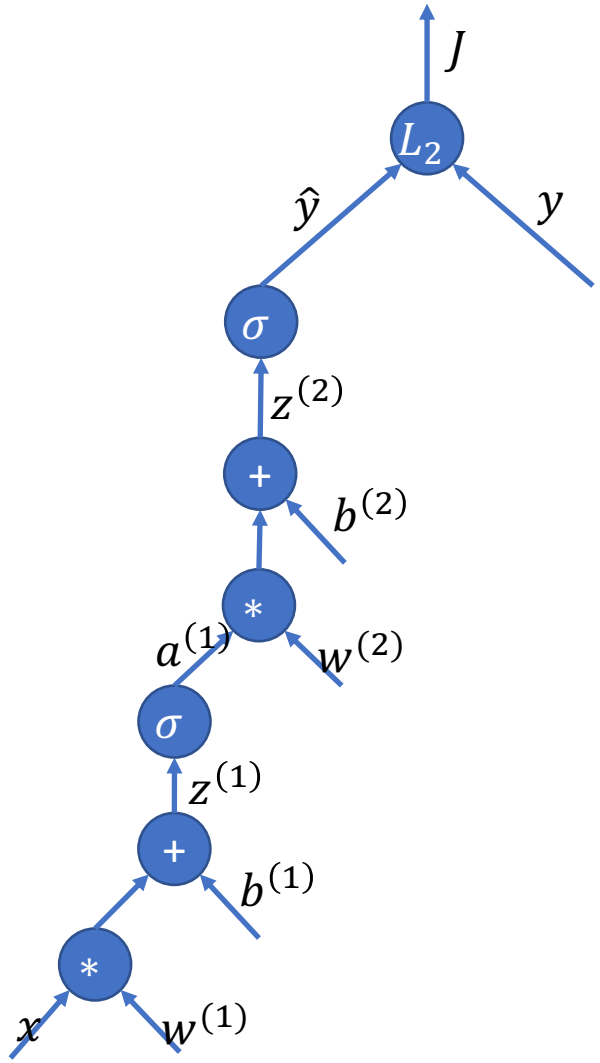
Regularization

Lecture 4

Overview

- **Background**
- Weight Norm Regularization
- Other Types of Regularization
- Dropout

Review: Backpropogation



$$\frac{dJ}{d\hat{y}} ; \frac{dJ}{dy}$$

$$\frac{d\hat{y}}{dz^{(2)}}$$

$$\frac{dz^{(2)}}{dw^{(2)}} ; \frac{dz^{(2)}}{db^{(2)}} ; \frac{dz^{(2)}}{da^{(1)}}$$

$$\frac{da^{(1)}}{dz^{(1)}}$$

$$\frac{dz^{(1)}}{dw^{(1)}} ; \frac{dz^{(1)}}{db^{(1)}} ; \frac{dz^{(1)}}{dx}$$

Review: Cost Function

Mean Squared Error: $J = \frac{1}{2}(\hat{y} - y)^2$ $\frac{dJ}{d\hat{y}} = \hat{y} - y$

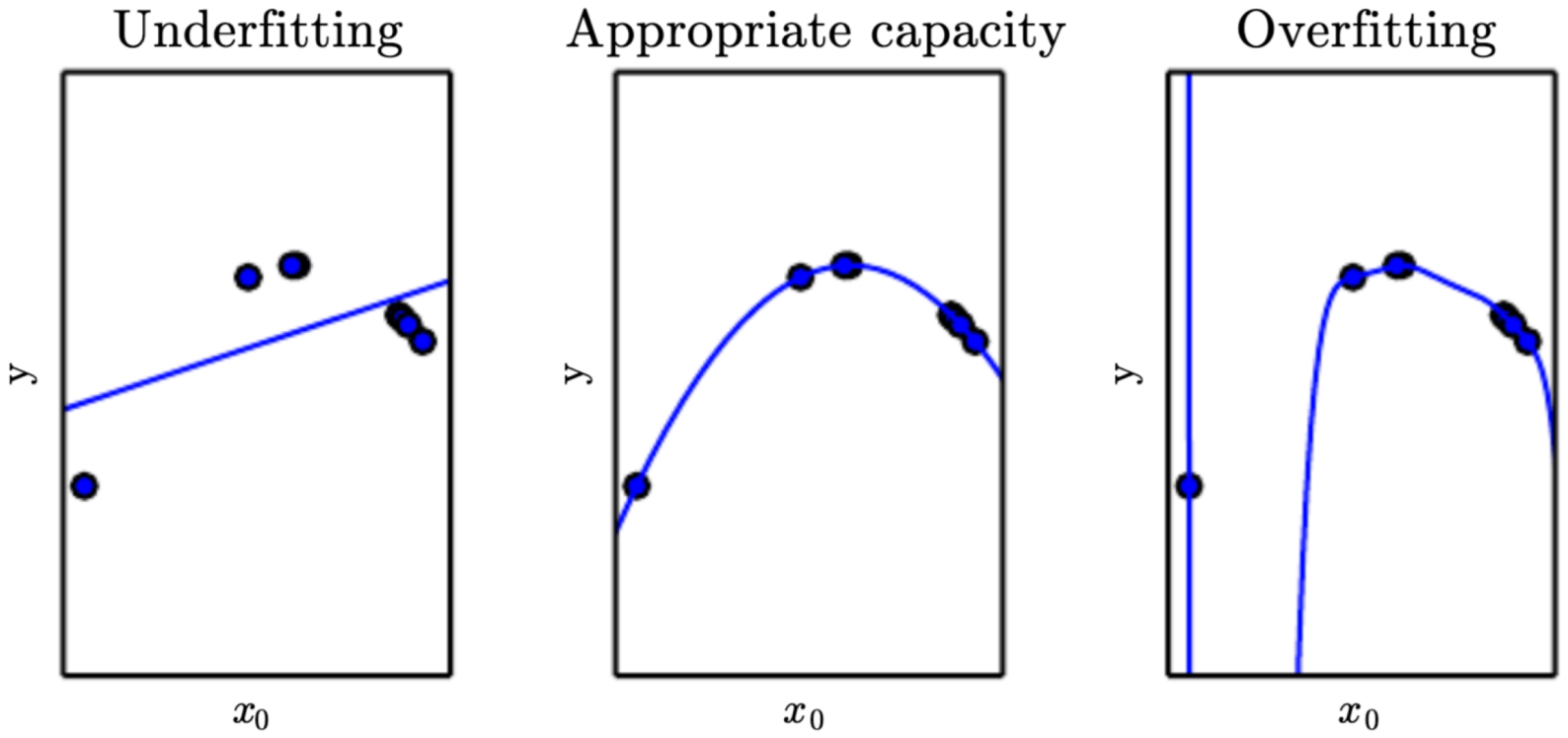
Cross Entropy: $J = y \log \hat{y} + (1 - y) \log 1 - \hat{y}$ $\frac{dJ}{d\hat{y}} = y - \hat{y}$

Review: Over- and Under-fitting

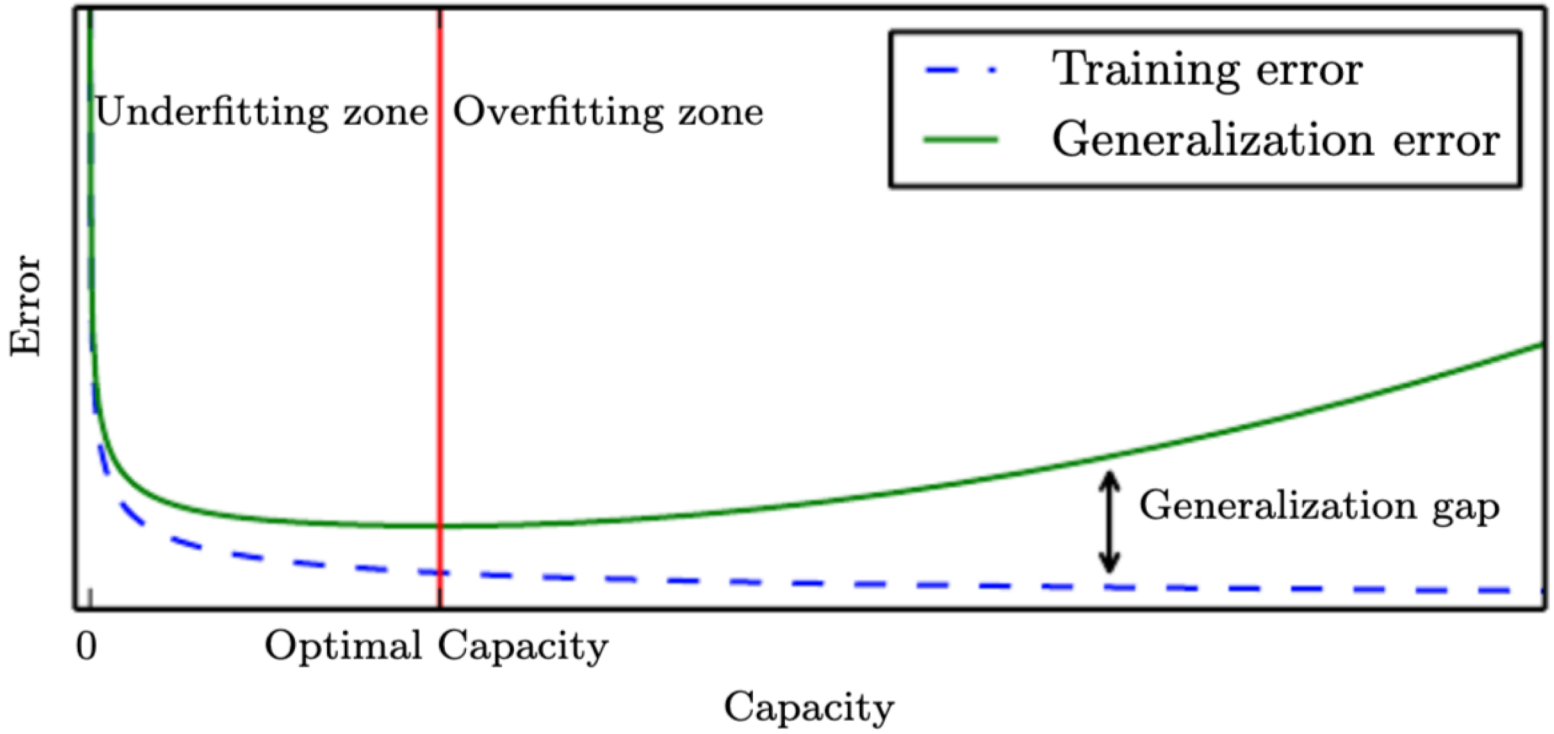
$$\hat{y} = b + wx$$

$$\hat{y} = b + w_1 x + w_2 x^2$$

$$\hat{y} = b + \sum_{i=1}^{k} w_i x^i$$



Review: Generalization Gap



Regularization

From the book –

“any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

- Can be intended to encode specific prior knowledge
- In other cases, simplify the model structure to promote generalization

Overview

- Background
- **Weight Norm Regularization**
- Other Types of Regularization
- Dropout

Weight Norm Based Regularization

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + a\Omega(\theta)$$

$J(\theta; X, y) \rightarrow$ The original loss function

$a \rightarrow$ Arbitrary hyperparameter $\in [0, \infty)$

$\Omega(\theta) \rightarrow$ Norm penalty

L^2 Parameter Regularization

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + a\Omega(\theta)$$

$$\Omega(\theta) \rightarrow \frac{1}{2}\sum_i \theta_i^2$$

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \frac{a}{2}\sum_i \theta_i^2$$

- Also known as “ridge” or “Tikhonov” regularization
- Balances high weight values vs cost function

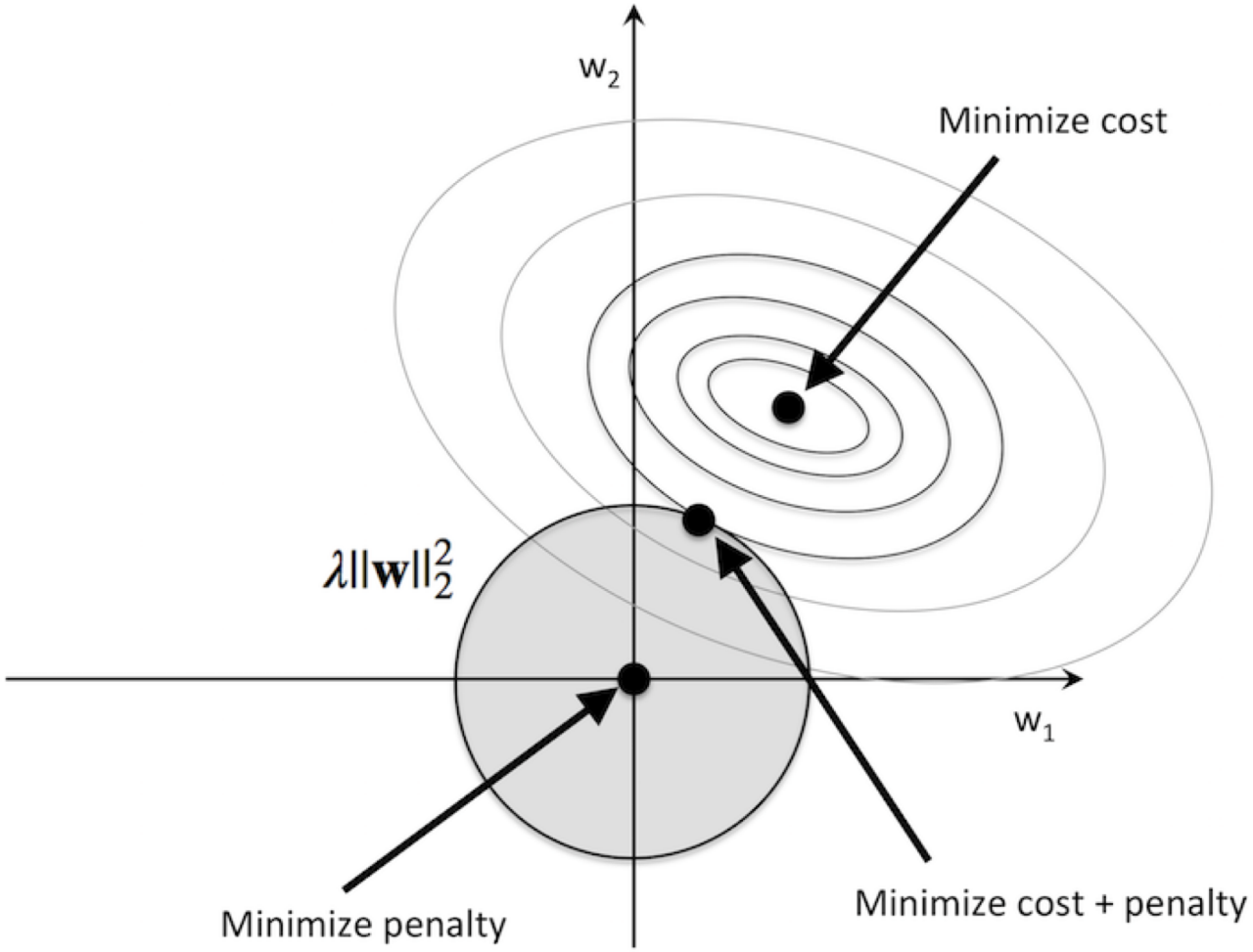
L^2 Parameter Regularization

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \frac{a}{2}\sum_i \theta_i^2$$

$$\frac{d\tilde{J}}{d\theta_i} = \frac{dJ}{d\theta_i} + a\theta_i$$

- Shrinks model weight values

L^2 Parameter Regularization



L^1 Parameter Regularization

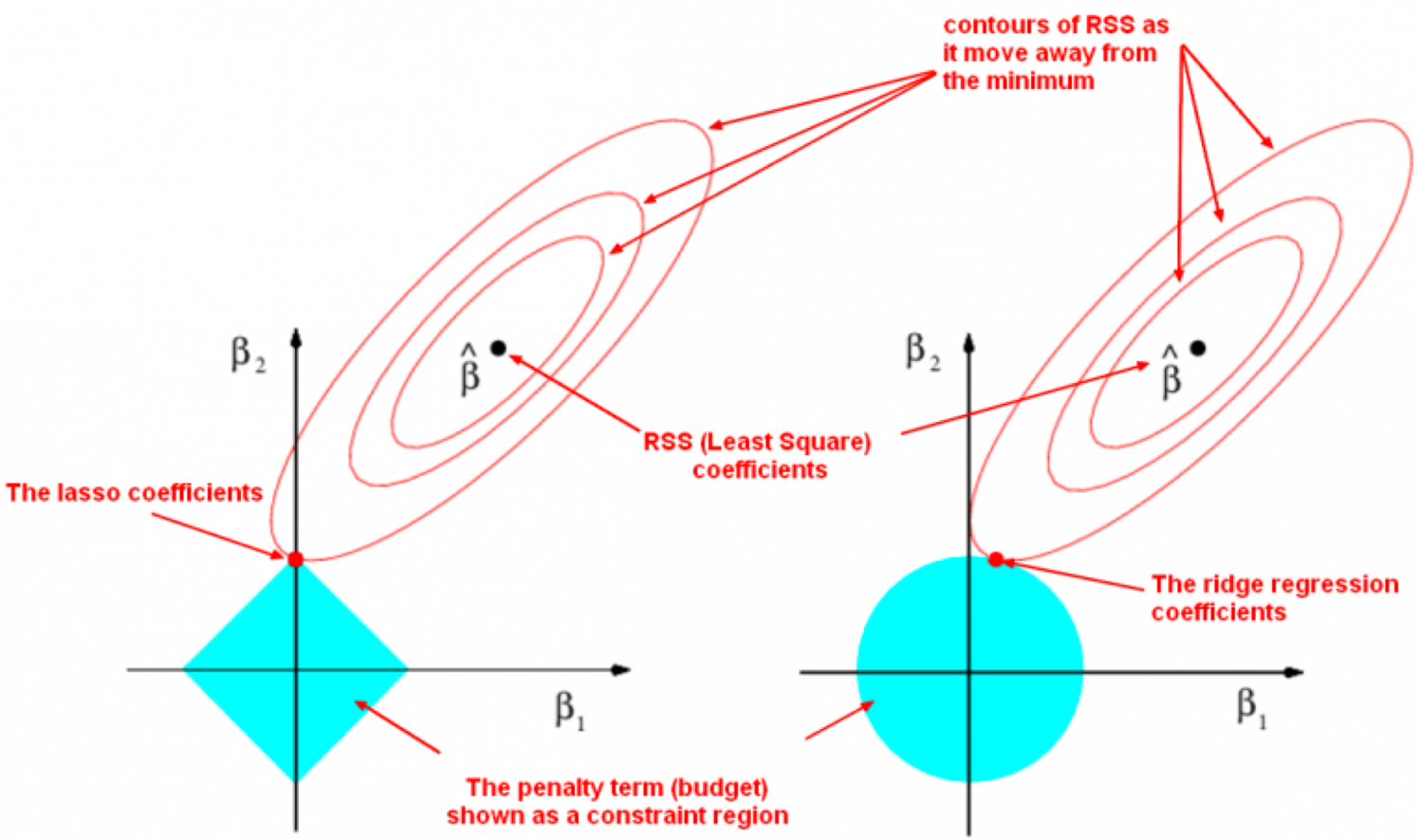
$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + a\Omega(\theta)$$

$$\Omega(\theta) \rightarrow \sum_i |\theta_i|$$

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \sum_i |\theta_i|$$

- Also known as “lasso” regularization
- Balances high weight values vs cost function

L^1 vs L^2 Parameter Regularization



L^1 L^2

Weight norm regularization

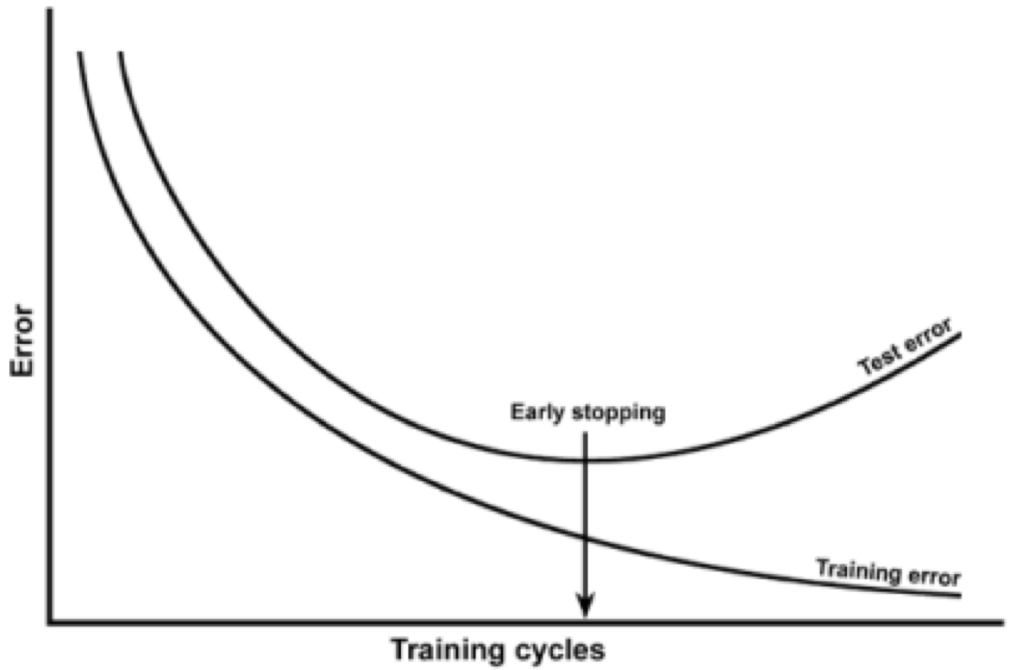
- Relies on the assumption that smaller weights create a simpler model
- Two most common weight norms are L^1 and L^2
- L^1 creates sparse models
- L^2 creates balance between weights

Overview

- Background
- Weight Norm Regularization
- **Other Types of Regularization**
- Dropout

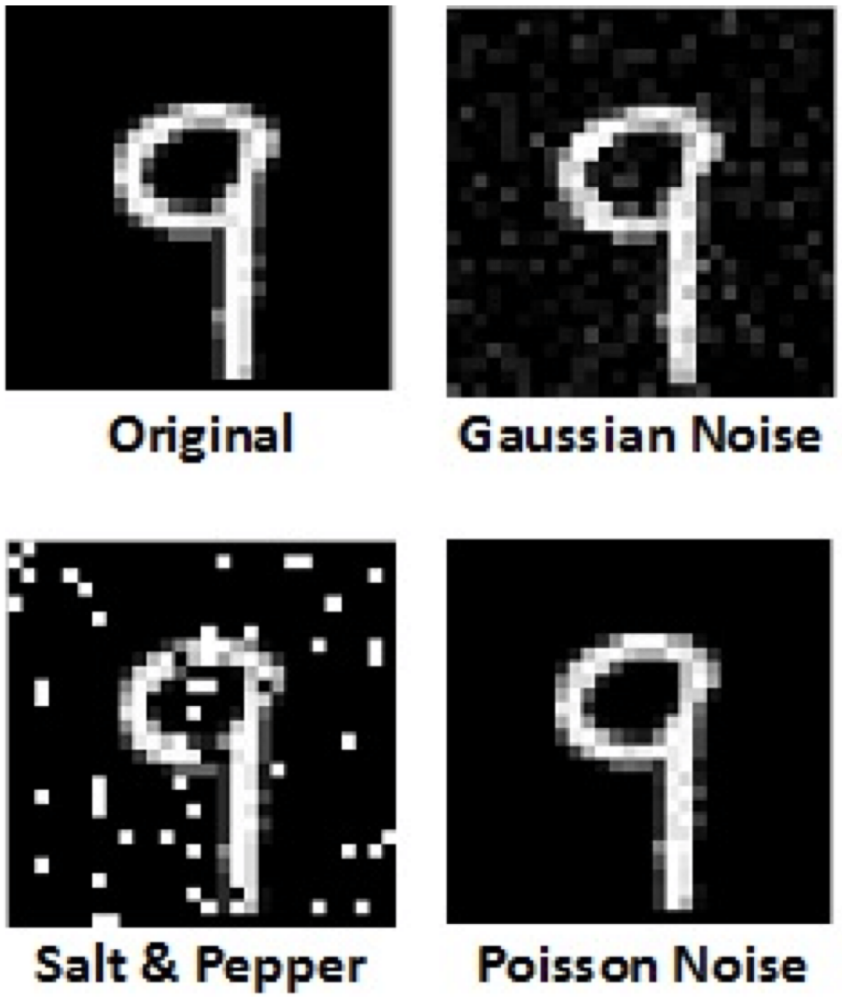
Early Stopping

- Stop training the model when the generalization error stops decreasing
- Typically set some threshold (i.e. 1%) and specify if generalization error does not drop by the threshold over some number of epochs, stop training the model



Adding Noise

- Add noise, either randomly or following some meaningful distribution, to your data
- Prevents model from learning patterns that are not meaningful to your results



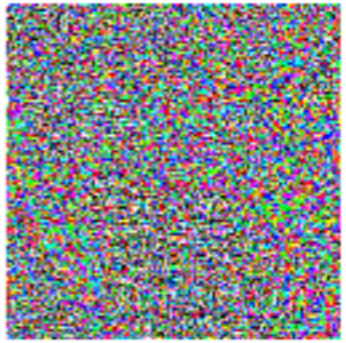
Adversarial Training

- Use a neural network to create a noise pattern that optimally throws off the network you are training

$\boldsymbol{x}$

$y =$"panda"
w/ 57.7%
confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
w/ 8.2%
confidence

$=$

$\boldsymbol{x} +$
$\epsilon \, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
w/ 99.3%
confidence

Overview

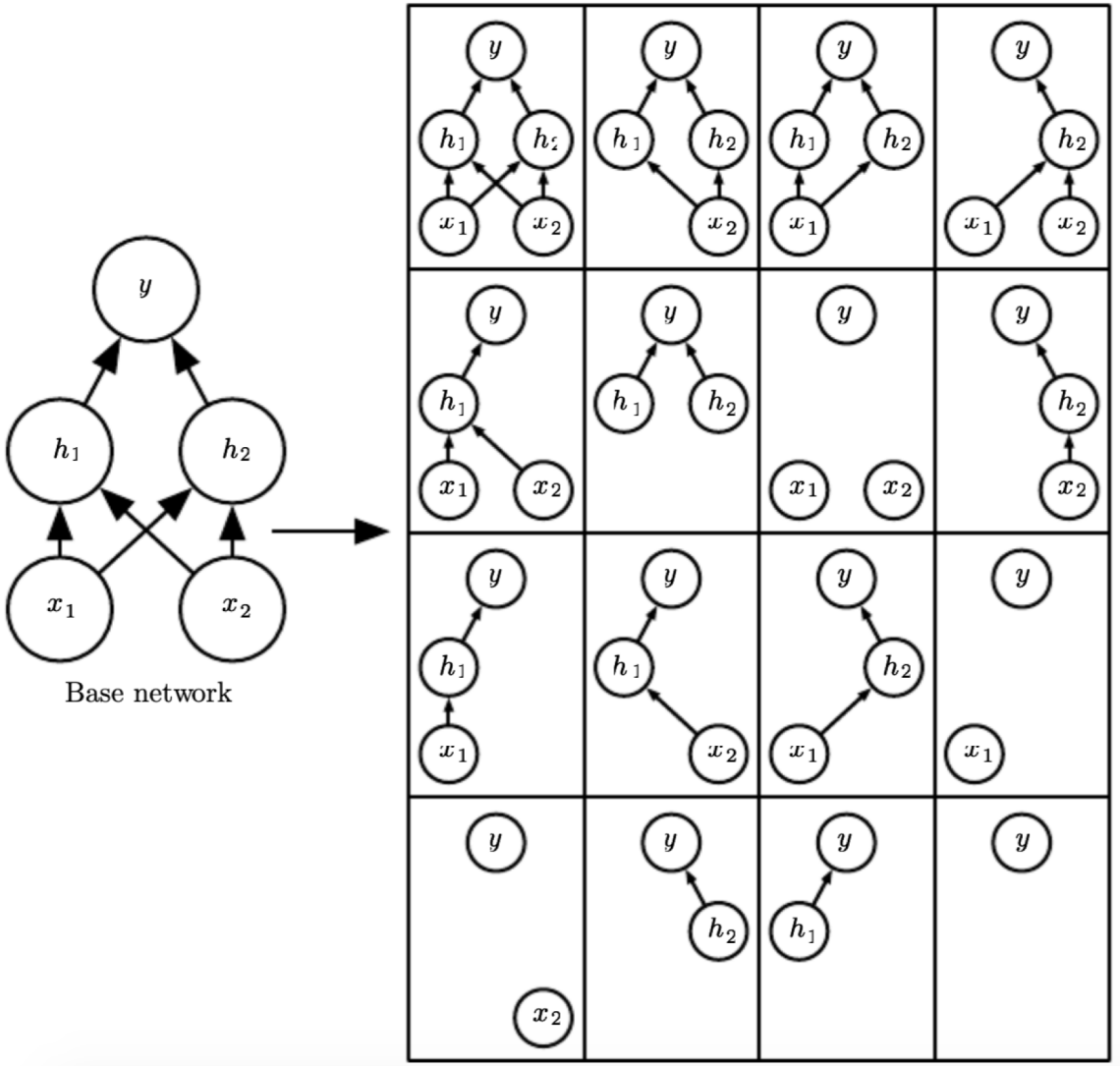
- Background
- Weight Norm Regularization
- Other Types of Regularization
- **Dropout**

Ensemble Learning

- For many problems, learning one model is good but learning multiple models is better
- Models are trained with with subsets of the data, different parameter values, etc
- Collection of final model is then averaged together

Ensemble Learning for Neural Networks

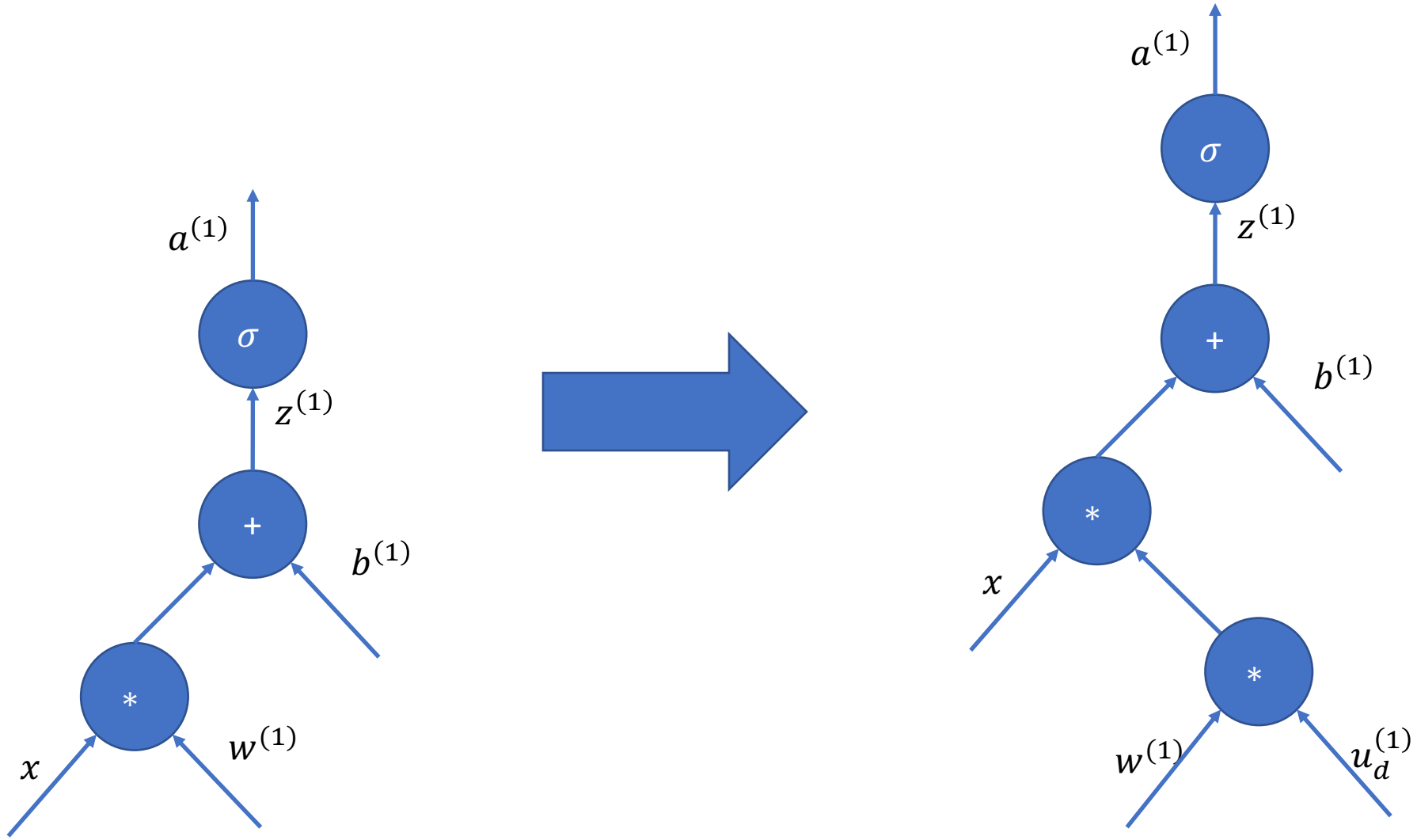
- Training multiple neural networks can be very costly
- We can approximate this by sampling portions of our network at a time during training
- “Specifically, dropout trains the ensemble consisting of all subnetworks that can be formed by removing nonoutput units from an underlying base network”



Dropout Algorithm

- Goal: Train all possible subnetworks of your network
- During training, randomly set some portion of weight values to zero during each minibatch
- Only train the weights associated that were not set to zero

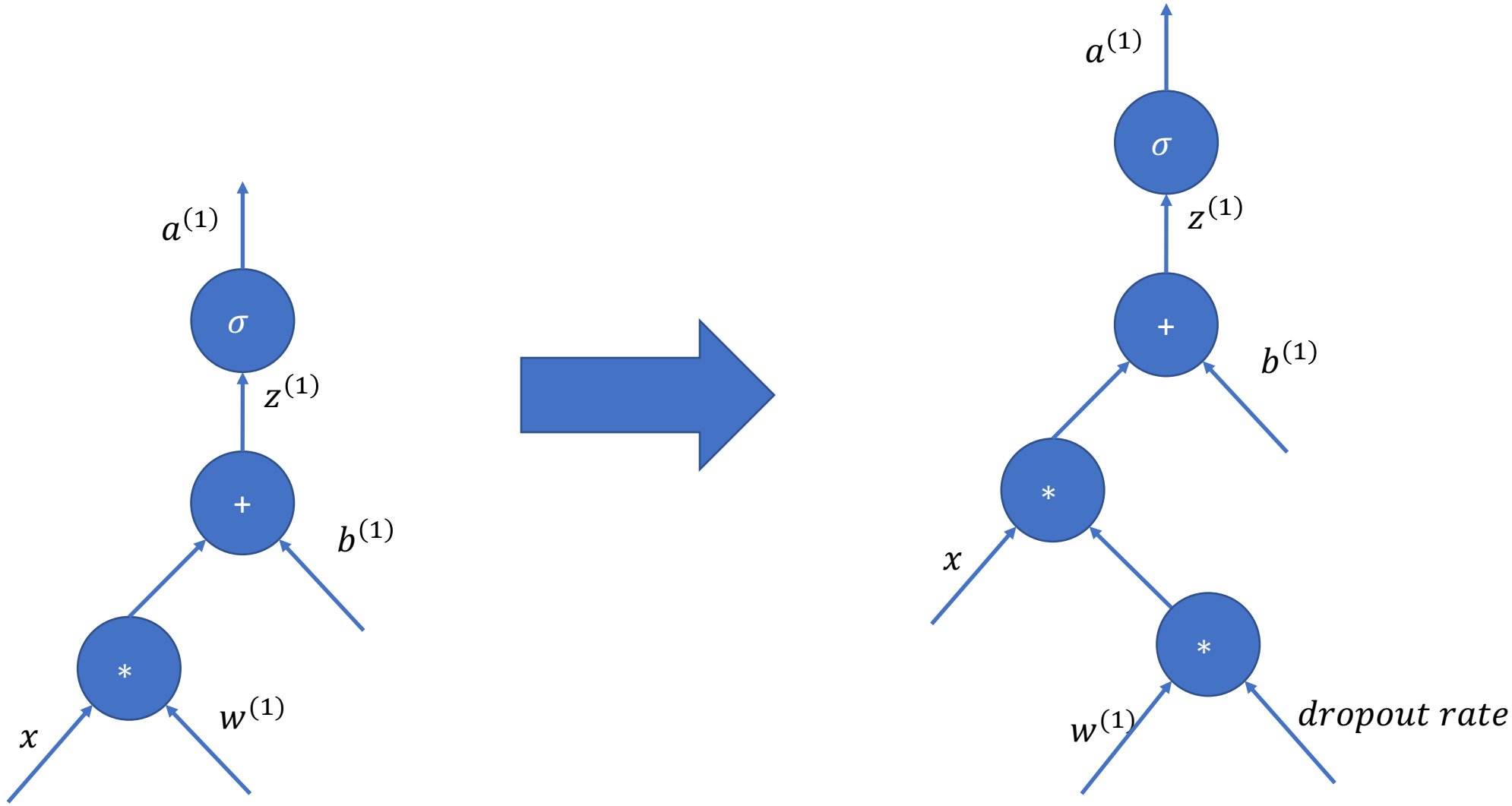
Dropout Training Computational Graph



Dropout Inference

- Lets say during training we sample 50% of the weights from hidden layers during each minibatch
- When we want to make predictions, we can approximate the ensemble of all networks by using the network as one complete network
- Each hidden network will then have twice the number of input weights to it than during training
- As a result, we scale the weights of during inference by the dropout rate, in this case 50%

Dropout Inference Computational Graph



Practical Dropout Considerations

- The weight scaling rule for inference is only an approximation of the true geometric mean of all of the subnetworks
  - But it seems to work well
- There are no guarantees for best dropout rates, methods, values, etc
- Typical setup is to remove 20% of the input layer and 50% of hidden layers at random.