ADAM OPTIMIZATION ALGORITHM

* ADAM (Adaptation Moment Estimation) = momentum + RMS prop

* compute dW, db using current mini-batch

$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)\, dW$, $V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db$ → MOMENTUM

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\, dW^2$, $S_{db} = \beta_1 S_{db} + (1-\beta_1)\, db^2$ → RMS PROP

$V_{dW}^{corrected} = V_{dW} / (1-\beta_1^t)$, $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

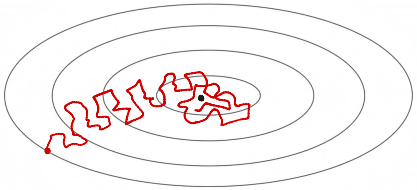
$S_{dW}^{corrected} = S_{dW} / (1-\beta_2^t)$, $S_{db} = S_{db} / (1-\beta_2^t)$

$$W = W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}} + \varepsilon}, \quad b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$ } UPDATE

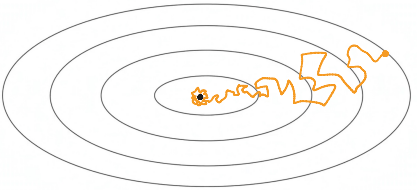
* hyperparameters → α (learning rate), β_1, β_2

LEARNING RATE DECAY

* reduce learning rate over time



REGULAR α (doesn't converge)

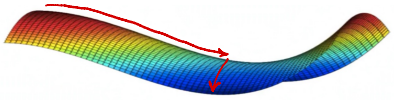


SLOWLY REDUCED α
(you oscilate in a tighter region)

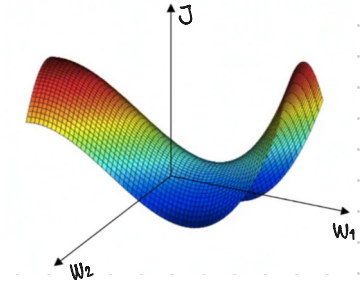
$$\alpha = \frac{1}{1 + \text{decay rate} * \text{epoch}}\, \alpha_0$$ LEARNING RATE DECAY

$$\alpha = 0.95^{\text{epoch}}\, \alpha_0$$ EXPONENTIAL DECAY

THE PROBLEM OF LOCAL OPTIMA



* the derivative is close to 0 for a long time. They can make learning slow.



SADDLE POINT:
* minimum along w_1
* maximum along w_2

TUNING PROCESS

* hyperparameters
  - α (learning rate) — MOST IMPORTANT!
  - $\beta_1, \beta_2, \varepsilon$ (Adam optimization)
  - # hidden units
  - mini-batch size
  - } SECOND MOST IMPORTANT
  - # layers
  - learning rate decay
  - } THIRD MOST IMPORTANT

* try random values. Don't use a grid!!
  - → coarse-to-find search process

* appropriate scale for hyperparameters:

(#1) learning rate: search for a value between 0.0001 and 1 within a log scale

0.0001 — 0.001 — 0.01 — 0.1 — 1

(#2) exponentially weighted averages: $\beta = 0.9 \dots 0.999$ (log scale!)

$1-\beta = 0.1 \dots 0.001$

0.9 — 0.99 — 0.999

0.1 — 0.01 — 0.001

* Tunning approach → babysitting one model: focus on one model one day at a time (PANDA STRATEGY)
* Tunning approach → train many models in parallel (CAVIAR STRATEGY)

how much computational power you have?
- A LOT! → Caviar
- NOT MUCH → Panda

BATCH NORMALIZATION

* speeds up learning
* normalize → mean, variance } layer outputs → to make the training of (w), (b) more efficient
  - before or after passing through the activation function?

$$\mu = \frac{1}{m}\sum_{i=1}^{n} z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{n} (z^{(i)} - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$ (LEARNABLE PARAMETERS) $\Longrightarrow$ if $\gamma = \sqrt{\sigma^2 + \varepsilon}$ and $\beta = \mu$, then $\tilde{z}^{(i)} = z^{(i)}_{norm}$

* mostly used with mini-batches

$X^{\{t\}} \rightarrow W^{[1]}, b^{[1]} \rightarrow z^{[1]} \rightarrow \beta^{[1]}, \gamma^{[1]}$ (BN) $\rightarrow \tilde{z}^{[1]}$

* gradient descent + batch normalization:

for t = 1 ... (# mini-batches):

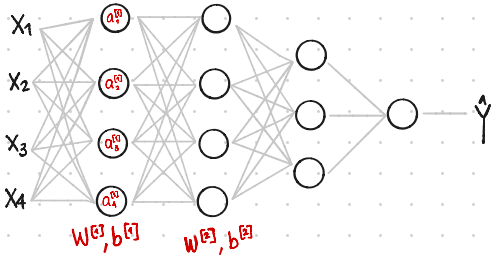
compute forward prop on $X^{\{t\}}$

in each hidden layer, use (BN) to replace $z^{[l]}$ with $\tilde{z}^{[l]}$

use back prop to compute $dw^{[l]}$, $db^{[l]}$, $d\beta^{[l]}$, $d\gamma^{[l]}$

update parameters (works with momentum, RMSprop, Adam, etc).

(#1) It makes weights in deeper layers of the (NN) more robust to changes compared to those in the early layers.



THE CHANGES IN $W^{[2]}, b^{[2]}$ WILL CHANGE $a_1^{[2]}, a_2^{[2]}, a_3^{[2]}$, AND $a_4^{[2]}$; THAT WILL ALSO CHANGE $W^{[3]}, b^{[3]}$, GENERATING A COVARIATE SHIFT PROBLEM.

→ BUT...

(BN) LIMITS THE AMOUNT TO WHICH UPDATING THE PARAMETERS IN THE EARLIER LAYERS CAN AFFECT THE DISTRIBUTION OF THE VALUES IN THE UPCOMING LAYERS. IT REDUCES THE PROBLEM OF THE INPUT VALUES CHANGING. THESE VALUES BECOME MORE STABLE.

(#2) By using a bigger mini-batch size, you reduce the noise together with the regularization effect. But don't use (BN) as a regularizer.

* at test time, you come up with a separate estimate of μ and σ^2. You estimate these parameters using exponentially weighted averages across the mini-batches.

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}}$

$\mu^{\{1\}[l]}$ $\sigma^{2\{1\}[l]}$ $\mu^{\{2\}[l]}$ $\sigma^{2\{2\}[l]}$ $\mu^{\{3\}[l]}$ $\sigma^{2\{3\}[l]}$

$\Rightarrow \mu, \sigma^2 \Rightarrow z_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}}$

SOFTMAX REGRESSION

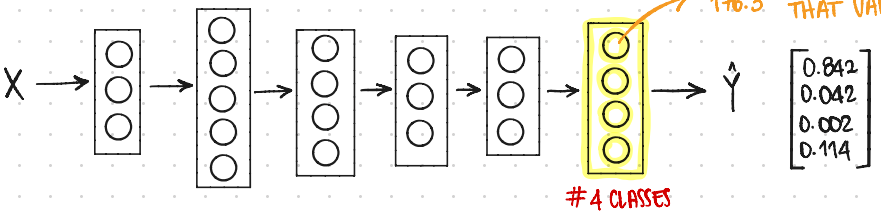
* multiclass classification

$$z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$$

$$a^{[L]}_{(4,1)} = \frac{e^{z^{[L]}}}{\sum_{i=i}^{4} t_i} \Longrightarrow a^{[L]}_i = \frac{t_i}{\sum_{i=i}^{4} t_i}$$

ACTIVATION FUNCTION

example:



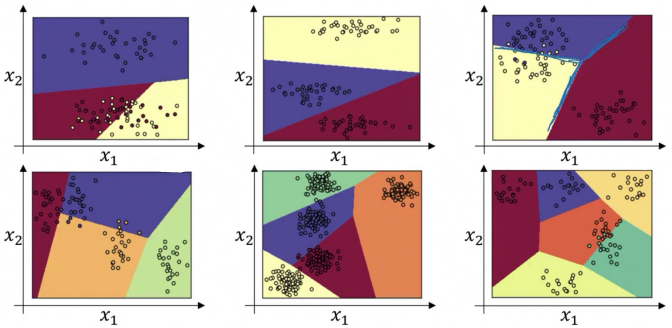
$\frac{e^5}{176.3} = 0.842$ (THE CHANCES OF THAT VALUE BEING 0 IS 84.2%)

#4 CLASSES

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

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

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



* these are examples with no hidden units. As you add layers the (NN) will be able to classify more complex (non-linear) points.

* Softmax regression generalizes logistic regression to (C) classes.

* If C=2, softmax reduces to logistic regression.

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$$

LOSS FUNCTION
(Max likelihood estimation)

* $\mathcal{L}(\hat{y}, y)$ small $\longrightarrow$ $\hat{y}_j$ big

$$J(w^{[1]}, b^{[1]}, \dots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

COST FUNCTION

* backpropagation:

$$\frac{\partial J}{\partial z} = dz^{[2]} = \hat{y} - y$$