lec3

Creation Date: 27/12/2019 10:09
Last Updated: 31/12/2019 11:32

Lec: Loss Function and Optmization

Loss Function:

- Loss function is to quantify how bad our training is at present.
- Optimization helps us to find the right parameters which minimizes the 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**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

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:

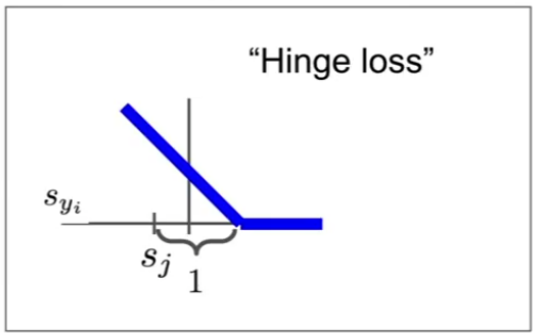
$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

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:



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

- The above loss is also called as "Hinge Loss"
- The term 1 value is just a tolerance value.

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	12.9

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, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

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	12.9

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:

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

Loss over full dataset is average:

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

L = (2.9 + 0 + 12.9)/3
= **5.27**

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	12.9

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:

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

Q: What happens to loss if car scores change a bit?

- Ans: Since we are changing it only a little bit(within the tolerance of 1) the loss value wont change.

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	12.9

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:

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

Q2: what is the min/max possible loss?

- Ans: Minimum value is 0, since it has max (0, value), Maximum can be infinity since it has a linearly increasing value.

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	12.9

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:

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

Q3: At initialization W is small so all s ≈ 0. What is the loss?

- Ans: C-1. Since if s'es are very small. The term s_j + 1 - s_y_i will result in C-1 for each training example, and when we add all for all the training samples it will be N(C-1), then when we take the average it becomes C-1.

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	12.9

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: $s = f(x_i, W)$

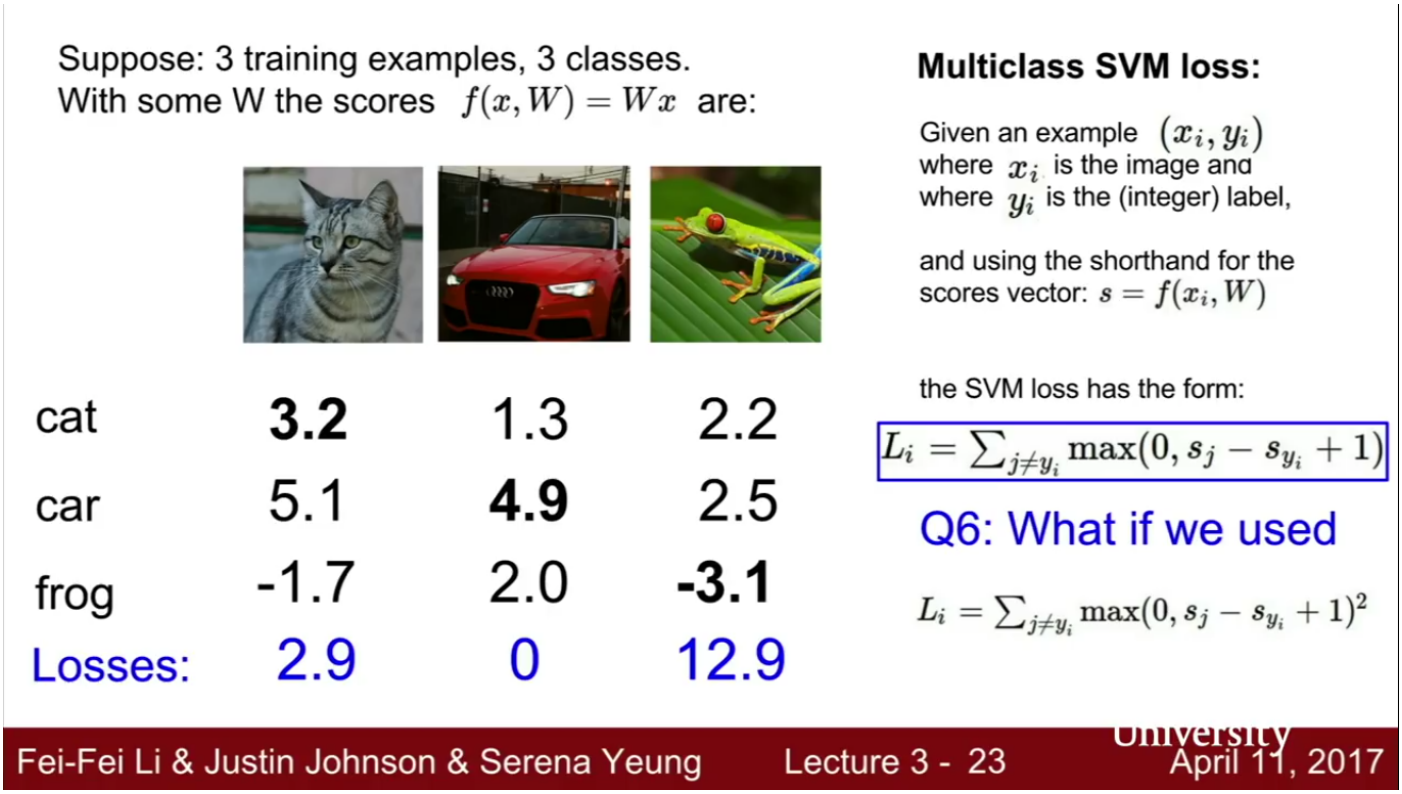
the SVM loss has the form:

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

Q4: What if the sum was over all classes? (including j = y_i)

- Ans: For each training sample the loss value increases by 1 and then overall the loss value will be N*1, then on average it adds just 1 to the previous loss where we not include the correct class.
- We prefer to use the method where we dont consider the correct class since it has the minimum loss of 0, which helps in visualizing the loss during training where the the optimizer reaches the 0 loss value during training.

- THE model WILL NOT CHANGE AND WOULD PRODUCE THE SAME MODEL AS THE ONE WHERE THE THE CORRECT LABEL IS NOT CONSIDERED.



- This is called as the 'Squared Hinge Loss', It is a different loss formulation since the tradeoff between good and bad example is non-linear in nature, but this loss function can be used too.
- **Why will one prefer squared hinge loss over hinge loss?**
  - The goal of a loss function is to quantify how wrong the predictions are, and how we penalize such mistakes. The penalty can be linear in nature, quadratic in nature or exponential in nature depending on how we want to quantify a bad prediction.
  - Eg. If we need the bad prediction to be penalized higher then squared or exponetial terms can be used.

Multiclass SVM Loss: Example code

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

```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```


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

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

E.g. Suppose that we found a W such that L = 0. Is this W unique?

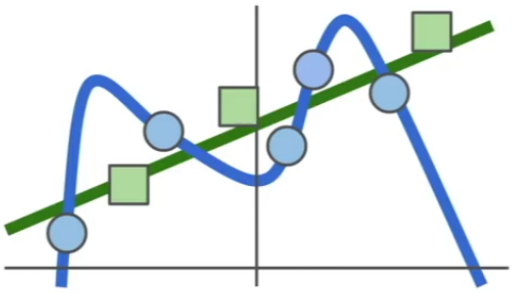
No! 2W is also has L = 0!

- Because our predictions will also have double their values and the relative difference will also get doubled and the loss value for each training example wont change.

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

- We want our model to generalize to test data and just fit the training data too well. The regularization loss term helps in simplifying the model to avoid overfitting through complex models via higher order parameters.
- We can either constraint our model to not include higher order terms or we can introduce a soft penalty via regularization to use higher order terms.

Regularization

λ = regularization strength (hyperparameter)

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization	$R(W) = \sum_k \sum_l W_{k,l}^2$
L1 regularization	$R(W) = \sum_k \sum_l \lvert W_{k,l}\rvert$
Elastic net (L1 + L2)	$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + \lvert W_{k,l}\rvert$

Max norm regularization (might see later)

Dropout (will see later)

Fancier: Batch normalization, stochastic depth

- L2 regularization is commonly used.
- L1 regularization is used to introduce sparsity.
- Most of the mentioned regularization are commonly used in ML and not just DL.

- In DL, Dropout and Batch Normalization is commonly used.

L2 Regularization (Weight Decay)

$x = [1, 1, 1, 1]$

$R(W) = \sum_k \sum_l W_{k,l}^2$

$w_1 = [1, 0, 0, 0]$

$w_2 = [0.25, 0.25, 0.25, 0.25]$

(If you are a Bayesian: L2 regularization also corresponds MAP inference using a Gaussian prior on W)

$$w_1^T x = w_2^T x = 1$$

- **The choice of the Regularization is problem dependent and data dependent**.
- L2 reg. chooses W which is spread out.
- L1 reg. chooses W which has more number of 0 (sparse is better)

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 $s = f(x_i; W)$

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

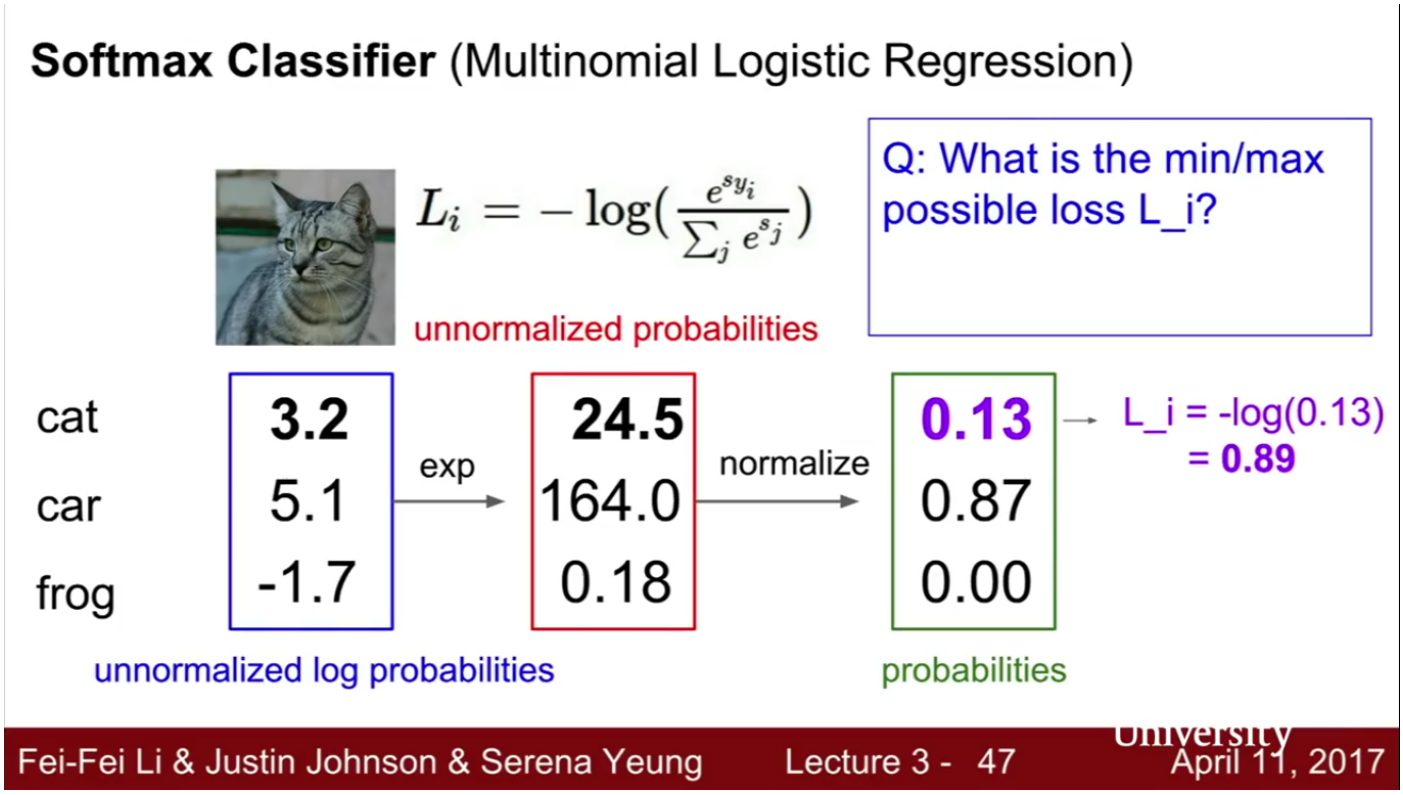
$L_i = -\log P(Y = y_i|X = x_i)$

cat	**3.2**
car	5.1
frog	-1.7

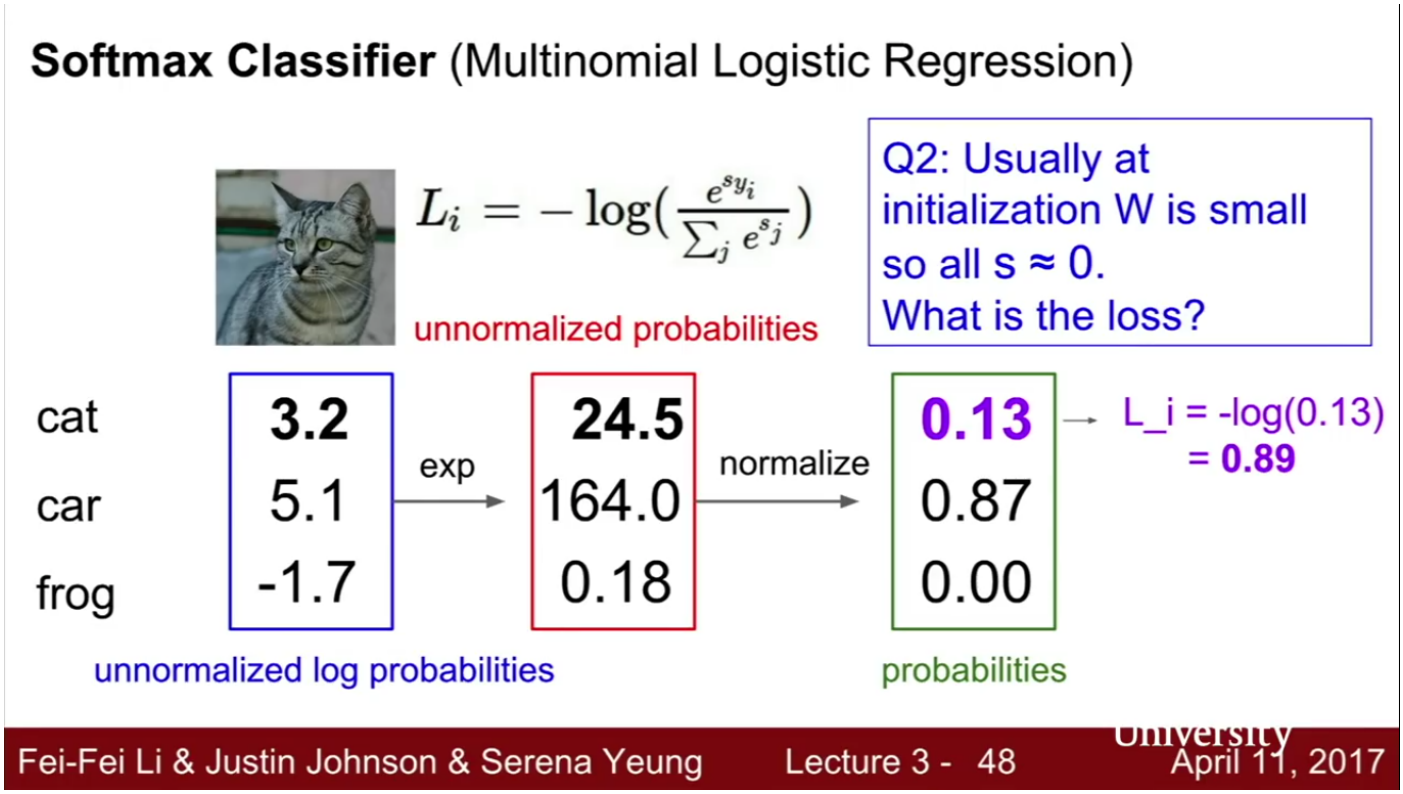
in summary: $L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$

- Its easier to maximize log probability than just the raw prob scores. So if we maximize the logP of correct class. But since loss measures badness we maximize the -logP

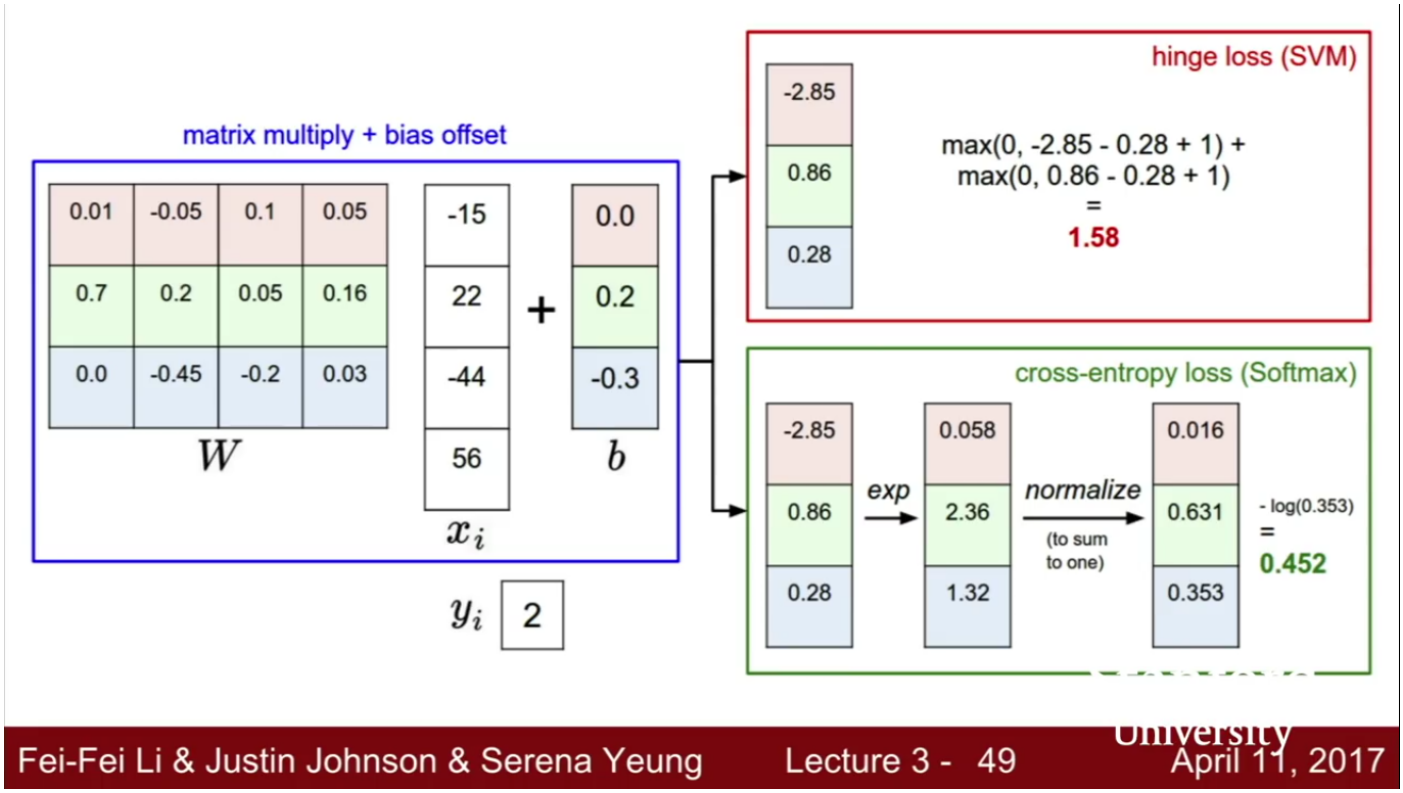
- The exponentiation in softmax is to make the scores positive. We divide the terms with sum to normalize the scores.



- Ans:
  - **Min Value**: We will get minimum value when we have a vector having 1 at the correct class and 0 at other places, and that will result in -log(1) which will give -0 which is 0.
  - What should the scores(unnormalized log prob) be for having a softmax of $[1, 0, 0]$?
    - For softmax prob vector to be $[1, 0, 0]$ our unnormalized prob score should be $[e^x, e^y, e^z]$ where $e^y = 0$ and $e^z = 0$, Now $e^y = 0$ will exist only when $y = -\infty$. Same with $e^z = 0$. Now the unnormalized log prob will be $[+number, -\infty, -\infty]$.
  - **Max Value**: If we get $-log(0)$ then we can get $+\infty$. and that will happen when the normalized log prob be $-\infty$



- Ans: Since the unnormalized log prob scores be $[0, 0, 0]$, this will result in the unnormalized prob be $[1, 1, 1]$, which when normalized gives $[1/3, 1/3, 1/3]$. In general, each of the loss term will be $-log(1/C) => log(C)$, where C is the number of classes.



- The linear model is the same but we have different loss functions for the comparison.
- **Note**: The loss is for the i^{th} example which has the correct label to be 2, that is the 3^{rd} score. When we calculate the loss we calculate with respect to the 3^{rd} label and $\mathbf{NOT}$ including all the labels.

Softmax vs. SVM

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

assume scores:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and $y_i = 0$

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?

- Ans: In the case of multi-class SVM, jiggling the datpoint will not affect the loss term until the score crosses the tolerance of the margin value. In the case of Softmax, the loss term tries to make the score to sum to 1, so it tries to push the correct prob to go high and the incorrect prob to go low. So it affects all the other scores also, which is not the case for Multi-class SVM which affects the incorrect classes only when the correct class score crosses the tolerance of the margin.
- Recap:

Recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

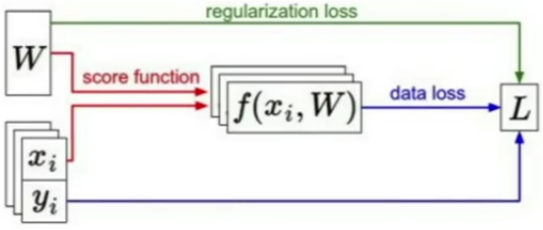
Softmax

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

SVM

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

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss



Optimization:

Walking man image is CC0 1.0 public domain

Strategy #1: A first very bad idea solution: **Random search**

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```


- Random Search: NOT a good strategy. Since it is not optimal and it cannot be reproduced.

Strategy #2: **Follow the slope**

- Works pretty well since we are utilizing the local geometry to approach the optimal W.

Strategy #2: **Follow the slope**

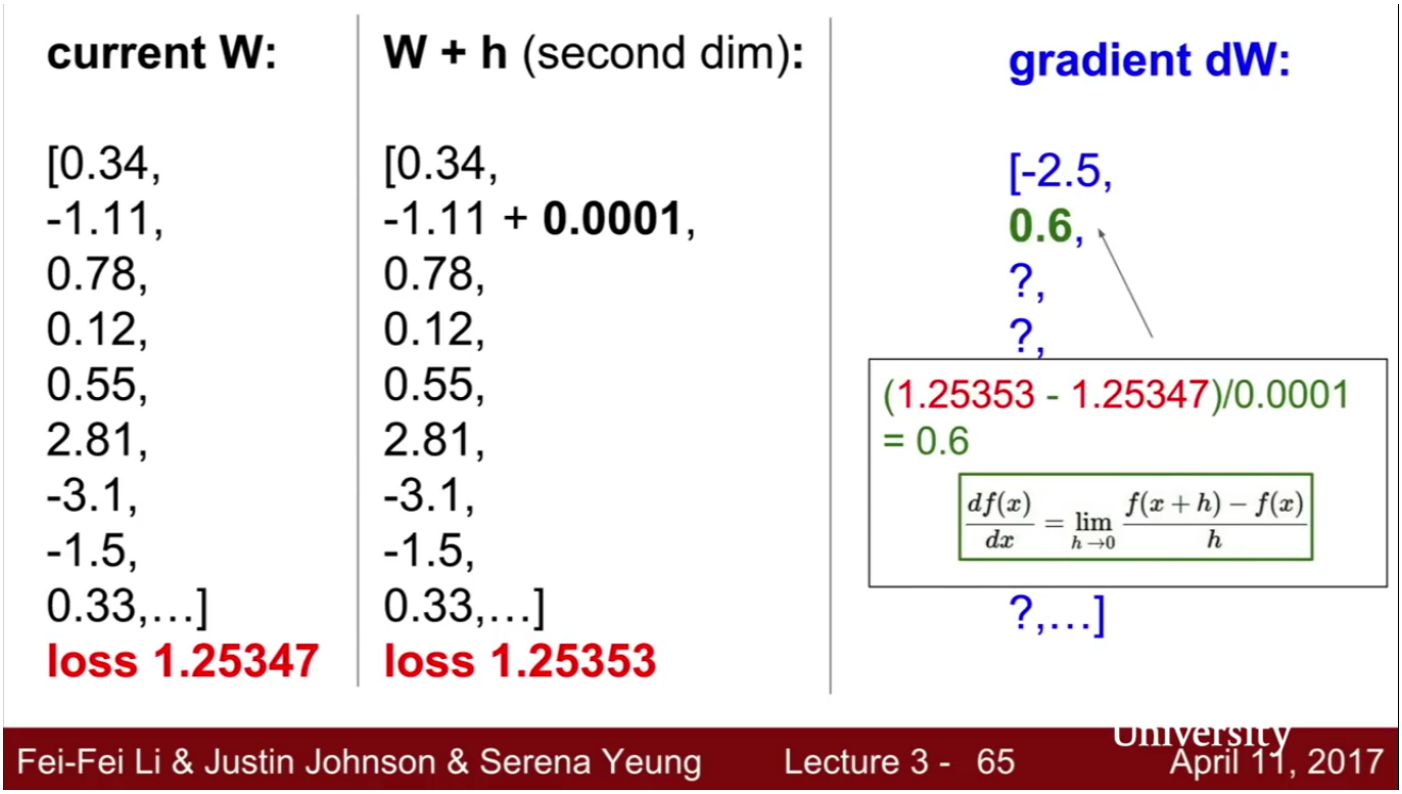
In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

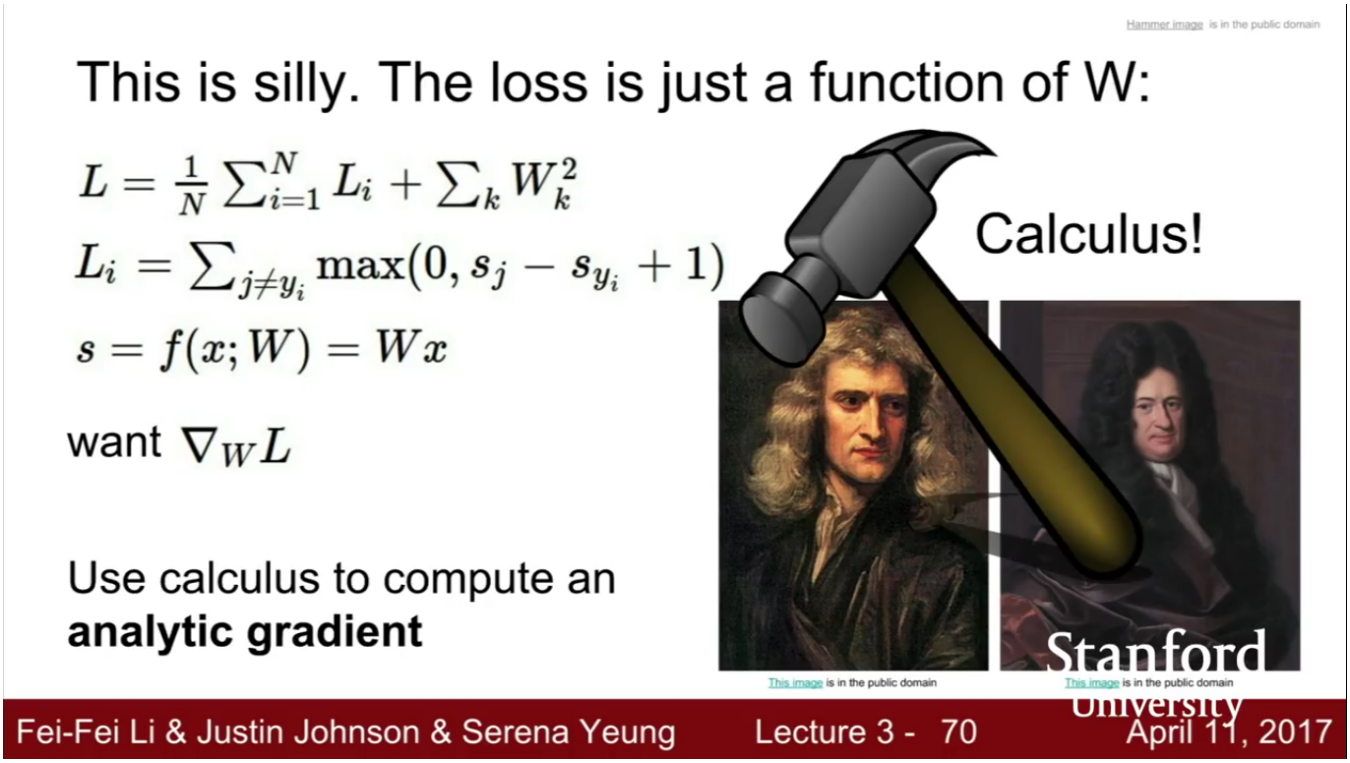
In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**

- **Each entry of the gradient vector tells us about the slope of the function f if we move in that co-ordinate direction.**
- **Gradient points in the direction of greatest increase of the function, so the negative of the gradient gives us the direction of greatest decrease.**
- **If we want to find the slope in a particular direction then we just perform dot product of the gradient with the direction vector corresponding to that direction.**
- Gradient is useful since it gives a linear, first order approximation to your function at the given point.



- We can compute gradient using finite difference technique, where we increment the value in a particular direction by a small value h, and find the gradient.
- THIS WORKS, but THIS IS VERY SLOW. Since in practice the vector W will have million of entries and we'll have many such Ws. And for each update we have to compute the gradient yielding a very time consuming task.



- Instead we can use calculus technique where we express the loss term L, and find the gradient for L.
  - Adv:
    - It'll be accurate.
    - It'll be fast.

In summary:

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone

=>

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check.**

- We usually check the result of analytical technique by comparing its result with numerical technique. We also reduce the dimensions for fast computation.

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

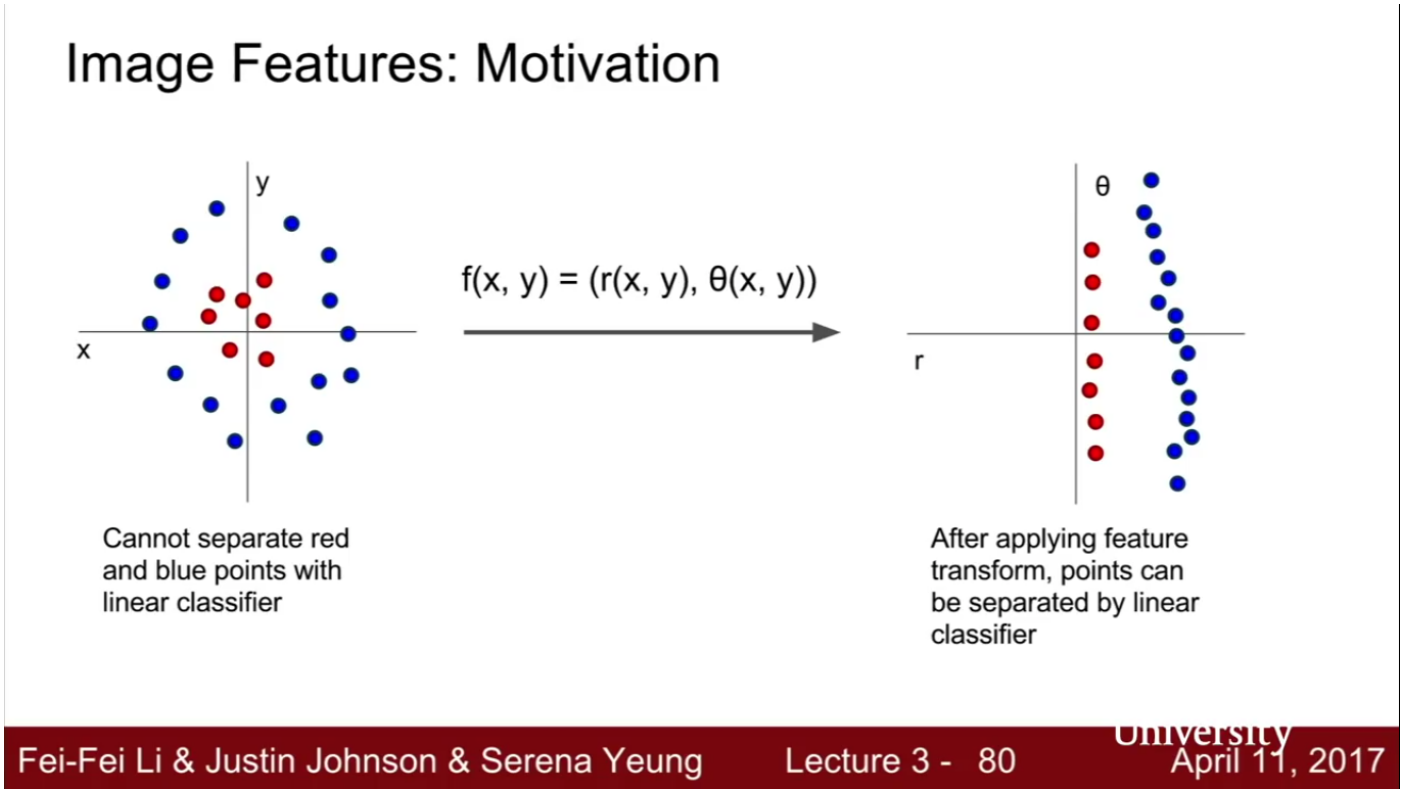
Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

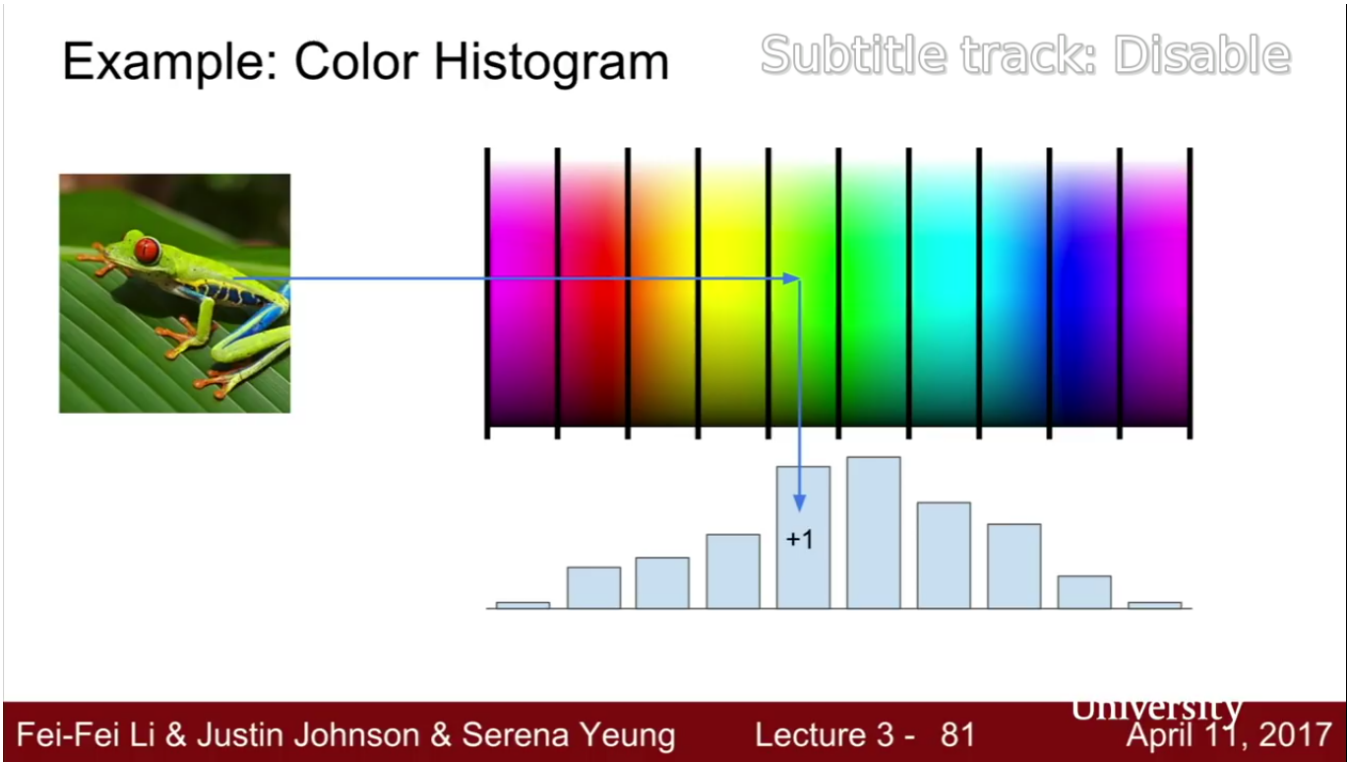
- MiniBatch is an approximation to Monte Carlo estimation of some expectation of the true value.

- Instead of using the image directly we commonly extract features from it and then combine them and later learn a classifier. This is done since the data(or problem) can be of multi-modality, and also to make it generalizable.

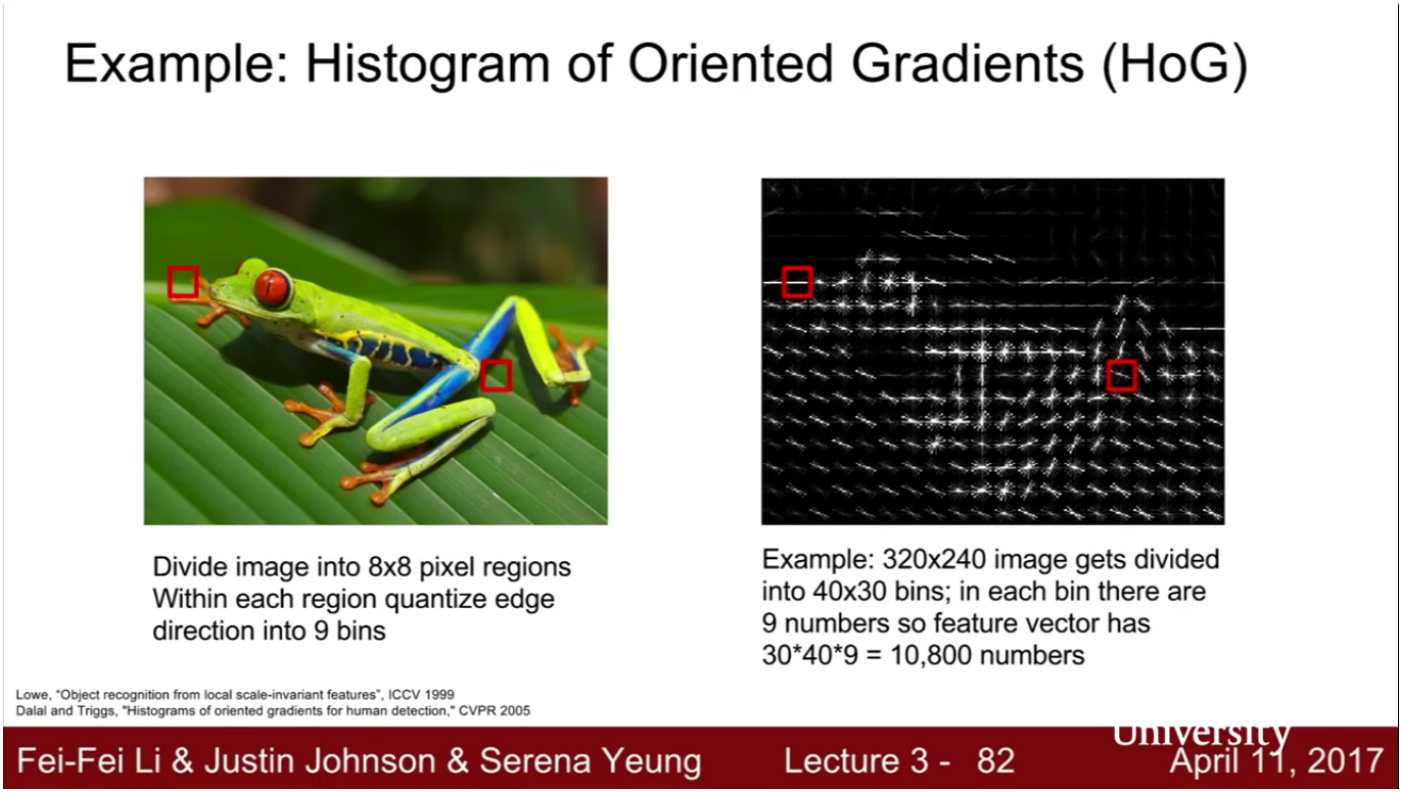


- Here we can notice that after the feature transform from cartesian to polar coordinates, it becomes easier to draw a decision boundary to classify them. Similarly we can perform some kind of a feature

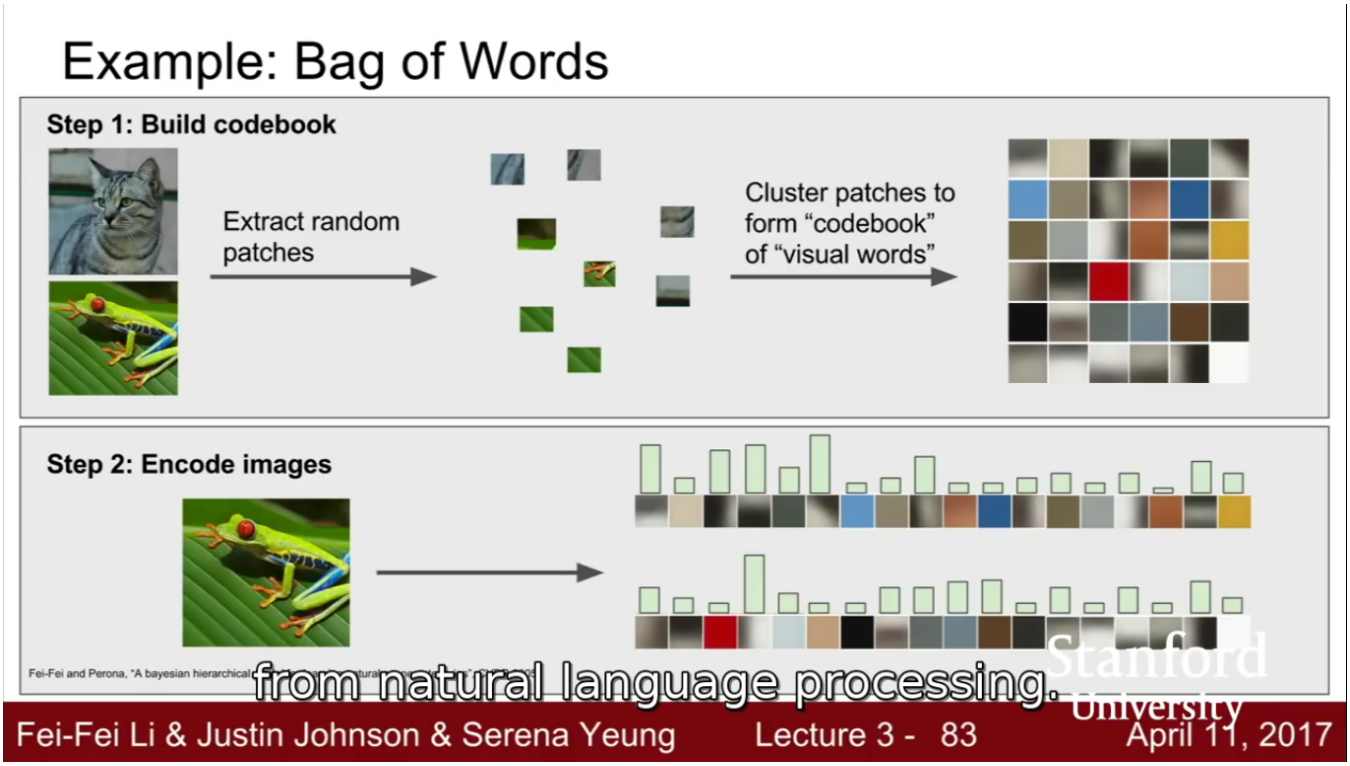
transform on images(or any data) to make it easier to classify.

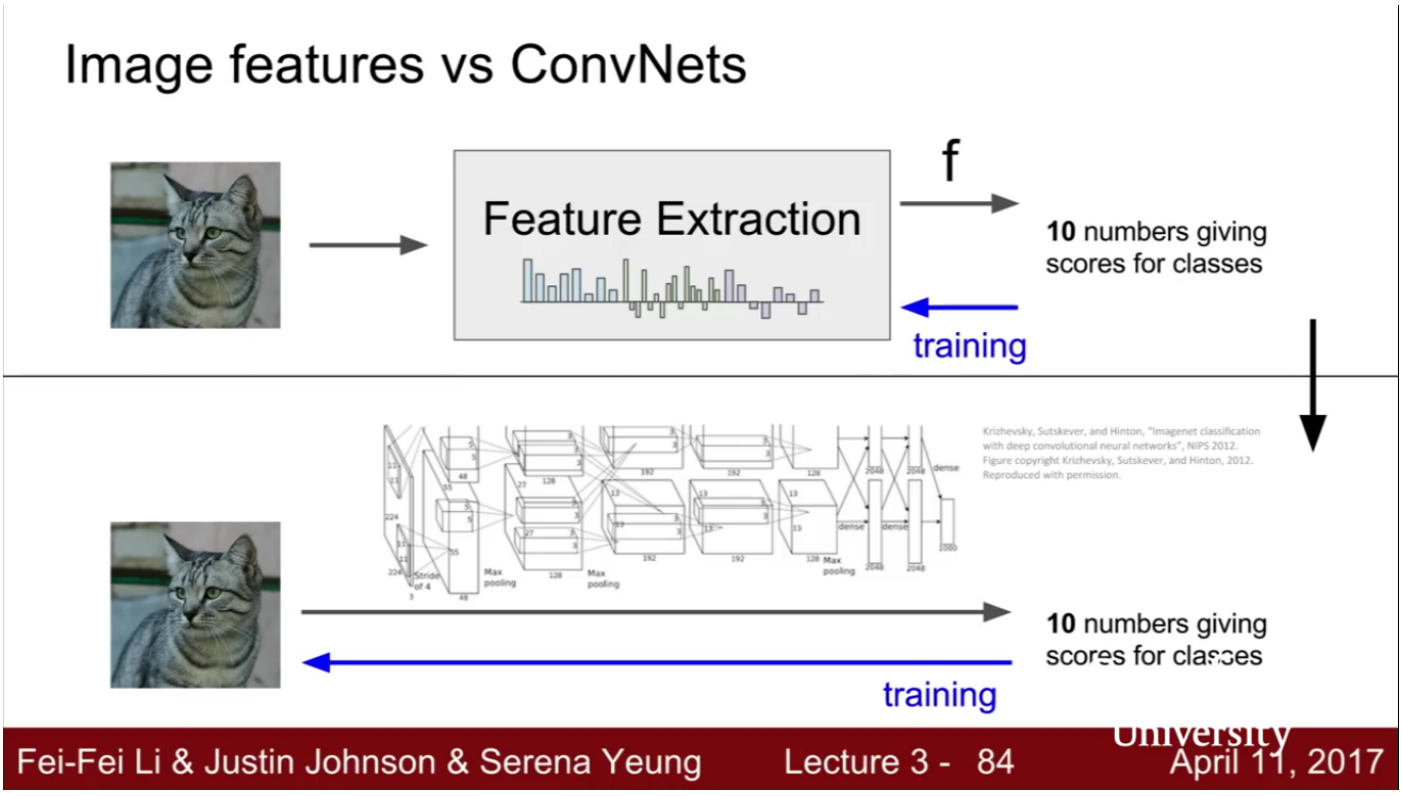


- Collecting histogram of a image is one kind of a feature extraction. We have made buckets of colors and we find the histogram.



- This is also a important feature which was used to identify humans





- Earlier techniques had this Feature Extraction block which was learnt only once. It was never updated again during training. Only the classifier was updated.

- In ConvNet, we learn the features and extract them and also learn the classifier weights.