5 area



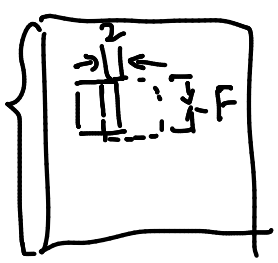
← connect 5×5×3 receptive field

but we have 32×32×5 neurons, for the same connection, we have 5 neurons connecting to same input locations, but different weight & activative function.

If we only focus on valid region, we may need

32×32×5 neurons to cover all 5×5 pixels.

possible locations.

If we set "stride = 2", say, 

we need $\frac{N-F}{\text{Stride}} + 1$ for one dimension

F: Filter size

If : input is a $[W, H, D]$ say $32 \times 32 \times 3$

receptive field is $F_1 \times F_2$

output volumn is $[W', H', D']$

$$W' = \frac{W - F_1}{S} + 1 \qquad H' = \frac{H - F_2}{S} \qquad D' = D$$

Problem.

We have a $32 \times 32 \times 3$ image.

We have a network with depth 30 (30 neurons on same area focusing on 5x5 area).

we have $32 \times 32 \times 30$ neurons

each of them has $5 \times 5 \times 3$ weights.

We will have $\sim 3M$ weights.

→ We only give weights to $5 \times 5 \times 3$ for 30 neurons.

Treat the output as a convolution..

But sometimes, we don't want to share parameters. for neurons focusing on different locations. Say, we know background is different from the face in the middle; They should focusing on different things.

Images → Low-level features → Mid-level features → High-level features → Classifier

Image → Conv → ReLU → Conv → ReLU ..... Conv → ReLU → Classifier

↑

Introduce Non-linearality

Pool layer:

Downsampling. say, take the max of 2x2 unadjacent block

So, some 5x5 filter could see a larger region. 10x10 this time.

We can have

→ [Conv - ReLU - Pool]$_{xN}$ + [Classifier]$_{x1}$ →