Optimization

Resana Machine Learning Workshop

prepared by Hosein Hasani

Summer 1399

Outline

- Loss Functions
  - Mean Squared Error
  - Hinge Loss
  - Cross Entropy Loss
- Regularization
  - Add Regularization Terms to Loss Functions
  - Early Stopping
  - Drop-out
  - Data Augmentation
- Gradient Descend
- Error Backpropagation

Typical steps of solving (supervised) learning problems

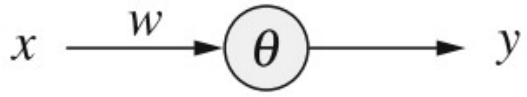
- Select the **hypothesis space**:
- Define a **loss function** that quantifies how much undesirable is each parameter vector across the training data.
- **Learning phase**: Apply an **optimization** algorithm that efficiently finds the parameters that minimize the loss function.
- **Evaluate** the obtained model.

Loss Functions

- Error: The difference between the actual outputs (e.g. **ground truth**) and the predicted outputs.
- The function that is used to compute this error is known as **Loss Function** $\boldsymbol{L}(.)$ or $\boldsymbol{J}(.)$.
- Generally, consists of **empirical risk term** and **regularization term**

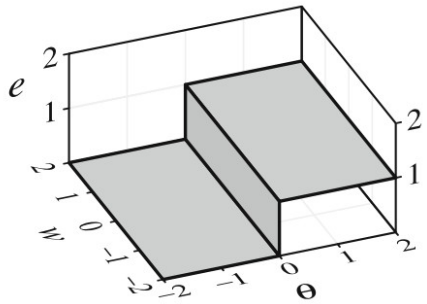
Loss Functions (Negation Problem)

- Consider a threshold logic unit with a single input and training examples for the negation:

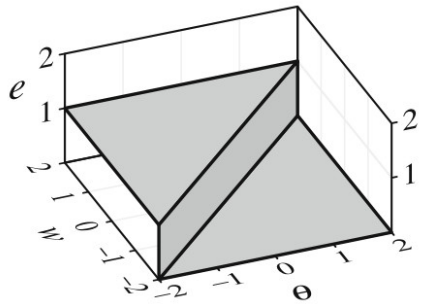


x	y
0	1
1	0

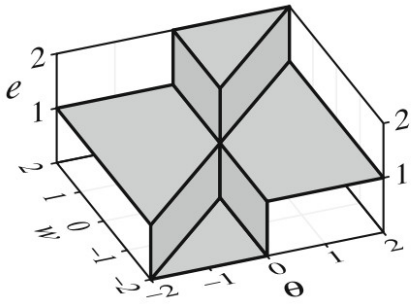
- Error of computing the negation w.r.t. the θ and w:



error for $x = 0$



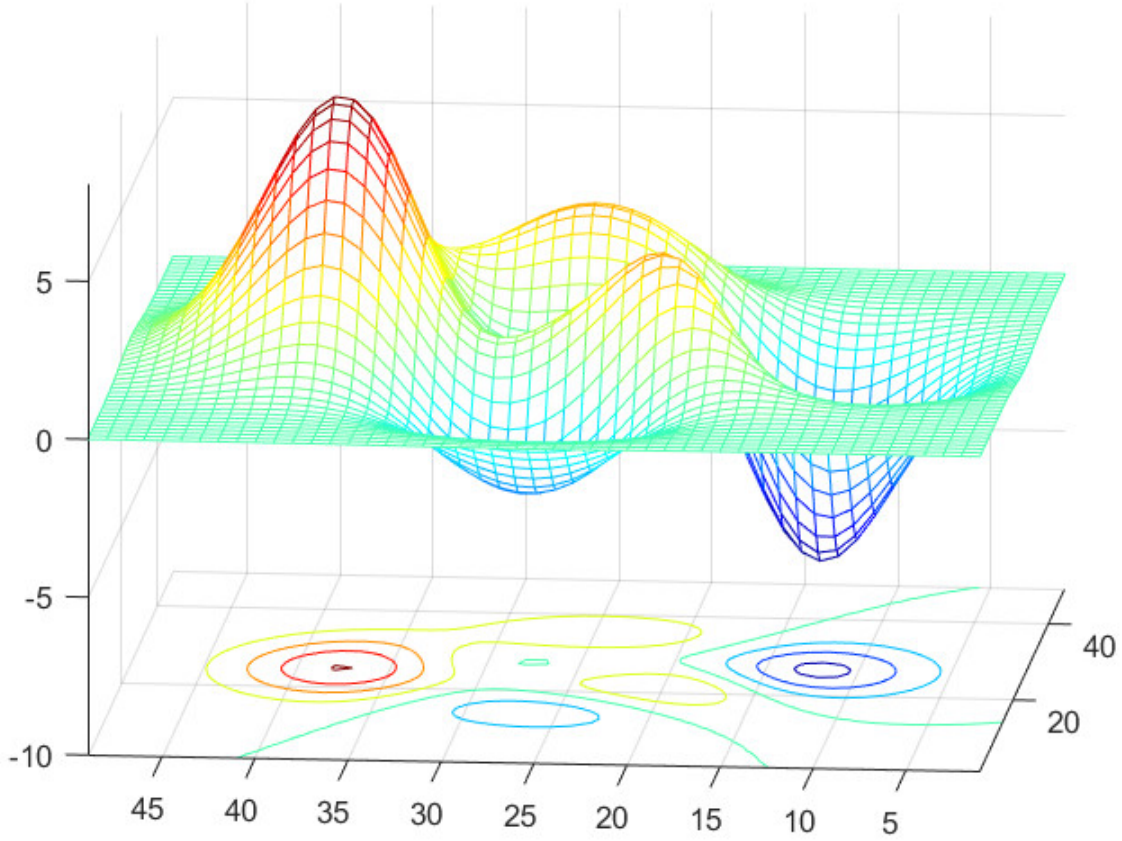
error for $x = 1$



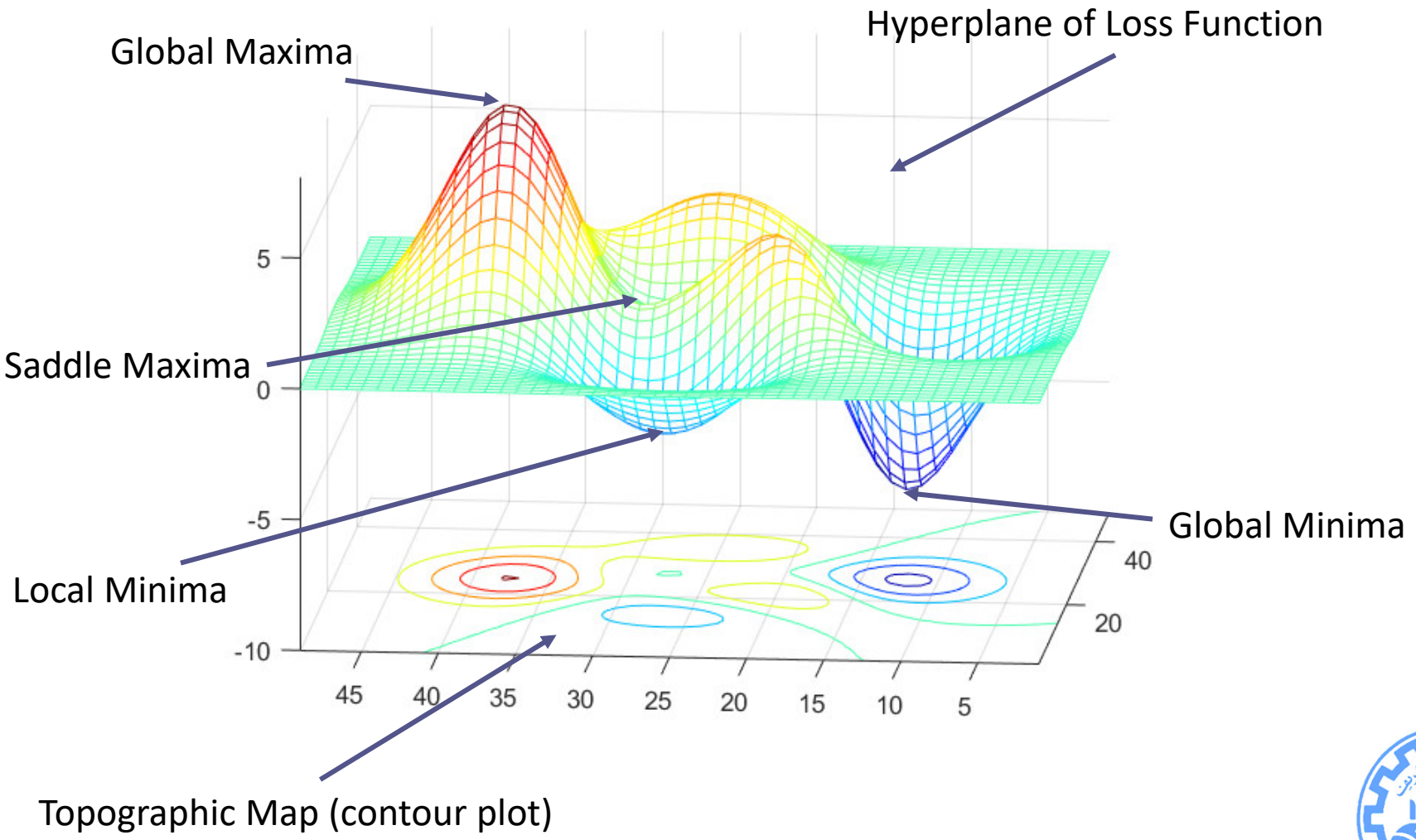
sum of errors

$$L = \sum_i L_i$$

Loss Functions: Loss Landscape



Loss Functions: Loss Landscape



Common Loss functions in machine learning

- Regression Losses
  - Mean Square Error (L2 Loss)
- Classification Losses
  - Hinge Loss (Multi class SVM)
  - Cross Entropy Loss

Mean Squared Error

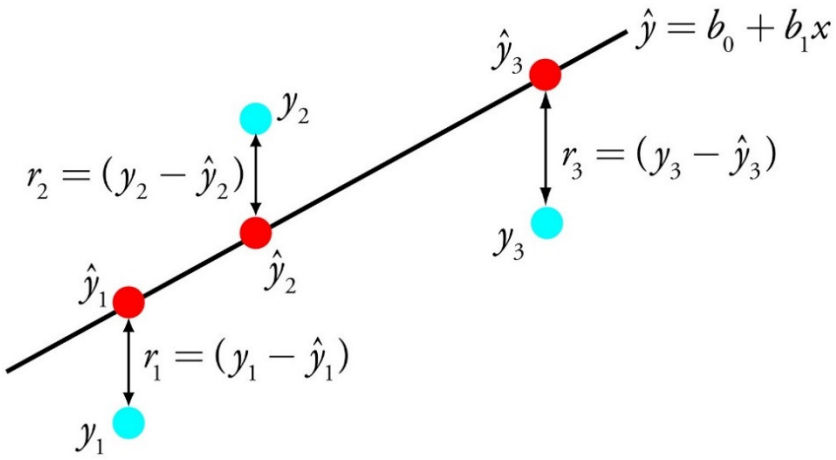
Mean Squared Error (MSE) loss function is widely used in regression problems.

$$J(w) = \sum_{i=1}^{N} (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y^{(i)}}})^2$$

Mean Squared Error

Mean Squared Error (MSE) loss function is widely used in regression problems.

$$J(w) = \sum_{i=1}^{N} (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y^{(i)}}})^2$$



Mean Squared Error

Mean Squared Error (MSE) loss function is widely used in regression problems.

$$J(w) = \sum_{i=1}^{N} (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y^{(i)}}})^2$$

Goal: Find w^* which minimizes J(w):

$$w^* = argmin_w \, J(w)$$

Linear Regression: Closed Form Solution

$$w^* = argmin_w J(w)$$

$$J(w) = \sum_{i=1}^{N} (y^{(i)} - w^T x^{(i)})^2 = (Y - Xw)^T (Y - Xw)$$

$$\nabla_w J(w) = X^T X w - X^T Y$$

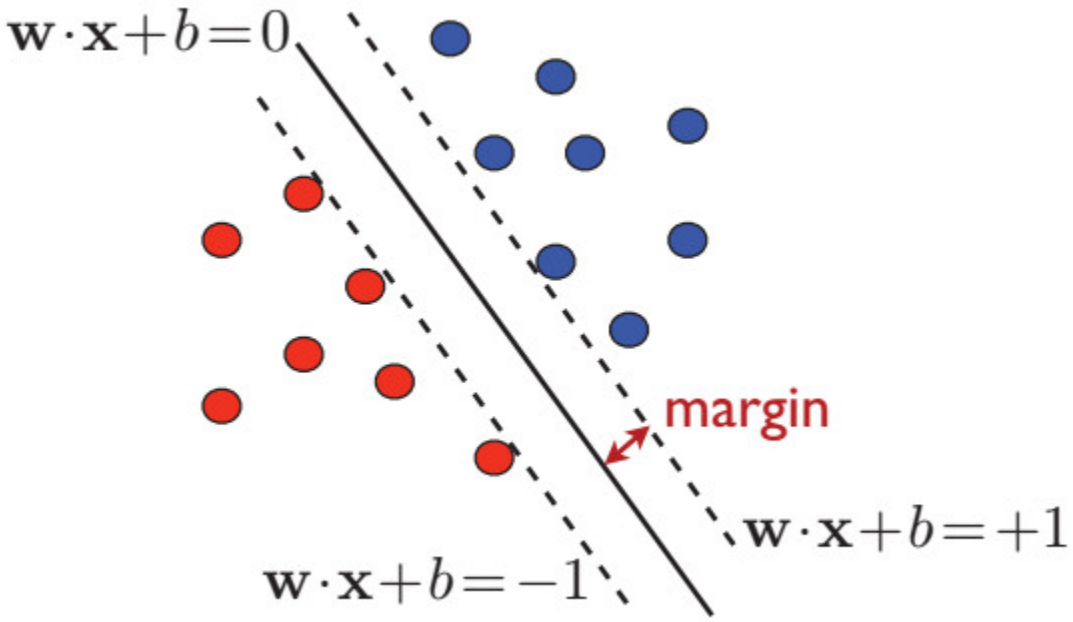
$$\nabla_w J(w^*) = 0 \rightarrow w^* = (X^T X)^{-1} X^T Y$$

Where:

$$X_{N\times d} = \begin{bmatrix} x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}, Y_{N\times 1} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}, w_{d\times 1} = \begin{bmatrix} w^{(1)} \\ \vdots \\ w^{(d)} \end{bmatrix}$$

Hinge Loss

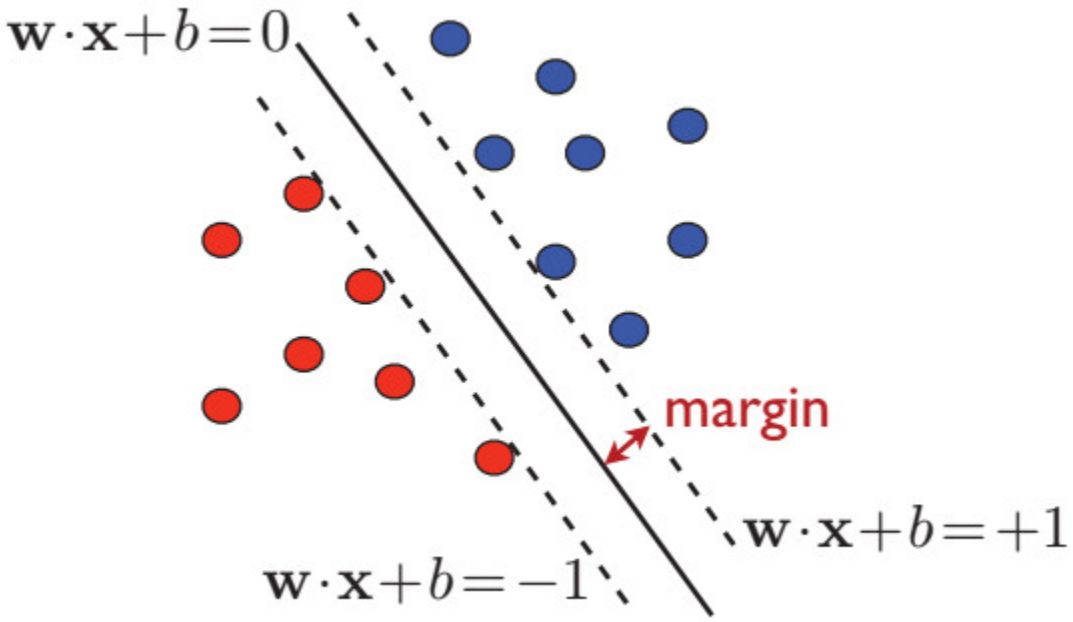
It's usually used in (multi-class) SVM problems.



[Mohri et al. 2012]

Hinge Loss

It's usually used in (multi-class) SVM problems.

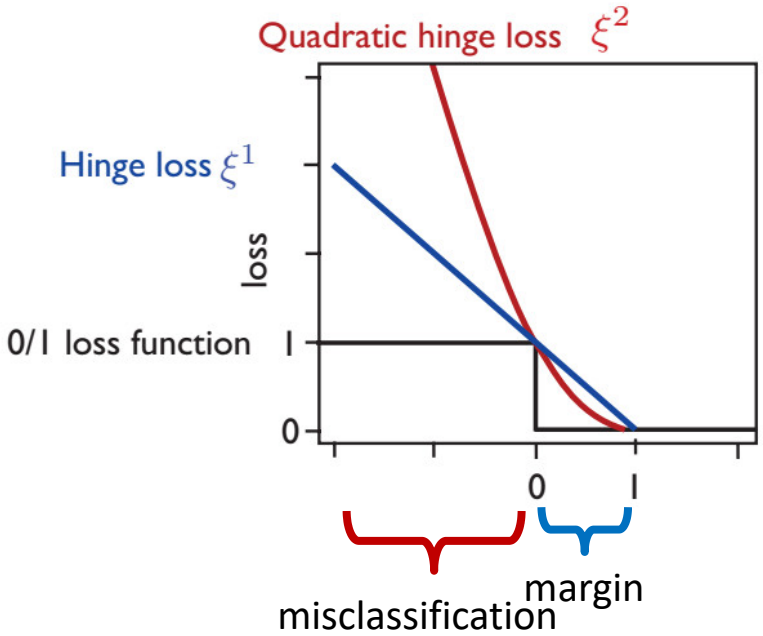


$$misclassification\ error: -sign(y^{(i)}[w.x^{(i)}+b])$$

[Mohri et al. 2012]

Hinge Loss

It's usually used in (multi-class) SVM problems.



horizontal axis~ $misclassification$: $y^{(i)}[w^T x^{(i)} + b]$

[Mohri et al. 2012]

Cross Entropy Loss

Works well for classification.

$$H(q,p) = -\sum_x q(x) \log p(x)$$

- P: The vector of estimated class probabilities

  ($p_i = \frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}}$ for **softmax** classifier)

- Q: The true distribution (one-hot vector: q = [0,0,0, …,1, …,0])

Cross Entropy Loss

Works well for classification.

$$H(q,p) = -\sum_x q(x) \log p(x)$$

Binary case:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

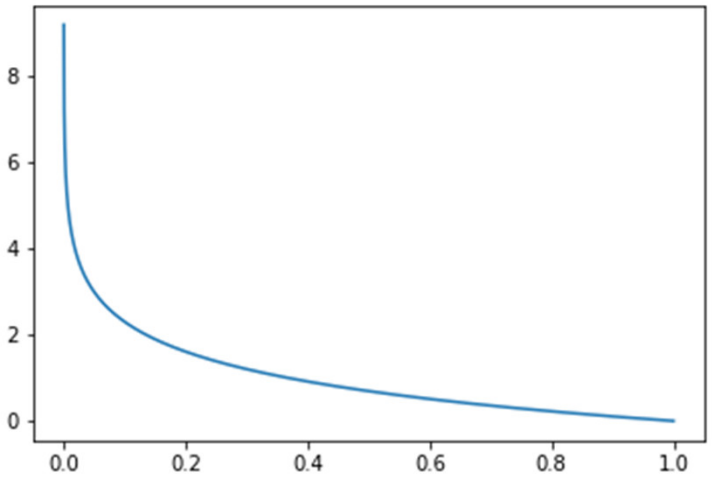
Here, $\hat{y}$ (and also y) is scalar.
$y \in \{0,1\}$ and $\hat{y}$ is the output of **sigmoid** function; $\hat{y} \in [0,1]$.

Cross Entropy Loss

Binary case:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

$(if\ y = 1)$: $J = -log\ \hat{y}$

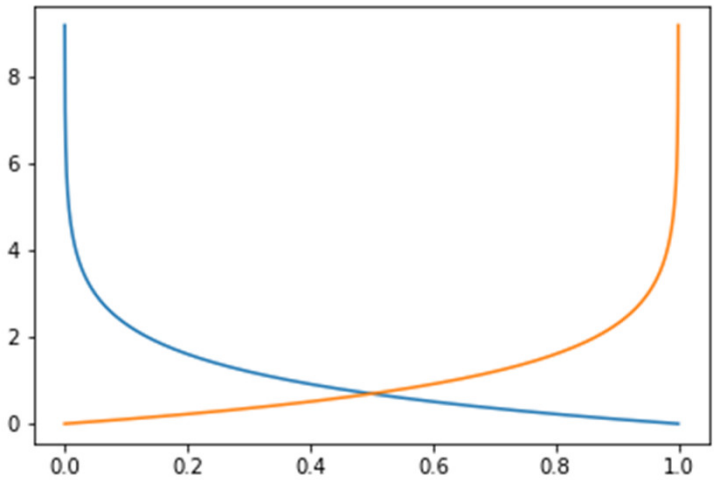


Cross Entropy Loss

Binary case:

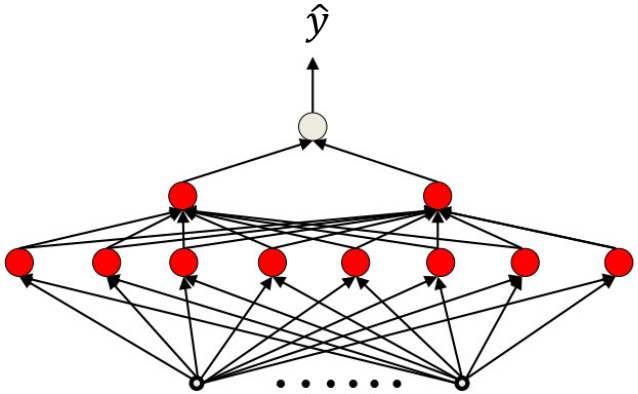
$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

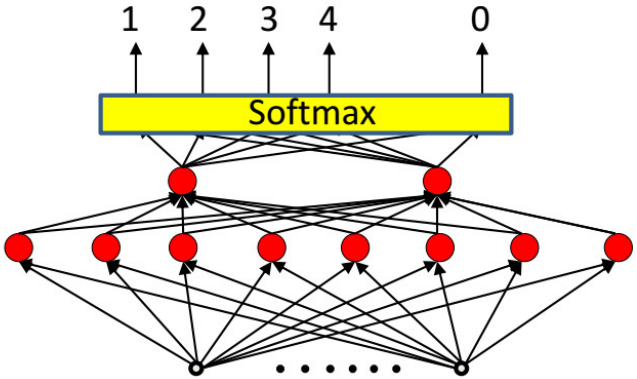
$(if\ y = 1)$: $J = -log\, \hat{y}$



$(if\ y = 0)$: $J = -log(1 - \hat{y})$

Cross Entropy vs. Mean Squared Error





	Desired output: y	Desired output: [0,0,…,1,…,0]
MSE	$L = (\hat{y} - y)^2$	$L = \sum_i (\hat{y}_i - y_i)^2$
CE	$L = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$	$L = \sum_i y_i \log(\hat{y}_i)$

Case Study: Linear Classification

Parametric approach: **Linear classifier**

$$f(x, W) = Wx$$

3072x1

(+b) 10x1

10x1

10x3072

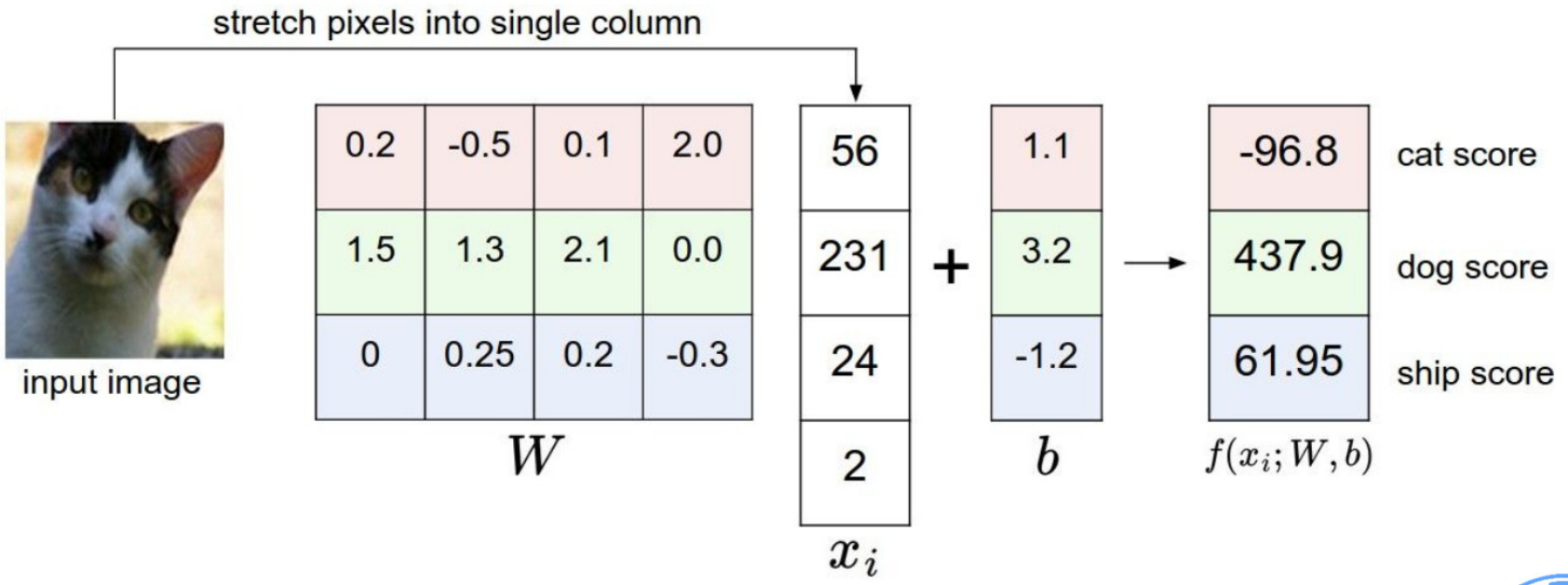
10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1

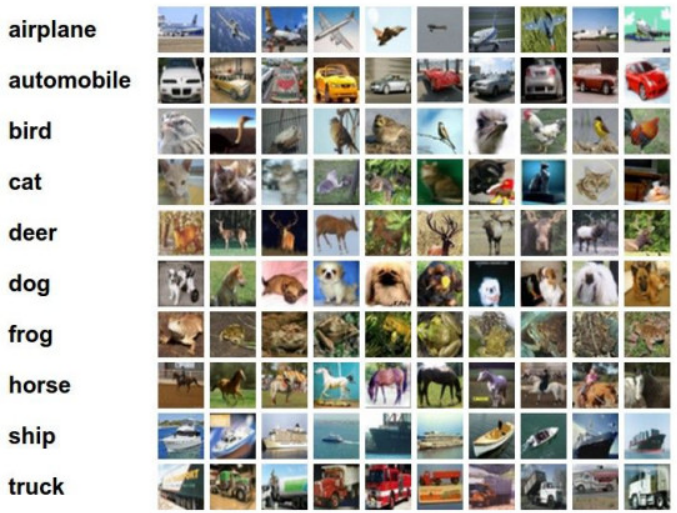
parameters, or “weights”

Case Study: Linear Classification

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



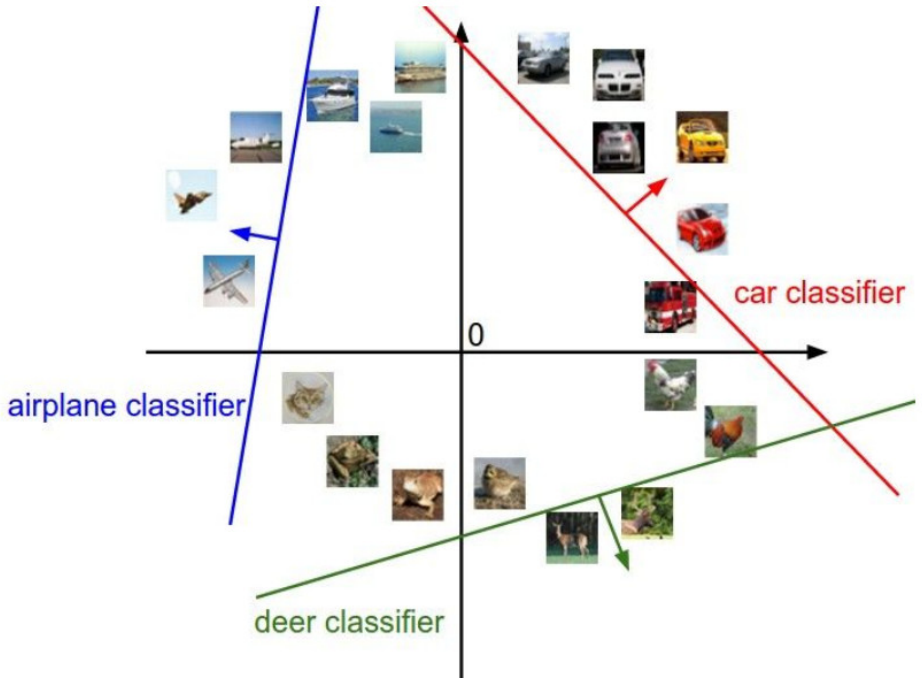
Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:

Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Linear Classifier: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Linear Classifier: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Linear Classifier: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

Linear Classifier: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Linear Classifier: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 10.9)/3
= **4.6**

Linear Classifier: Cross Entropy Loss

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Linear Classifier: Cross Entropy Loss

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

where $s = f(x_i; W)$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Linear Classifier: Cross Entropy Loss

Softmax Classifier (Multinomial Logistic Regression)

cat **3.2**

car 5.1

frog -1.7

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

---

in summary: $$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

Linear Classifier: Cross Entropy Loss

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

exp → normalize

unnormalized log probabilities

probabilities

Linear Classifier: Cross Entropy Loss

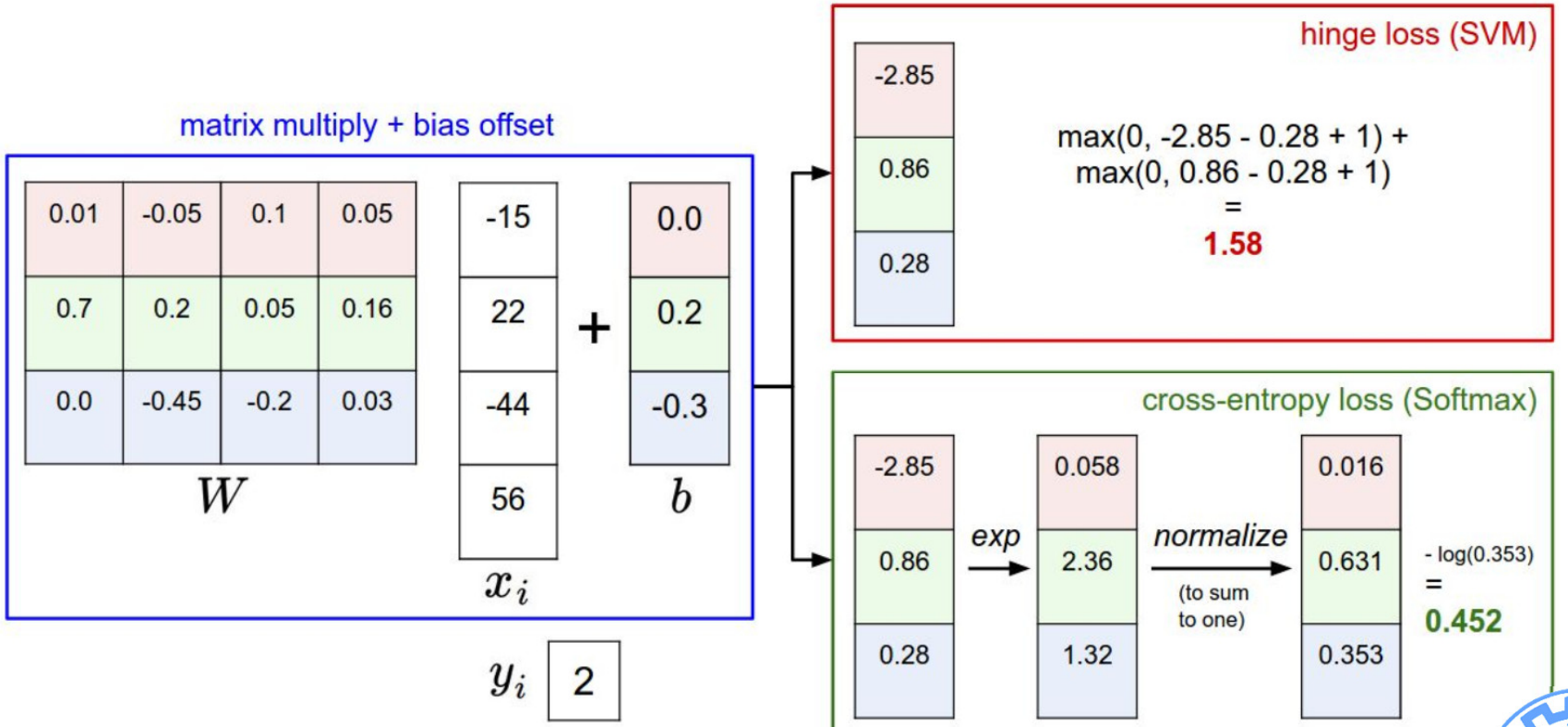
Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

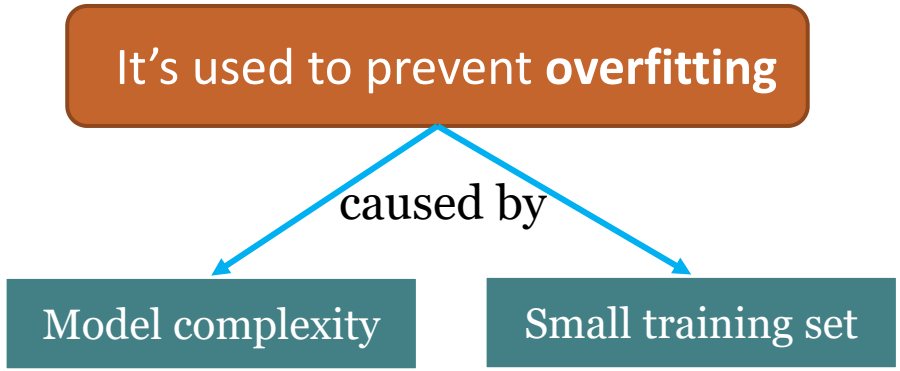
cat	**3.2**	**24.5**	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	164.0	0.87	
frog	-1.7	0.18	0.00	
		exp →	normalize →	
	unnormalized log probabilities		probabilities	

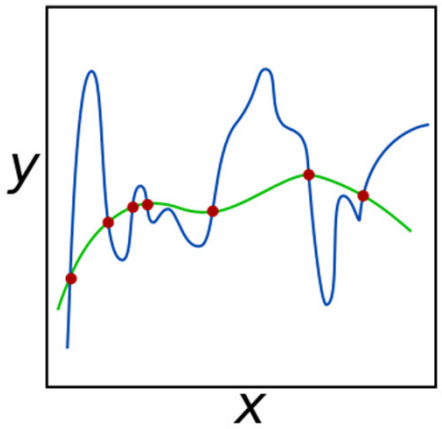
Linear Classifier:
Cross Entropy Loss vs. Hinge Loss



Regularization Terms

- Add regularization terms to loss functions
- Early Stopping
- Drop-out
- Dataset Augmentation





[wikipedia.rog]

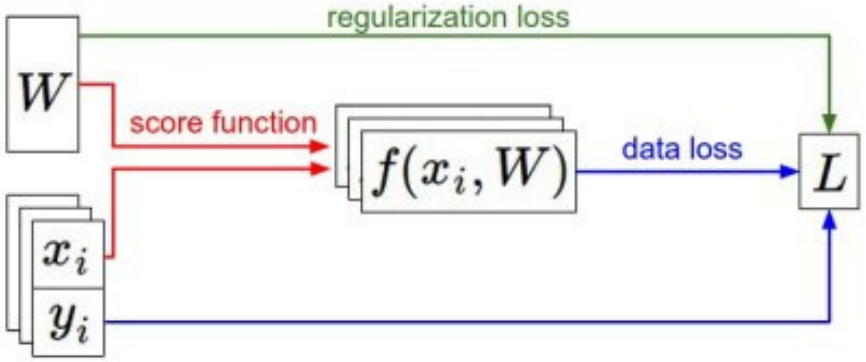
Regularization Terms

Prevent weight growing (**Weight Decay**).

- L2 (ridge) regularization $\mathrm{R(w)} = \sum_{k=1}^{K}\sum_{l=1}^{d} w_{lk}^2$
- L1 (lasso) regularization $\mathrm{R(w)} = \sum_{k=1}^{K}\sum_{l=1}^{d} |w_{lk}|$

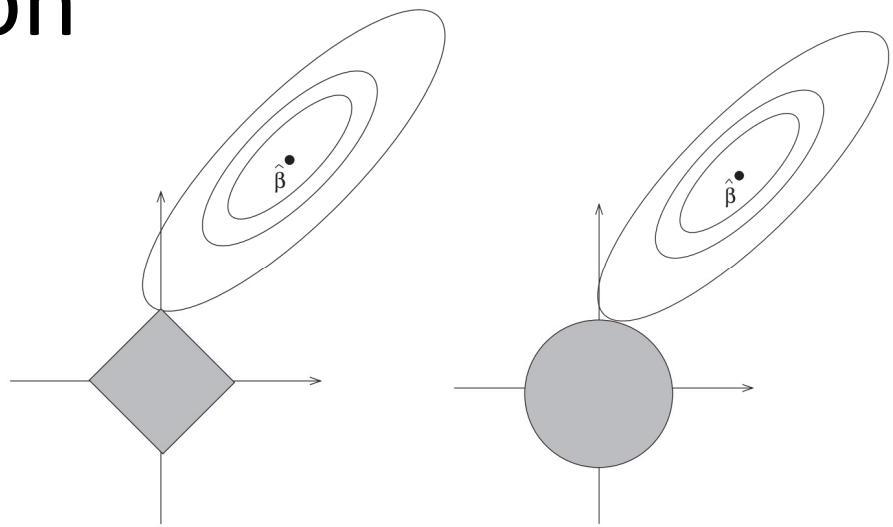
Full loss:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(w)$$



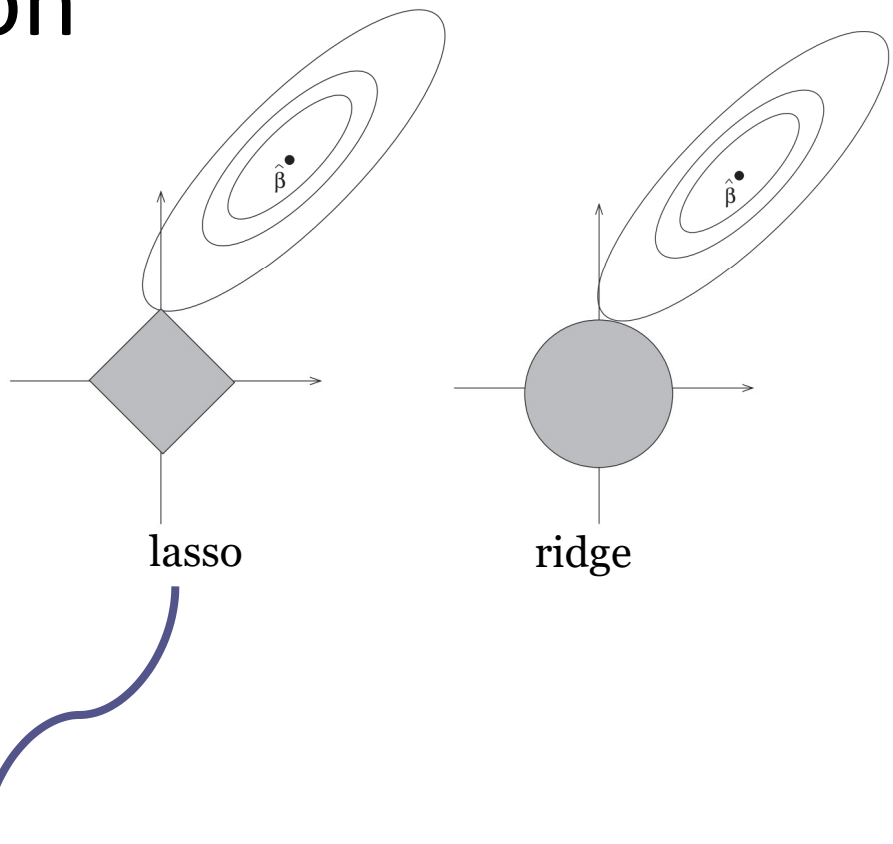
[http://cs231n.github.io/optimization-1/]

Comparison



[Tibshirani et al. 1996]

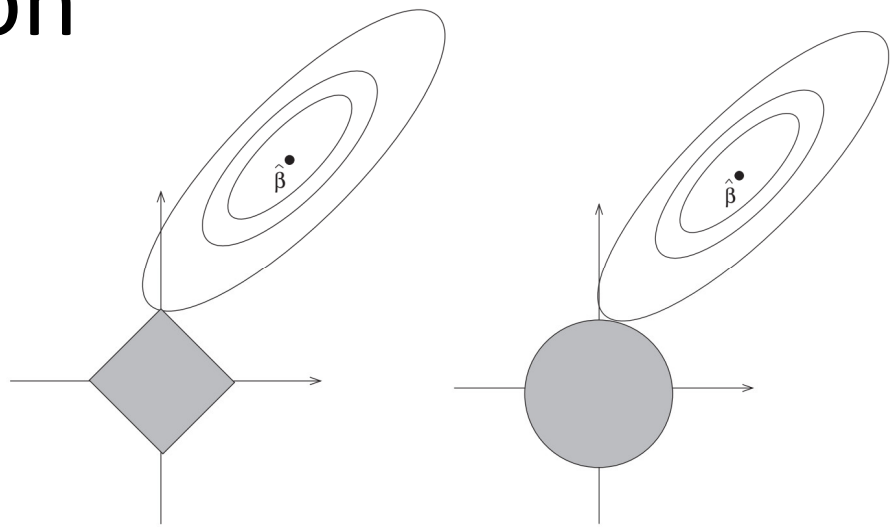
Comparison



regularization + feature selection

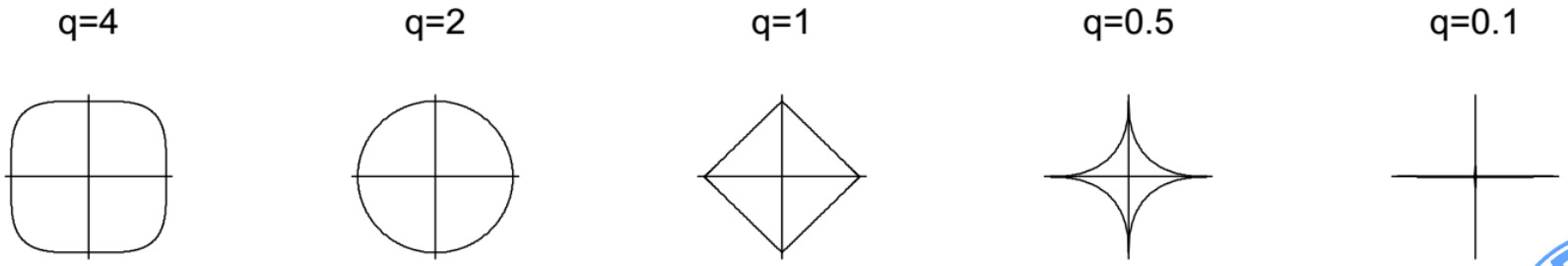
[Tibshirani et al. 1996]

Comparison



lasso ridge

Other norms:

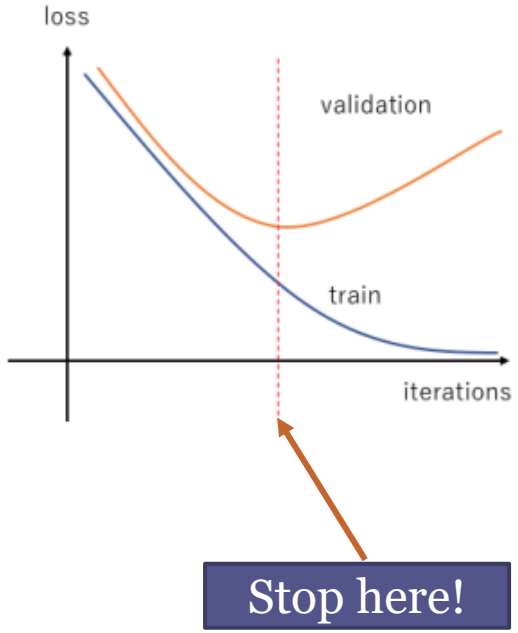


$$\sum_j |\beta_j|^q$$

[Tibshirani et al. 1996]

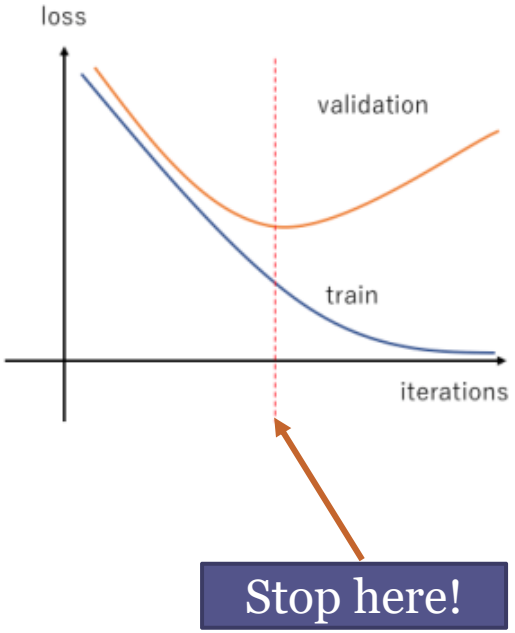
Early Stopping

- Stop training procedure when validation loss begins to increase.

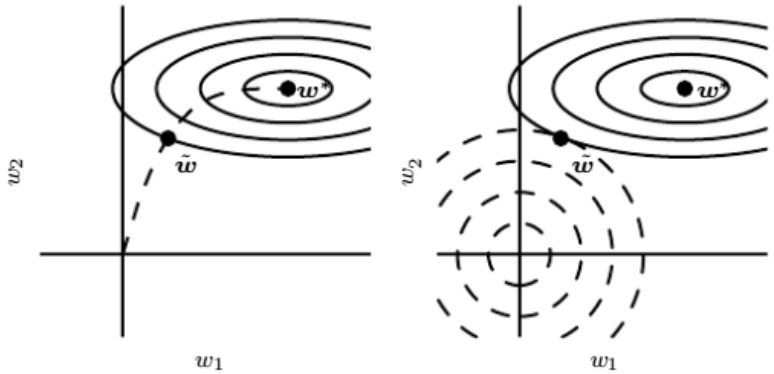


Early Stopping

- Stop training procedure when validation loss begins to increase.



- It's similar to L2 regularization

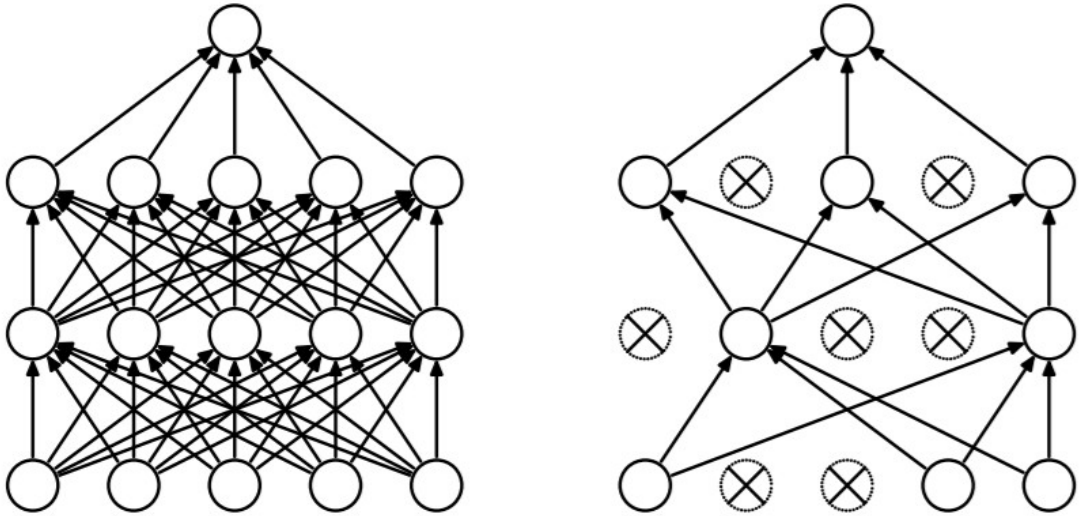


Early stopping L2 regularization

[Goodfellow et al. 2016]

Drop-out

- In each forward pass, randomly set some neurons to zero.

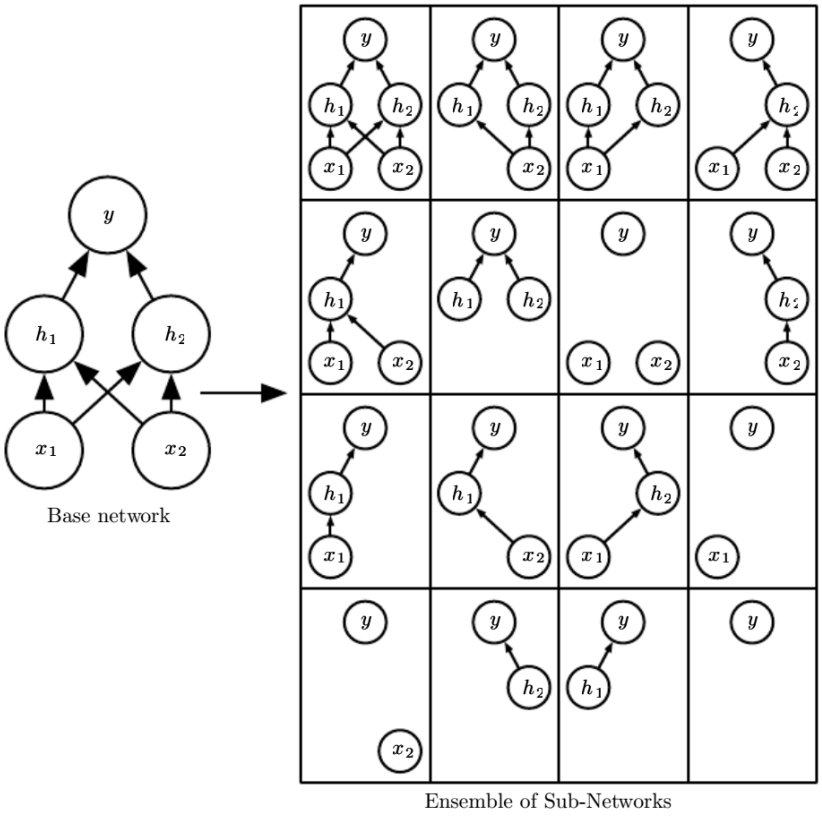


(a) Standard Neural Net (b) After applying dropout.

[Srivastava et al. 2014]

Drop-out

- In test time: average out all ensembles!



$$f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$

Adding randomness!

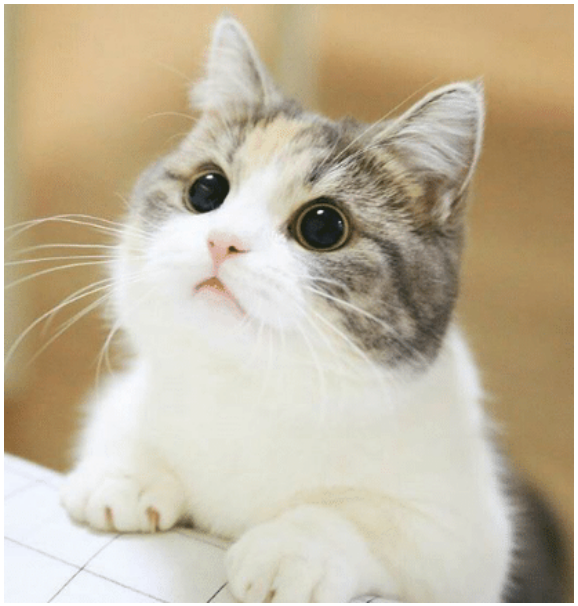
[Goodfellow et al. 2016]

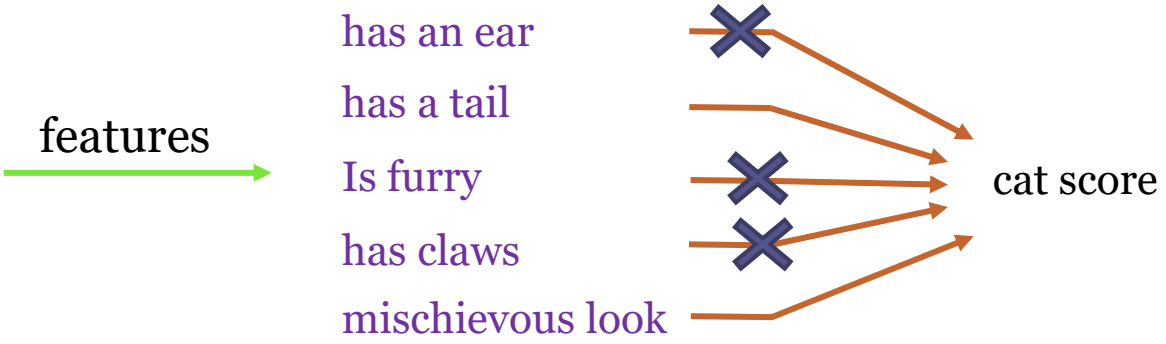
Effects of Drop-out

- Spread out weights: shrinking the weights (similar to L2 regression)
- Prevent co-adaptation of features

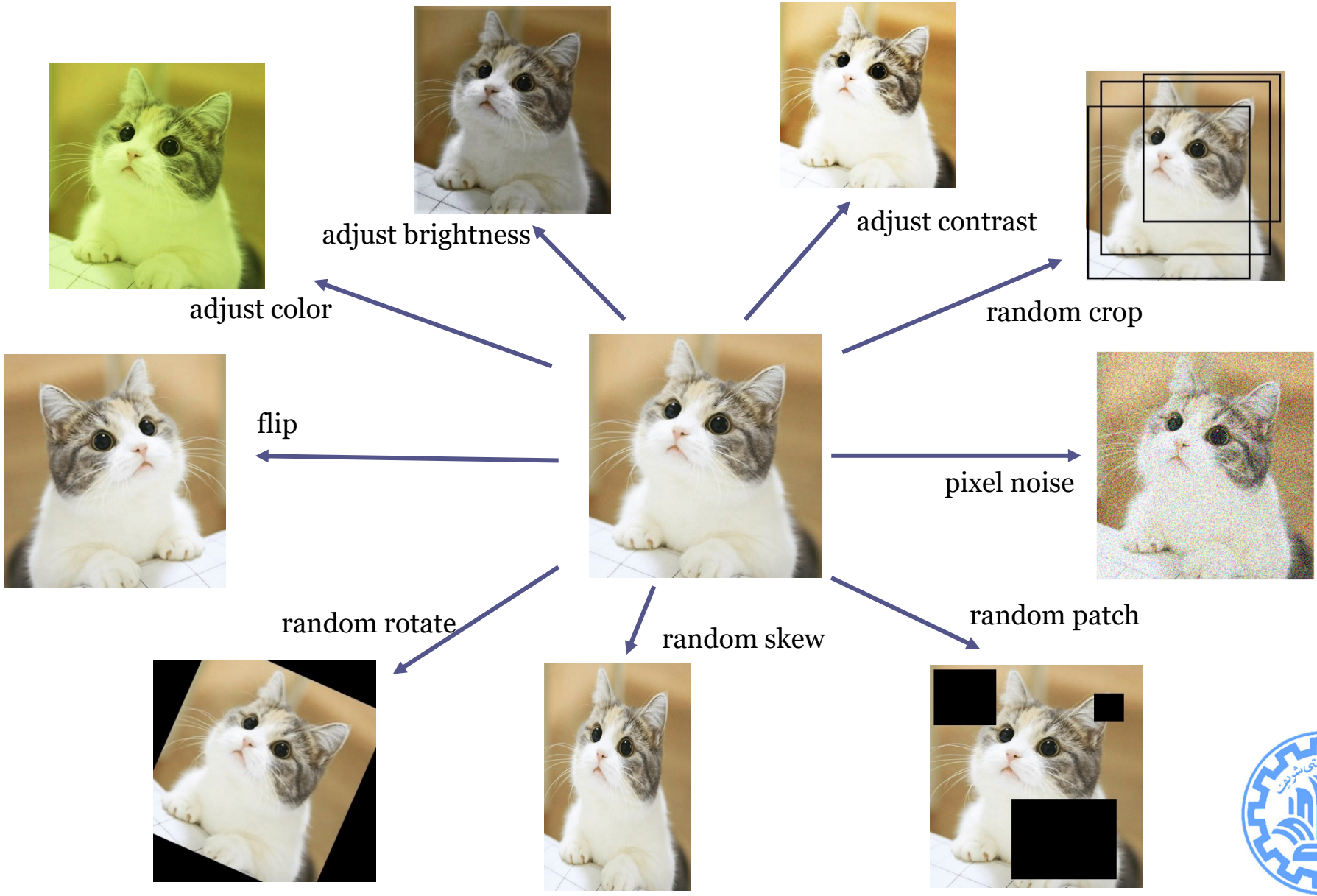
Effects of Drop-out

- Spread out weights: shrinking the weights (similar to L2 regression)
- Prevent co-adaptation of features



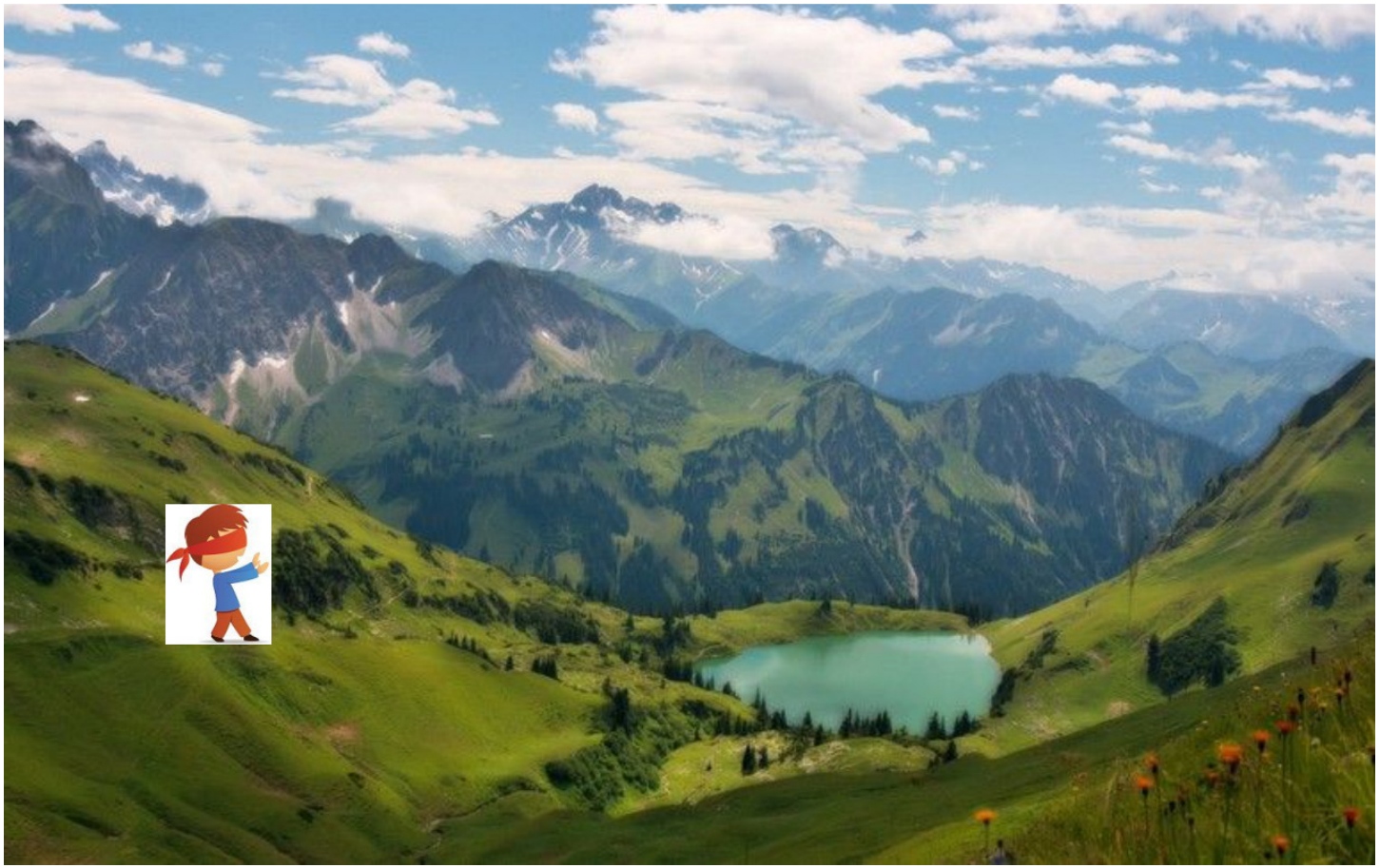


Dataset Augmentation



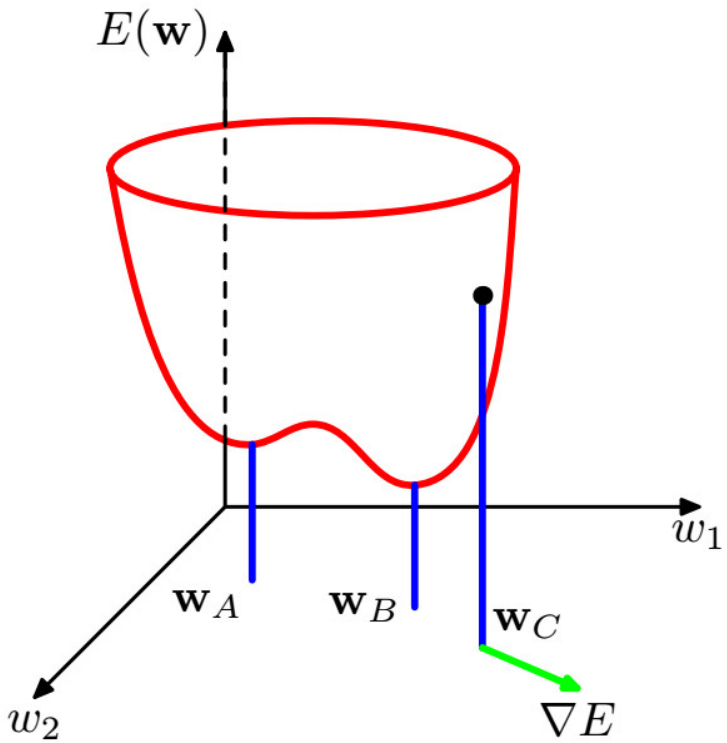
Optimization

- Gradient Based Optimizations



Optimization

- Gradient Based Optimizations



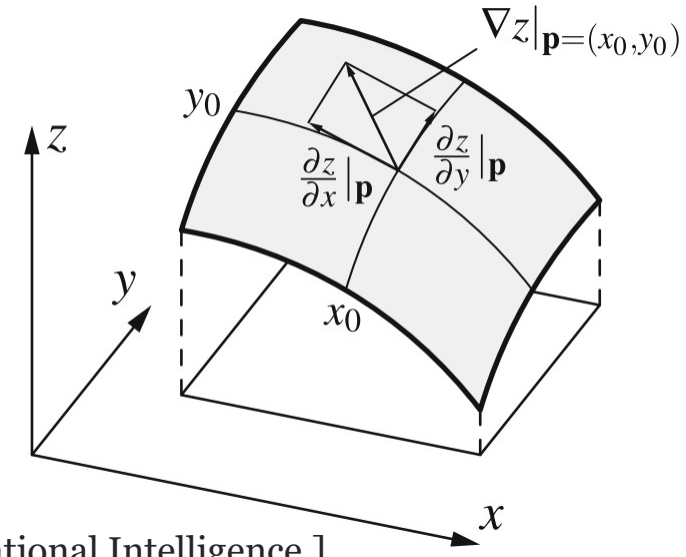
[Bishop. 2006]

Gradient Descend

- Minimize J(w)

$$w^{t+1} = w^t - \eta \nabla_w J(w^t)$$

$$\nabla_w J(w) = [\frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, \dots, \frac{\partial J(w)}{\partial w_d}]$$



[Kruse et al. Computational Intelligence.]

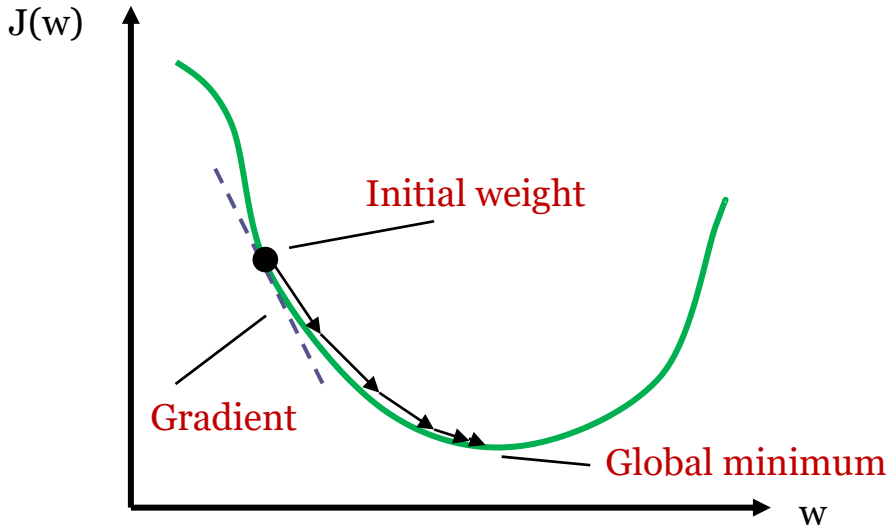
Gradient Descend

- Minimize J(w)

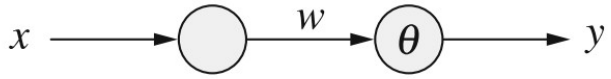
learning rate

$$w^{t+1} = w^t - \eta \nabla_w J(w^t)$$

$$\nabla_w J(w) = [\frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, \dots, \frac{\partial J(w)}{\partial w_d}]$$

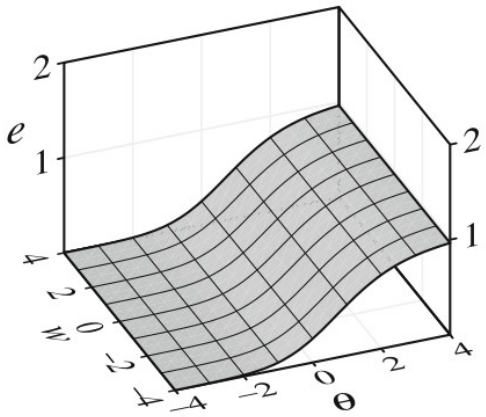


Gradient Descend

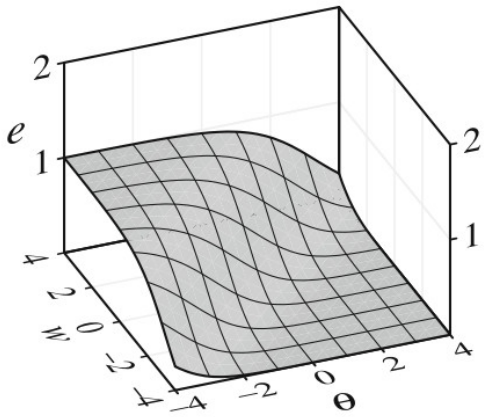


x	y
0	1
1	0

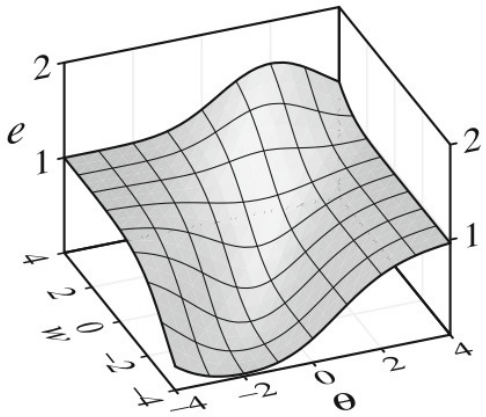
A two-layer perceptron with a single input and training examples for the negation



error for $x = 0$



error for $x = 1$

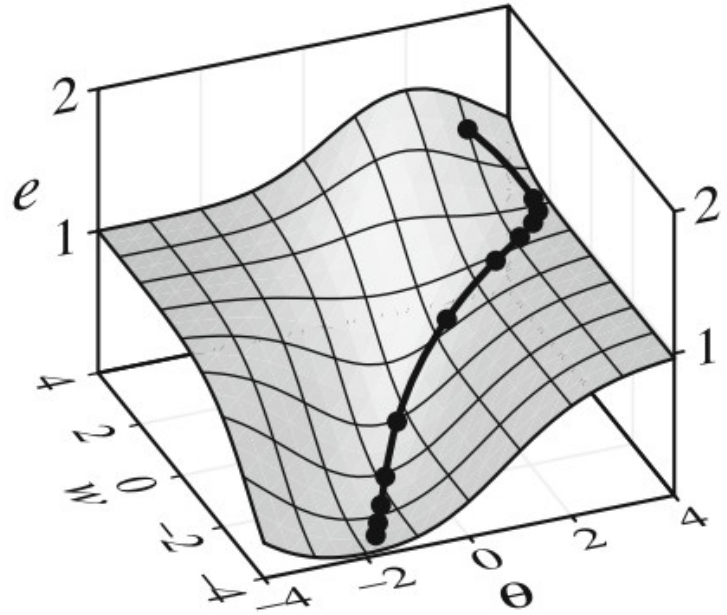


sum of errors

(Sum of) squared errors for computing the negation with a logistic activation function

[Kruse et al. Computational Intelligence.]

Gradient Descend



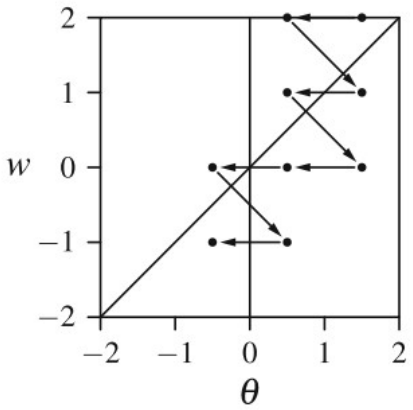
Training processes with initial values $\theta = 3$, $w = \frac{7}{2}$ and learning rate 1

[Kruse et al. Computational Intelligence.]

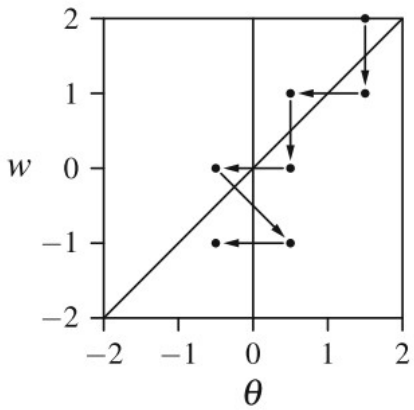
Learning Strategies

- **Online Learning**: Model will be updated by each training sample

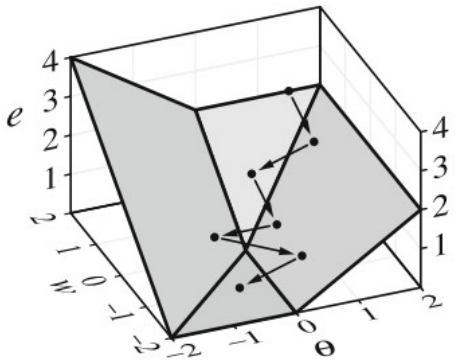
- **Batch Learning**: Don't perform the changes immediately after every training example, but aggregate them over all training examples and apply changes at the end (every epoch).
- For the negation problem with ReLU activation function, we have:



online training



batch training



batch training

Stochastic Gradient Descend

- Batch techniques process the entire training set in one go.
  - computationally costly for large data sets.

$$J(w) = \sum_{i=1}^{N} J^{(i)}(w)$$

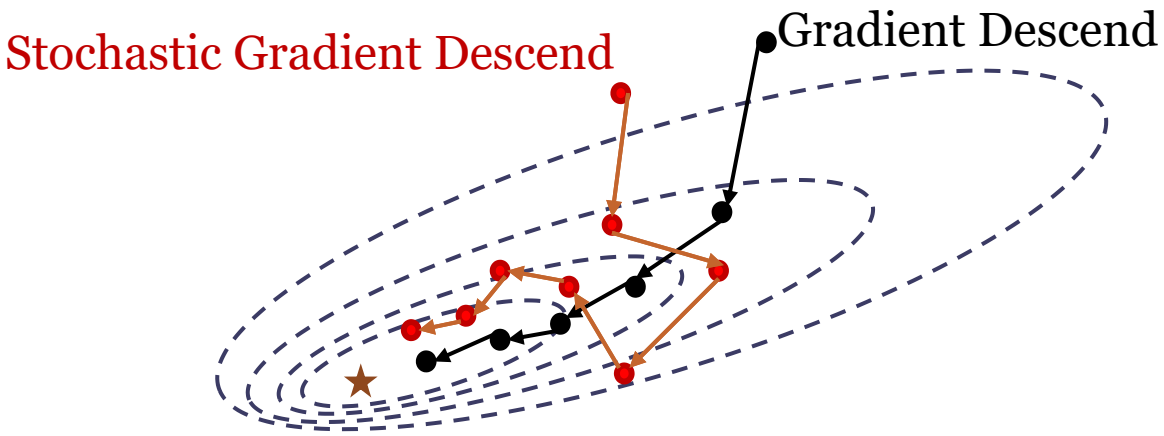
- SGD: Update after presentation of a mini-batch of data:

$$w^{t+1} = w^{t} - \eta \sum_{j \in S} \nabla_w J^{(j)}(w)$$

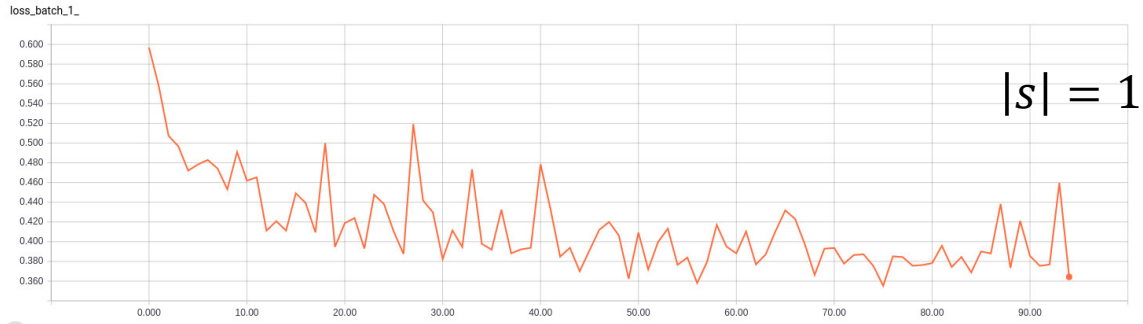
Stochastic Gradient Descend

- Update after presentation of a mini-batch of data:

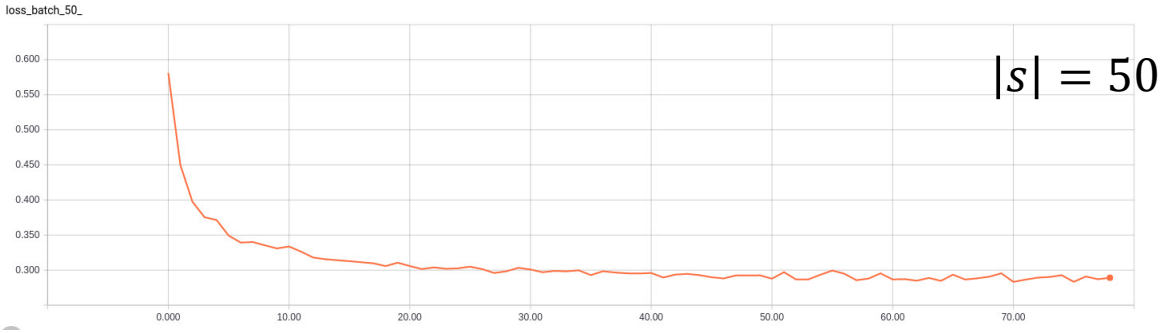
$$w^{t+1} = w^t - \eta \sum_{j \in S} \nabla_w J^{(j)}(w)$$



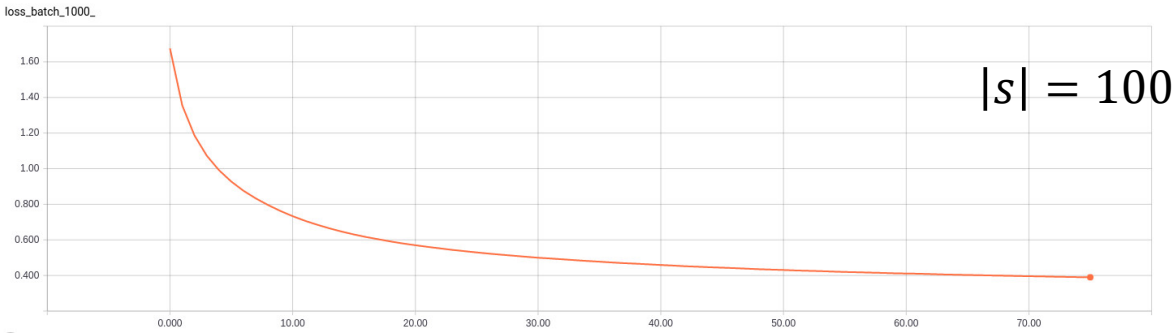
Stochastic Gradient Descend



$|s| = 1$



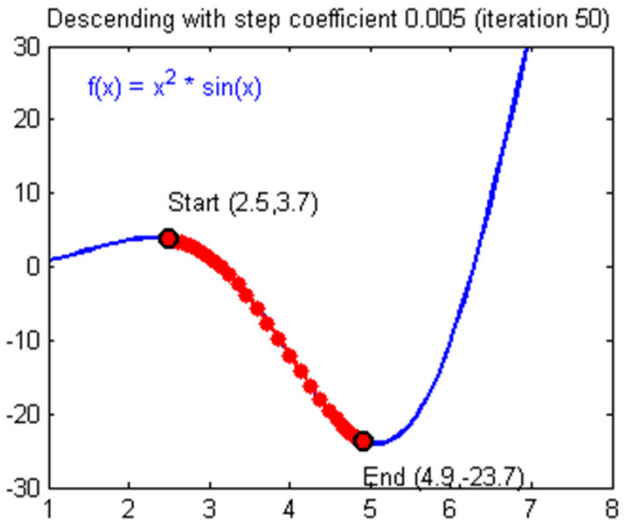
$|s| = 50$

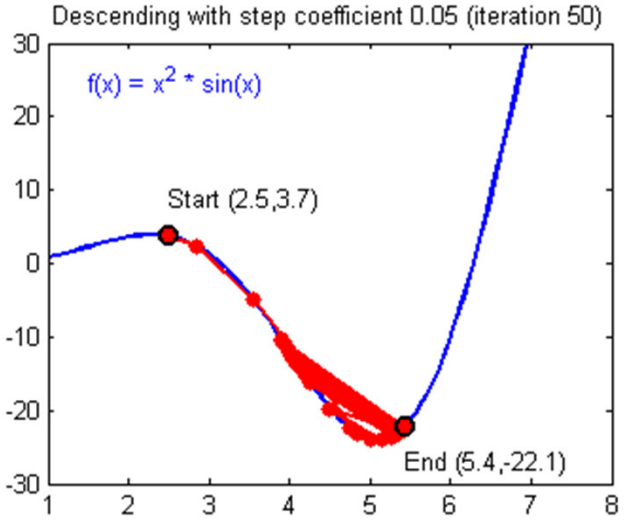


$|s| = 1000$

Learning Rate (Problem)

- Learning rate (η) is a hyperparameter and tuned by experiments.
  - If η is too small: gradient descend can be slow
  - If η is too large: gradient descend can overshoot the minimum. It may fail to converge, or even diverge!





[http://scs.ryerson.ca/~aharley/neural-networks/]

Learning Rate decay

- A good solution:
  Start with η_0, then slowly reduce learning rate over the time.

- exponential decay:

$$\eta = \eta_0 e^{-decay\_rate * epoch\_num}$$

- $1/t$ decay:

$$\eta = \frac{\eta_0}{1 + decay\_rate * epoch\_num}$$

Gradient Descend with Momentum

$v_{dW} = 0$

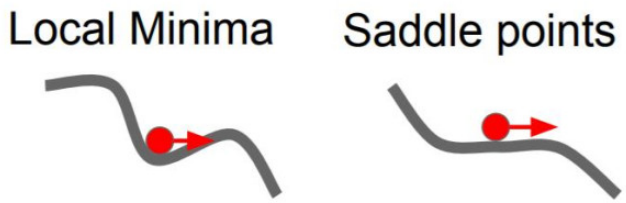
On each iteration:

Compute dW on the current mini-batch

$v_{dW} = \beta v_{dW} + dW$

$W = W - \eta v_{dW}$

- It can be shown that v_{dW} is moving average of recent gradients with exponentially decaying weights.
- Momentum helps to reduce problems with SGD

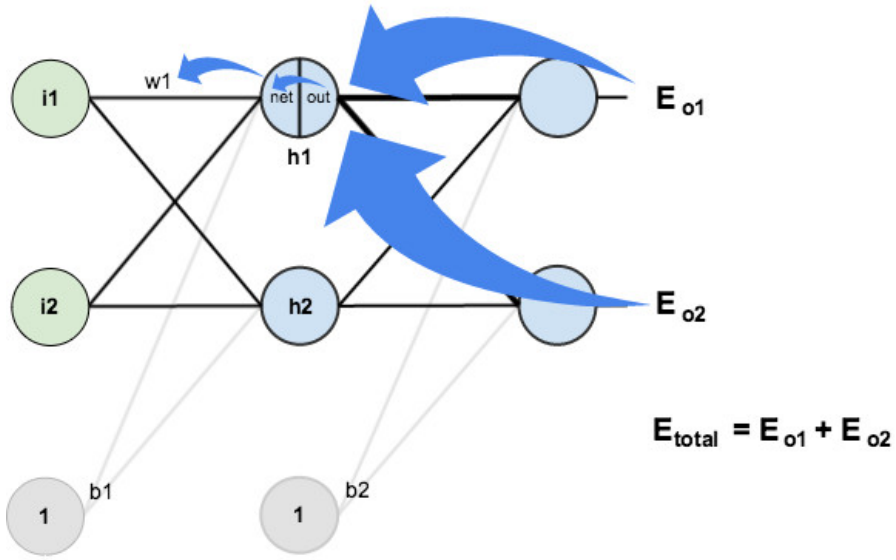


Error Backpropagation

- Employ gradients calculated by gradient descend to adjust weights in different layers by utilizing chain rule.

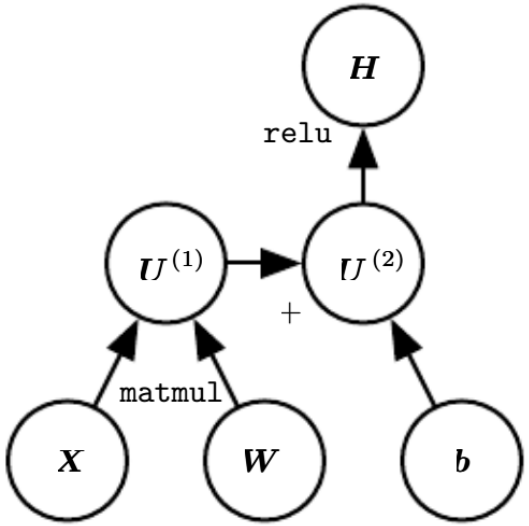
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

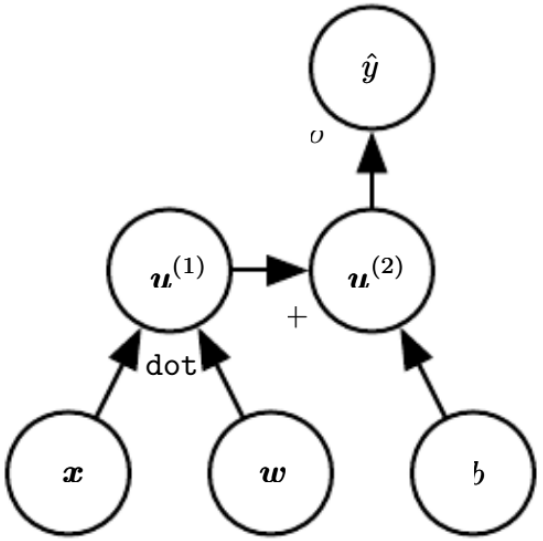


[https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/]

Computational graphs



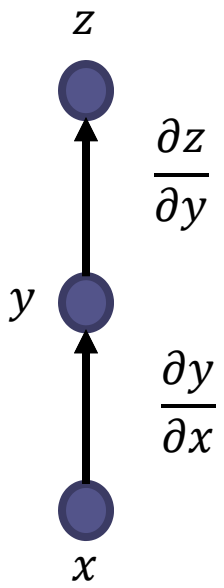
$$H = \max\{0, W^T X + b\}$$



$$\hat{y} = \sigma(W^T X + b)$$

[Goodfellow et al. 2016]

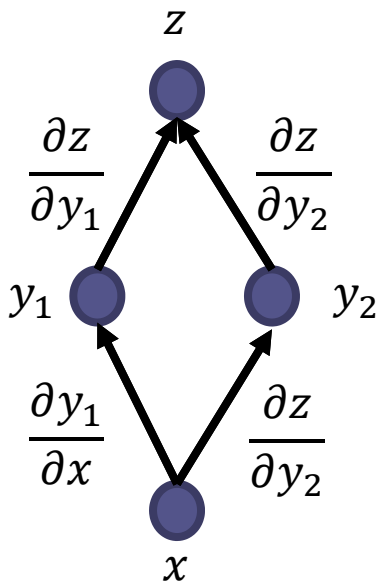
Simple Chain Rule



$$z = f\big(g(x)\big)$$
$$y = g(x)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Chain Rule: Multiple Paths



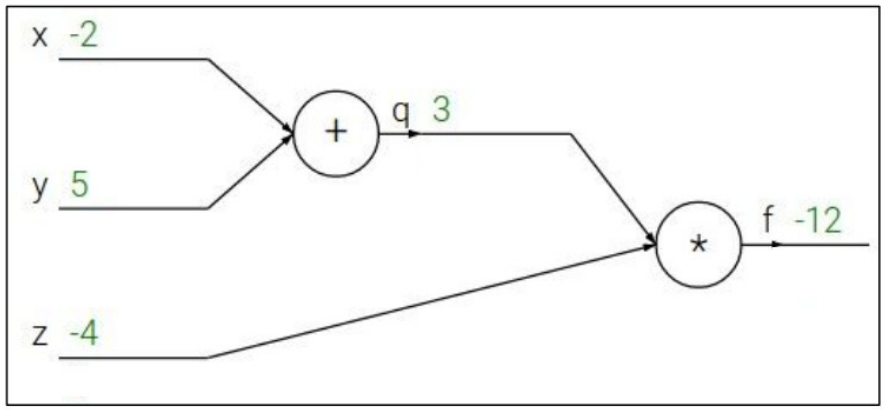
$$z = f_1(g_1(x)) + f_2(g_2(x))$$
$$y_1 = g_1(x)$$
$$y_2 = g_2(x)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{y_2}\frac{\partial y_2}{\partial x}$$

Backpropagation: Example

$$f(x, y, z) = (x + y)z$$

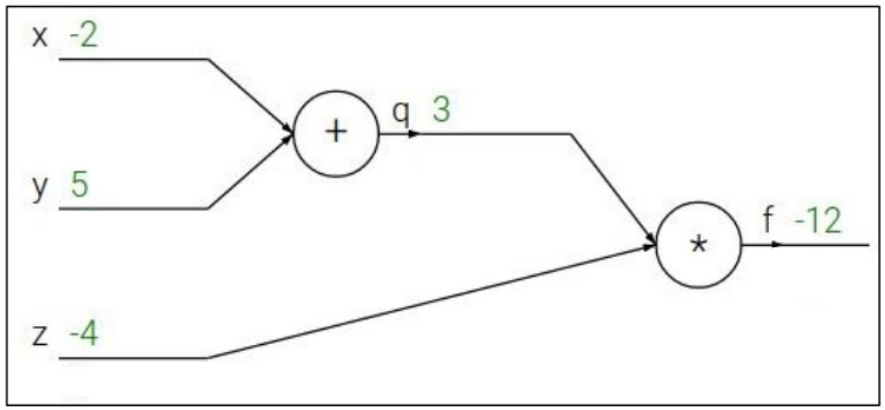
e.g. x = -2, y = 5, z = -4



Backpropagation: Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

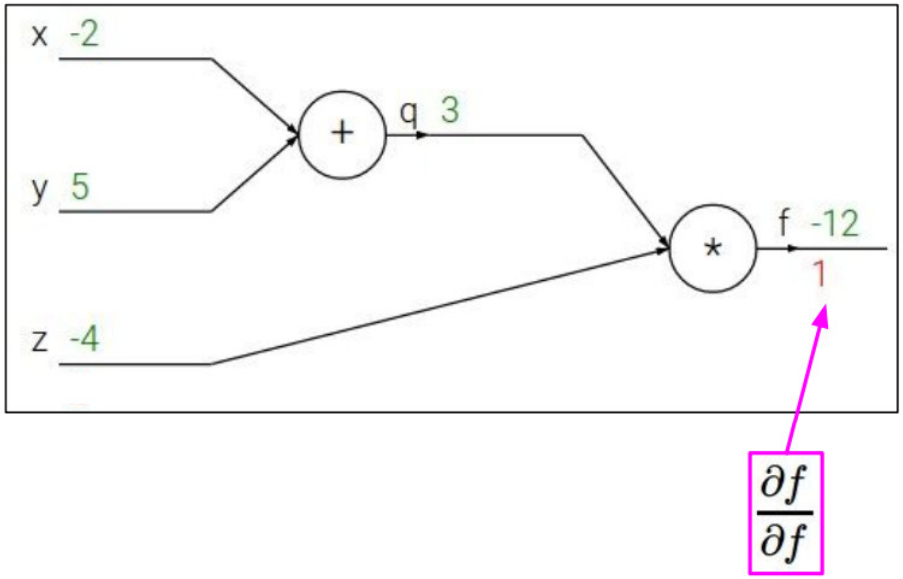
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: Example

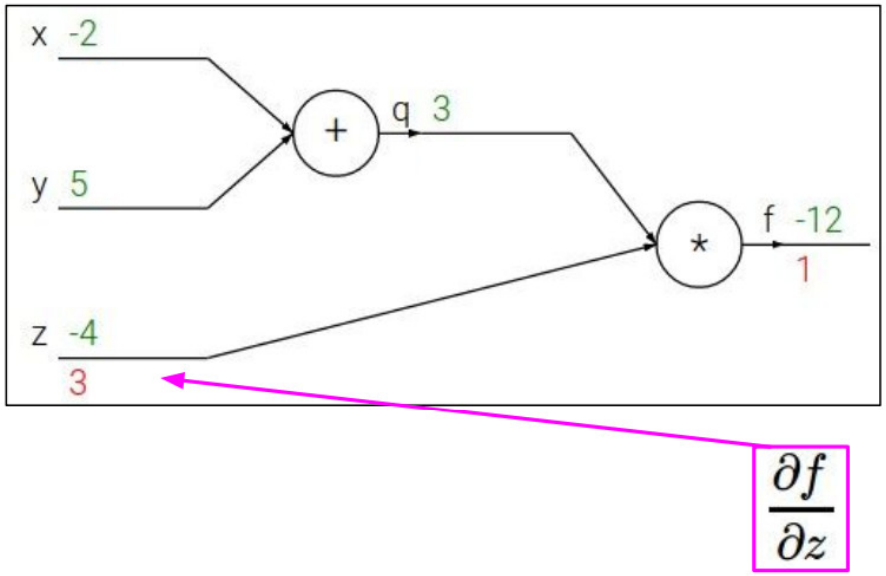
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Example

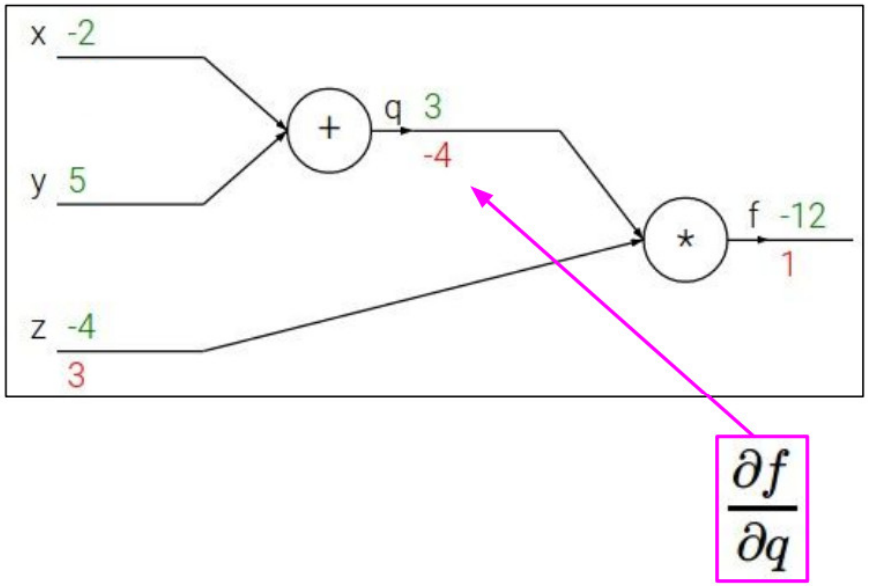
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: Example

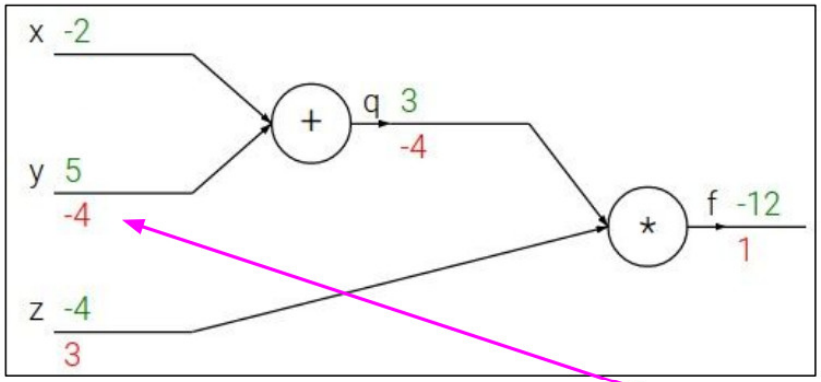
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\frac{\partial f}{\partial y}$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Backpropagation: Example

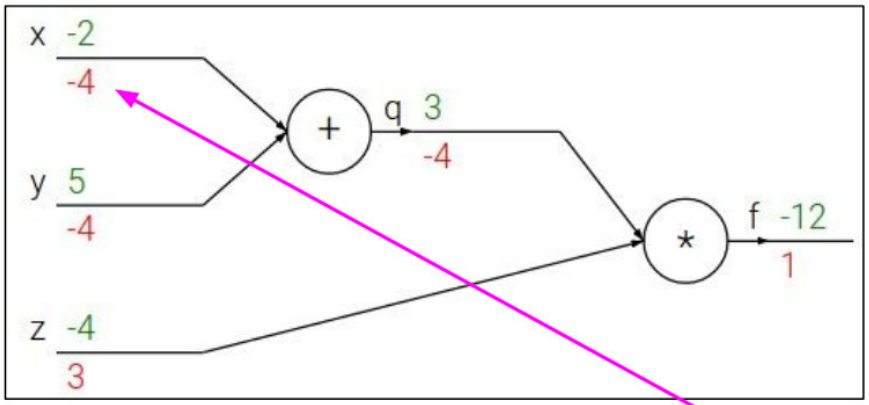
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Resources (recommended)

- CS231n slides (loss functions, optimization, backpropagation)
- Goodfellow, et al. Deep learning. (chapter 4, 6, 7)
- Bishop. Pattern Recognition And Machine Learning. (chapter 5)