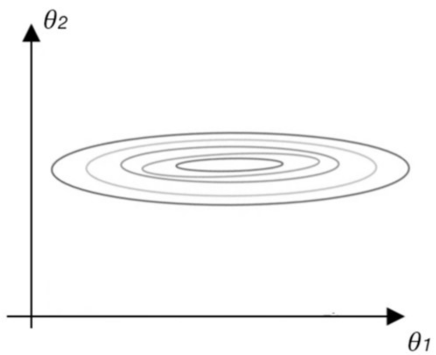
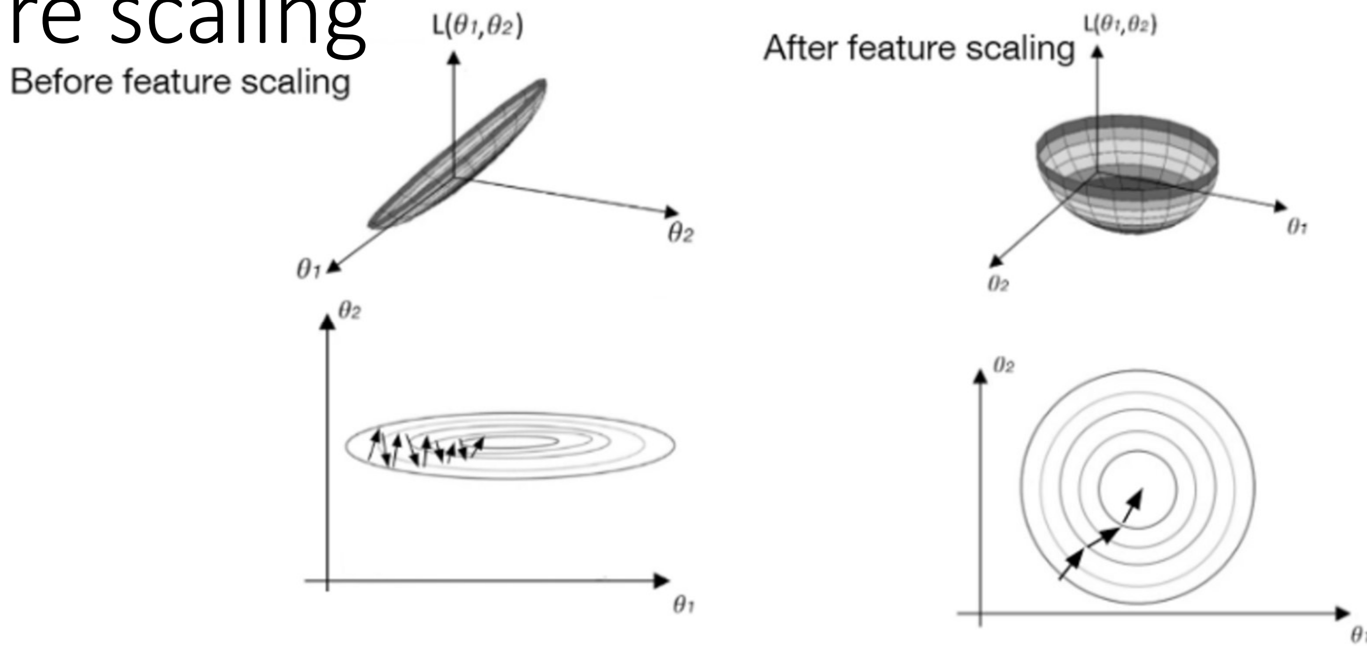
Need for feature scaling



θ_1=[0, 1000], θ_2=[0, 1]



- The GD optimizer would take bigger steps in the direction of smaller range feature, while making only little steps in the direction of higher range feature.

- The gradient would oscillate a lot, which would cause the model to take a longer time to find the global minimum or In the worst case, the minimum could not be found at all.

- We can prevent this problem by feature scaling

https://www.deeplearning-academy.com/p/ai-wiki-data-preprocessing

Data splitting strategy

Training	Validation	Test
The dataset on which we train the network. The model sees and learns from this data	The validation set serves to evaluate a model, adjust the hyperparameters, and try different model architectures. The model only sees this data during training to calculate some evaluation metrics, but never learns from this data	Once a model is completely trained its final performance is tested on the test set. The test dataset used to evaluate the final performance of a trained model.

Very often the validation set is used as the test set, but it is not good practice

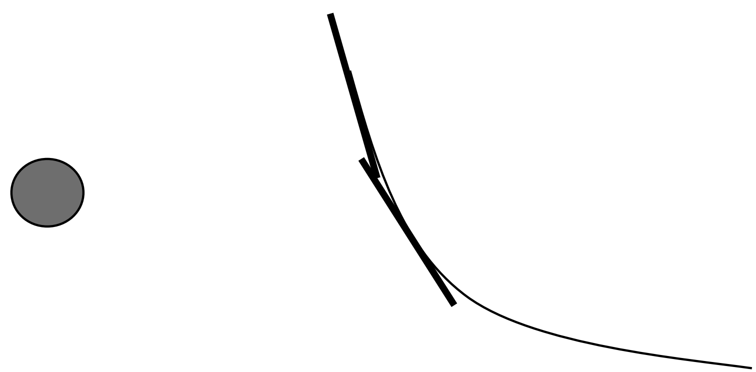
Data splitting strategy

- Training size has to be larger
- 1000 to 50,000 70% Train, 20% validation and 10% for test
- >>50,000, 90% Train, 5% validation and 5% for test

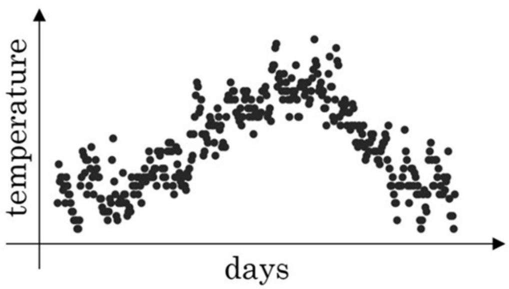
Andrew NG

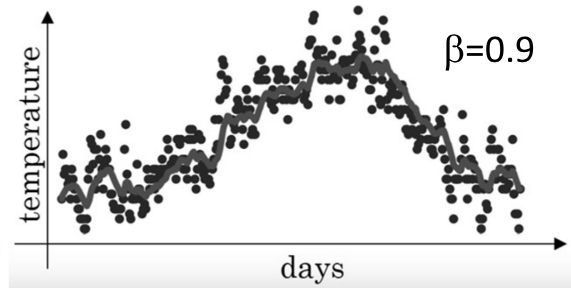
SGD

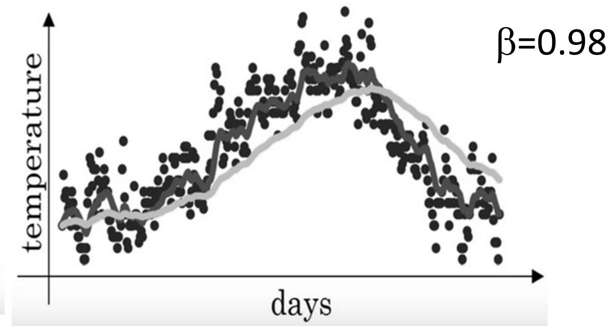
- 1) cannot adopt to changing gradient values
- 2) it cannot change the learning rate to accelerate/ decelerate the minimization based on the gradient values



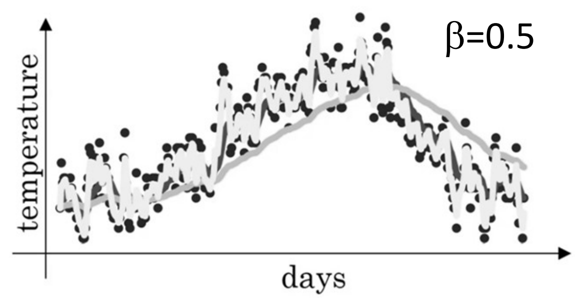
Exponentially weighted moving average







$$V_0 = 0$$
$$V_1 = 0.9\,V_0 \quad + 0.1\,\theta_1$$
$$V_2 = 0.9\,V_1 \quad + 0.1\,\theta_2$$
$$V_3 = 0.9\,V_2 \quad + 0.1\,\theta_3$$
$$\vdots$$
$$V_t = 0.9\,V_{t-1} \quad + 0.1\,\theta_t$$



$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$

Vt ~ averaging over 1/(1-β) values

- deeplearning.ai [Andrew NG]

Exponentially weighted moving average

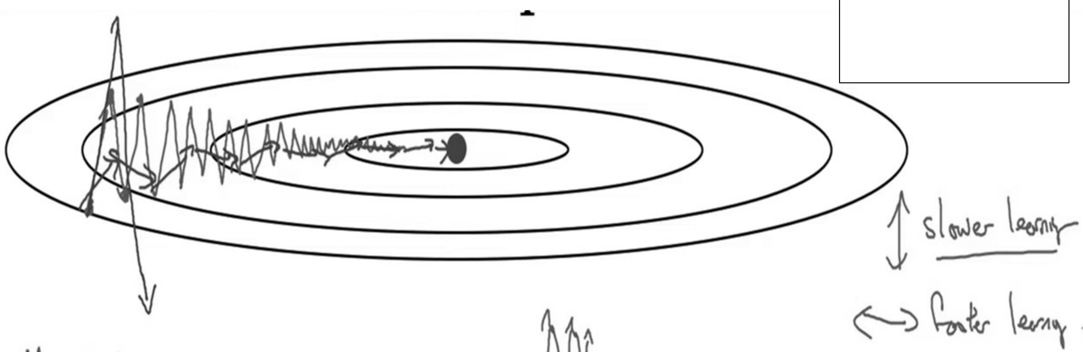
$$v_n = \sum_{k=0}^{n} (1-\beta) \times \beta^n \theta_n$$

V_{10}= $0.1\theta_{10}$+0.1X(0.9) θ_9+0.1X$(0.9)^2\theta_8$+0.1X$(0.9)^3\theta_7$+-----+0.1X$(0.9)^{10}\theta_1$

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

Advantage Less memory

Gradient descent with momentum



Momentum:

On iteration t:

Compute dW, db on current mini-batch.

$V_{dW} = \beta V_{dW} + (1-\beta) dW$ "$V_\theta = \beta V_\theta + (1-\beta)\theta_t$"

$V_{db} = \beta V_{db} + (1-\beta) db$

$W := W - \alpha V_{dW}$, $b := b - \alpha V_{db}$

W=W-αdw

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + (1-\beta) dW$$

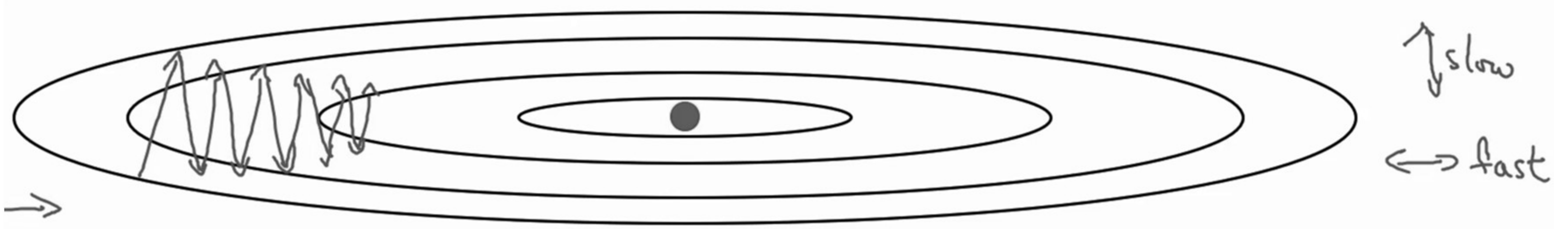
$$v_{db} = \beta v_{db} + (1-\beta) db$$

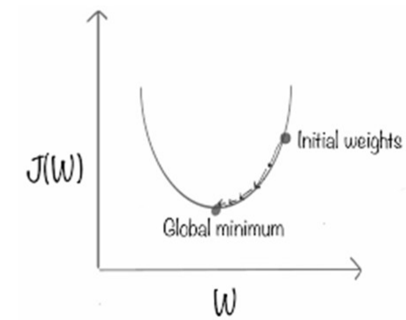
$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

Hyperparameters: α, β $\beta = 0.9$

- Large gradient → smaller weight update

Adaptive learning rate





dw= dCf/dw

On iteration t:

Compute dW, db on current mini-batch

$$S_{dW} = \beta S_{dW} + (1-\beta)\, \underline{dW^2} \quad \text{element-wise}$$

$$S_{db} = \beta S_{db} + (1-\beta)\, db^2$$

$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

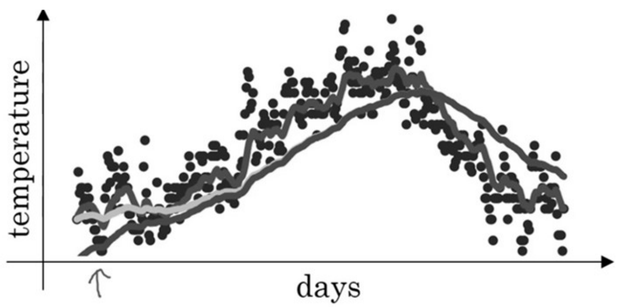
Adaptive learning rate

$$\frac{\alpha}{\sqrt{S}}$$

Large gradient → smaller weight update

small gradient → larger weight update

Bias correction in EWMA



$$v_t = \frac{\beta}{(1-\beta^t)} v_{t-1} + \frac{(1-\beta)}{(1-\beta^t)} \theta_t$$

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

ADAM optimizer (Adaptive Moment Estimate)

$V_{dw} = 0,\ S_{dw} = 0.\ V_{db} = 0,\ S_{db} = 0$

On iteration t:

Compute dW, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db \leftarrow$ "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) db \leftarrow$ "RMSprop" β_2

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

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

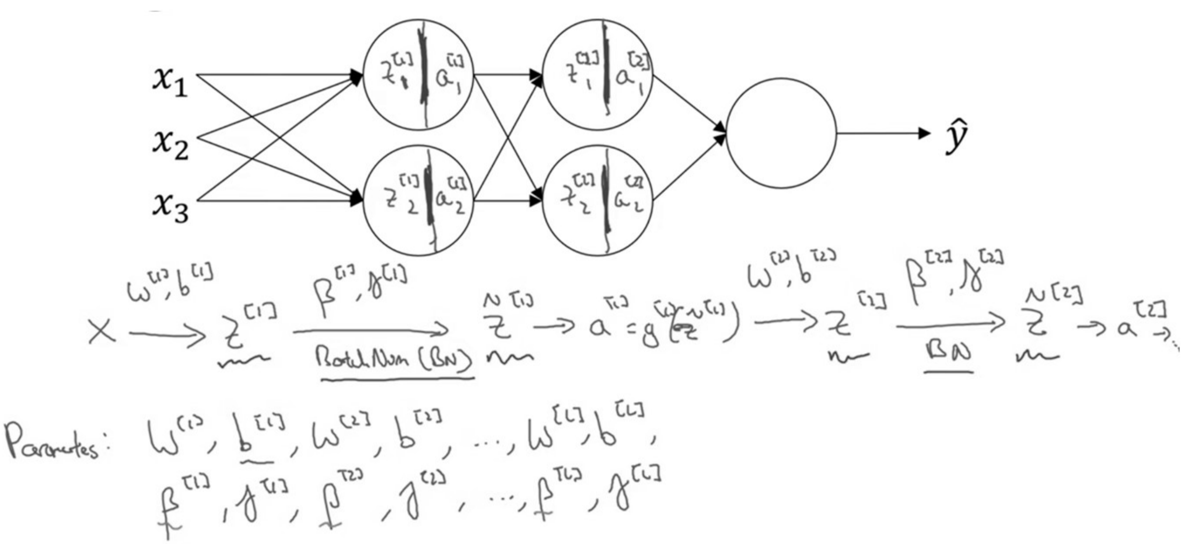
$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

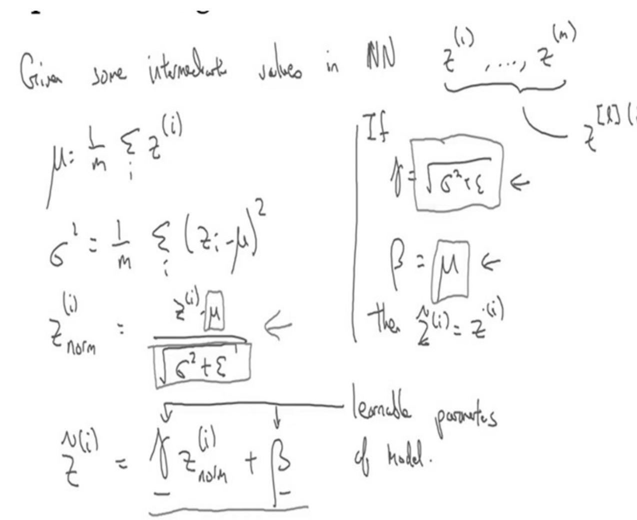
$\alpha \rightarrow$ needs to tuned

$\beta 1 = 0.9 \rightarrow$ momentum

$\beta 2 = 0.999 \rightarrow$ RMS

Batch normalization works





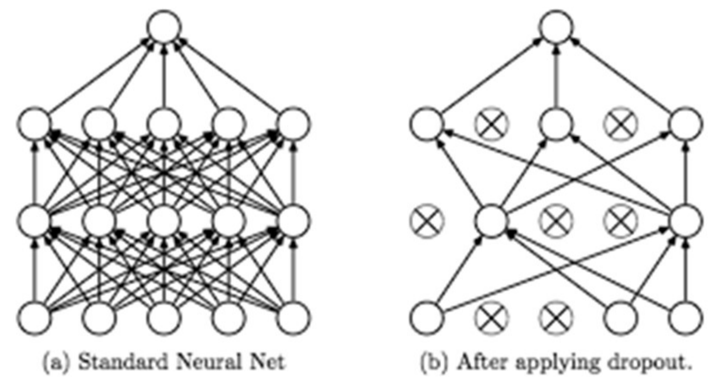
- internal covariance shift: The change in the distribution of inputs to layers in the network is referred to "internal covariate shift"

- Batch normalization accelerates training, in some cases by halving the epochs or better,

- provides some regularization, reducing generalization error

dropout

- → drop out 20% to 50% of the neurons randomly during training
- → all the neurons are activated during testing
- Weights are multiplied by the probability value (probability of drop out)
- Weights are updated for a simpler neural net → generalization capability of NN



Best practices

- Normalize the features
- Use Kaiming He initialization for wights
- Use ReLu activation function for hidden layers
- Use Softmax AF for output layer for classification, no activation for regression
- Use RMSE cost function for regression
- Use CrossEntropy (Cat/ Sp cat) for classification→ large dataset with lot of class labels → use sparse categorical cross entropy
- Use batch normalization
- Use drop out or L2 regularization if over fit
- Use ADAM optimizer

- With all these figured out generally, the architecture of the NNet is what needs to be hyper tuned for a larger range
- Best architecture→ NAS
- Network pruning