Machine Learning in Healthcare

# #L16-Neural Networks II

Technion-IIT, Haifa, Israel

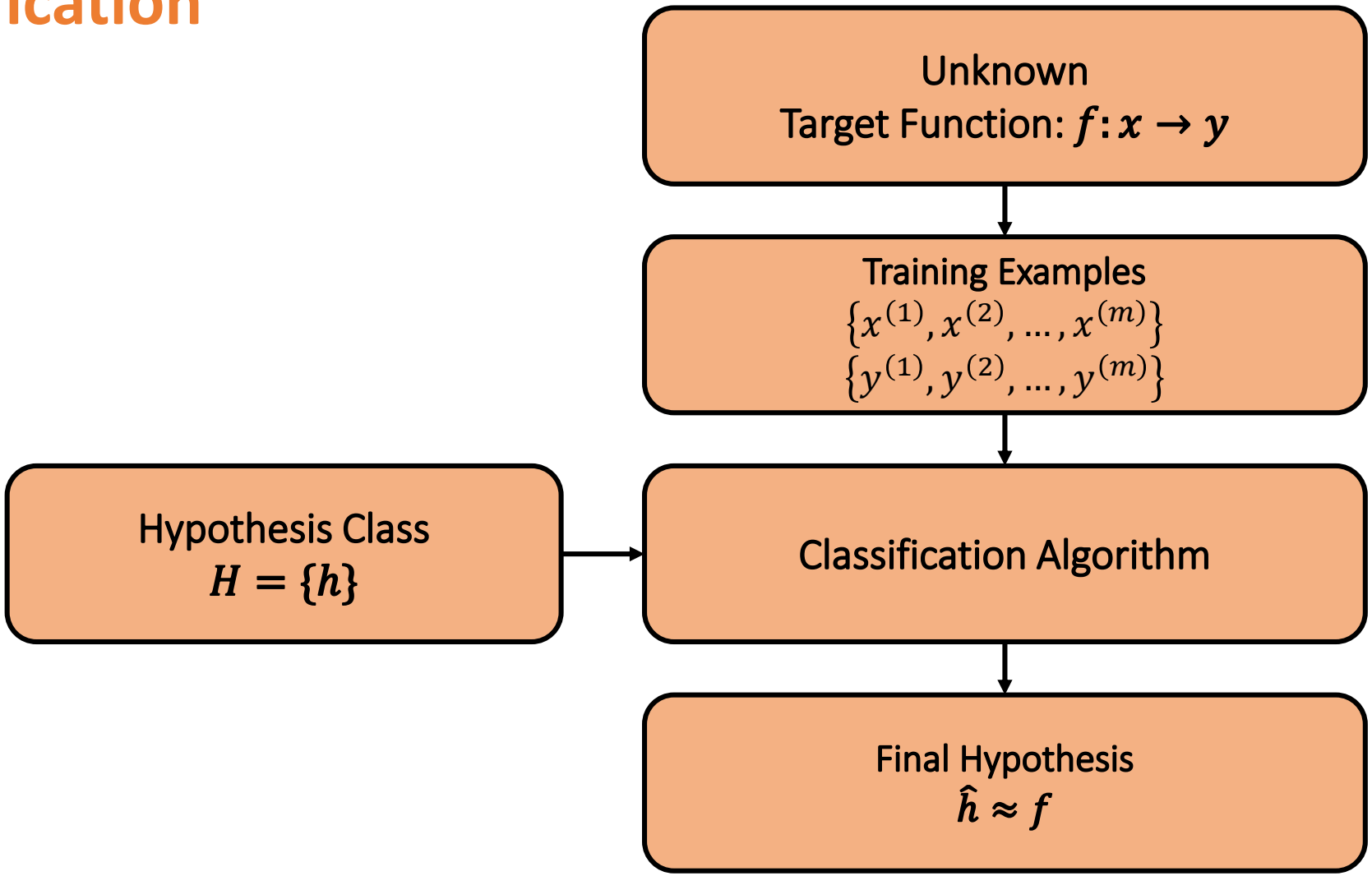
Asst. Prof. Joachim Behar
Biomedical Engineering Faculty, Technion-IIT
Artificial intelligence in medicine laboratory (AIMLab.)
https://aim-lab.github.io/
Twitter: @lab_aim

# Agenda

- Revisiting train-validation-test split.
- Weight initialization.
- Optimization algorithms.
- Bias-variance tradeoff.
- Batch normalization.

AIMLab.

# Classification



# Revisiting Train-Validation-Test split

AIMLab.

# Train-Validation-Test

- In the first part of the course we explained that one should divide the data into three sets: **train, validation and test sets**.
- The reason for this split was to ensure:
  - Good hyperparameters selection.
  - Good generalization.
- We have seen that a typical split is 60%-20%-20%.

AIMLab.

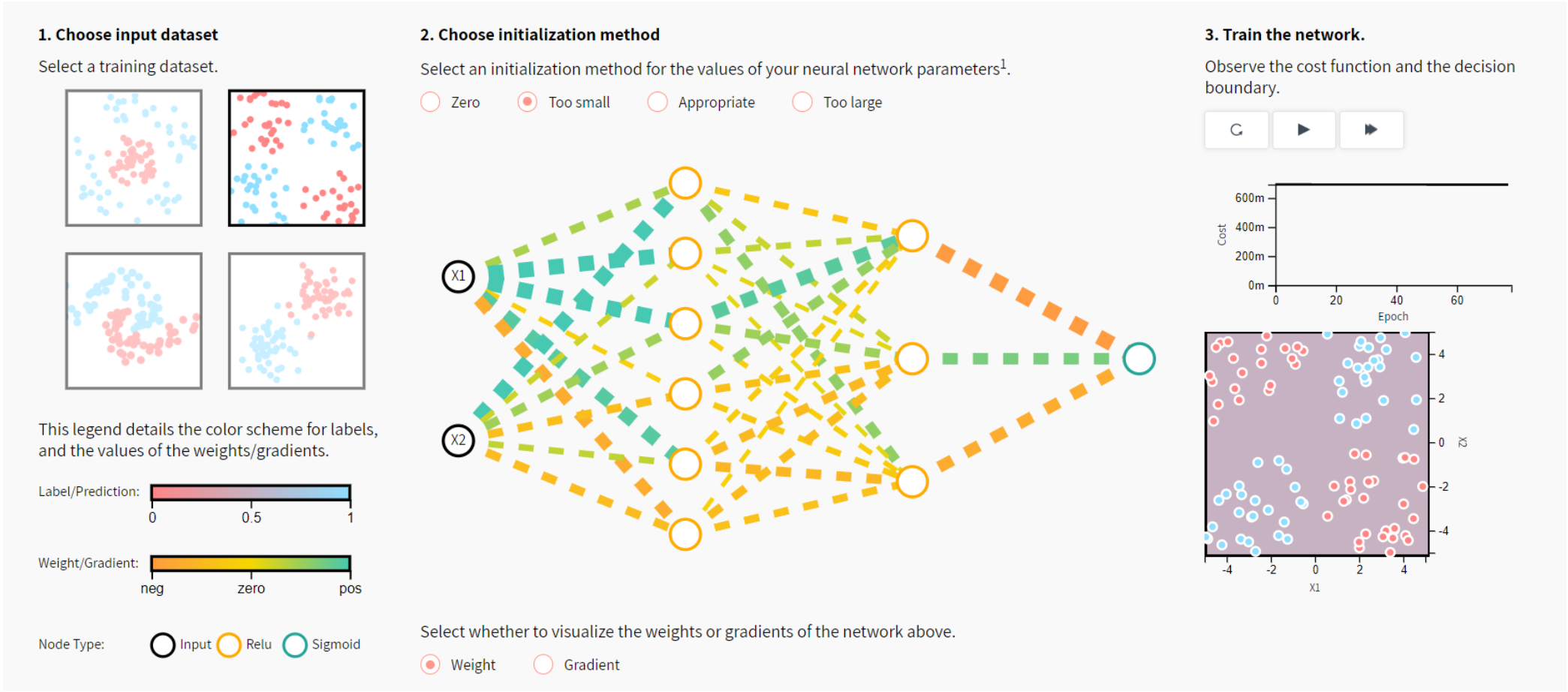
# Train-Validation-Test

- The motivation for this split was to leave enough data in the validation and test set that are representative of the population sample that is being studied.
- For medium sized datasets it is perfectly ok but what about "**big data**" e.g. 1 Million examples?
- For "big data" it is okay to use most of the data in the training set i.e. >95% of the examples in the training set.
- However, you must check that the validation and test set come from a similar distribution.

# Weights initialization

# Weights initialization

- https://www.deeplearning.ai/ai-notes/initialization/



# Weights initialization

- Constant initialization: neurons will evolve symmetrically during training thus preventing different neurons to learn different information.
- Very small weights: vanishing gradient.
- Very high weights: exploding gradient.
- Thus we seek to find an initialization with weights that are not too small or too big and not constant either so that the back propagated gradient signal be not too small or too big.

AIMLab.

# Weights initialization

- In practice this is achieved by assuming that:
  - The mean of the activation function is zero: $E\left[a^{[l-1]}\right] = E\left[a^{[l]}\right] = 0$
  - The variance of the activations should stay the same across every layer: $Var\left[a^{[l-1]}\right] = Var[a^{[l]}]$.
  - This is used to sort of make sure that activations will not be too small or too large at each layer.

AIMLab.

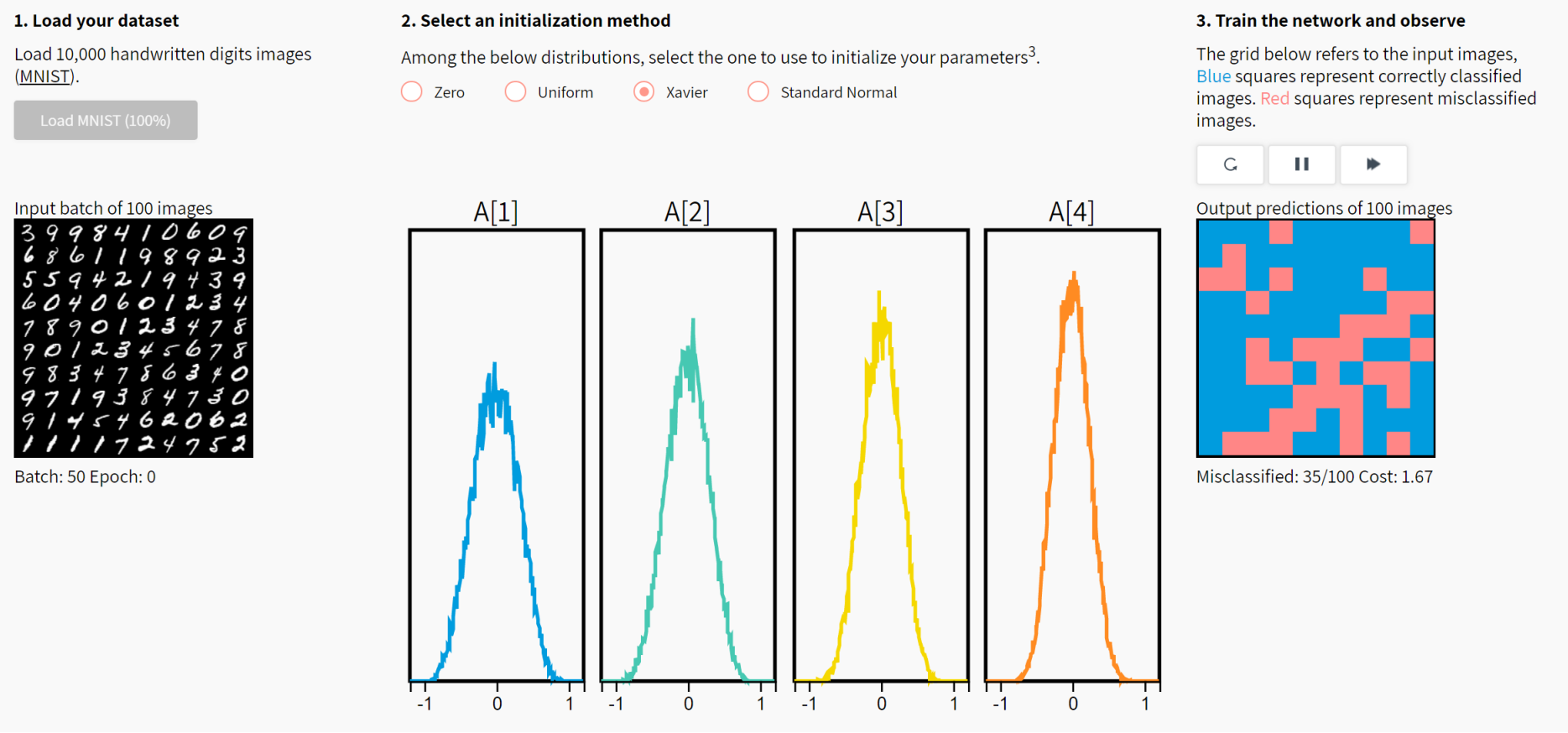
# Weights initialization

- Xavier initialization ($tanh$):
  - $W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}\right)$
- He initialization ($ReLU$):
  - $W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]}}\right)$.
- Prove that under these initialization: $Var\left[a^{[l-1]}\right] = Var[a^{[l]}]$.

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

# Weights initialization

- https://www.deeplearning.ai/ai-notes/initialization/

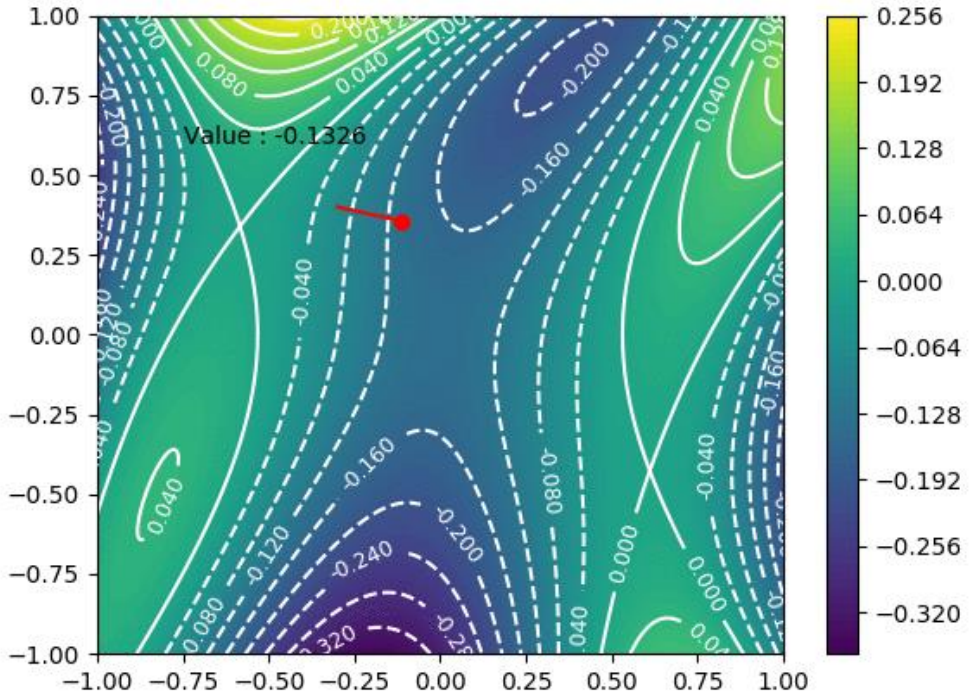


# Optimization algorithms

AIMLab.

# Gradient descent

- One of the most popular algorithms to perform optimization. Used extensively for optimizing neural networks.




https://github.com/Shathra/gradient-descent-demonstration

AIMLab.

# Gradient descent

- Reminder:
  - Given $w$ we compute
    - $J(w)$
    - $\frac{\partial J(w)}{\partial w_j}$ for $j \in [1, \dots, n]$
  - Gradient descent, update $w_j$
    - $w_j := w_j - \alpha \frac{\partial J(w)}{\partial w_j}$
- We can divide gradient descent in three variants, each of which differ in how many examples are used to compute the gradient of the objective function.

# Gradient descent

- We can divide gradient descent in three variants, each of which differ in how many examples are used to compute the gradient of the objective function.
- Why does the number of examples matter?
- We demonstrated how to vectorize forward and backward propagation for $m$ examples. The motivation was to make things faster (versus using a loop).

AIMLab.

# Gradient descent

- However, let's now assume we are dealing with a very large dataset e.g. $m = 10{,}000{,}000$ examples.
  - If we perform the matrix operations this will be slow ($A \in \mathbb{R}^{n_h^{[l]} \times m}, W^{[l]} \in \mathbb{R}^{n_h^{[l]} \times n_h^{[l-1]}}$).
  - It requires the loading of the whole dataset into memory which may not be technically feasible.
- An idea is then not to process all the examples at the same time but rather by **mini-batches**.

AIMLab.

# Gradient descent

- **Batch gradient descent** (BGD): compute the gradient of the cost function for the entire training set:
  - $w_j := w_j - \alpha \frac{\partial J(w)}{\partial w_j}$
- **Stochastic gradient descent** (SGD): compute the gradient of the cost function for each training example:
  - $w_j := w_j - \alpha \frac{\partial J(w; x^{(i)}, y^{(i)})}{\partial w_j}$
- **Mini-batch gradient descent**: compute the gradient of the cost function for $p$ training examples:
  - $w_j := w_j - \alpha \frac{\partial J(w; x^{(i:i+p)}, y^{(i:i+p)})}{\partial w_j}$

AIMLab.

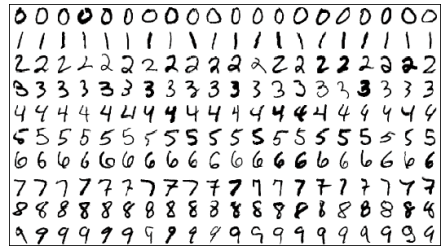
# Gradient descent

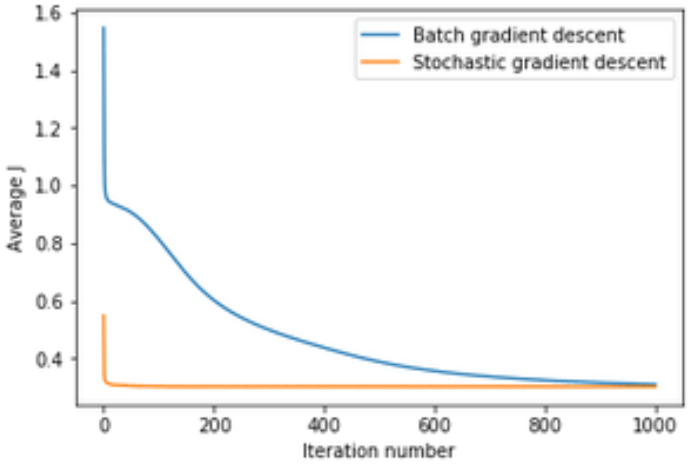
- What is the difference between the three options?
- Trade-off between the **accuracy of the parameter update** and the **time it takes to perform an update:**
  - **Batch gradient descent (BGD):** can be very slow for large datasets. However, objective function does not fluctuate heavily.
  - **Stochastic gradient descent (SGD):** frequent updates with a high variance that cause the objective function to fluctuate heavily. However, performing an update is quick.
  - **Mini-batch gradient descent:** a compromise between the two.

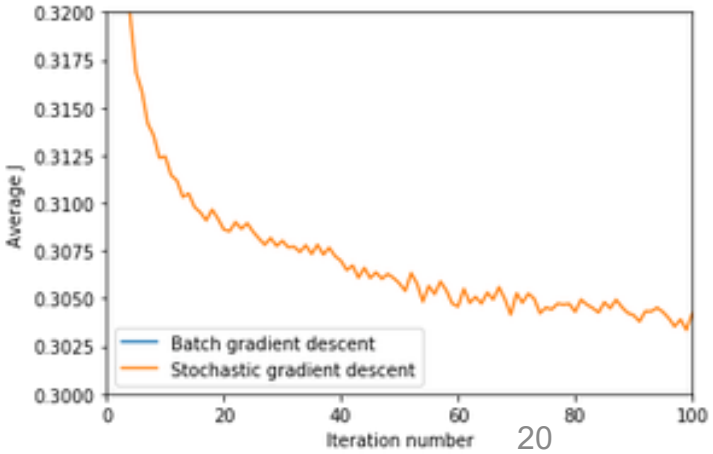
| | Accuracy of parameter update | Computational time to perform an update |
|---|---|---|
| BGD | + | - |
| SGD | - | + |
| Mini-batch | + | + |

# Gradient descent

- What is the consequence on convergence rate and accuracy?
- Vocabulary: an “epoch” corresponds to a single pass through all training set examples
- Let’s run batch and stochastic GD on the MNIST dataset
  - SGD converges much more rapidly.
  - BGD accuracy > SGD accuracy on the long run.
  - Batch gradient descent might lead to better results than SGD because SGD is more sensitive to noise which might hinder setting in a good minima.



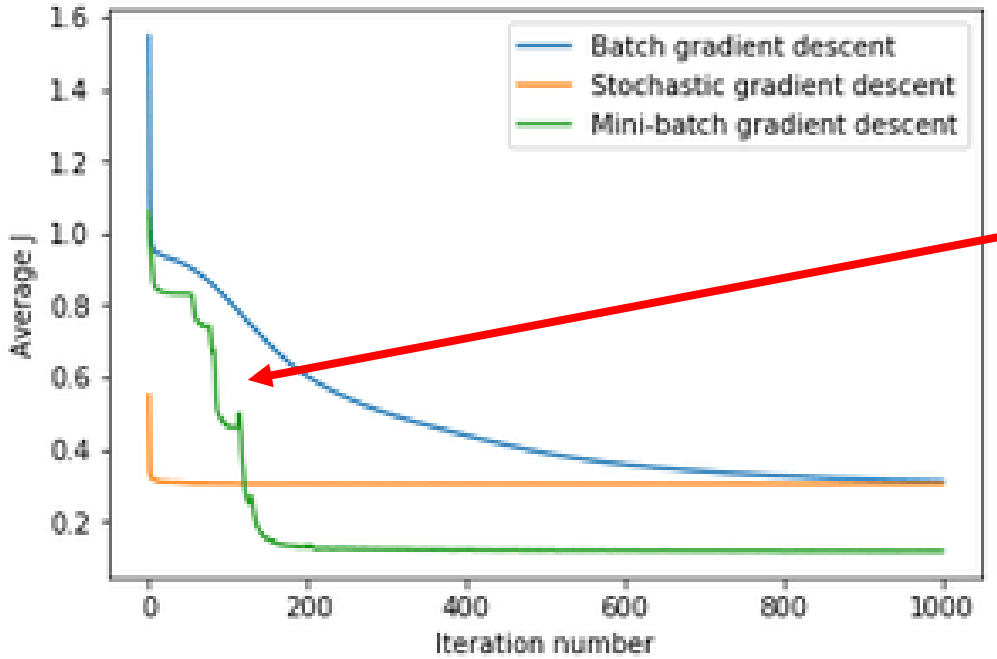



Images: https://adventuresinmachinelearning.com/stochastic-gradient-descent/
#Iteration on this graphs == #epoch

# Gradient descent

- The middle road: mini-batch gradient descent
  - Better performance than SGD and BGD,
  - Convergence is faster than BGD.




The jagged decline in the cost function illustrates that mini-batch gradient descent is kicking the cost function out of a local minima.

Images: https://adventuresinmachinelearning.com/stochastic-gradient-descent/

# Gradient descent

- In practice how do we perform Mini-batch gradient descent?
  - Assume the training set examples are divided into $k$ mini-batch of size $p$
  - **For** each mini-batch $\{X^k, Y^k\}$
    - Forward propagation on $X^k$,
    - Compute cost function $J^k$,
    - Backpropagation: compute gradients with respect to $J^k$,
    - Update weights and bias.

# Gradient descent

- How to choose the size $p$ of the mini-batch?
  - If the training set is small then just use batch gradient descent,
  - A power of 2,
  - Size needs to fit in CPU/GPU memory,
  - You might search for this parameter and thus consider it as an hyperparameter.

# Gradient descent

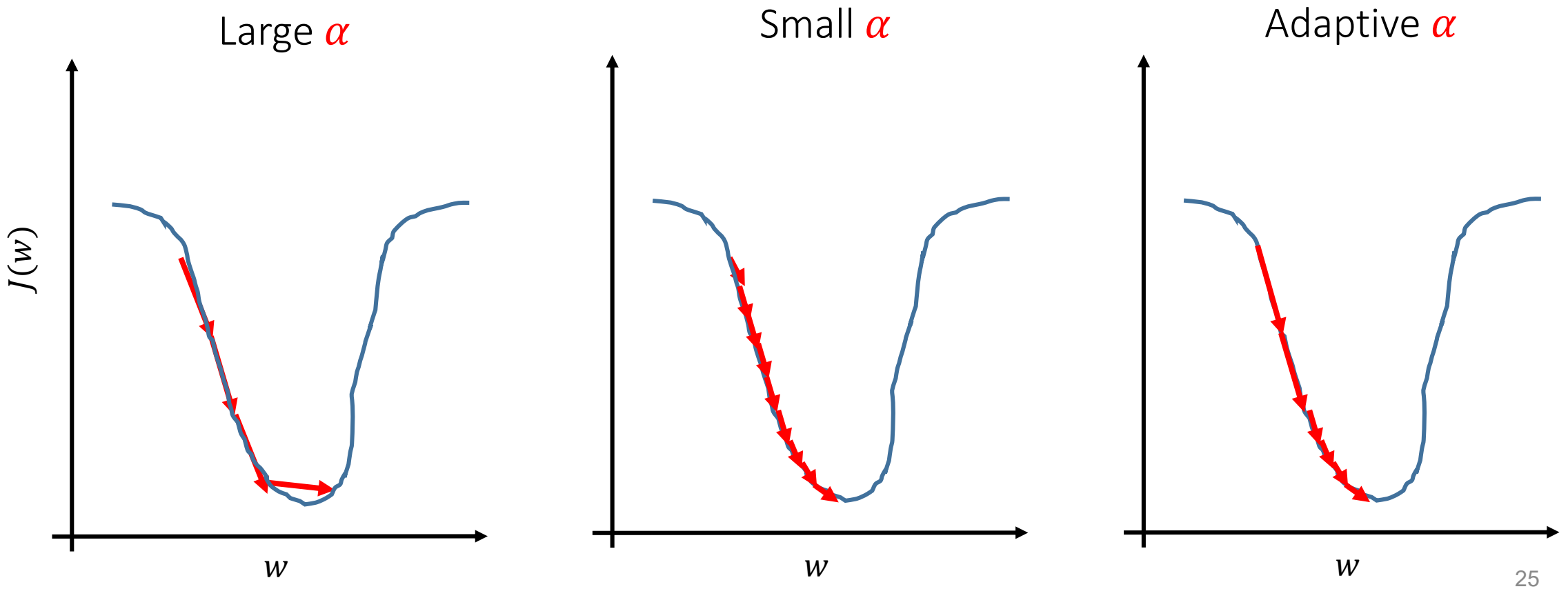
- There are some optimization gradient descent algorithms that will improve the speed or accuracy over classical gradient descent:
  - **Gradient descent with momentum**
  - RMSprop
  - Adam optimization algorithm (momentum + RMSprop)
  - Nesterov
  - …
- See for a good overview:
  - Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747(2016).

AIMLab.

# Learning rate decay

$$w_j = w_j - \alpha \frac{\partial J(w)}{\partial w_j}$$

- We want to adapt the learning rate as a function of #epochs.
- Motivation:



AIMLab.

# Learning rate decay

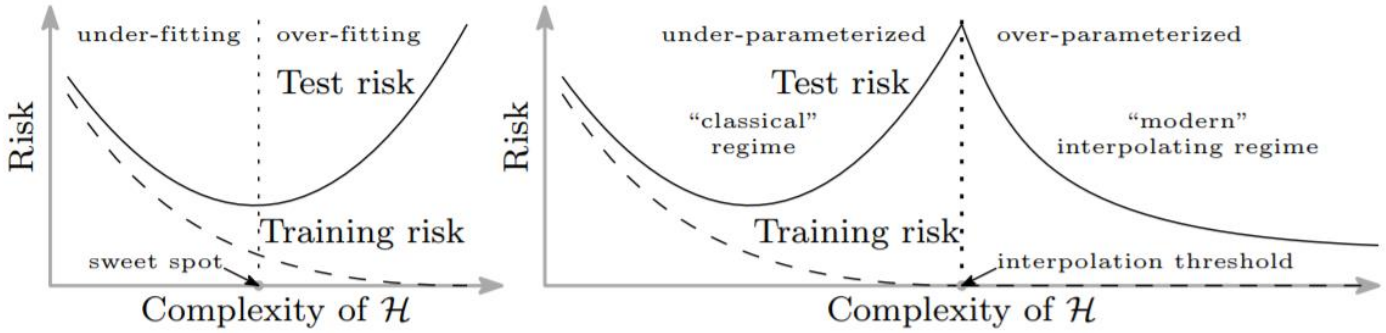
$$w_j = w_j - \alpha \frac{\partial J(w)}{\partial w_j}$$

- Different ways to adapt:
  - Time based decay: $\alpha = \alpha_0 / (1 + \alpha_1 * \#epoch)$
  - Step decay: reduce the learning rate by a factor every few epochs.
  - Exponential decay: $\alpha = \alpha_0 e^{-\alpha_1 * \#epoch}$
- This will add an hyperparameter $\alpha_1$ to search for.

# Bias-Variance Tradeoff

# Revisiting the concept (Advanced)

- In deep learning (and other modern learners) the need for a bias-variance tradeoff is not always necessary. Empirically it has been shown that while fitting the training set data almost perfectly we can still obtain good generalization performance.
- The traditional "**U-shape curve**" ➔ "**Double descent**" risk curve.
- In deep learning it is possible to decrease the bias without affecting the variance and the variance without affecting the bias.




(a) U-shaped "bias-variance" risk curve (b) "double descent" risk curve

Belkin, Mikhail, et al. "Reconciling modern machine learning and the bias-variance trade-off." arXiv preprint arXiv:1812.11118 (2018).

AIMLab.

# Dealing with overfitting

- Reminder of LR regularization using L1 and L2:
  - $J(w) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}log\left(h_w\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log(1 - h_w\left(x^{(i)}\right))\right] - \frac{\lambda}{2m}\sum_{j=1}^{n_x} w_j^q$
  - If $q = 2$ this is known as **Ridge Regression**. It makes use of the $L2$ norm.
  - If $q = 1$ this is known as **Lasso Regression**. It makes use of the $L1$ norm.

AIMLab.

# Dealing with overfitting – Frobenius norm regularization

- **Frobenius norm and regularization**:
  - How about regularizing a neural network?
    - $J\left(W^{[1]}, b^{[1]}, \dots, W^{[\mathrm{L}]}, b^{[\mathrm{L}]}\right) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|W^{[l]}\right\|_F^2$
    - $F$ stands for the **Frobenius norm**: matrix norm defined as the square root of the sum of the absolute squares of its elements.
    - Where $W^{[\mathrm{L}]} \in \mathbb{R}^{n_h^{[l]} \cdot n_h^{[l-1]}}$ and, $\left\|W^{[l]}\right\|_F^2 = \sum_{i=1}^{n_h^{[l-1]}}\sum_{i=1}^{n_h^{[l]}}(w_{ij}^{[l]})^2$

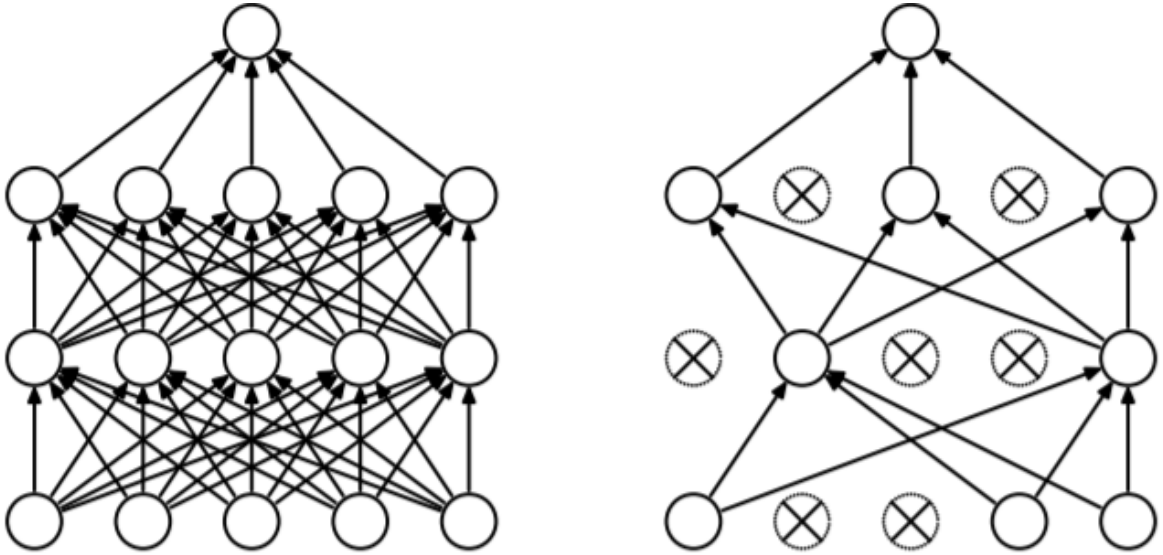
AIMLab.

# Dealing with overfitting – Frobenius norm regularization

- Frobenius norm and regularization:
  - Backpropagation:
    - $dW^{[l]} := dW^{[l]}_{no\ reg} + \frac{\lambda}{m} W^{[l]}$
    - $W^{[l]} := W^{[l]} - \alpha \cdot dW^{[l]} = W^{[l]} \left(1 - \frac{\alpha\lambda}{m}\right) - \alpha \cdot dW^{[l]}_{no\ reg}$
    - Thus the effect of regularization is to weight down a bit the $W^{[l]}$ during backpropagation. The $W^{[l]}$ will get smaller.

# Dealing with overfitting – Dropout

- Learn a fraction of the weights in the network at each training iteration.
- Intuition: spread out the weights across the network in order to rely on many features.



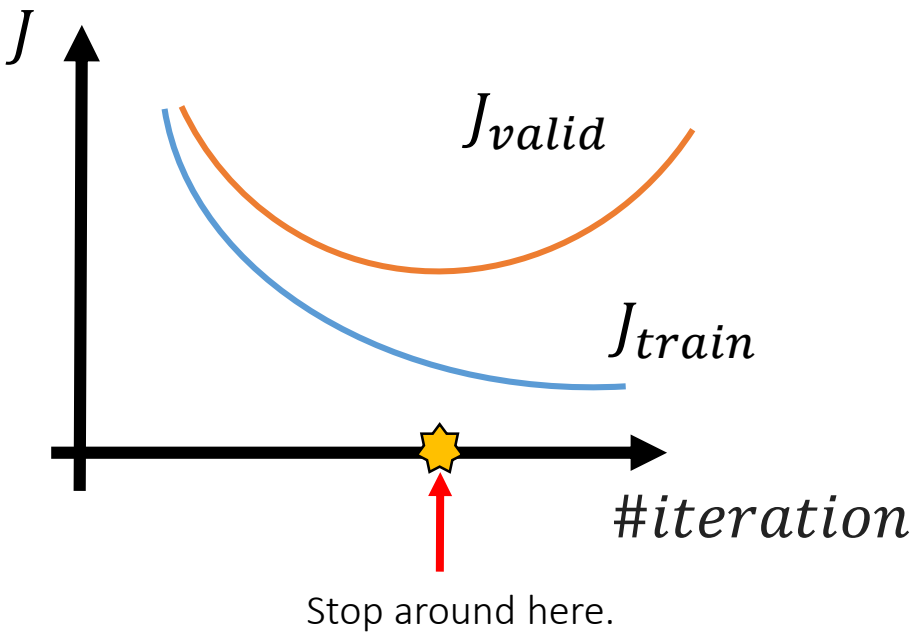
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

# Dealing with overfitting and high bias

- Other methods for dealing with overfitting:
  - Data augmentation
    - E.g. rotation, distortion.
  - Early stopping.
- Dealing with high bias:
  - Bigger network.
  - Train longer.



# Batch normalization

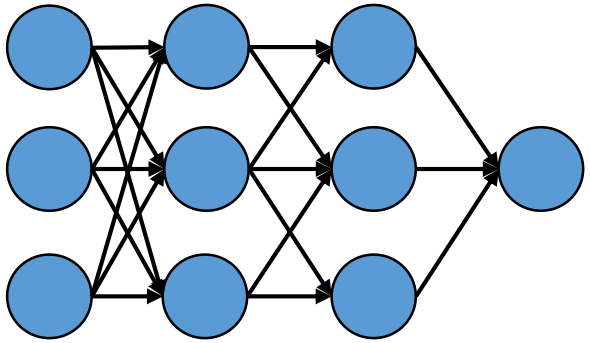
# Batch normalization

- Batch normalization: consists of normalizing the mean and variance of each of the features at every level of representation during training.
- It enables:
  - Using higher learning rates thus enables to accelerate the learning process.
  - Help with the training of very deep networks.
- Intuition:
  - **Covariance shift** refers to the change in the distribution of the input values to a given learning algorithm.

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

# Batch normalization

- In neural network:
  - The weights at each layer change at each iteration of gradient descent.
  - This means that, from the point of view of a layer further in the network, the values of its inputs may change importantly at each iteration because the weights of the previous layers have been updated.
  - This leads to a situation similar to the “covariance shift” concept but for each layer.
  - Batch normalization enables to learn more stable distribution of inputs.



AIMLab.

# Batch normalization

- **Batch norm**: normalization of the features across the $p$ examples in each mini-batch:
  - $\mu^{[l]} = \frac{1}{p}\sum_{i=1}^{p} z^{[l](i)}$
  - $(\sigma^{[l]})^2 = \frac{1}{p}\sum_{i=1}^{p}(z^{[l](i)} - \mu^{[l]})$
  - $z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]^2} + \varepsilon}}$
- To leave some flexibility on the distribution mean and variance:
  - $\tilde{z}^{[l](i)} = \gamma^{[l]} z_{norm}^{[l](i)} + \beta^{[l]}$
- The set of parameters we want to learn becomes $\{W^{[l]}, \gamma^{[l]}, \beta^{[l]}\}$.

# Batch normalization

- Batch normalization: normalize the activations of each layer and this is improving the learning speed and outcomes.
- The intuition of “limiting covariance shift” behind Batch normalization but this has been challenged since.

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
Santurkar, Shibani, et al. "How does batch normalization help optimization?." Advances in Neural Information Processing Systems. 2018.

# Take home

- Optimization algorithms:
  - Gradient descent: stochastic, batch and mini-batch.
  - Momentum, RMSprop, Adam etc.
  - Learning rate adaptation.
- Vocabulary:
  - **Epoch**: a single pass through all training set examples.
  - **Mini-batch**: compute the gradient of the cost function for $p$ training examples.
    - Trade-off between the accuracy of the parameter update and the time it takes to perform an update.

# Take home

- Revisiting some paradigms:
  - Train-validation-test set split.
  - Bias-variance and “double descent” curves.
  - Regularization using Frobenius norm, dropout, data augmentation etc.
- Batch normalization.

# References

[1] Andrew Ng, Coursera, Neural Networks and Deep Learning. Coursera.

[2] Initializing neural network: https://www.deeplearning.ai/ai-notes/initialization/

[3] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747(2016).

[4] Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research 13.Feb (2012): 281-305.

[5] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.