CS 461: Machine Learning Principles

Class 18: Nov. 4

Deep Neural Network:

Architecture, Training Methods,

Performance and Adversarial Examples

Instructor: Diana Kim

In the last class:

- Deep CNN Architecture : AlexNet
  - (1) Feature blocks
  - (2) Convolutional blocks
    - Robust to small translation, scaling, and rotations.
    - Equivariance: through Local Receptive and Parameter Sharing of Filter bank
    - Invariance: through Max pooling
    - Similarities of learned filters to those found in biological vision systems.

Today:

- Deep CNN Architecture : AlexNet

  (2) Classifier block : **F**ully **C**onnected Layer (FC) + activation $\{N\}$

  - Different activation functions: Logistic / Tanh vs. ReLU

  Q: Why ReLU is preferred in the modern deep CNN architecture?

  Q: What problem can we have as we training a deep CNN with ReLU?

- Training Schemes for Deep-CNN

  (1) preprocessing of data

  (2) optimization methods

  (3) regularization methods

Today:

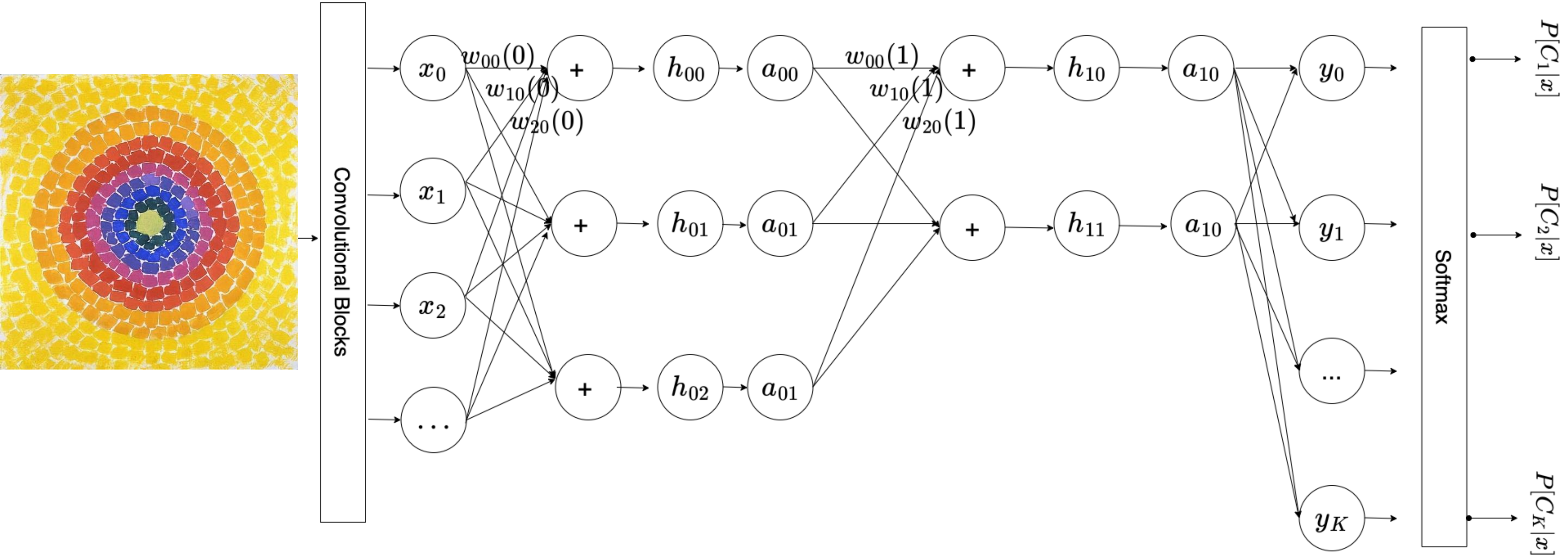
- Deep-CNN Inference Results ✓

(1) CNN performance competitive to human object recognition
(2) the performance unseen unusual data?
(3) Adversarial Examples

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /Users/dianakim/.cache/torch/hub/v0.10.0.zip
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /Users/dianakim/.cache/torch/hub/c
100%|
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)   → F-C
    (2): ReLU(inplace=True)   action
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

[Pytorch Pretrained AlexNet]

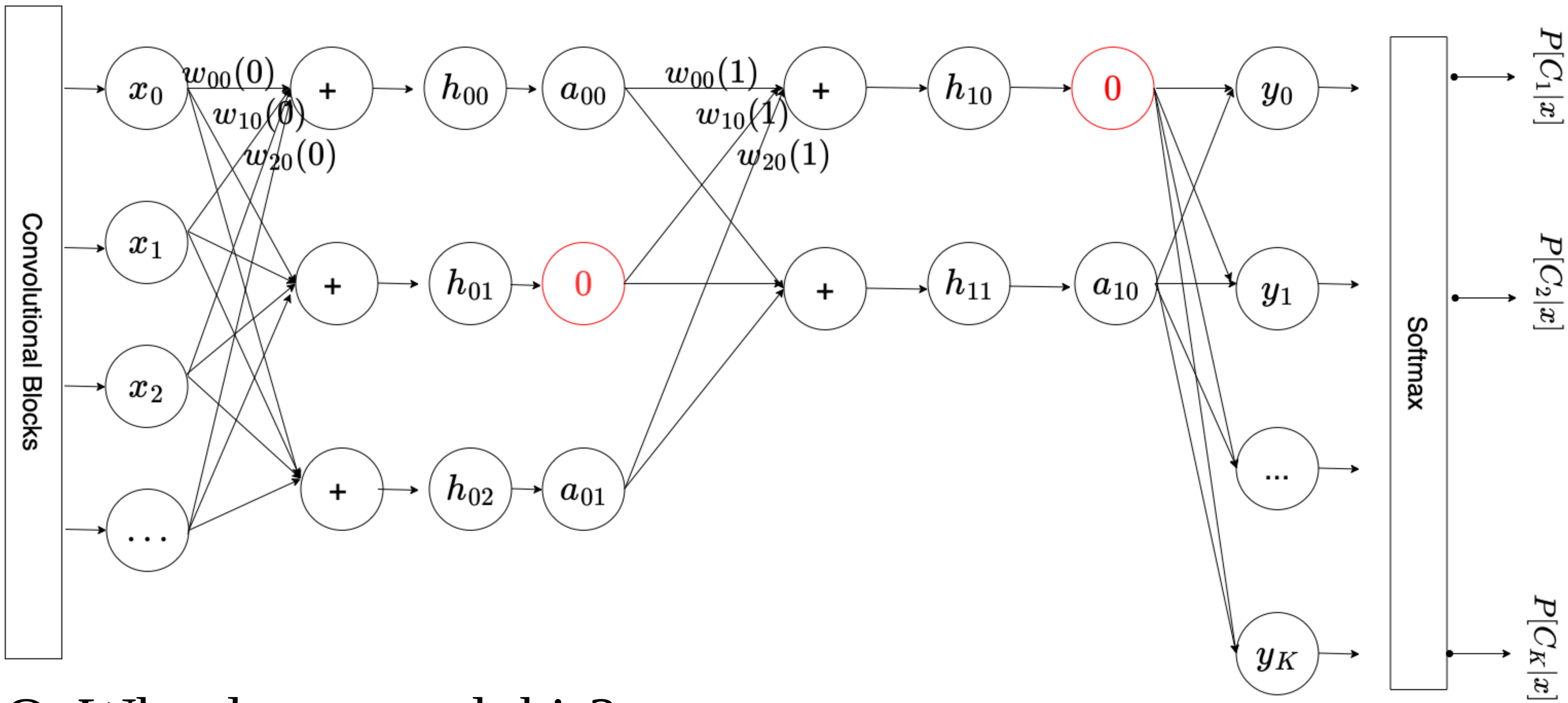
The last few layers of AlexNet
alternate between fully connected layers, activation layers, and dropout.



$\rightarrow p$

[1] Drop Out Layer (0.5)

: 50% of units are set as zero in training process. (training)



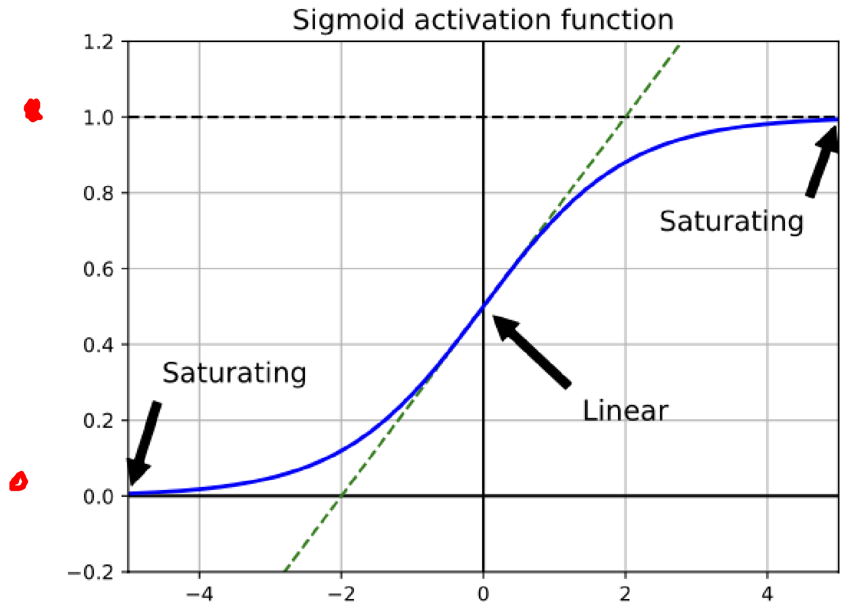
Q: Why do we need this?

How drop out operation is different in training and inference stage?

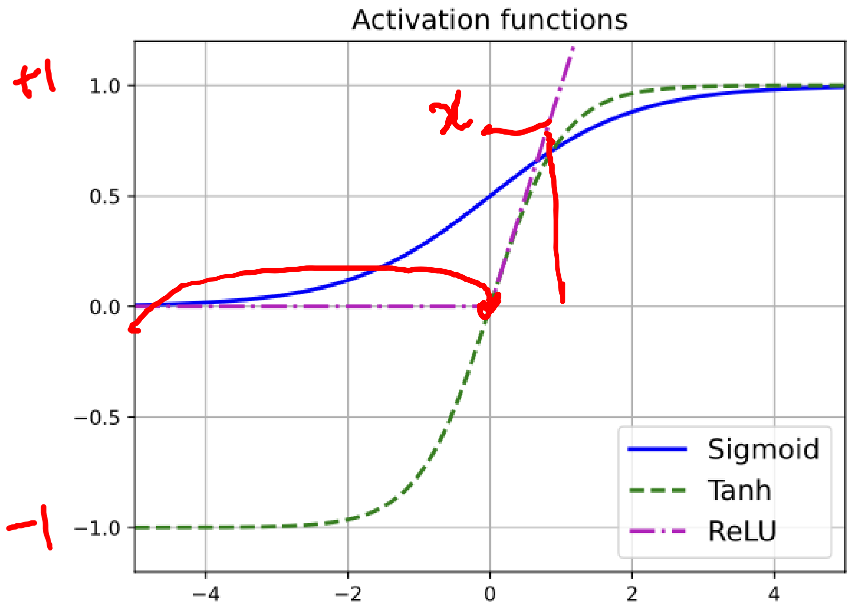
[2] Activation Functions: gives non linearity to neural net

$$\text{Relu}(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Fig 13.2 Textbook Murphy



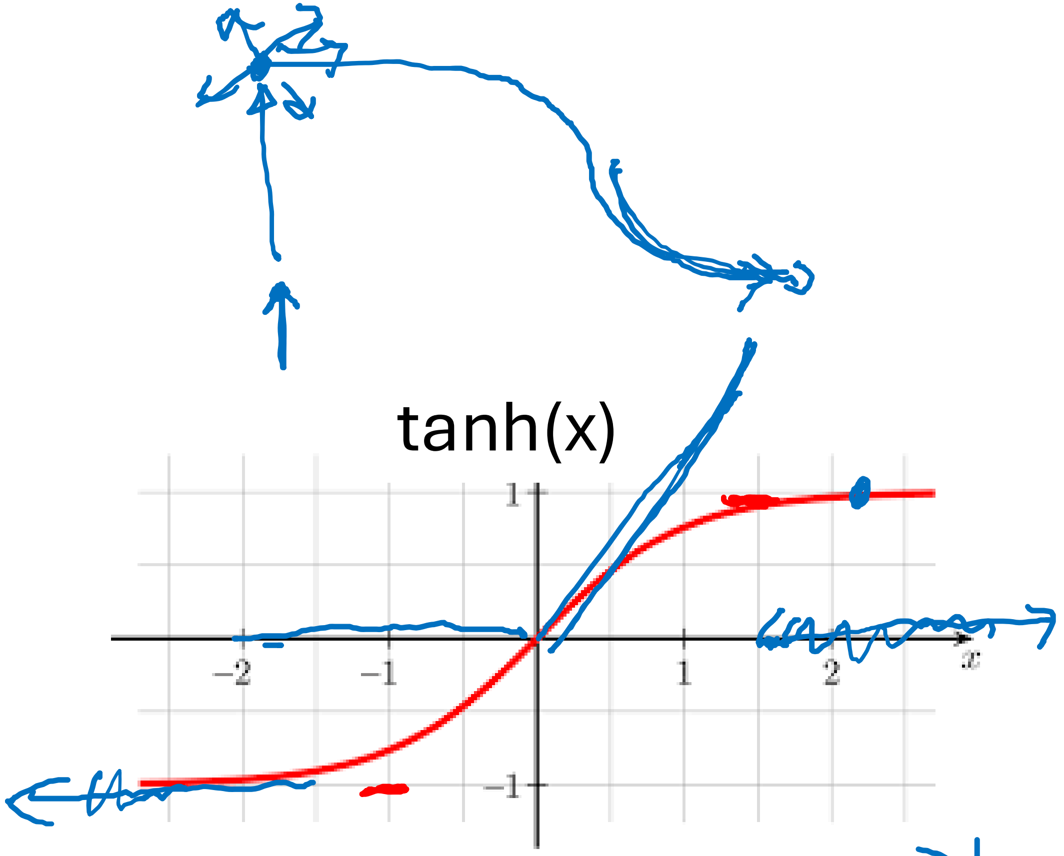
Sigmoid Activation

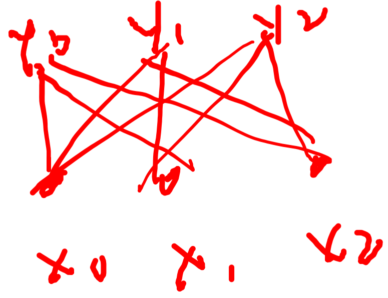


Activation Functions of Neural Nets

- Sigmoid 1
- Tanh 2
- ReLU 3 ✓

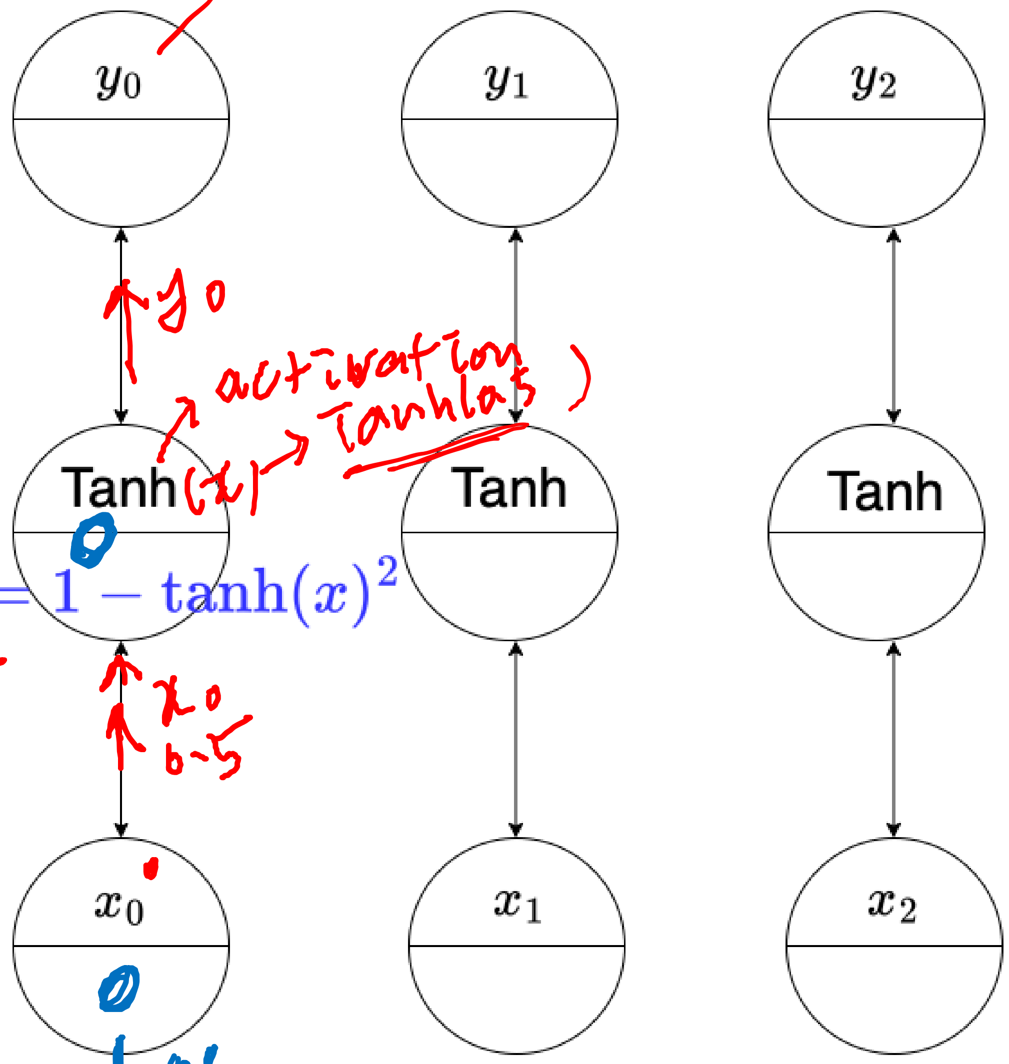
(2) Activation Function:
Tanh (Hyperbolic Tangent Unit)





out

$$\frac{\partial J}{\partial y_0}$$



y_0

activation

$$\frac{dy_0}{dx_0} = 1 - \tanh(x)^2$$

x_0
0.5

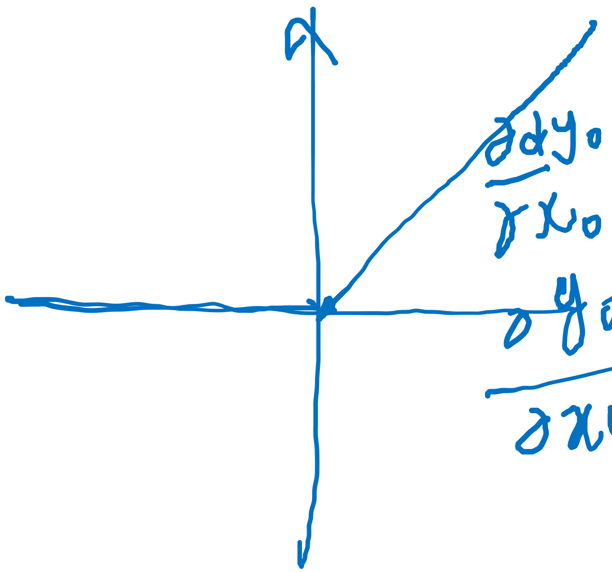
$$\frac{\partial L}{\partial W} = 0$$

$$\frac{dJ}{dx_0} = \frac{dJ}{dy_0} \cdot \frac{dy_0}{dx_0}$$

w

(1) Activation Function: (popular block)
ReLU (**R**ectified **L**inear **U**nit)

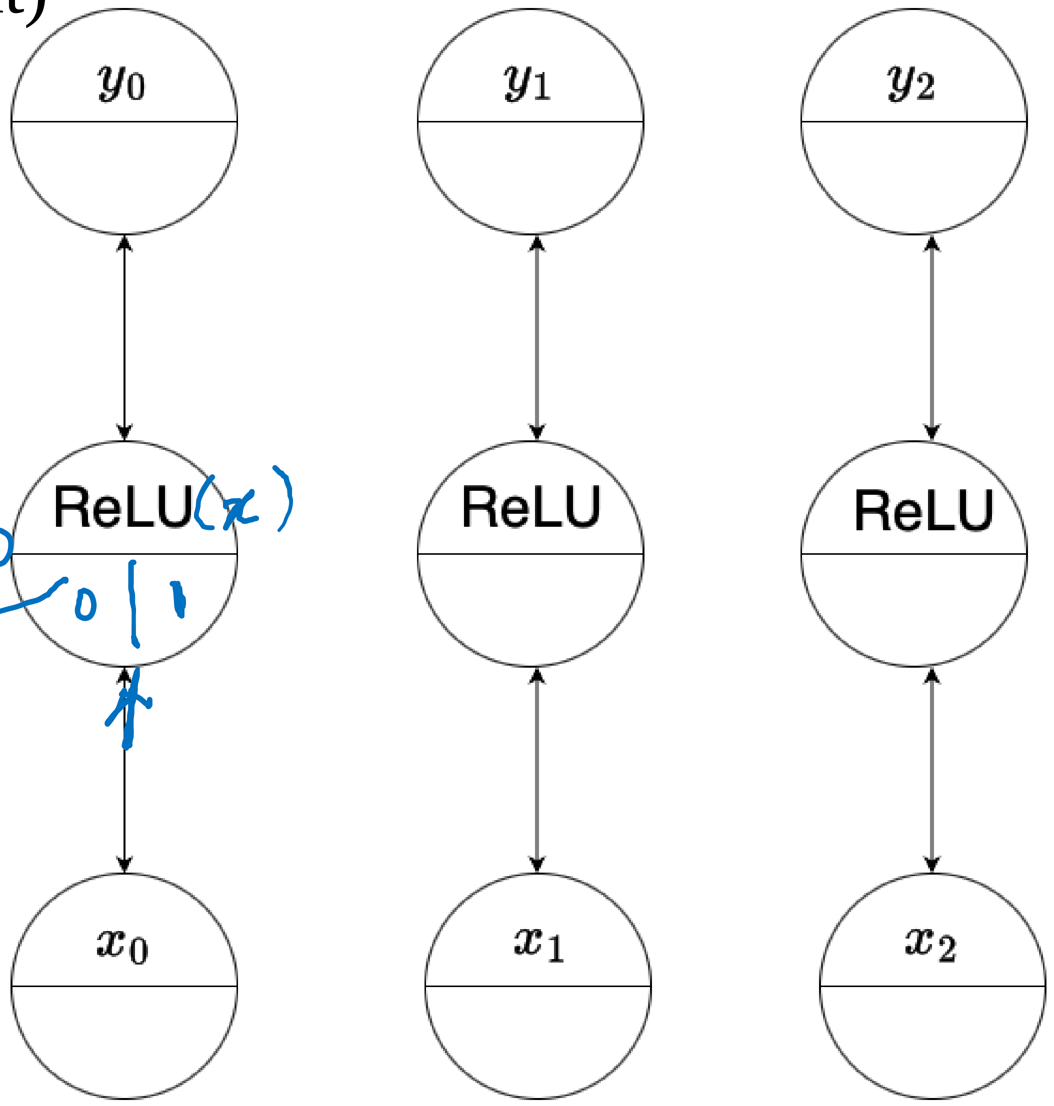
ReLU(x)



$$\frac{\partial y_0}{\partial x_0} = 0 \quad x < 0$$

$$\frac{\partial y_0}{\partial x_0} = 1 \quad x \geq 0$$

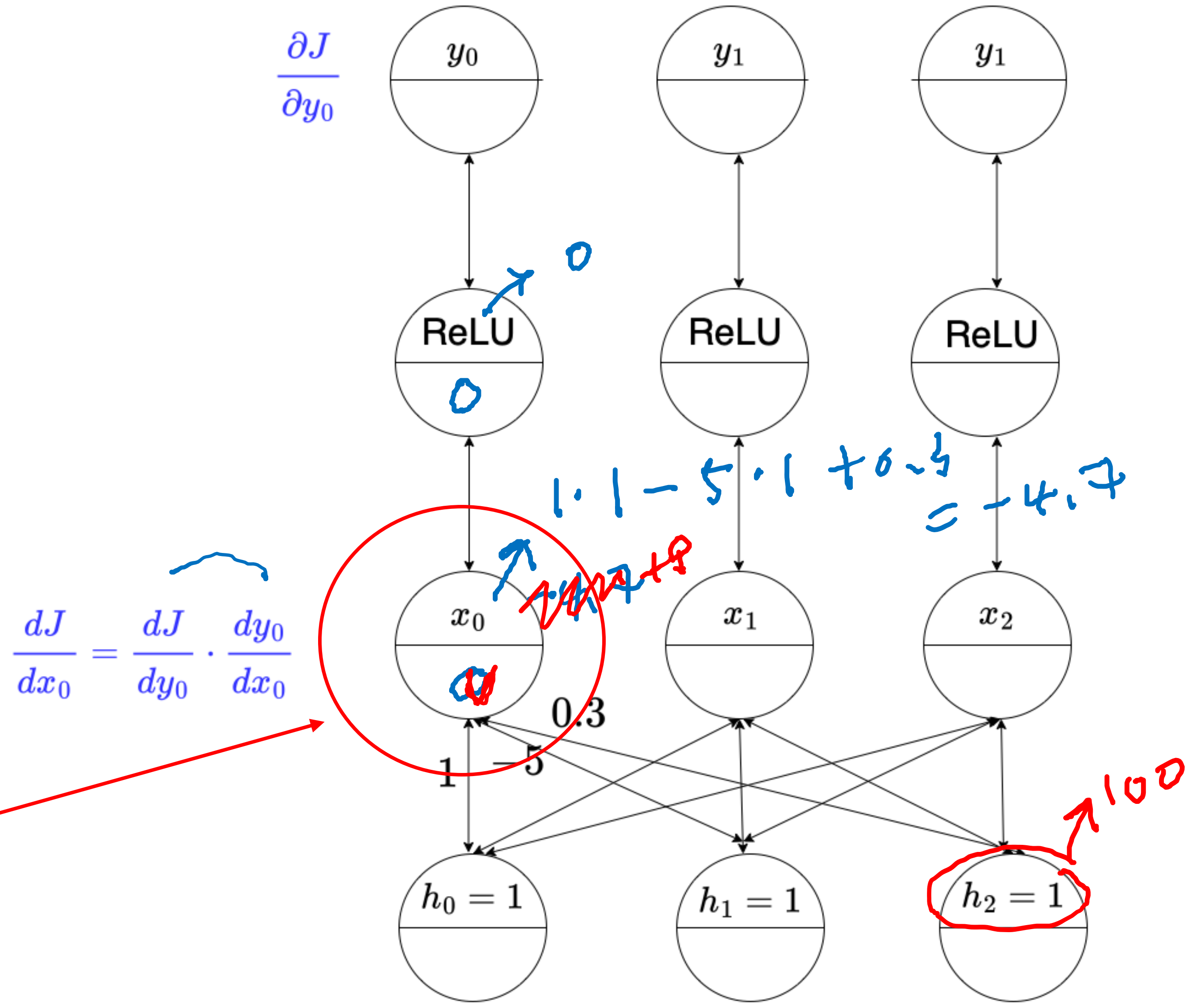
$$\frac{\partial J}{\partial y_0}$$



?

$$\frac{dJ}{dx_0} = \frac{dJ}{dy_0} \cdot \boxed{\frac{dy_0}{dx_0}}$$

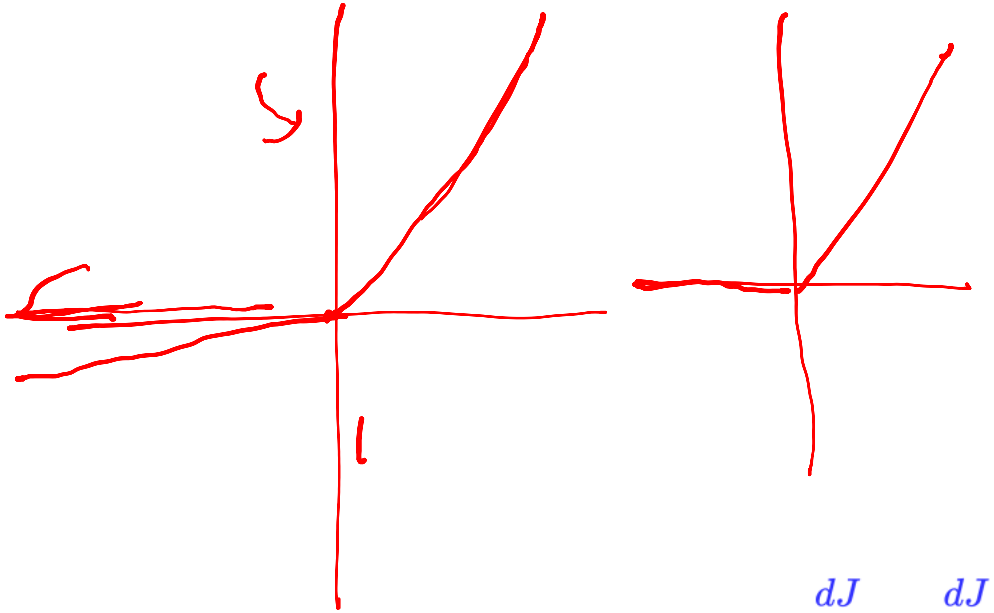
(1) Dead Neuron Case for ReLU



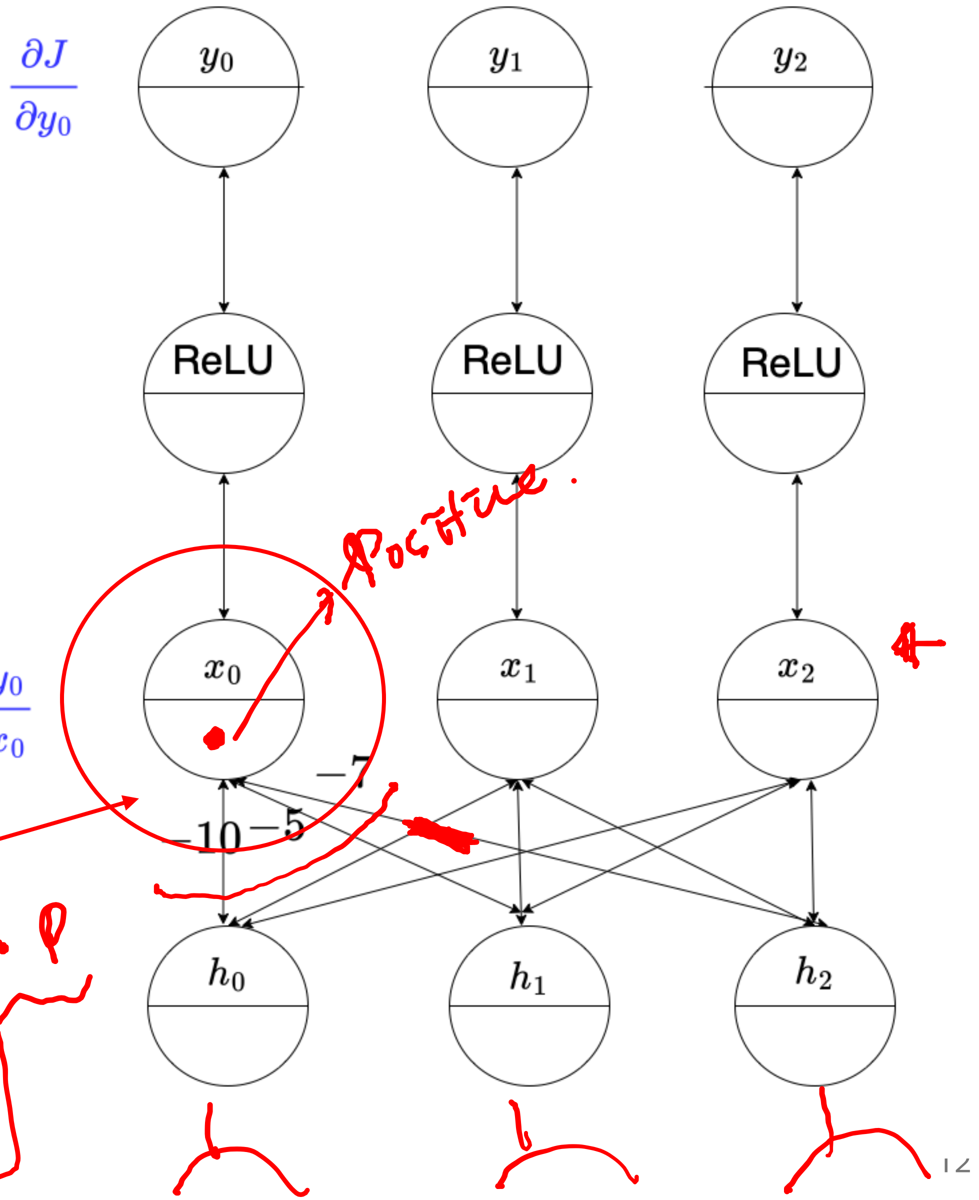
Q: can the unit x_0 $revive$?

(2) Dead Neuron Case for ReLU

leaky Relu



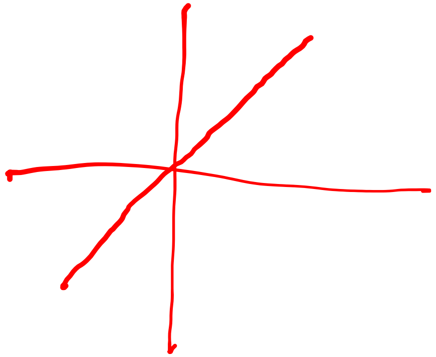
$$\frac{dJ}{dx_0} = \frac{dJ}{dy_0} \cdot \frac{dy_0}{dx_0}$$



$\nabla W = P \cdot P$

$-\eta \times \begin{bmatrix} + \\ + \\ + \end{bmatrix}$

Q: can the unit x_0 *revive*?



The possible reasons for dead neurons:

- Large step size: making all parameters as big negatives.

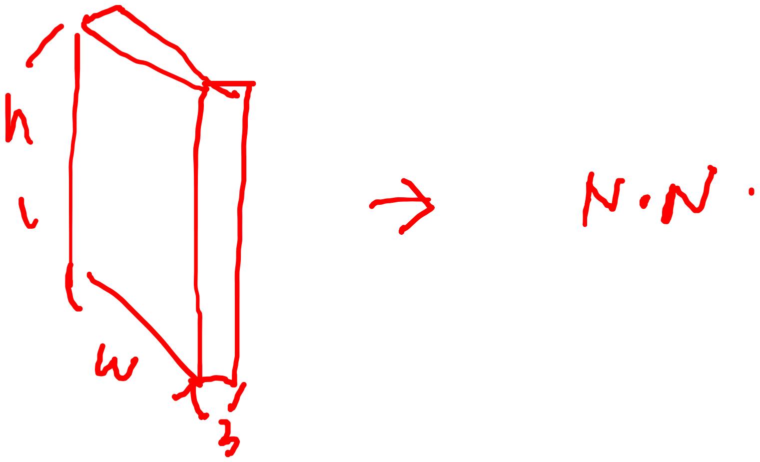
$$W_{t+1} = W_t - \eta \nabla J(W_t)$$

- Large negative bias.

The possible solutions for dead neurons:

- Lowering the step size
- Careful weight initialization
- What if ReLU leak small gradients for the negative values?

Training Schemes for Deep-CNN



(1) Preprocessing of Data

: deep-CNN takes images as input ; there is internal preprocessing steps before facing the first conv block.

Input Transformation in Pytorch (https://github.com/pytorch/examples/blob/main/imagenet/main.py)

[1] input image

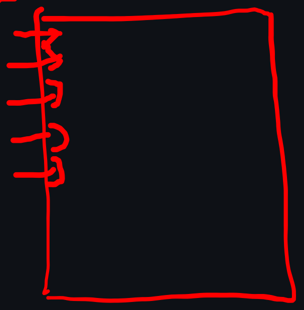
```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))

val_dataset = datasets.ImageFolder(
    valdir,
    transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize,
    ]))
```

Imagenet data

0 - 255 → [0, 1]



p

[Preprocessing for Training]

[Preprocessing for Test]

Q) Why the preprocessing for training/test different?

Q) What the input dimension of the network?

- **RandomResizedCrop (size, scale, ratio, interpolation)**

  : Corp a random portion of image and resize it to a given size

  Size: expected output size of crop

  Scale (s): the lower and upper bounds for the random area of the crop

  Ratio (r): the lower and upper bounds for the random aspect ratio of the crop

Ex) Suppose Scale = (0.08, 1.0)

Ratio= (0.75, 1.33)

Then, one possible crop example is

s=np.uniform (0.08,1.0) r=np.uniform(0.75,1.33)

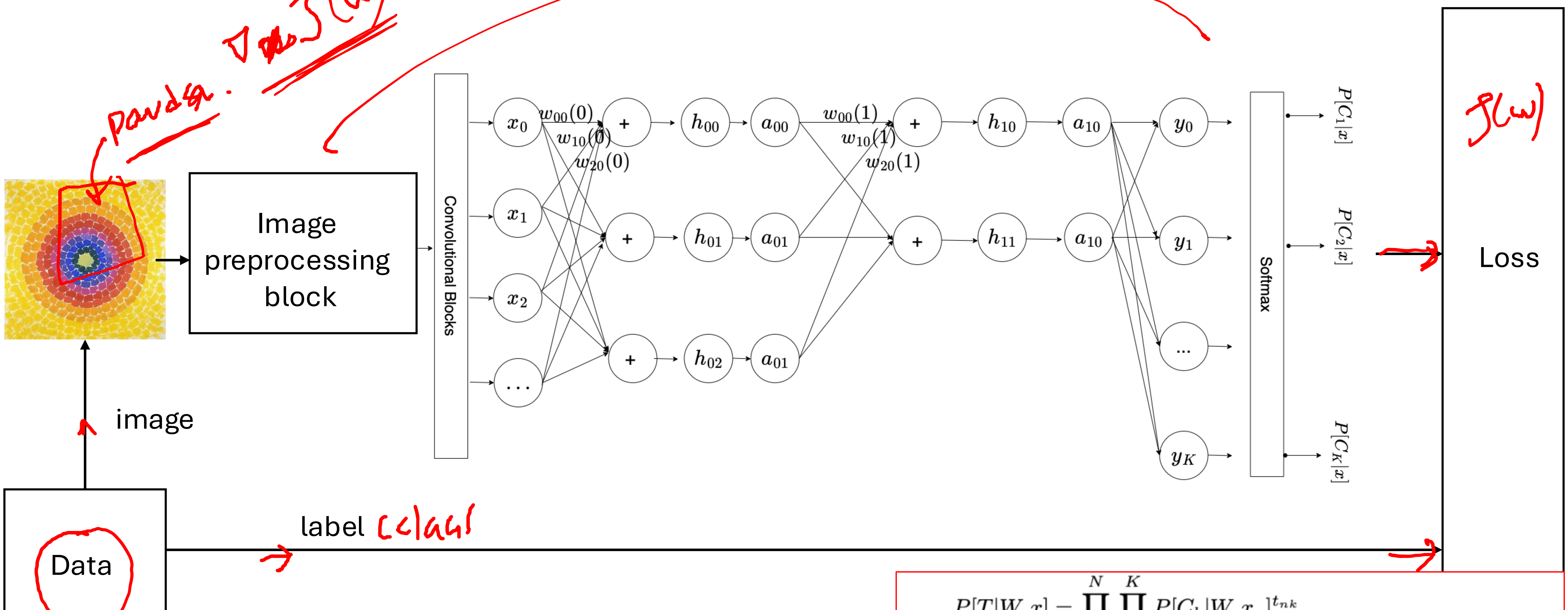
A= original image area × s # the area to crop

W= $\sqrt{A \times r}$ # the area to crop # width to crop

H = $\sqrt{A/r}$ # the area to crop # height to crop

$W \times H = A = \sqrt{A \times r \cdot A / r} = A$

Overall architecture of Deep CNN



The loss function with softmax
: cross entropy

$$P[T|W,x] = \prod_{n=1}^{N}\prod_{k=1}^{K} P[C_k|W,x_n]^{t_{nk}}$$

$$J(W) = -\ln P[T|W,x] = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln P[C_k|W,x_n]$$

(2) Optimization Methods

Neural Networks are trained (updating parameters) by using iterative and gradient based algorithms.

$$W_{t+1} = W_t - \eta_t D_t$$

(Descent Direction)

(Step Size)

One iteration in training ($D_t = \nabla J(W_t)$ and $\eta_t =$ constant)

#100

1. One batch data D flows into a neural net in the feedforward direction.

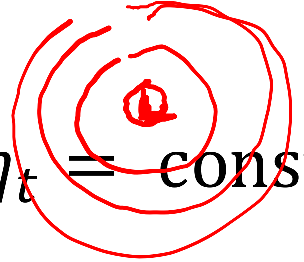
   : compute loss $J(w_t, D)$ at the w_t.

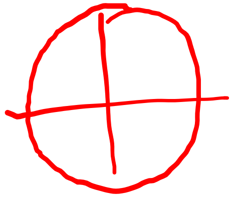
2. Backpropagation algorithm to compute gradients $\nabla J(W_t)$

3. Updates W: $W_{t+1} = W_t - \eta \nabla J(W_t)$

- Steepest Descent Algorithm ($D_t = \nabla J(W_t)$ and $\eta_t =$ constant)

  $W_{t+1} = W_t - \eta \nabla J(W_t)$





suppose.

large

converge rate

$$J(w) = \frac{1}{2} w^t Q w + b^t w + c$$

From bishop 286p

small

$\uparrow \approx 1$

- Steepest descent convergence rate with line search :

$$\frac{||J(w_{k+1}) - J(w*)||}{||J(w_k) - J(w*)||} \leq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2 = \left(\frac{\frac{\lambda_{max}}{\lambda_{min}} - 1}{\frac{\lambda_{max}}{\lambda_{min}} + 1}\right)^2 = \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 \approx 0$$

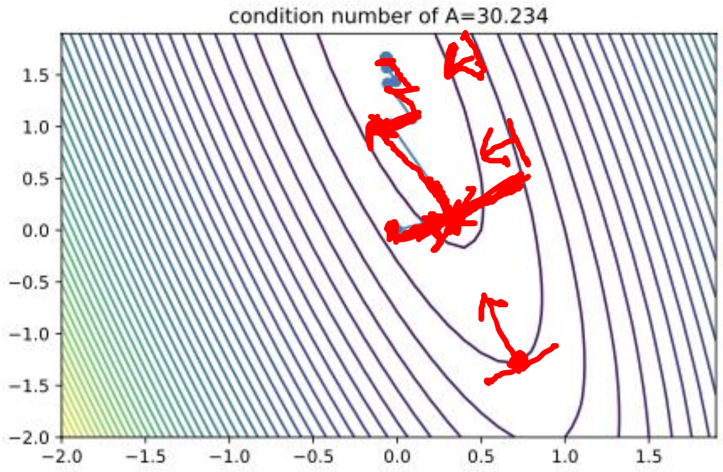
[the eigenvalue of Q matrix in the quadratic function]

$\kappa \uparrow$ large

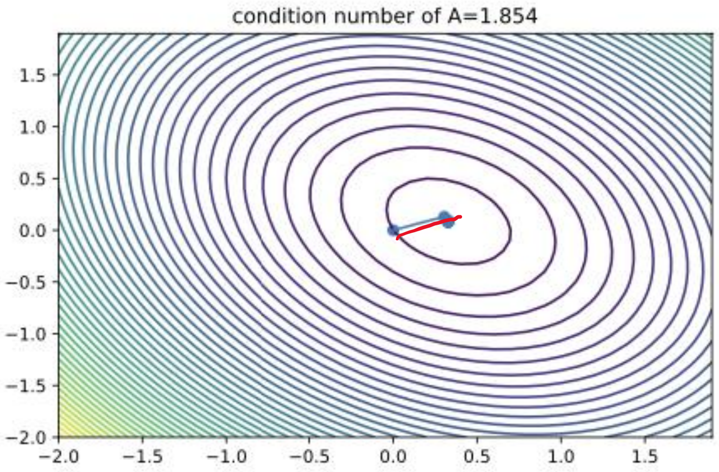
The condition value $\kappa = \frac{\lambda_{max}}{min}$ measures how skewed the space is, in the sense of being far from a symmetrical bowl. Hence, for a skewed quadratic objective function, steepest descent algorithm can be quite slow.

Steepest Descent Convergence Speed for different condition number κ

From bishop 8.2.4

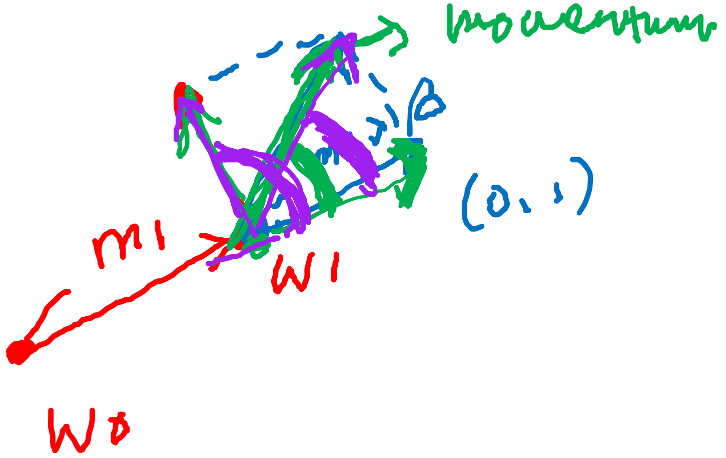


Large $\kappa(30.234)$: Slow



Small $\kappa(1.854)$: Fast

- Steepest Descent Algorithm with Momentum ✓



D_0 $\quad$ D_{t-1}

$$W_{t+1} = W_t - \eta\,(\beta m_t + \nabla J(W_t))$$

0.8 or 0.9

$m_{t+1} : \beta m_t + \nabla J(W_t)$ *(the new direction at* $t+1$ *step)*

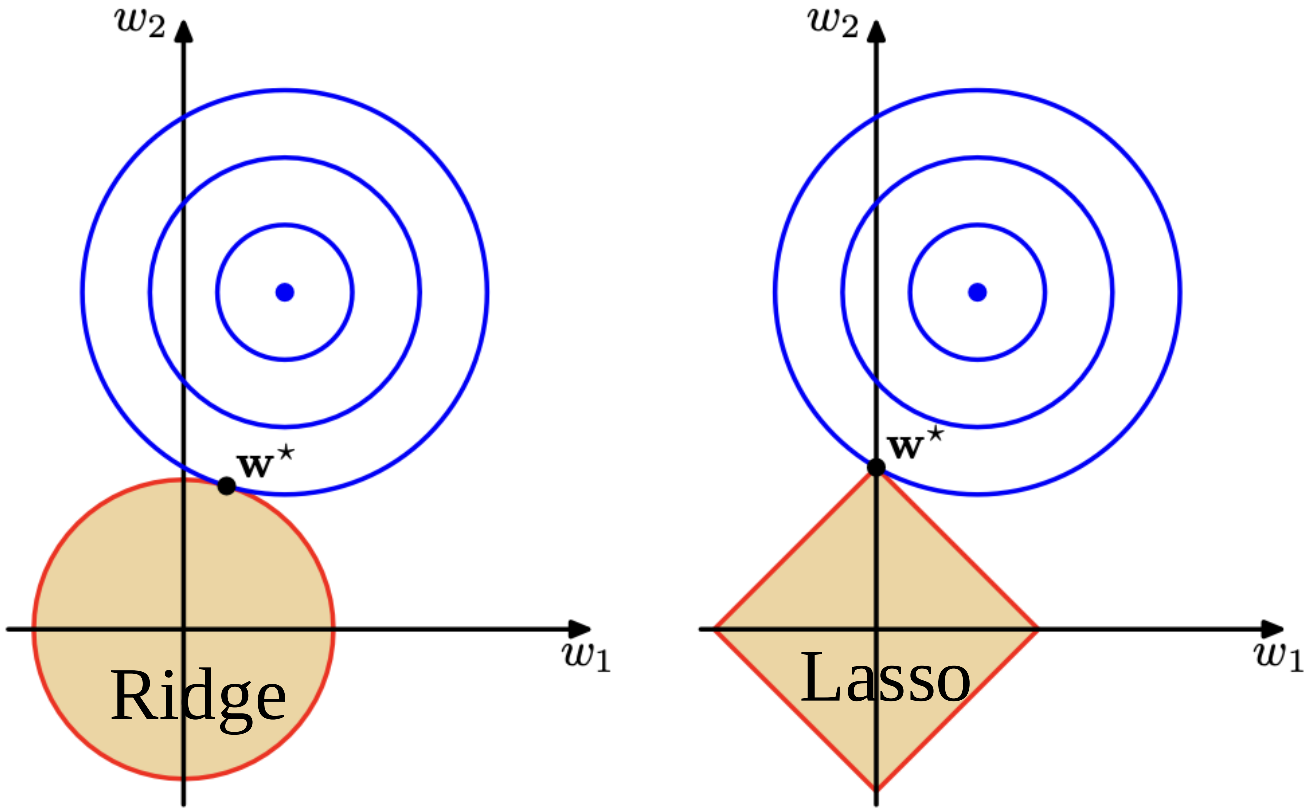
β: momentum [0,1)

Stochastic Gradient Descent with Momentum is a popular optimization choice to achieve faster convergence.

(3) Regularization for DNN

To control the effective complexity

Geometric Interpretation of Ridge / Lasso Regression [Sept 23rd Slide]



- as λ getting bigger
  the constraint range getting smaller!
- Q: which one gives a sparse solution?

From Bishop Chap Figure 3.4

+ the constraints regulate the magnitude of W, so the model complexity. Lasso gives sparse solution.

- Lasso Regularization

$\nabla J_w(x.y) + C \cdot w.$

$w_t - \eta \Big(\quad \Big)$

$$\tilde{J}_w(x,y) = J_w(x,y) + C||w||^2$$

$$\nabla \tilde{J}_w(x,y) = \nabla J_w(x,y) + Cw$$

$$w_{t+1} = w_t - \eta C w_t - \eta \nabla J_{w_t}(x,y)$$

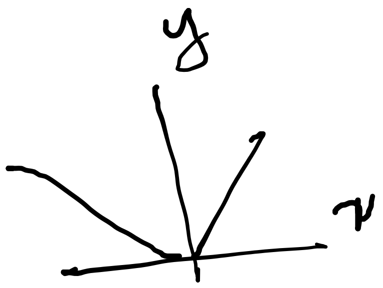
$$w_{t+1} = (1 - \eta C) w_t - \eta J_{w_t}(x,y)$$

+ The updating rule with ridge constraint

$$W_{t+1} = 1 \cdot W_t - \eta \nabla J(W_t)$$

+ without ridge constraint

- Ridge Regularization



J'

$\frac{\partial}{\partial x} f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

$$\tilde{J}_w(x,y) = J_w(x,y) + C||w||$$

$$\nabla \tilde{J}_w(x,y) = \nabla J_w(x,y) + C\text{sign}(w)$$

$$w_{t+1} = w_t - \eta C\text{sign}(w_t) - \eta \nabla J_{w_t}(x,y)$$

$$W_{t+1} = W_t - \eta \nabla J(W_t)$$

+ The updating rule with ridge constraint

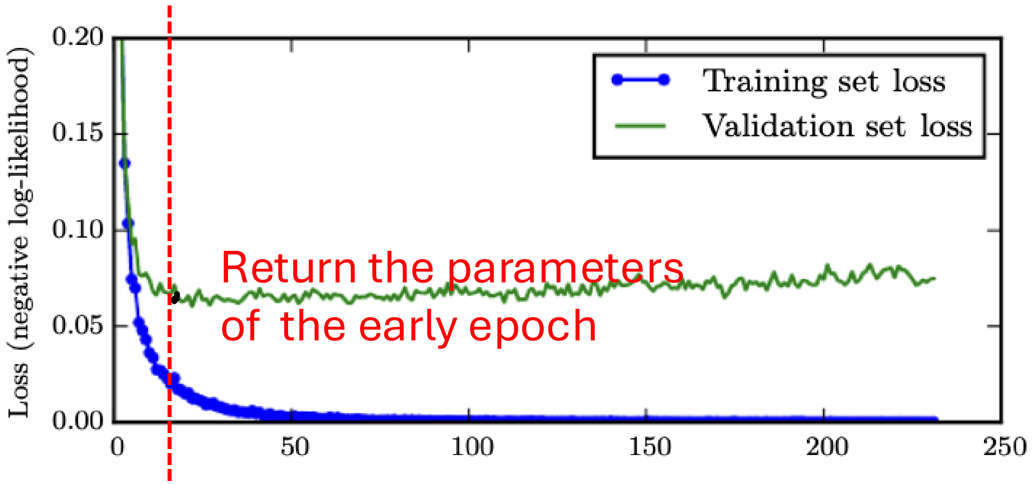
+ without ridge constraint

- Early Stopping (effective and simple regularization method)

When training large models with sufficient representational capacity, we often observe that training error decreases steadily over time, but validation error begins to rise again.

When the training algorithm terminates,
we return the parameters at the moment of the lowest validation error.

Loss (Negative Log Likelihood)



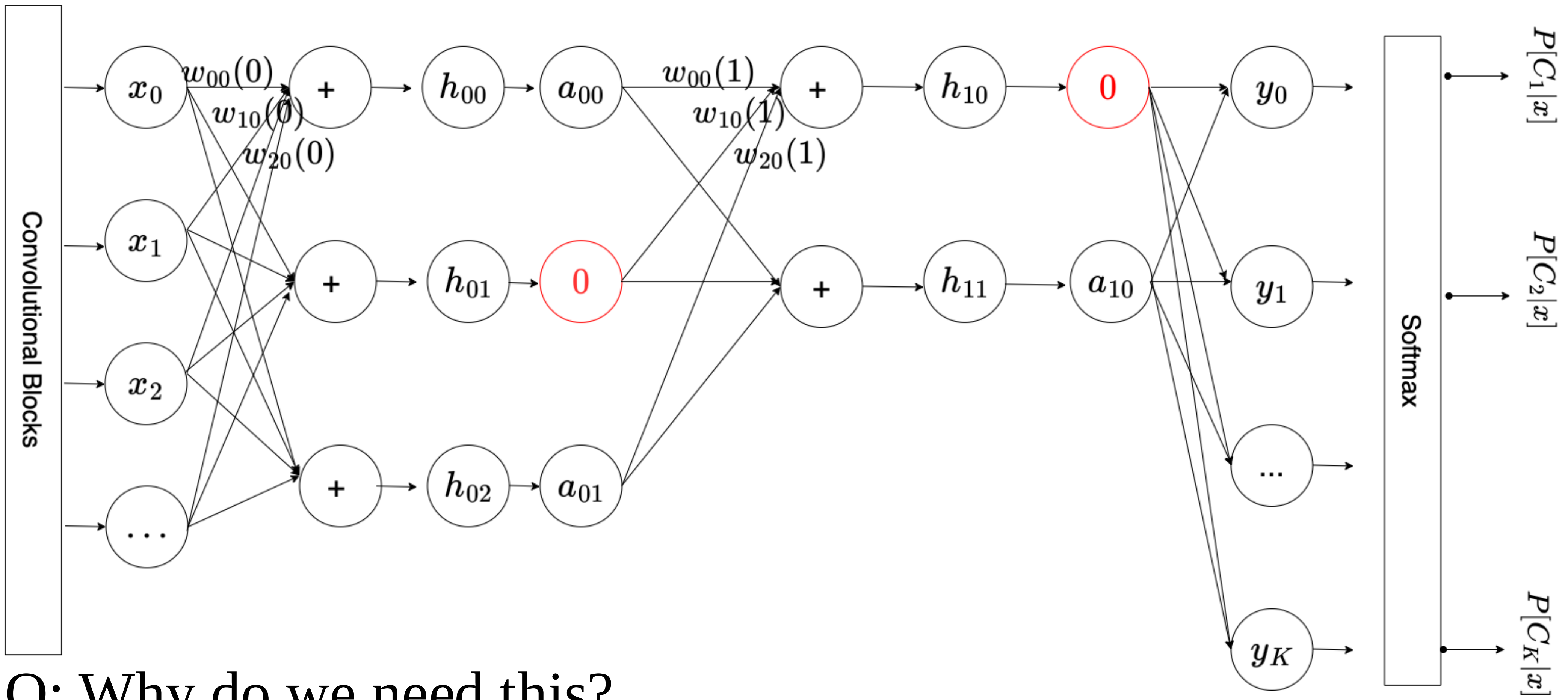
epochs

One epoch: work through entire training set.
Q: Suppose we have 3,500 data samples. but if batch size is 50, then how many steps are in one epoch?

100 step
save parameter

- w 0
- w 100
- w 200
- w 300

- Drop Out Layer (0.5)
  : 50% of units are set as zero in training process.



Q: Why do we need this?

How drop out operation is different in training and inference stage?

(4) Choosing the Right Batch size

it can vary depending on the task, model architecture, and hardware limitations.

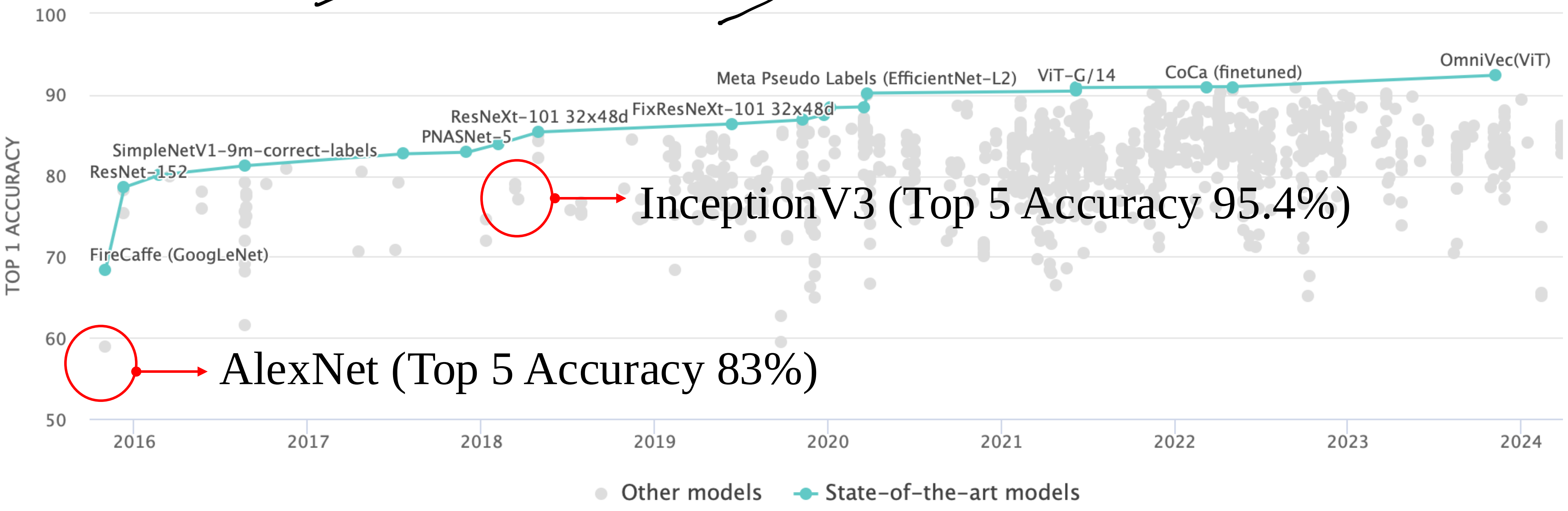
- **small** batch size(1 – 32 data samples)
  + pro: good generalization and less memory use
  + cons: longer training time

- **large** batch size(more than 512 data samples)
  + pro: fast and parallelism (GPU: Graphic Processing Unit)
  + cons: less generalization (bring less stochasticity)

$J(w)$ / small subset of traing

Deep CNN object recognition performance and generalization to unseen data

Comparison the performance of large scale image classification method with the performance of humans on this task
(H1: top-five 94.9% and H2: top-five 88%)



CNN models achieved human level of performance on object recognition. (ImageNet)

Human Annotation Interface

(from the paper: ImageNet Large Scale Visual Recognition Challenge, Olga Russakovsky et. al)

- One test set image and a list of 1000 ILSVRC categories on the side.
- Categories are sorted in the topological order of the ImageNet hierarchy, which places semantically similar concepts nearby in the list. (ex 120 kinds breeds: Beagle, Border terrier, Scottish deerhound, Shih-Tzu, etc)

- The user of the interface selects 5 categories from the list by clicking on the desired items.
- They found the task of annotating images with one of 1000 categories to be an extremely challenging task for an untrained annotator.

- H1 trained on 500 images and annotated 1500 test images.
- H2 trained on 100 images and then annotated 258 test images.

ImageNet Inference Demonstration

[Wikipedia: European Rabbit]

[“Object” by Meret Oppenheim , 1936]

- AlexNet Prediction

```
hare 0.9713024497032166
wood rabbit 0.02861912176012993
wallaby 1.7613450836506672e-05
ibex 1.6985746697173454e-05
fox squirrel 6.545086307596648e-06
```

- Inception V3 Prediction

```
hare 0.8869689106941223
wood rabbit 0.032969459891319275
Angora 0.0008774788584560156
sarong 0.0006217487971298397
ibex 0.0004210647603031248
```

Consistent Results

Top Five Inference Results

- AlexNet

```
mortar 0.69256192445755
cup 0.03658083826303482
hook 0.02810201235115528
mushroom 0.018149923533201218
spindle 0.017833007499575615
```

- VggNet11

```
bath towel 0.32794177532196045
wool 0.0928330346941948
cup 0.08141378313302994
pug 0.03363395854830742
hair slide 0.025715012103319168
```

Different models give different classification results.

Adversarial Examples

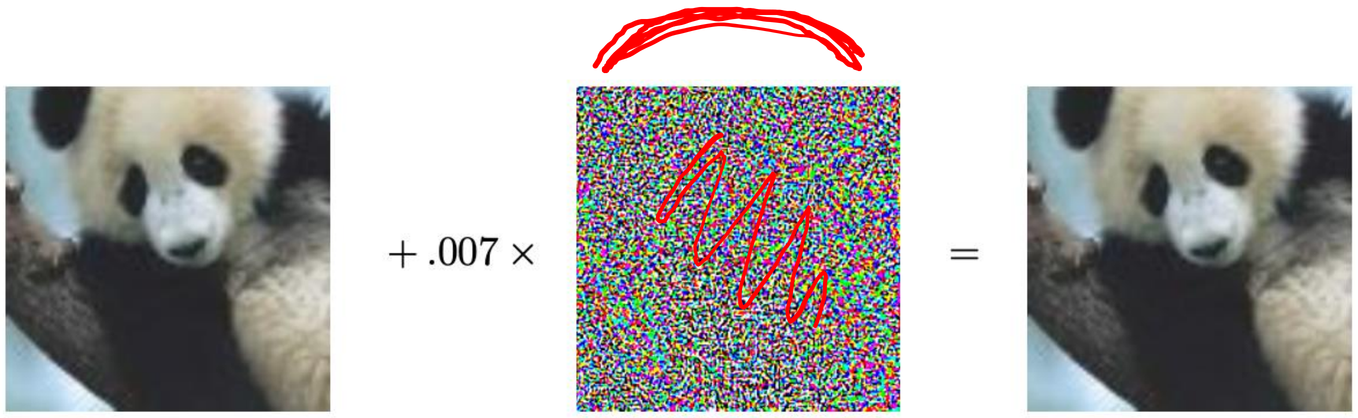
(Unusual Mistake DNN makes)

Adversarial Example is the example
that has been carefully computed to be misclassified. (classifier)
to be not detected. (object detector)

- Adversarial Example

From the paper
``EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES” Ian J. Goodfellow et. al

“Panda” 57.7% | “Nematode” Confidence | “Gibbon” 99.3%



$\boldsymbol{x}$
“panda”
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“nematode”
8.2% confidence

$\boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
99.3 % confidence

Human observer cannot tell the differenced between original example and the adversarial example, but the network can make highly different prediction.

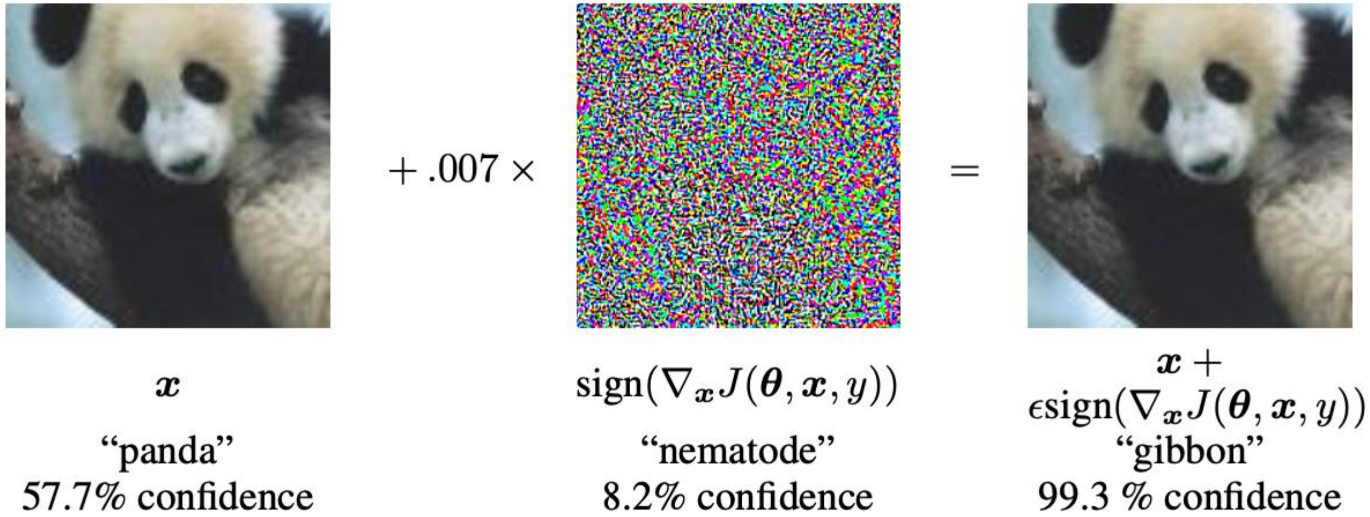
[Gibbon]

From the paper
``EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES” Ian J. Goodfellow et. Al

“Panda” 57.7%

“Nematode” Confidence

“Gibbon” 99.3%



Fast Gradient Sign Method

$$\tilde{x} = x + \epsilon \cdot \mathrm{sign}(\nabla_x J(w, x, y))$$

Reference: https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

Resource from : https://www.iangoodfellow.com/slides/2016_10_27_uber.pdf

Turning Objects into Airplanes

They are recognized as airplanes with high confidence but no difference is detected form human observations.

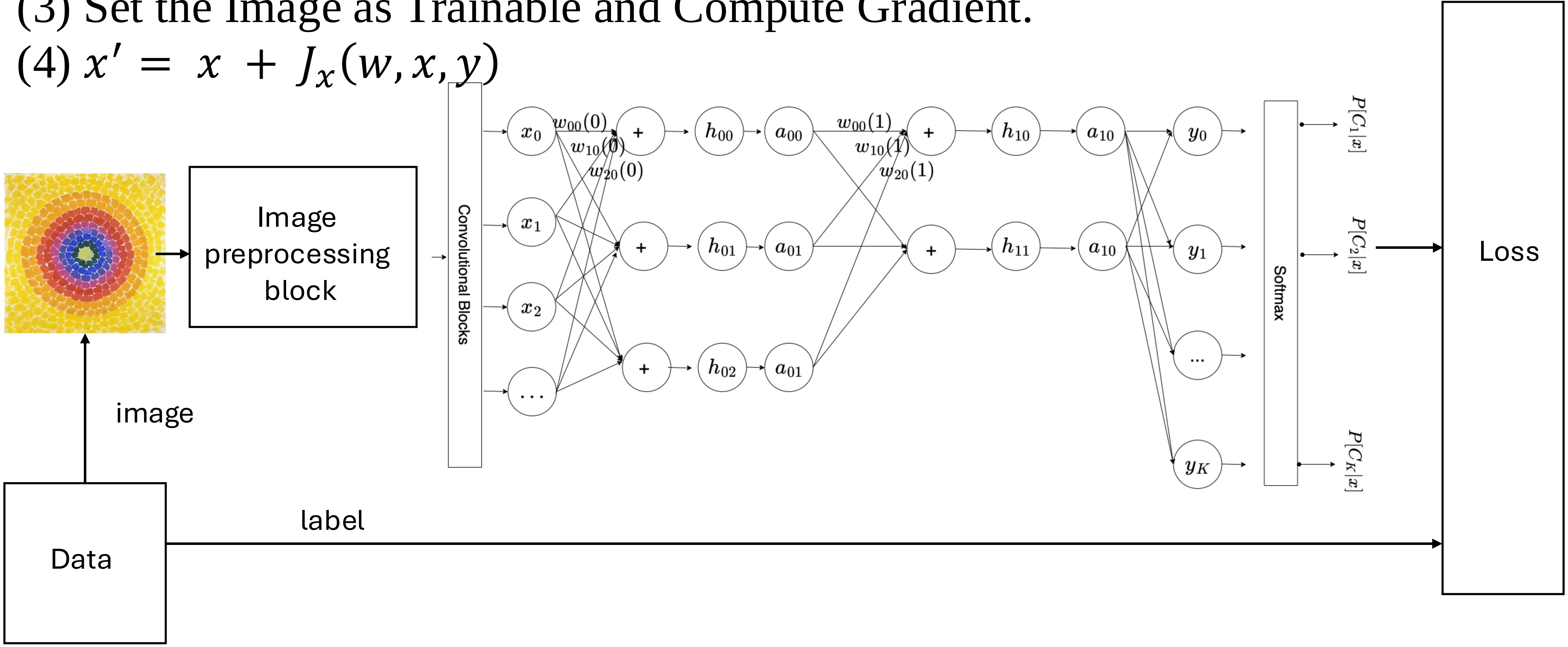
Q: How we can generate the adversarial examples?

(1) Load a pretrained DNN

(2) Freeze the parameters

(3) Set the Image as Trainable and Compute Gradient.

(4) $x' = x + J_x(w, x, y)$



Resource from : https://www.iangoodfellow.com/slides/2016_10_27_uber.pdf

Nearly Linear Responses in Practice