Deep learning

Dr. Aissa Boulmerka
a.boulmerka@centre-univ-mila.dz

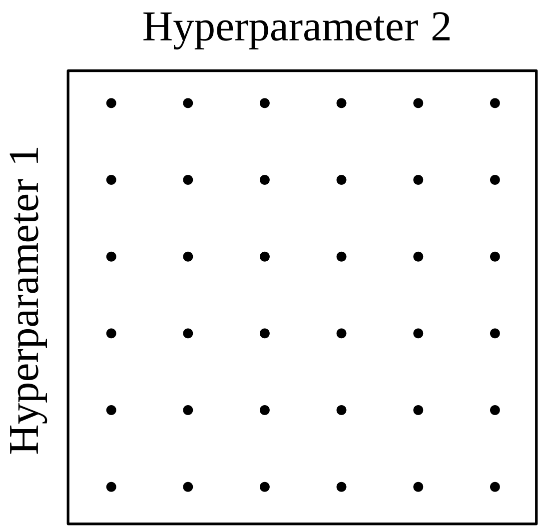
2023-2024

CHAPTER 6
HYPERPARAMETER TUNING, BATCH NORMALIZATION AND PROGRAMMING FRAMEWORKS

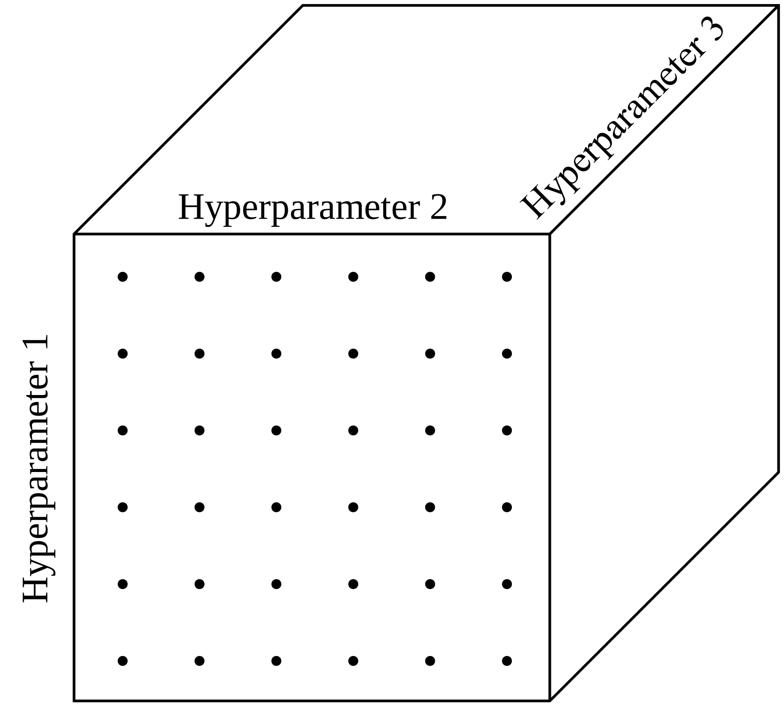
Tuning process

- We need to **tune** our hyperparameters to get the best out of them.
- Important hyperparameters are :
    i. Learning rate (α).
    ii. Momentum beta (β).
    iii. Mini-batch size.
    iv. Number of hidden units.
    v. Number of layers.
    vi. Learning rate decay.
    vii. Regularization lambda.
    viii. Activation functions.
    ix. Adam beta1 & beta2 .
- Its hard to decide which hyperparameter is the **most important in a problem**. It depends on your problem.

Try random values: Don’t use a grid

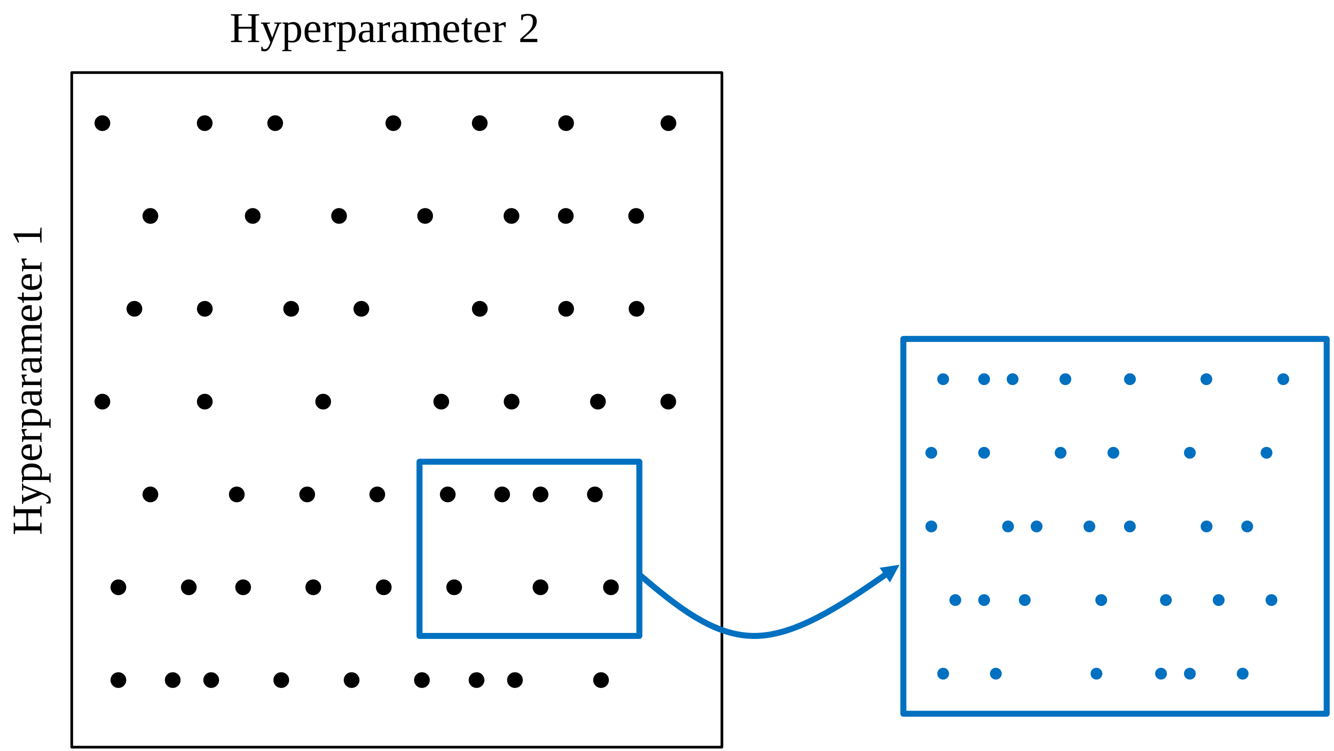


GRID SEARCH



- One of the ways to tune is to sample a **grid with N hyperparameter** settings and then try **all settings combinations** on your problem.
- **PROBLEM**: One iteration takes a long time ==> **NOT AS IMPORTANT.**
- **SOLUTION**: Try **random values**: don't use a grid.

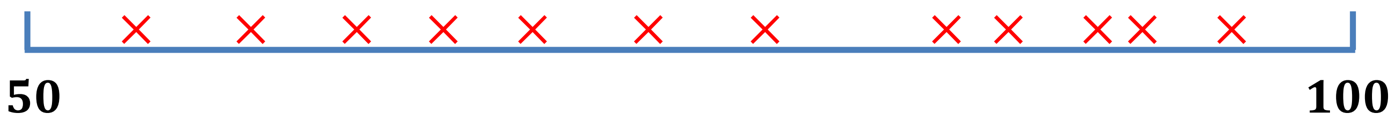
Coarse to fine



- You can use **Coarse to fine** sampling scheme :
  - When you find some hyperparameters values that give you a better performance : **zoom into** a smaller region around these values and sample more densely within this space.
- These methods can be **automated**.

Picking hyperparameters at random

$$\boldsymbol{n^{[l]} = 50, \cdots, 100}$$

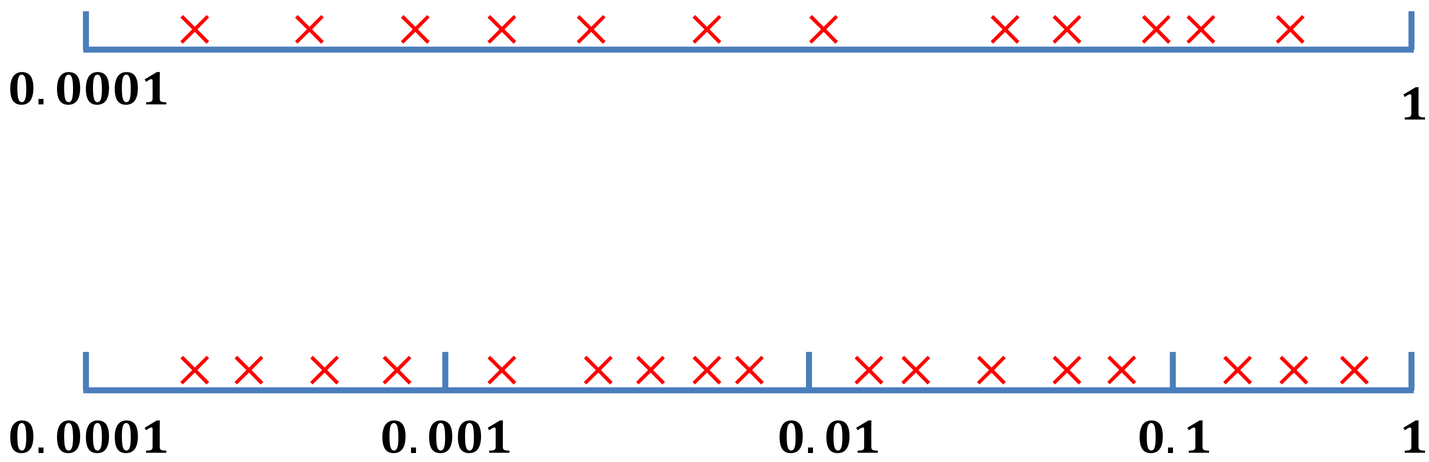


#layers : $\boldsymbol{L = 2, \cdots, 5}$

$$\boldsymbol{2, 3, 4, 5}$$

Appropriate scale for hyperparameters

$$\alpha = 0.0001, \cdots, 1$$



Using an appropriate scale to pick hyperparameters

- Let's say you have a specific range for a hyperparameter from "a" to "b". It's better to search for the right ones using the logarithmic scale rather than in linear scale:

```
Calculate: a_log = log(a) # e.g. a = 0.0001 then a_log = -4
Calculate: b_log = log(b) # e.g. b = 1 then b_log = 0
Then:
r = (a_log - b_log) * np.random.rand() + b_log
# In the example the range would be from [-4, 0] because rand range [0,1)
result = 10^r
```

- It uniformly samples values in log scale from [a,b].

Hyperparameters for exponentially weighted averages

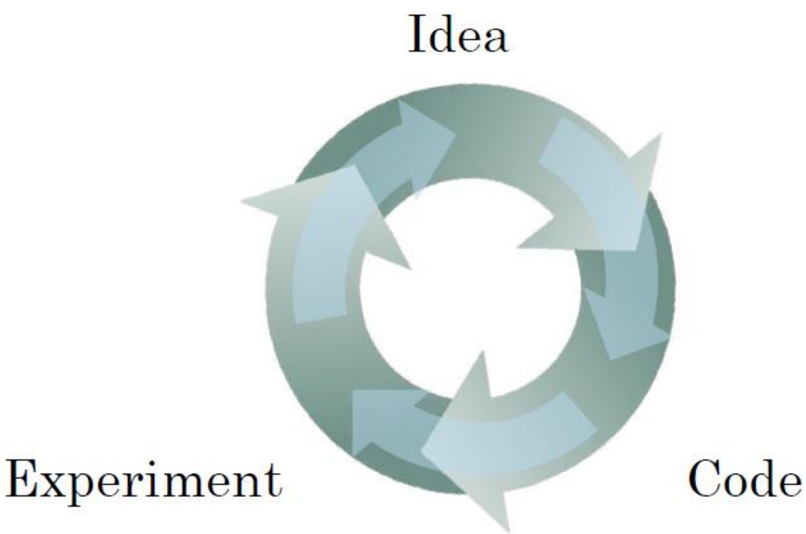
- If we want to use the last method on exploring on the "momentum beta":
  - Beta (β) best range is from 0.9 to 0.999.
  - You should search for 1 - beta in range 0.001 to 0.1 (1 - 0.9 and 1 - 0.999) and then use

    a = 0.001 and b = 0.1

  - Then:

```
a_log = -3
b_log = -1
r = (a_log - b_log) * np.random.rand() + b_log
beta = 1 - 10^r   # because 1 - beta = 10^r
```

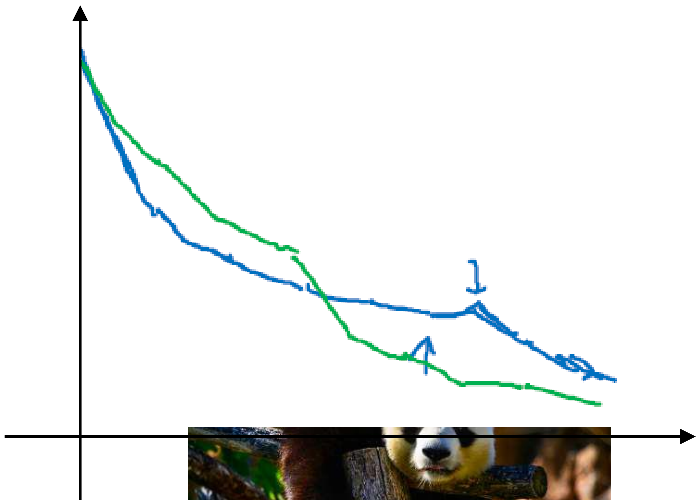
Re-test hyperparameters occasionally



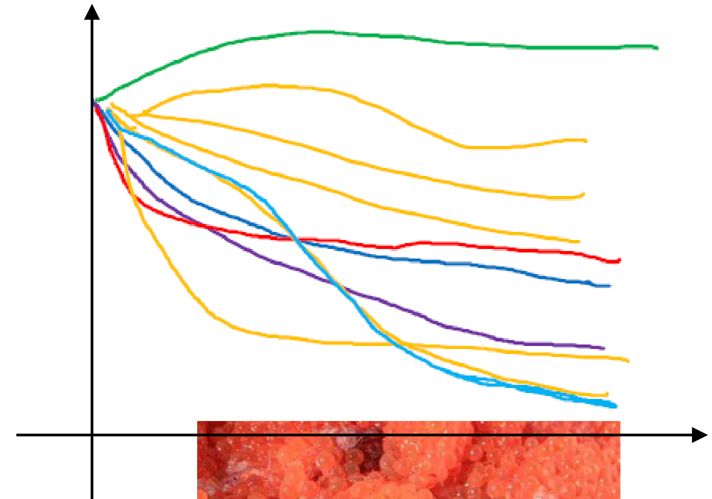
- NLP, Vision, Speech, Ads, logistics, ….
- Intuitions do get stale.
- Re-evaluate occasionally.

- Intuitions about hyperparameter settings from one application area may or may not transfer to a different one.

Hyperparameters tuning in practice: Pandas vs. Caviar



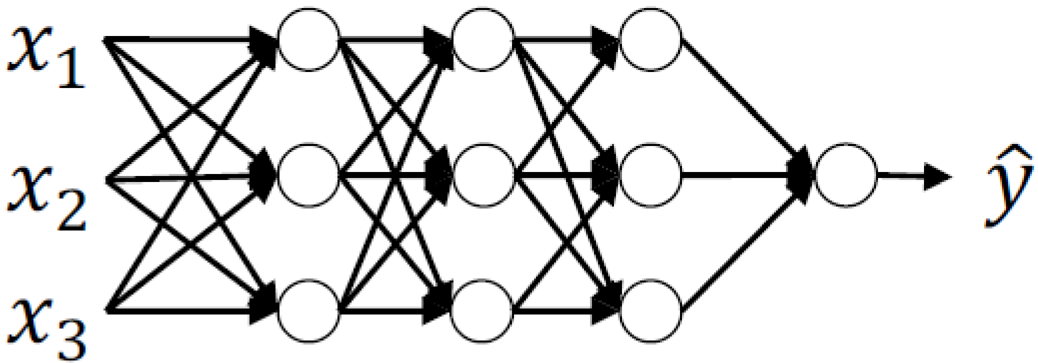
Panda



Caviar

- If you don't have much computational resources you can use the "babysitting model":
  - Day 0 you might initialize your parameter as random and then start training.
  - Then you watch your learning curve gradually decrease over the day.
  - And each day you nudge your parameters a little during training.
  - Called panda approach.
- If you have enough computational resources, you can run some models in parallel and at the end of the day(s) you check the results.
  - Called Caviar approach.

Batch Normalization



$$\mu = \frac{1}{m}\sum_i x^{(i)}$$
$$X = X - \mu$$
$$\sigma^2 = \frac{1}{m}\sum_i x^{(i)^2}$$
$$X = X/\sigma^2$$

- Batch norm is one of the most important ideas in the rise of deep learning (*).
- Batch Normalization speeds up learning.
- Before we normalized input by subtracting the mean and dividing by variance. This helped a lot for the shape of the cost function and for reaching the minimum point faster.
- The question is: for any hidden layer can we normalize $A^{[l]}$ to train $W^{[l]}, b^{[l]}$ faster? This is what batch normalization is about.

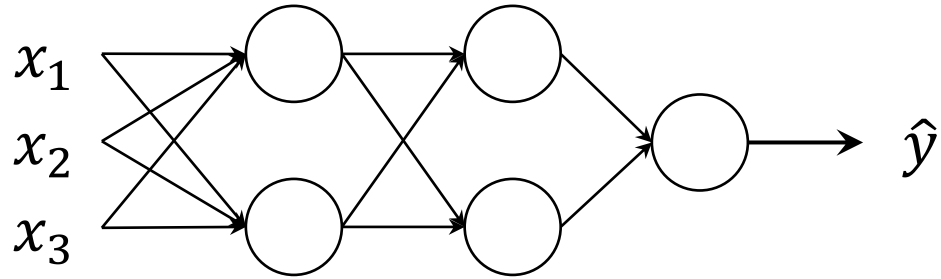
(*) Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *International conference on machine learning*. ICML, 2015.

Implementing Batch Norm

Given $Z^{[l]} = \left[z^{(1)}, \cdots, z^{(m)}\right]$, $i \ = \ 1 \ to \ m$ (for each input)

- Compute the **mean**: $\mu = \frac{1}{m}\sum_i z^{(i)}$
- Compute the **variance**: $\sigma^2 = \frac{1}{m}\sum_i \left(z^{(i)} - \mu\right)^2$
- $z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$ (add ε epsilon for numerical stability if $\sigma^2 = \ 0$)
    - Forcing the inputs to a distribution with zero mean and variance of 1.
- $\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$
    - To make inputs belong to other distribution (with other mean and variance).
    - γ (gamma) and β (beta) are **learnable parameters** of the model.
    - Making the NN learn the distribution of the outputs.
    - Note: if $\gamma = \sqrt{\sigma^2 + \varepsilon}$ and $\beta = \mu$ then $\tilde{z}^{(i)} = z^{(i)}$

Adding Batch Norm to a network



- The NN parameters will be:

$$X \xrightarrow{W^{[1]},\, b^{[1]}} Z^{[1]} \xrightarrow[\text{Bach Norm (BN)}]{\gamma^{[1]},\, \beta^{[1]}} \tilde{Z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{Z}^{[1]}) \xrightarrow{W^{[2]},\, b^{[2]}} Z^{[2]} \xrightarrow[\text{Bach Norm (BN)}]{\gamma^{[2]},\, \beta^{[2]}} \cdots$$

- Parameters:

$$\left.\begin{array}{c} W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \cdots, W^{[L]}, b^{[L]} \\ \gamma^{[1]}, \beta^{[1]}, \gamma^{[2]}, \beta^{[2]}, \cdots, \gamma^{[L]}, \beta^{[L]} \end{array}\right\}$$

Back-prop:

$$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$$

- If you are using a deep learning framework, you should not implement batch norm yourself. For example, in Tensorflow you can add this line:

```
tf.nn.batch-normalization()
```

Working with mini-batches

- Batch normalization is usually applied with mini-batches.

$$X^{\{1\}} \xrightarrow{W^{[1]},\, b^{[1]}} Z^{[1]} \xrightarrow[\text{BN}]{\gamma^{[1]},\, \beta^{[1]}} \tilde{Z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{Z}^{[1]}) \xrightarrow[\text{BN}]{W^{[2]},\, b^{[2]}} Z^{[2]} \; \cdots$$

$$X^{\{2\}} \xrightarrow{W^{[1]},\, b^{[1]}} Z^{[1]} \xrightarrow[\text{BN}]{\gamma^{[1]},\, \beta^{[1]}} \tilde{Z}^{[1]} \longrightarrow \cdots$$

$$X^{\{3\}} \xrightarrow{W^{[1]},\, b^{[1]}} \cdots$$

- If we are using batch normalization, parameters $b^{[1]}, \cdots, b^{[L]}$ doesn't count because they will be eliminated after mean subtraction step because taking the mean of a constant $b^{[l]}$ will eliminate the $b^{[l]}$.
- So if you are using batch normalization, you can remove $b^{[l]}$ or make it always zero.
- So the parameters will be $W^{[l]}$, $\beta^{[l]}$, and $\gamma^{[l]}$.
- Shapes:
  - $Z^{[l]} : (n^{[l]}, 1)$
  - $\beta^{[l]} : (n^{[l]}, 1)$
  - $\gamma^{[l]} : (n^{[l]}, 1)$

Implementing gradient descent

For t = 1 … numMiniBatches

1) Compute forwardprop on $X^{\{t\}}$

In each hidden layer l, use BN to replace $Z^{[l]}$ with $\tilde{Z}^{[l]}$

2) Use backprop to compute $dW^{[l]}$, $db^{[l]}$, $d\beta^{[l]}$, $d\gamma^{[l]}$

3) Update parameters:

$$\begin{cases} W^{[l]} = W^{[l]} - \alpha dW^{[l]} \\ b^{[l]} = b^{[l]} - \alpha db^{[l]} \\ \beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]} \\ \gamma^{[l]} = \gamma^{[l]} - \alpha d\gamma^{[l]} \end{cases}$$

- Works with momentum, RMSprop, Adam.

Why does Batch normalization work?

- The first reason is the same reason as why we **normalize** X.
- The second reason is that batch normalization reduces the problem of **input values changing (shifting)**.
- Batch normalization does some **regularization**.

Batch Norm as regularization

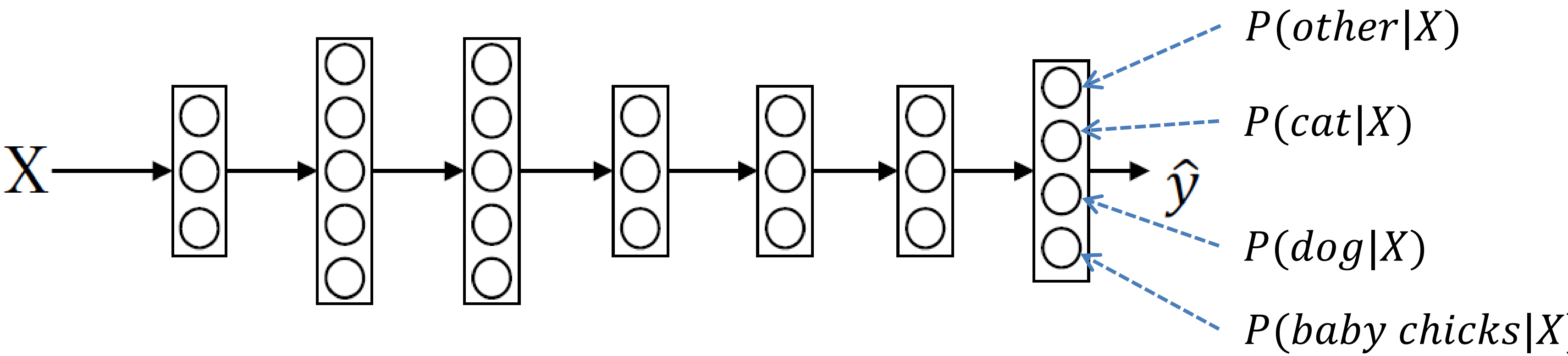
- Batch normalization does some **regularization**:
  - Each mini batch is scaled by the mean/variance computed of that mini-batch.
  - This adds some noise to the values $Z^{[l]}$ within that mini batch. So similar to dropout it adds some noise to each hidden layer's activations.
  - This has a slight regularization effect.
  - Using bigger size of the mini-batch you are reducing noise and therefore regularization effect.
  - Don't rely on batch normalization as a regularization. It's intended for normalization of hidden units, activations and therefore speeding up learning. For regularization use other regularization techniques (L2 or dropout).

Multi-class classification (Softmax regression)

- Recognizing cats, dogs, and baby chicks

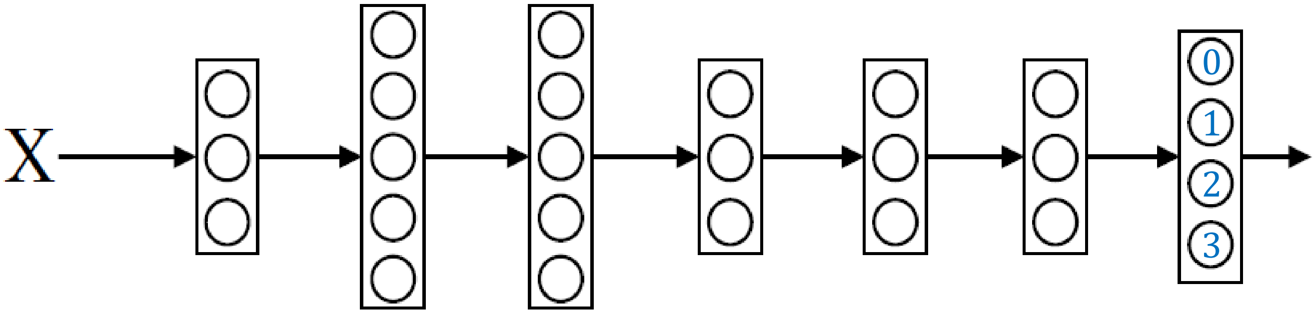
3	1	2	0	3	2	0	1

$$\boldsymbol{C = \#\, Classes = 4} \quad \boldsymbol{(0, 1, 2, 3)}$$



$\boldsymbol{\hat{y}\ has\ a\ size\ of\ (1, 4)}$

Softmax layer



$$\hat{y} = \begin{bmatrix} \frac{148.4}{176.3} \\ \frac{7.4}{176.3} \\ \frac{0.4}{176.3} \\ \frac{20.1}{176.3} \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

$$Z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]} \Rightarrow (4,1)$$

Activation function:

$$t = e^{(Z^{[L]})}$$

$$\hat{y} = a^{[L]} = \frac{e^{Z^{[L]}}}{\sum_{j=1}^{4} t_j}, a_i^{[L]} = \frac{t_i}{\sum_{j=1}^{4} t_j}$$

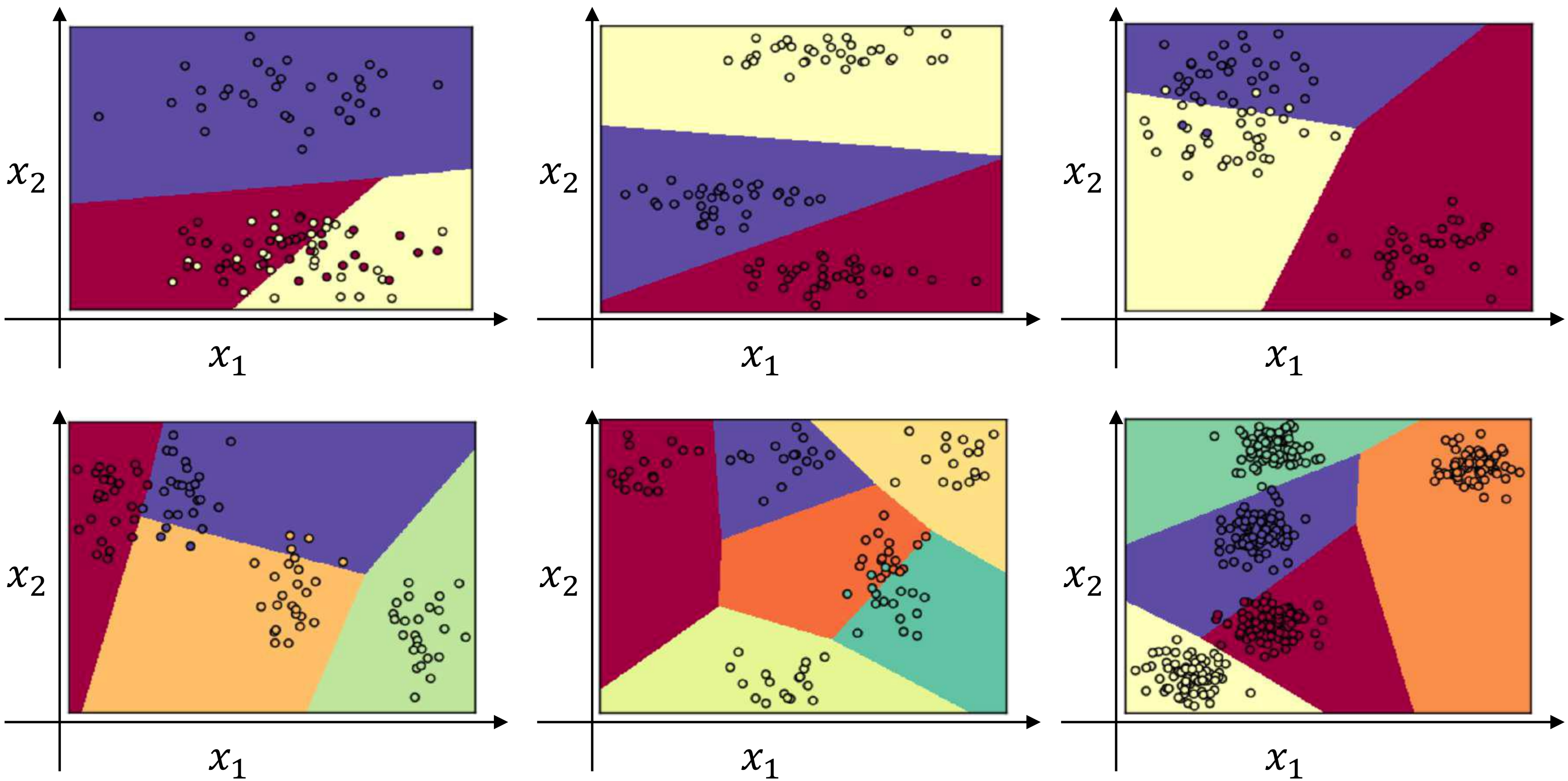
$$a^{[L]} = g^{[L]}\left(z^{[L]}\right)$$

$$Z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

$$t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}, \sum_{j=1}^{4} t_j = 176.3$$

$$a^{[L]} = \frac{t}{176.3}$$

Softmax examples



Understanding softmax

(4,1)

$$Z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} \qquad \boldsymbol{\#Clasees\ C = 4}$$

Softmax | Hard max

$$a^{[L]} = g^{[L]}\left(Z^{[L]}\right) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Softmax regression generalizes logistic regression to C classes

If C = 2 Softmax reduces to logistic regression.

Loss function

$$y^{(i)} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \qquad a^{[L](i)} = \hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \quad \#\boldsymbol{Clasees\ C = 4}$$

$(4,1)$

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j log(\hat{y}_j) \qquad J(W^{[1]}, b^{[1]}, \dots) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Example: $\mathcal{L}(\hat{y}, y) = -y_2 log(\hat{y}_2) = -log(\hat{y}_2)$

Small $\mathcal{L}(\hat{y}, y) \Rightarrow$ make $\hat{y}_2$ big.

Vectorization:

$$Y = [y^{(1)}, y^{(2)}, \cdots, y^{(m)}] \qquad \hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \cdots, \hat{y}^{(m)}]$$

$(4, m)$

$$= \begin{bmatrix} 0 & 0 & 1 & . \\ 1 & 0 & 0 & . \\ 0 & 1 & 0 & . \\ 0 & 0 & 0 & . \end{bmatrix} \qquad = \begin{bmatrix} 0.3 & . & . & . \\ 0.2 & . & . & . \\ 0.1 & . & . & . \\ 0.4 & . & . & . \end{bmatrix}$$

$(4, m)$

Deep learning frameworks

- It's not practical to implement everything from scratch. Our numpy implementations were to know how NN works.
- There are many good deep learning frameworks.
- Deep learning is now in the phase of doing something with the frameworks and not from scratch to keep on going.
- Here are some of the leading deep learning frameworks:

Caffe/Caffe2	CNTK	DL4J
Keras	Torch	mxnet
PaddlePaddle	TensorFlow	Theano

- Choosing deep learning frameworks
  - Ease of programming (development and deployment)
  - Running speed
  - Truly open (open source with good governance)

TensorFlow

- In this section we will learn the basic structure of TensorFlow programs.

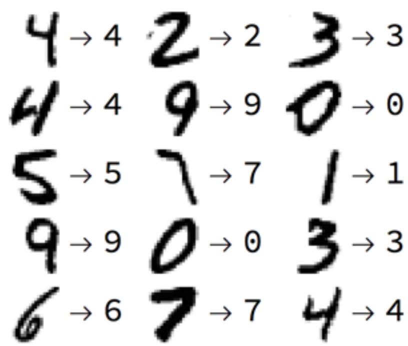
- **Demo 1: Optmization of a simple quadratic equation.**

  - Implement a minimization function. For example the function:

$$\boldsymbol{J(w) \ = \ w^2 - 12w + 36}$$

  - The result should be $\boldsymbol{w} \ = \boldsymbol{6}$ as the function is $(\boldsymbol{w - 6})^{\boldsymbol{2}} = \boldsymbol{0}$

- **Demo 2: Classification of digit images**



References

- Andrew Ng. Deep learning. Coursera.
- Geoffrey Hinton. Neural Networks for Machine Learning.
- Kevin P. Murphy. Probabilistic Machine Learning An Introduction. MIT Press, 2022.
- MIT Deep Learning 6.S191 (http://introtodeeplearning.com/)