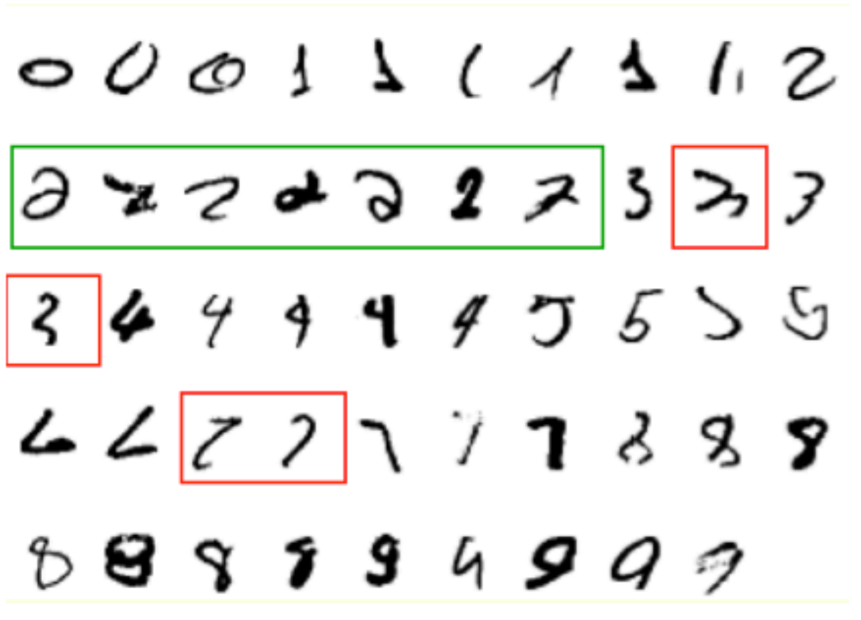
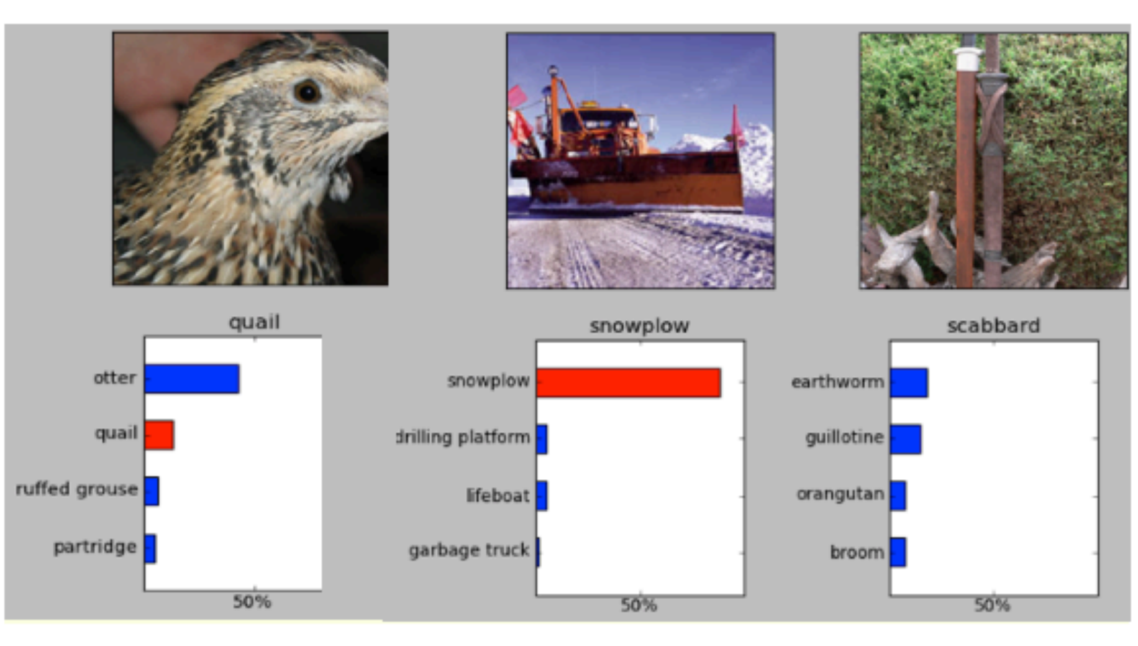
Video 19

Monday, October 26, 2020 9:04 PM





MNIST dataset

task: recognize handwritten digits in images

outputs: 10 possible classes.
corresponding to digits
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

ImageNet Dataset

task: object recognition

outputs: 1000 possible classes
include ≠ species of dogs / birds
≠ types of trucks
etc.

Targets are from k possible classes (classes are the discrete outputs)
Rather than index classes $\in \{1, \dots, k\}$, we use a one-hot vector to represent them:

$$\vec{t} = (0, \dots, 0, 1, 0, \dots, 0)$$

(the 1 is at the index of the class the example belongs to.)

Example for MNIST dataset
10 classes
let's take an image of a 6, its target would be

$$\vec{t} = (0,0,0,0,0,0,1,0,0,0)$$

(index of the class of the input which is in class 6.)

Linear model:
it now has
- D input dimensions (features) (has not changed)
- K output dimensions (classes) (has changed)

Rather than have a scalar output for each input, we now have a vector output for each input.

$$z_k = \vec{w}_k^T \vec{x} + b_k$$

- k: index of a class k among K classes (K: total # of classes)
- we have one weight vector $\vec{w}_k$ and one bias scalar b_k for each class of the task

Model output is the vector $\vec{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_k \\ \vdots \\ z_K \end{pmatrix}$ } K dimensions (total # of classes)

Linear model for multiclass setting:

$$\vec{z} = W\vec{x} + \vec{b}$$

- $\vec{z}$: vector of dimension K
- $\vec{x}$: input vector of dimension D
- W: matrix of dimensions $K \times D$ (K: # of classes; D: # of input dimensions (features))
- $\vec{b}$: vector of dimension K

Activation function:

Recall in the binary classification setting, we used logistic function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Here in the multiclass classification setting, we use the softmax function:
(multivariate generalization of the logistic function)

For $k \in 1..K$ (k: index of a class; K: # of classes):

$$y_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}}$$

} output of the model for one class $k \in 1..K$

Output of the linear model w/ a softmax activation function:

$$\vec{y} = \overrightarrow{\text{softmax}}(\vec{z}) = \begin{pmatrix} \frac{e^{z_1}}{\sum_{k'=1}^{K} e^{z_{k'}}} \\ \vdots \\ \frac{e^{z_K}}{\sum_{k'=1}^{K} e^{z_{k'}}} \end{pmatrix}$$

(vector of K dimensions)

The components of $\vec{z}$, values $z_1, \dots, z_K$ are called **logits** (inputs to the softmax)

Properties of the softmax:

- outputs are all positive
- outputs all sum to 1 ← nice because they can be interpreted as probabilities (each component k of the softmax output, which is a vector, is the probability of the input being in class k)
- If one of the logits z_k is much larger than the other logits,

$$\overrightarrow{\text{softmax}}(\vec{z}) = \overrightarrow{\text{softmax}}(z_1, \dots, z_K) \approx (0, 0, \dots, 0, 1, 0, \dots 0)$$

(the 1 is at position k)

In this case the softmax behaves like an argmax.

Cross-entropy for multiclass classification

$$\mathcal{L}_{CE}(\vec{y}, \vec{t}) = -\sum_{k=1}^{K} t_k \log y_k$$

- $\vec{y} = \overrightarrow{\text{softmax}}(\vec{z})$: model output; $\vec{t}$: target (vectors of k dimensions)
- t_k: k^{th} component of the target (one-hot vector)
- y_k: k^{th} component of the model output

$$= -\vec{t}^T (\log \vec{y})$$

(we apply the log elementwise to the (vector) model output.)

Here again, we can combine the cross-entropy loss function and the softmax activation function into the softmax-cross-entropy function

Gradient descent update rule:

Requires computing

$$\frac{\partial \mathcal{L}_{CE}}{\partial \vec{z}} = \vec{y} - \vec{t}$$

($\vec{z}$: now a vector)