Introduction to Deep Learning for Computer Vision

Adhyayan '23 - ACA Summer School
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Lecture 2

Training Neural Networks!

Loss Optimization

- We want to find network weights that ***achieve the lowest loss***.

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

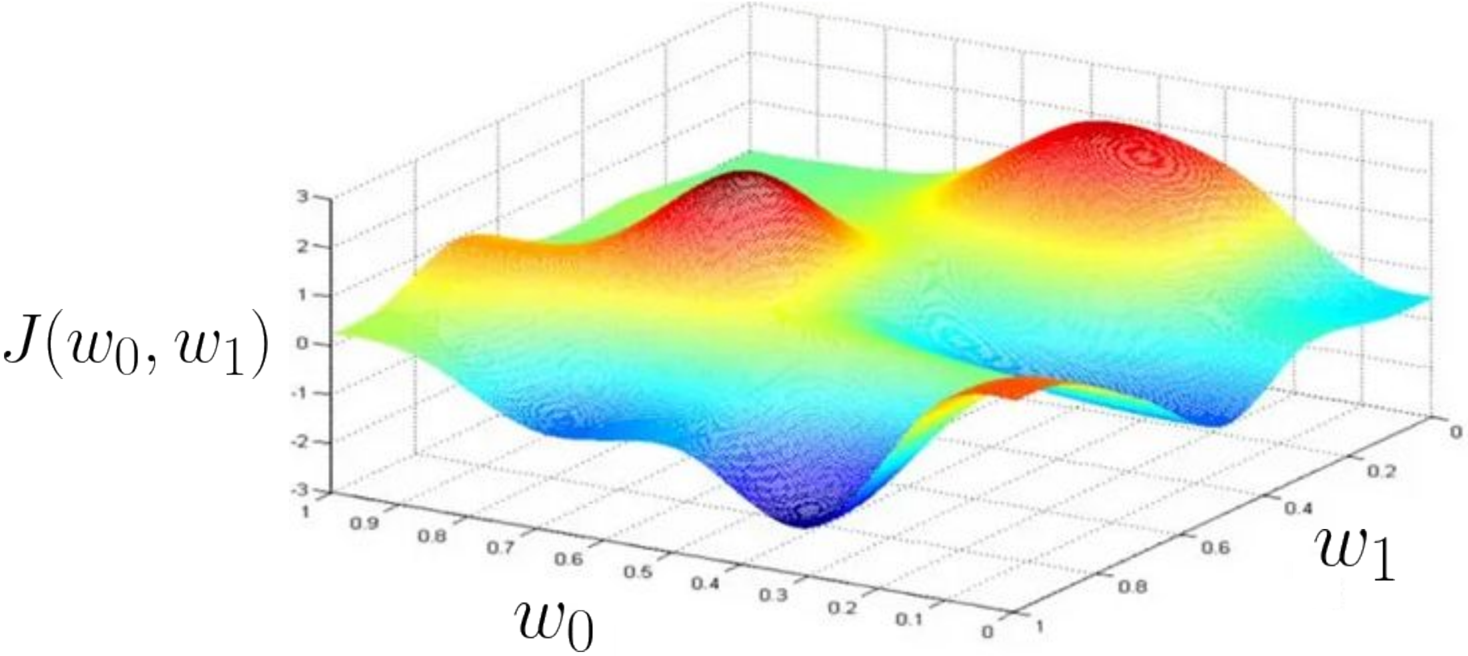
Remember:

$$\mathbf{W} = \{W^{(0)}, W^{(1)}, \cdots\}$$

Loss Optimization

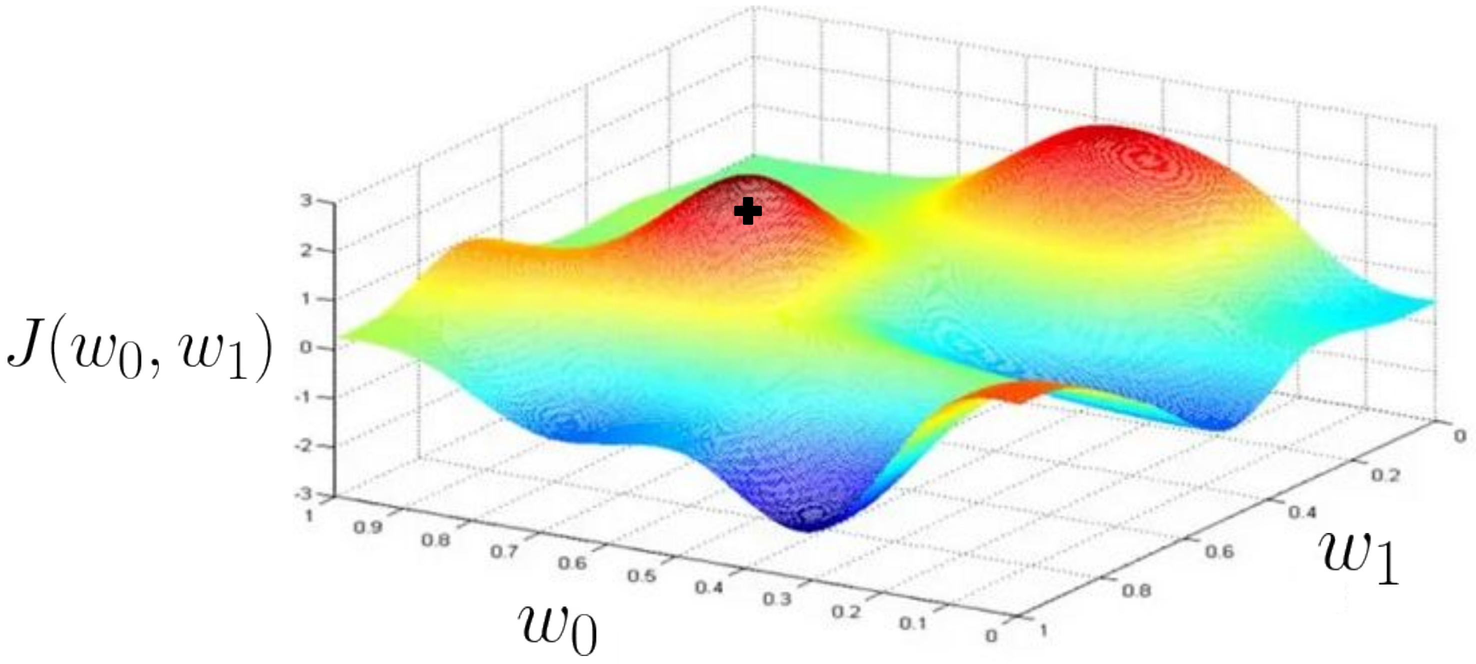
$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

Remember:
Our loss is a function of the network weights.



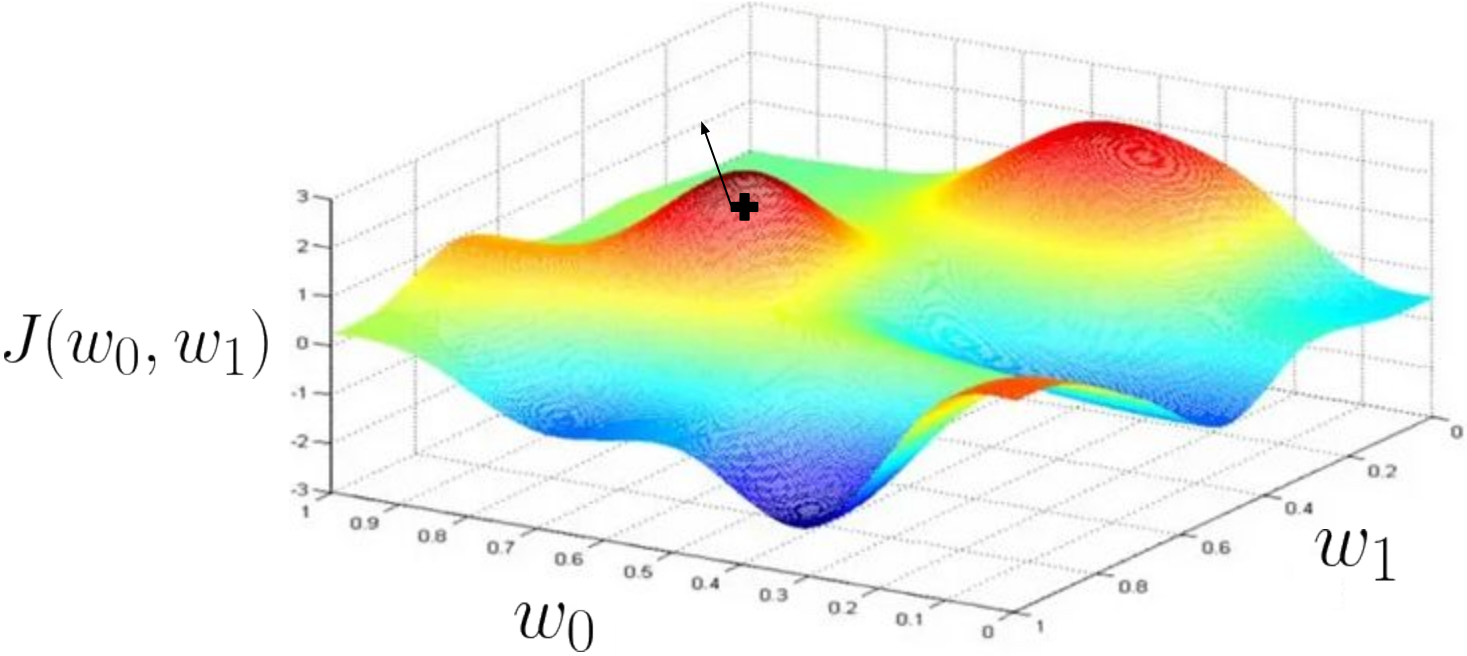
Loss Optimization

Randomly pick an initial (w_0, w_1)



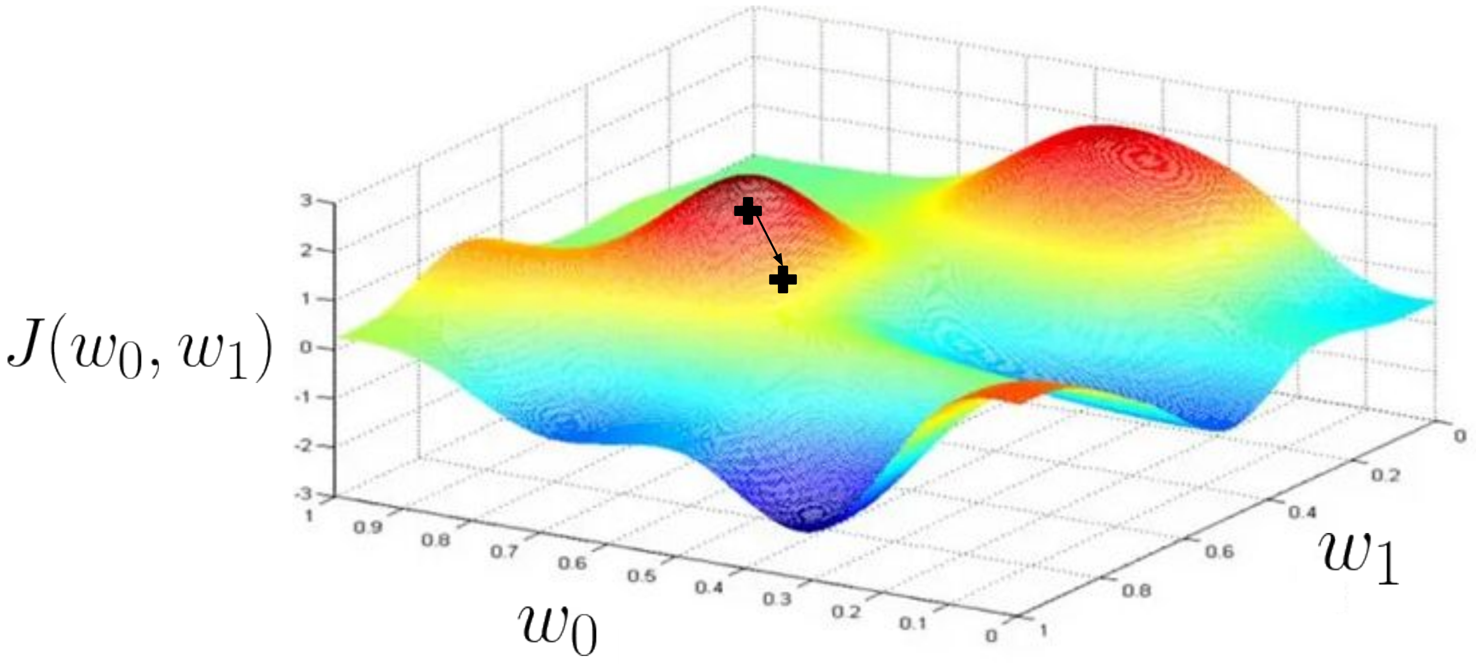
Loss Optimization

Compute gradient, $\frac{\partial J(W)}{\partial W}$



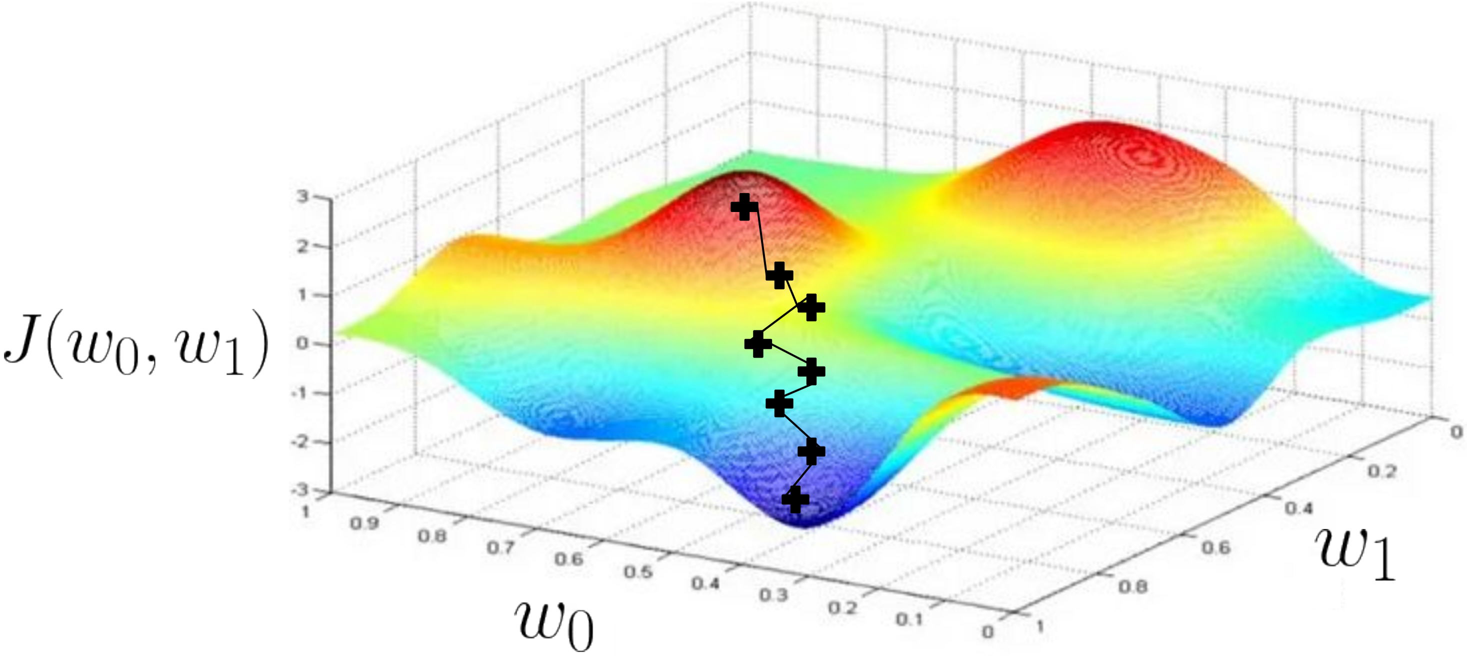
Loss Optimization

Take small step in opposite direction of the gradient



Gradient Descent

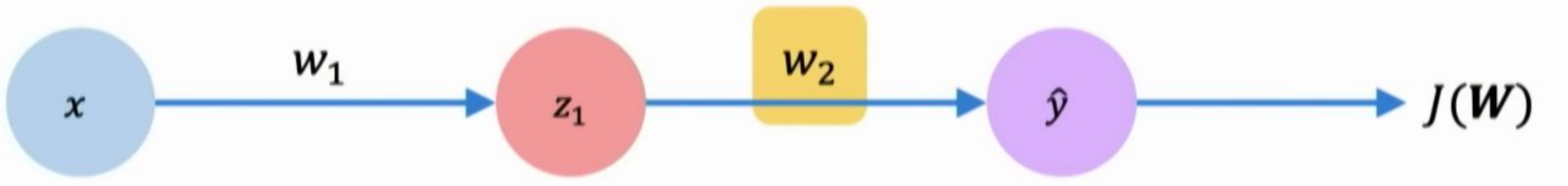
Repeat Until Convergence



Gradient Descent

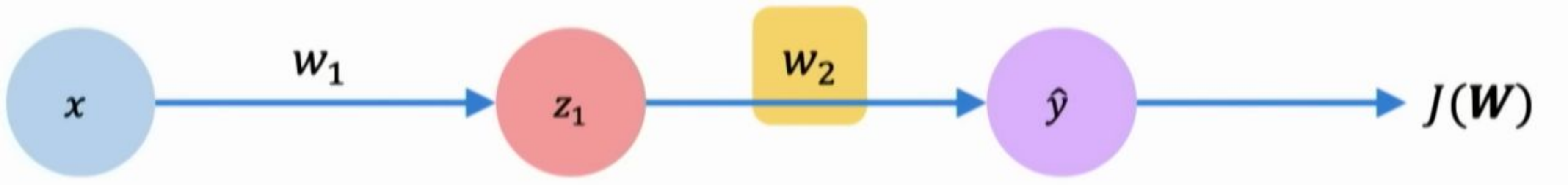
- Algorithm:
  1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
  2. Loop until convergence:
  3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
  4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
  5. Return weights

Computing Gradients: Backpropagation



How does a small change in one weight (ex. $\boldsymbol{w_2}$) affect the final loss $J(\boldsymbol{W})$

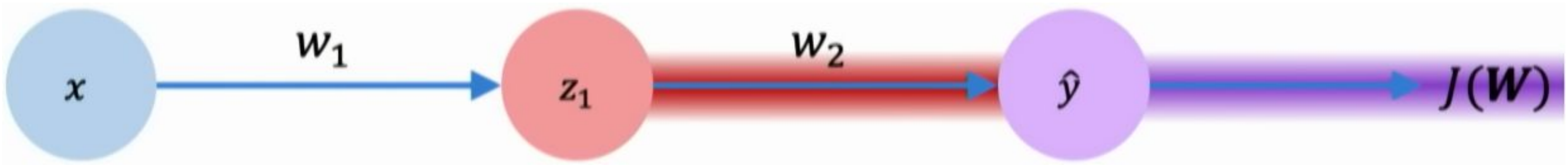
Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_2}$$

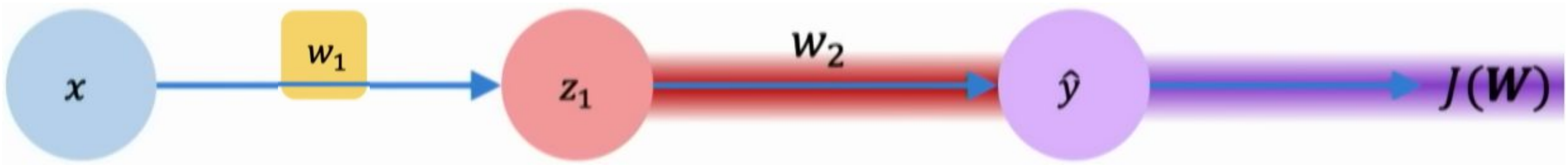
Let's use chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(w)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2}$$

Computing Gradients: Backpropagation

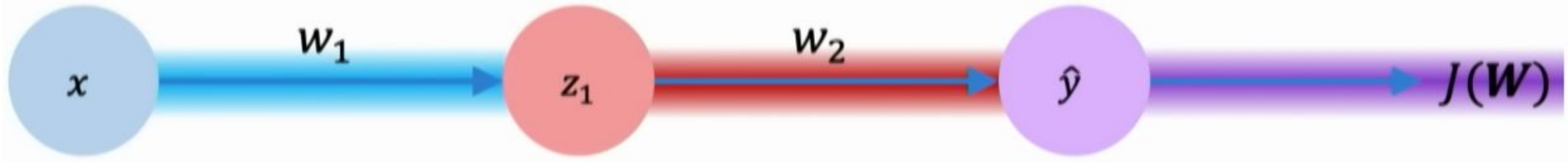


$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(w)}{\partial \hat{y}}} \times \underbrace{\frac{\partial \hat{y}}{\partial w_1}}$$

Apply Chain Rule

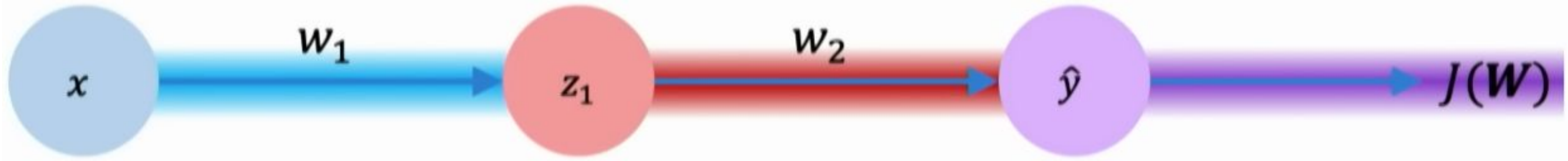
Apply Chain Rule

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(w)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(w)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Neural Networks in Practice: Optimization!

Training Neural Networks is Difficult!

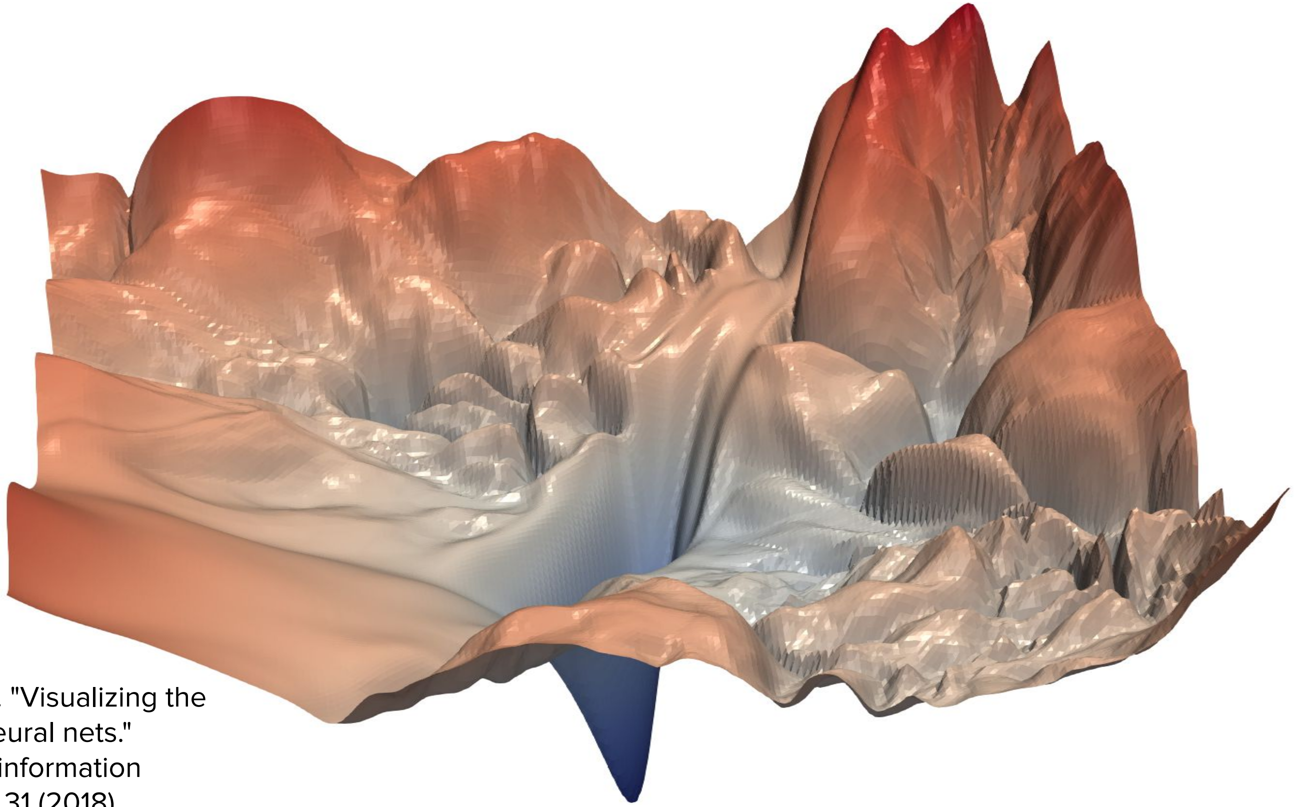


Image: Li, Hao, et al. "Visualizing the loss landscape of neural nets." Advances in neural information processing systems 31 (2018).

Loss Functions Can Be Difficult To Optimize

Remember: Optimization through Gradient Descent

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

Loss Functions Can Be Difficult To Optimize

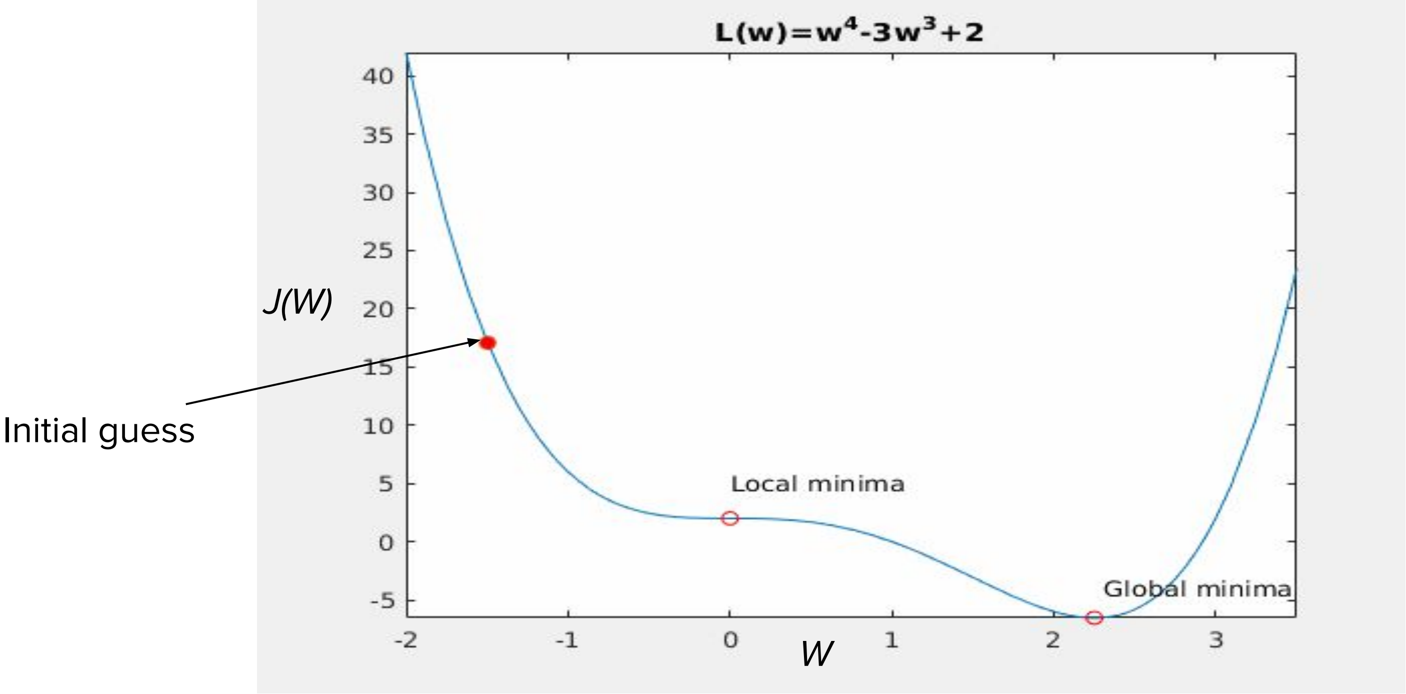
Remember: Optimization through Gradient Descent

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

How can we set the learning rate?

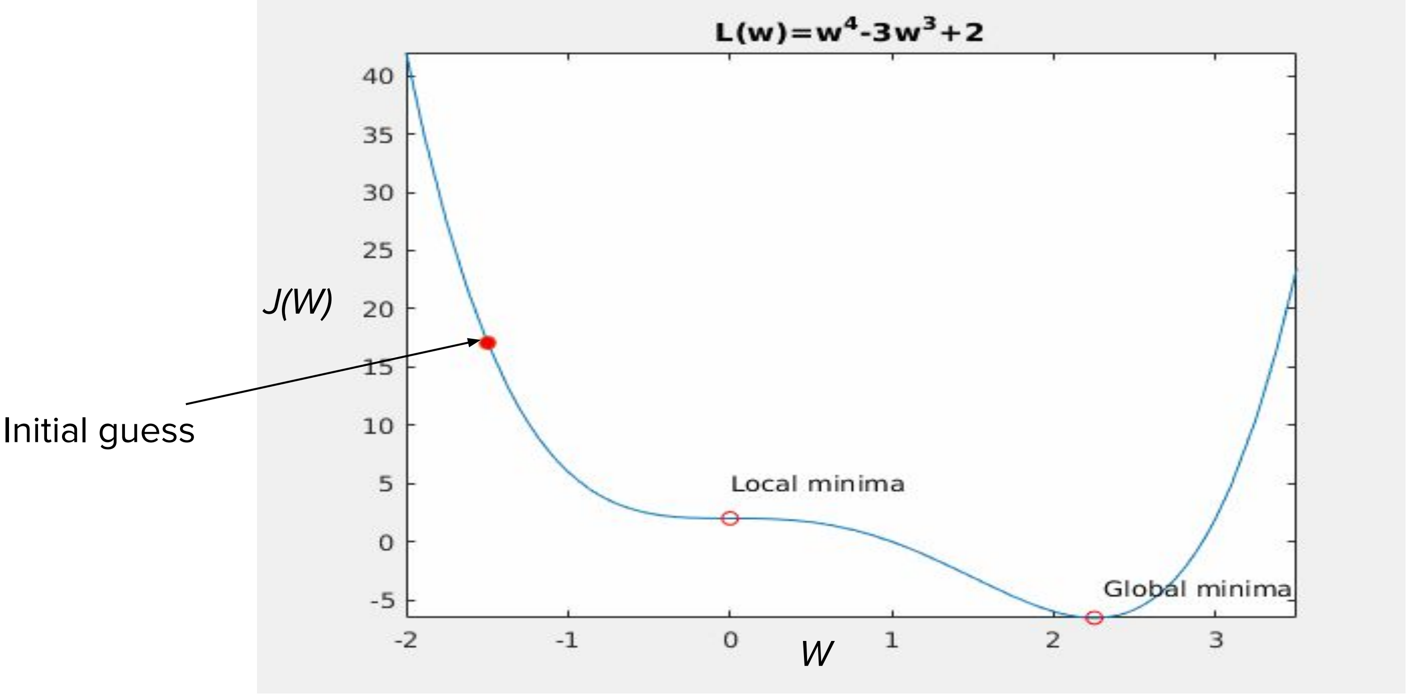
Setting the Learning Rate

Small learning rates *converge slowly and get stuck in a false local minima.*



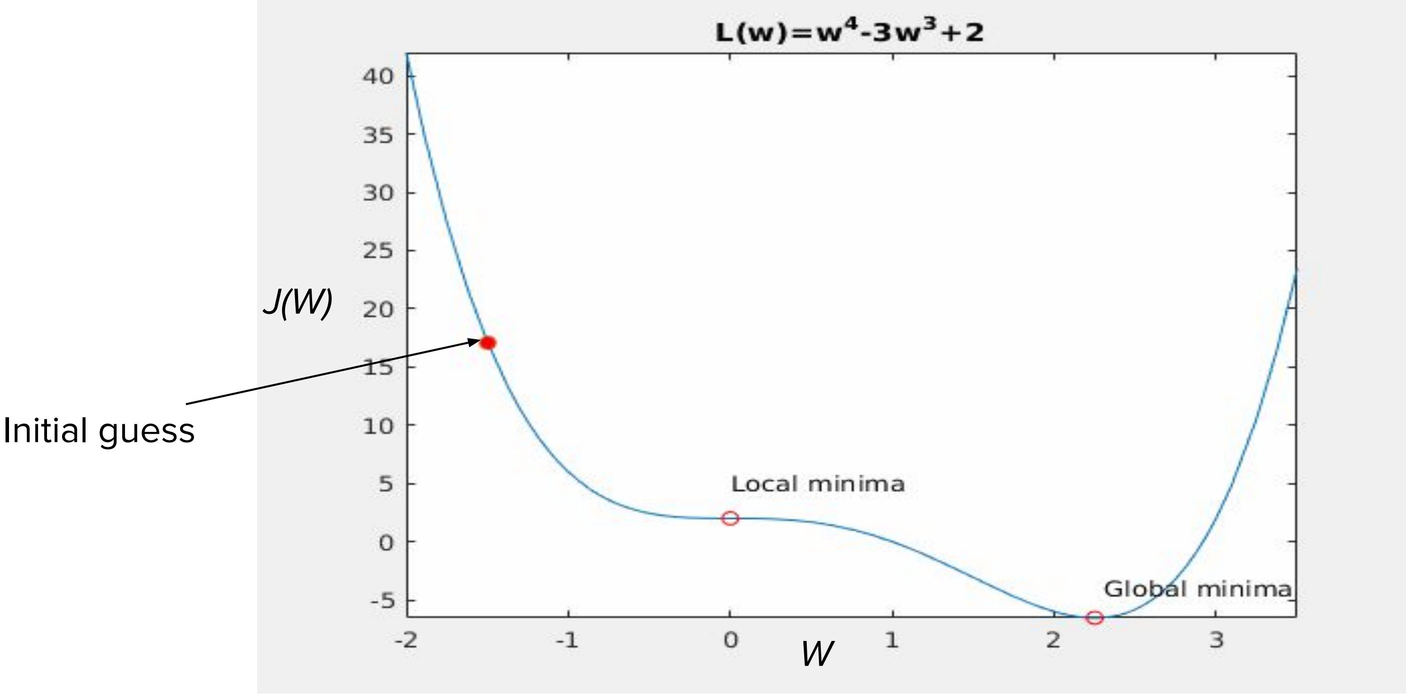
Setting the Learning Rate

Large learning rates *overshoot, become unstable and diverge.*



Setting the Learning Rate

Stable learning rates *converge smoothly and avoid local minima.*



How to deal with this?

Idea 1:

Try lots of different learning rates and see what works “just right”.

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works “just right”.

Idea 2:

Do something smarter!

Design an adaptive learning rate that “adapts” to the landscape!

Adaptive Learning Rates

- Learning Rates are no longer fixed
- Can be made larger or smaller depending on:
  - How large gradient is
  - How fast learning is happening
  - Size of particular weights
  - etc…

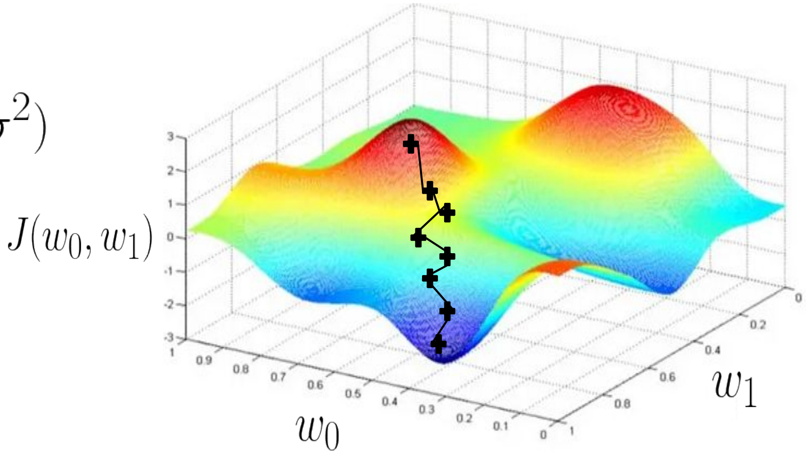
Gradient Descent Algorithms

Algorithm	Reference
SGD	Kiefer, Jack, and Jacob Wolfowitz. "Stochastic estimation of the maximum of a regression function." *The Annals of Mathematical Statistics* (1952): 462-466.
Adam	Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).
Adadelta	Zeiler, Matthew D. "Adadelta: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).
Adagrad	Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research* 12.7 (2011).
RMSProp	Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26–31.

Neural Networks in Practice: Mini-batches!

Gradient Descent

- Algorithm:
  1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
  2. Loop until convergence:
  3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
  4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
  5. Return weights

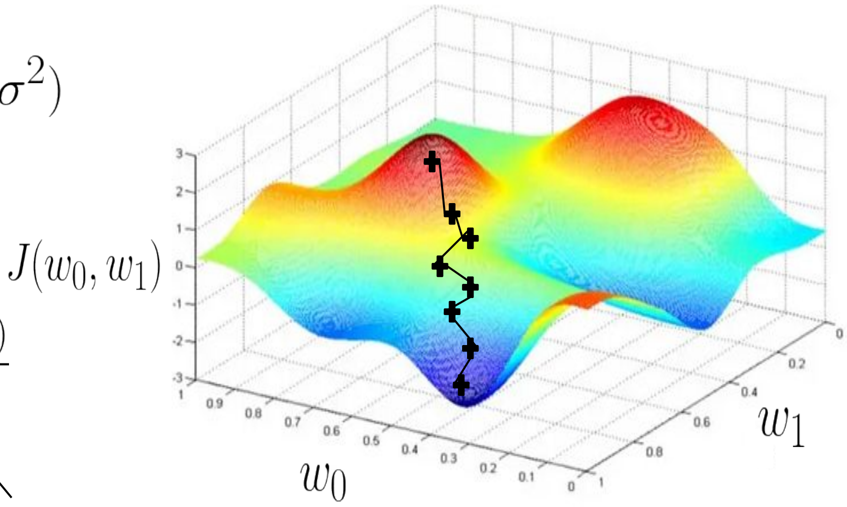


Can be very **computationally intensive** to compute.

Stochastic Gradient Descent

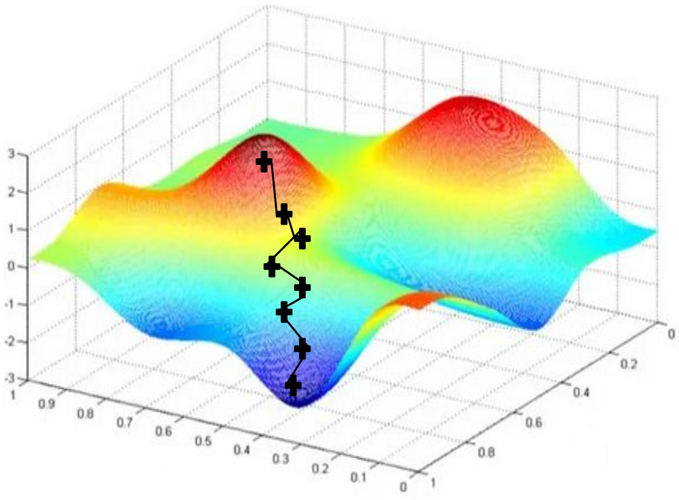
- Algorithm:
  1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
  2. Loop until convergence:
  3. Pick a single data point ***i***
  4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$
  5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
  6. Return weights

Easy to compute but
very noisy (stochastic)!



Stochastic Gradient Descent

- Algorithm:
  1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
  2. Loop until convergence:
  3. Pick a batch of ***B*** data points.
  4. Compute gradient, $\frac{\partial J(W)}{\partial W} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$
  5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
  6. Return weights



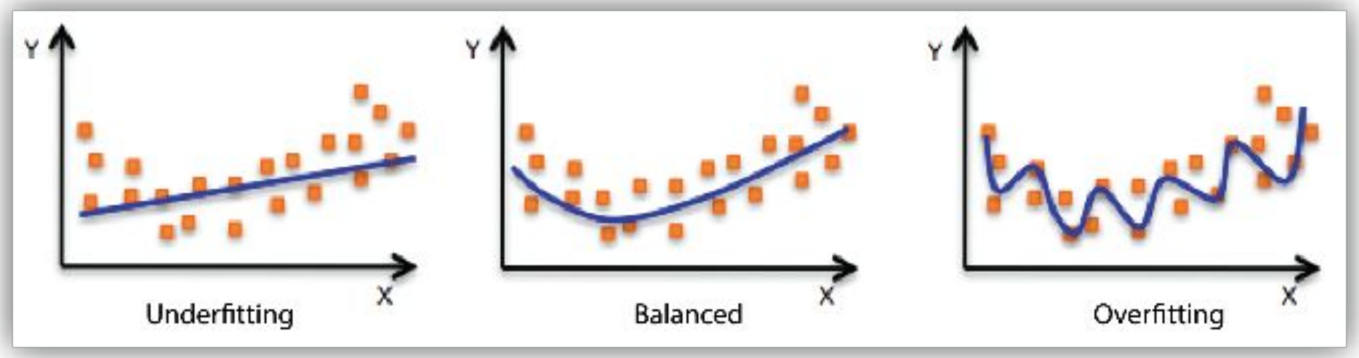
Fast to compute and a much better estimate of the true gradient!

Mini-batches while training

- More accurate estimation of gradient:
  - Smoother convergence
  - Allows for larger learning rate
- Mini-batches lead to fast training!
  - Can parallelize computation
  - Achieve significant speed increases of GPUs!

Neural Networks in Practice: Overfitting!

The Problem of Overfitting



Model does not have capacity to fully learn the data

Too complex, extra parameters do not generalize well

Regularization

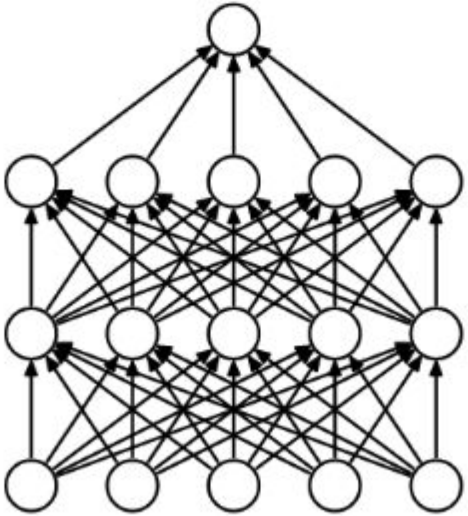
- What is it?
  - Technique that constrains our optimization problem to discourage complex models

Regularization

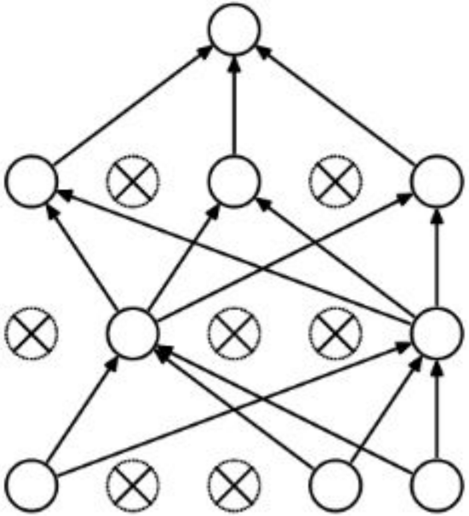
- What is it?
  - Technique that constrains our optimization problem to discourage complex models
- Why do we need it?
  - Improve generalization of our model on unseen data.

Regularization I : Dropout!

- During training, randomly set some activations to 0.
  - Typically ‘drop’ 50% of activations in layer
  - Forces the network to not rely on any one node.



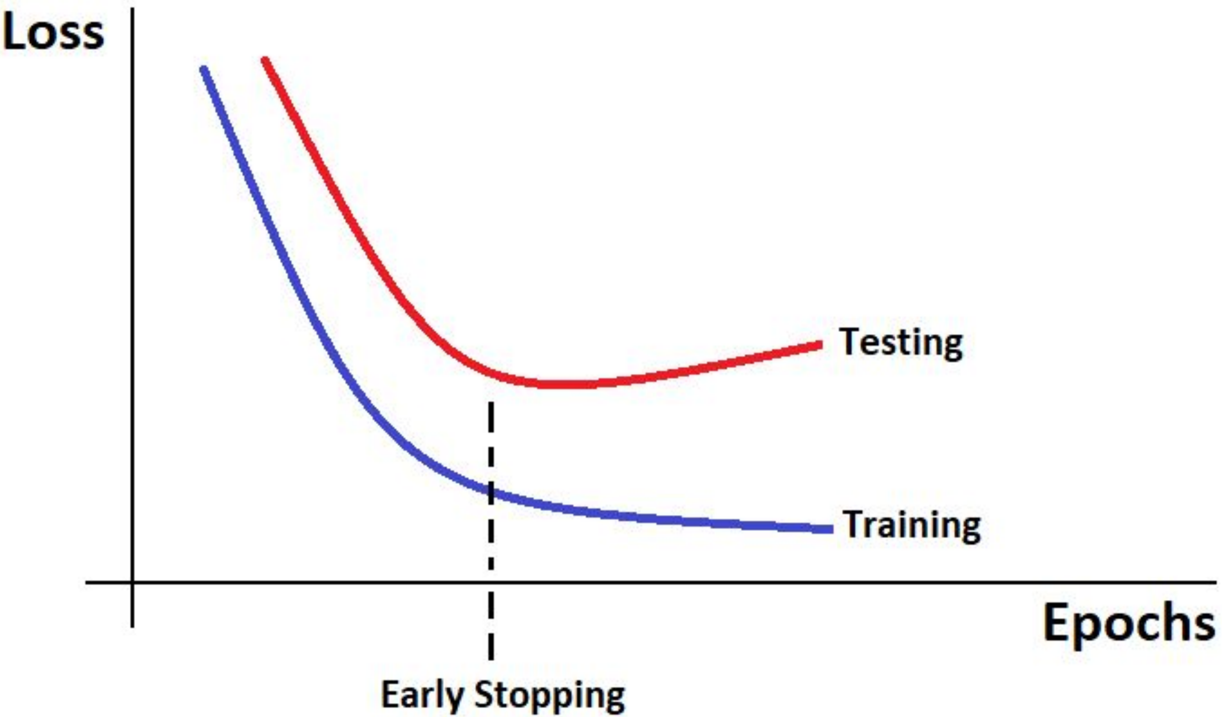
(a) Standard Neural Net



(b) After applying dropout.

Regularization II: Early Stopping

- Stop training before we have a possibility to overfit.



Next Lecture: Convolutional Neural Networks!