EBU7240

Computer Vision

- Introduction to Deep Learning -

Semester 1, 2021

Changjae Oh

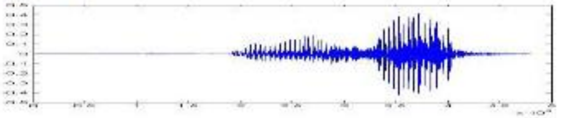
Outline

- Machine learning basics++
- Introduction to deep learning
- Linear classifier

What is Machine Learning?

- **Learning = Looking for a Function**

Speech Recognition

$f($  $)=$ **"Hello"**

Handwritten Recognition

$f($  $)=$ "2"

Weather forecast

$f($  Thursday $)=$ "  Saturday"

Play video games

$f($  $)=$ "move left"

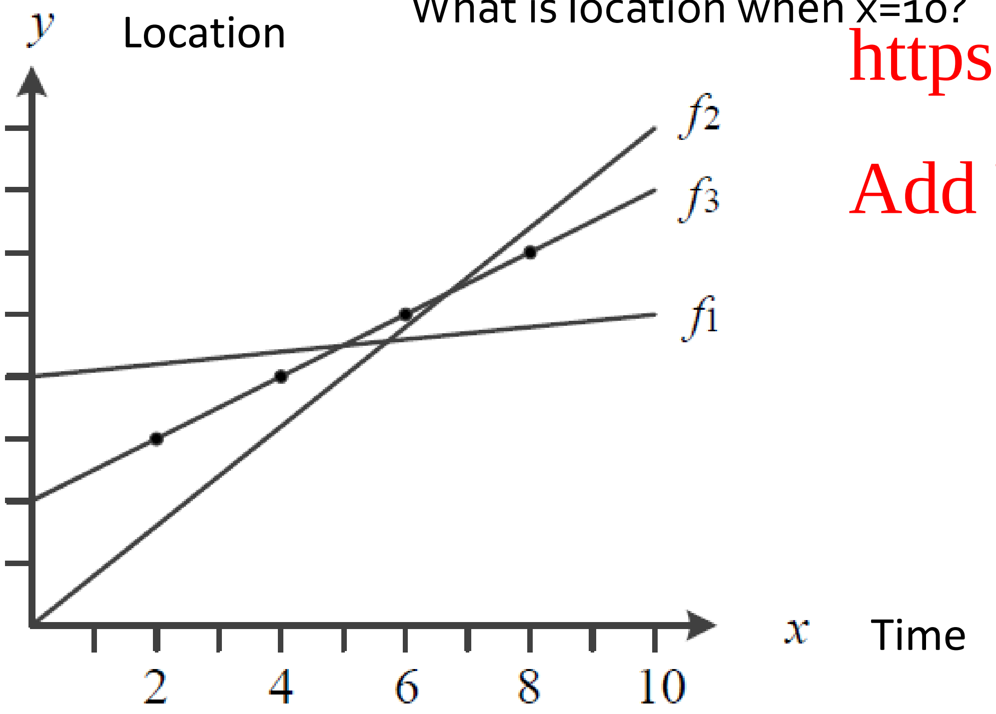
http://www.slideshare.net/ckmarkohchang

Machine Learning: Basic Concept

- **Prediction task**
  - Regression: returns a specific value
  - Classification: returns a class label

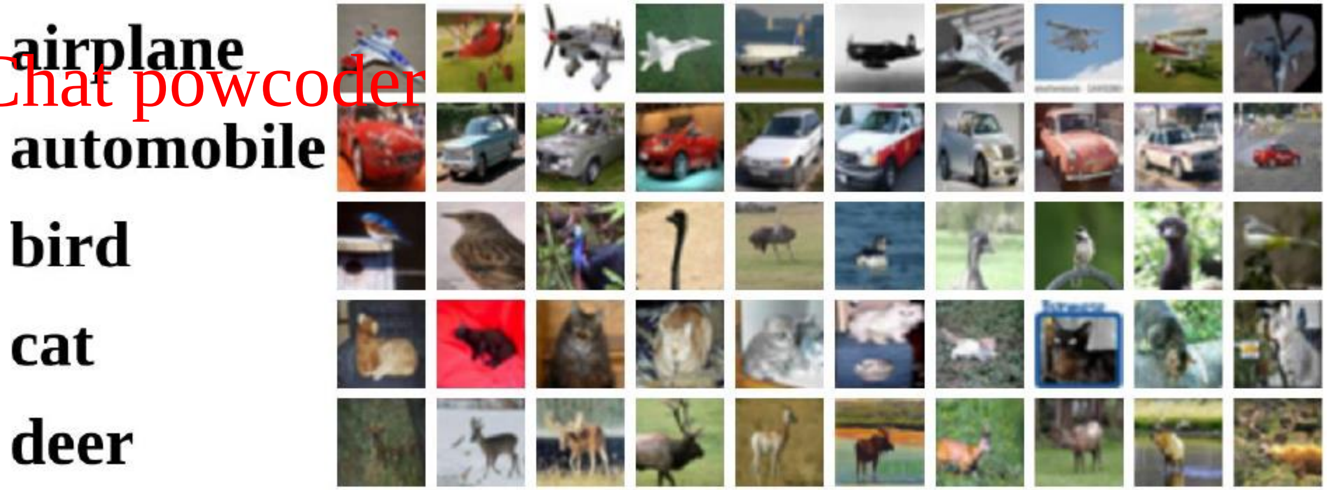
Regression example

What is location when x=10?



Classification example

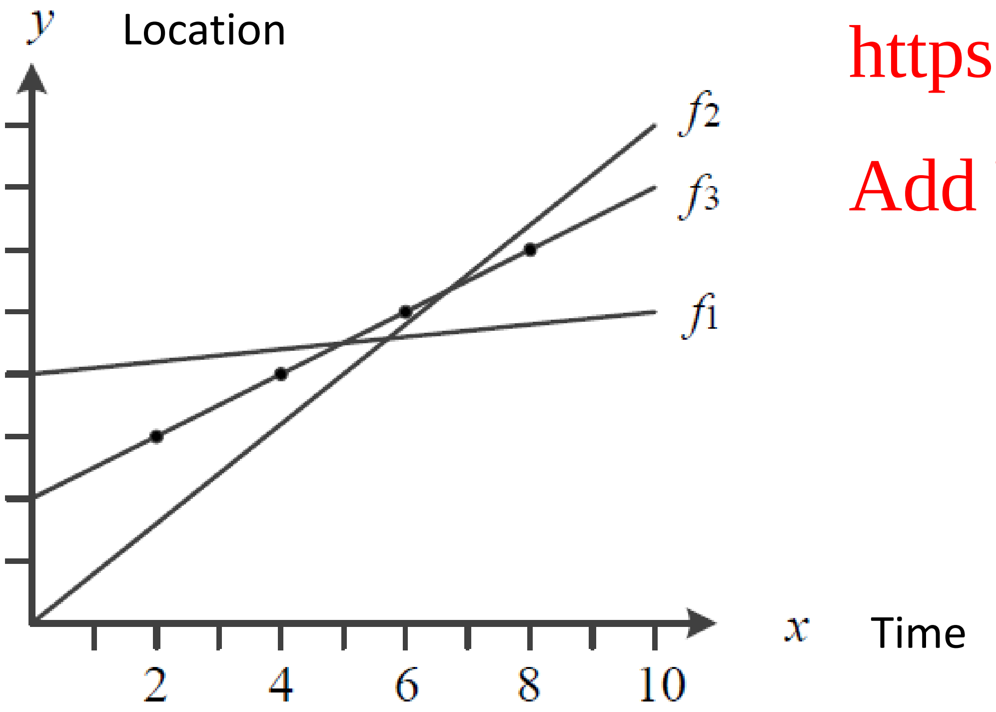
For a new image, what class does it belong to?



Machine Learning: Basic Concept

- **Training data**

$\mathbb{X} = \{\mathbf{x}_1 = (2.0), \mathbf{x}_2 = (4.0), \mathbf{x}_3 = (6.0), \mathbf{x}_4 = (8.0)\}$
$\mathbb{Y} = \{y_1 = 3.0, y_2 = 4.0, y_3 = 5.0, y_4 = 6.0\}$



Training the model with data

- Finding optimal parameters
- Starting at a random value, increasing accuracy to compute optimal parameters

$$y = wx + b$$

Optimal parameter w=0.5 b=2.0

Goal

- Minimize errors for new samples (test set)
- Generalization
  refers to high performance for test sets

Machine Learning: Basic Concept

- **Multi-dimensional feature space**
  - d-dimensional data: $\mathbf{x}=(x_1,x_2,\ \ldots\ ,x_d)^{\mathrm{T}}$ Note) d=784 in MNIST data

- **Linear classifier for *d*-dimensional data**
  - 1-D linear classifier # of variables = $d+1$
    - Widely used in machine learning

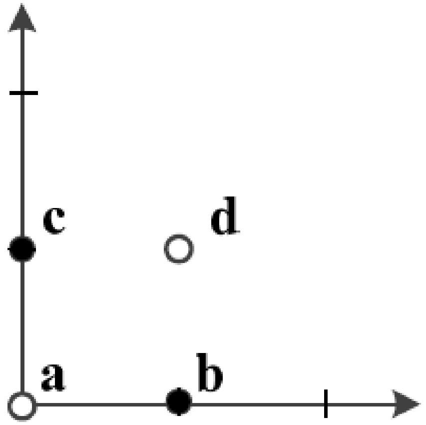
$$y = \underline{w_1}x_1 + \underline{w_2}x_2 + \cdots + \underline{w_d}x_d + \underline{b}$$

  - 2-D linear classifier # of variables = $\frac{(d+1)(d+2)}{2}$

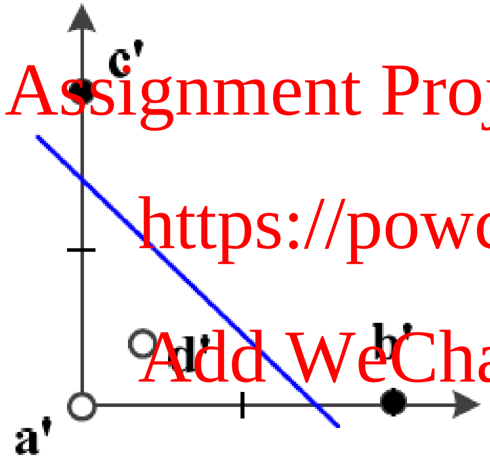
$$y = \underline{w_1}x_1^2 + \underline{w_2}x_2^2 + \cdots + \underline{w_d}x_d^2 + \underline{w_{d+1}}x_1x_2 + \cdots + w_{\underline{\frac{d(d+1)}{2}}}x_{d-1}x_d + w_{\underline{\frac{d(d+1)}{2}+1}}x_1 \cdots + w_{\underline{\frac{d(d+1)}{2}+d}}x_d + \underline{b}$$

Machine Learning: Basic Concept

- **Feature space transformation**
  - Map a linearly non-separable feature space into separable space



Original feature space



Transformed feature space, which is linearly separable

Toy example

$$\mathbf{x} = (x_1, x_2)^{\mathrm{T}} \longrightarrow \mathbf{x}' = \left(\frac{x_1}{2x_1x_2 + 0.5}, \frac{x_2}{2x_1x_2 + 0.5}\right)^{\mathrm{T}}$$

$$\mathbf{a} = (0,0)^{\mathrm{T}} \longrightarrow \mathbf{a}' = (0,0)^{\mathrm{T}}$$

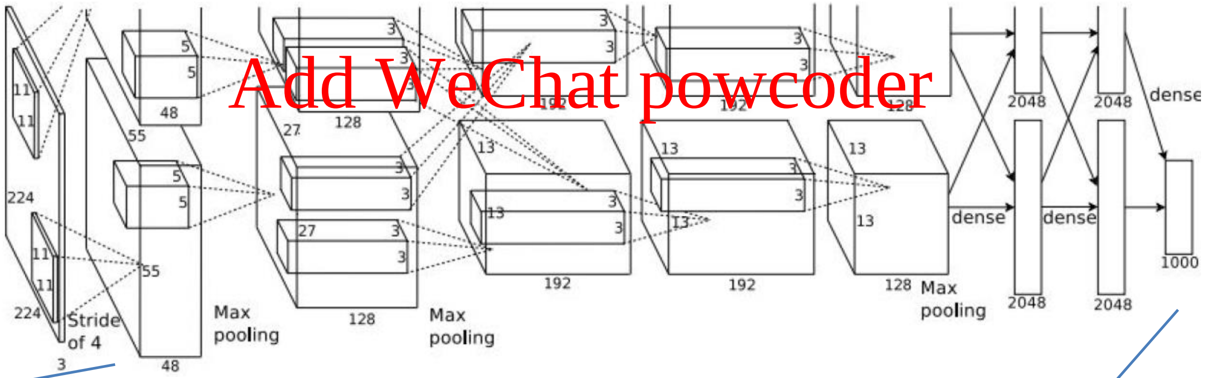
$$\mathbf{b} = (1,0)^{\mathrm{T}} \longrightarrow \mathbf{b}' = (2,0)^{\mathrm{T}}$$

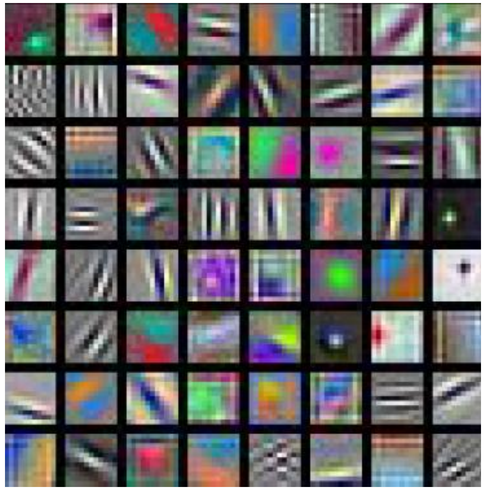
$$\mathbf{c} = (0,1)^{\mathrm{T}} \longrightarrow \mathbf{c}' = (0,2)^{\mathrm{T}}$$

$$\mathbf{d} = (1,1)^{\mathrm{T}} \longrightarrow \mathbf{d}' = (0.4,0.4)^{\mathrm{T}}$$

Machine Learning: Basic Concept

- **Representation learning**
  - Aims to find good feature space automatically
  - Deep learning finds a hierarchical feature space by using neural networks with multiple hidden layers.
    - The first hidden layer has low-level features (edge, corner points, etc.), and the right-hand side features advanced features (face, wheel, etc.)

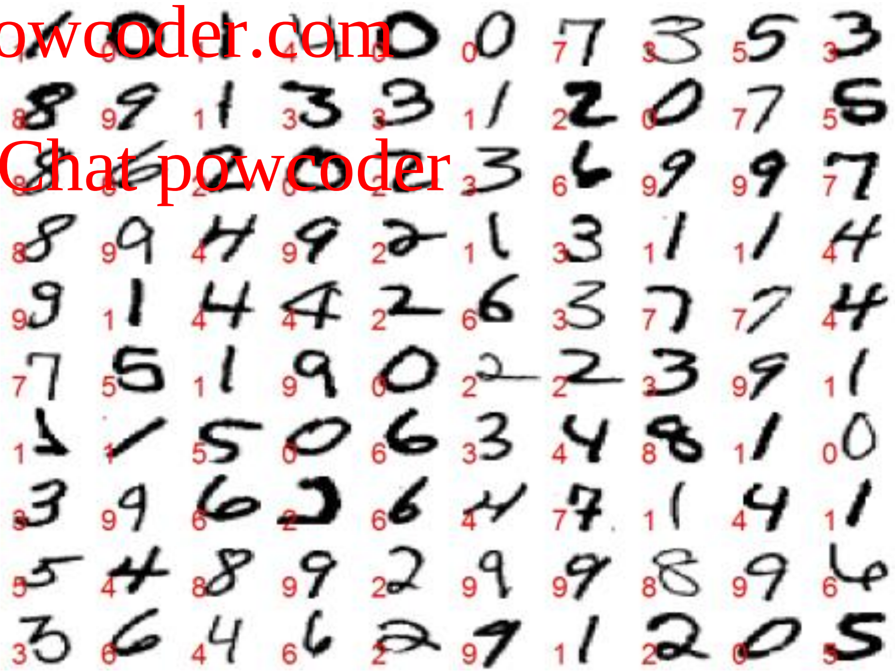




Data for Machine Learning

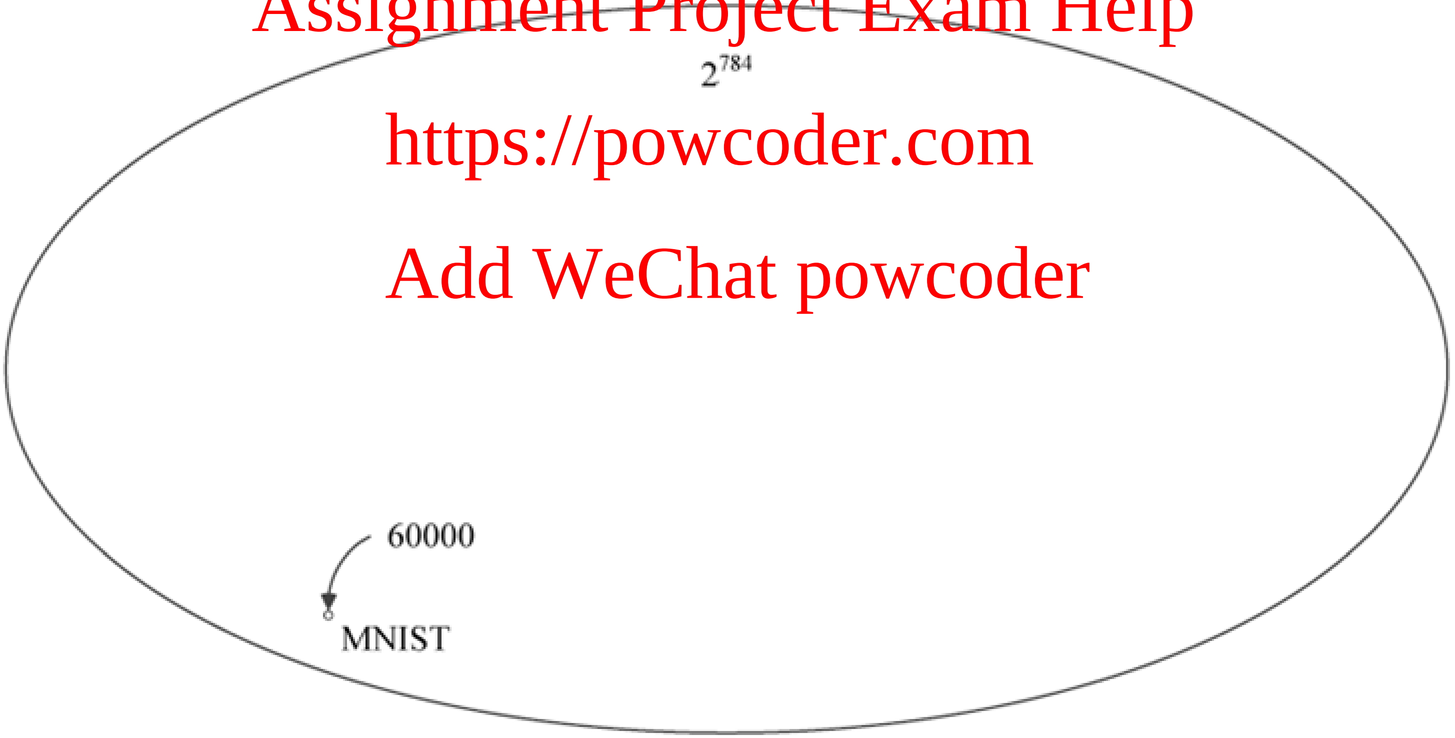
- **The quality of the training data**
  - To increase estimation accuracy, diverse and enough data should be collected for a given application.
  - Ex) After learning from a database with a frontal face only, the recognition accuracy of side face will be degraded.

- **MNIST database**
  - Handwritten numeric database
  - Training data: 60,000
  - Test data: 10,000



Data for Machine Learning

- **Database size vs. training accuracy**
  - Ex) MNIST: 28*28 binary image
    → The total number of possible samples is 2^{784}, but MNIST has 60,000 training images.



Data for Machine Learning

- **How does a small database achieve high performance?**
  - In a feature space, the actual data is generated in a very small subspace

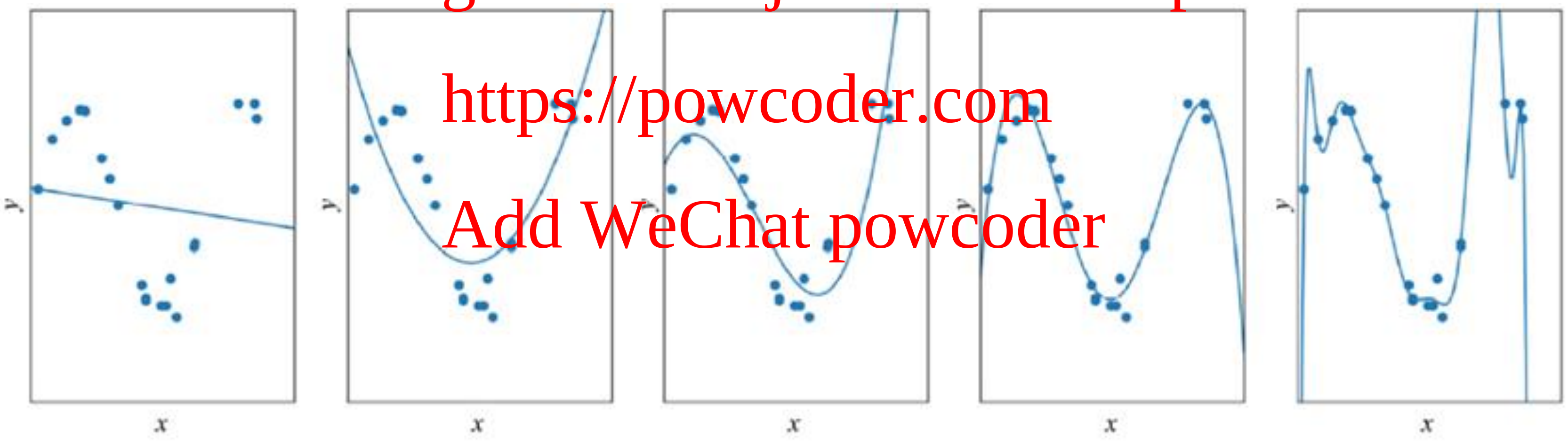
     ,  is unlikely to happen.

  - Manifold assumption

    Smooth change according to certain rules like 

Training Model: under-fitting vs. over-fitting

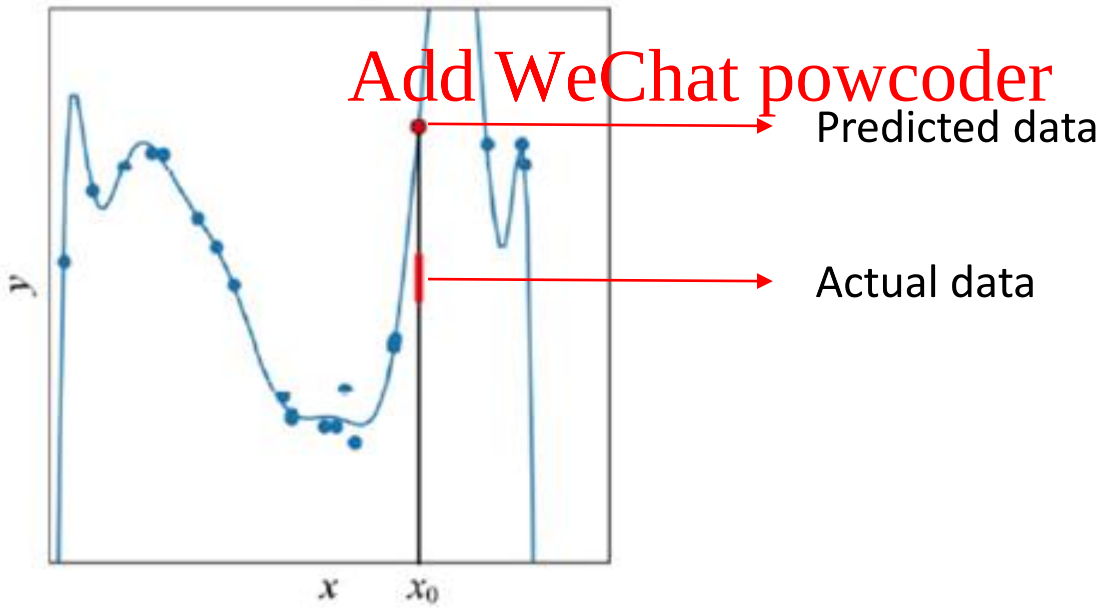
- **Under-fitting**
  - Model capacity is too small to fit the data accordingly.
  - Model with higher order can be used.



1st order   2nd order   3rd order   4th order   12th order

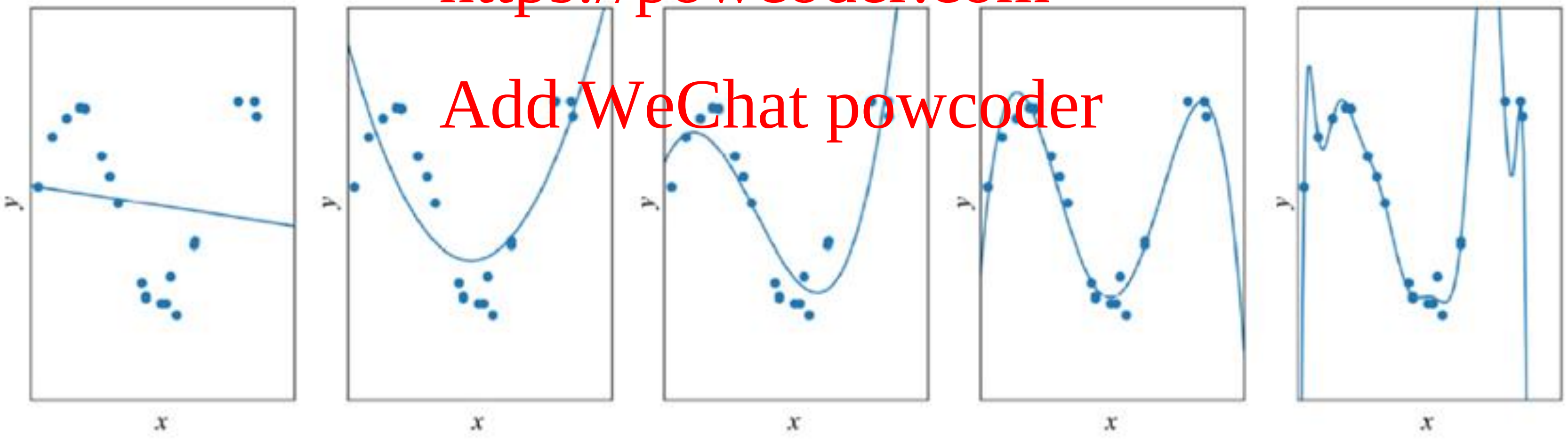
Training Model: under-fitting vs. over-fitting

- **Over-fitting**
  - 12^{th} order polynomial model approximates perfectly for the training set.
  - But if you anticipate "new" data, there's a big problem.
  - Since the model capacity is large, the training process also accepts data noise.
  - The model with the appropriate capacity should be selected.



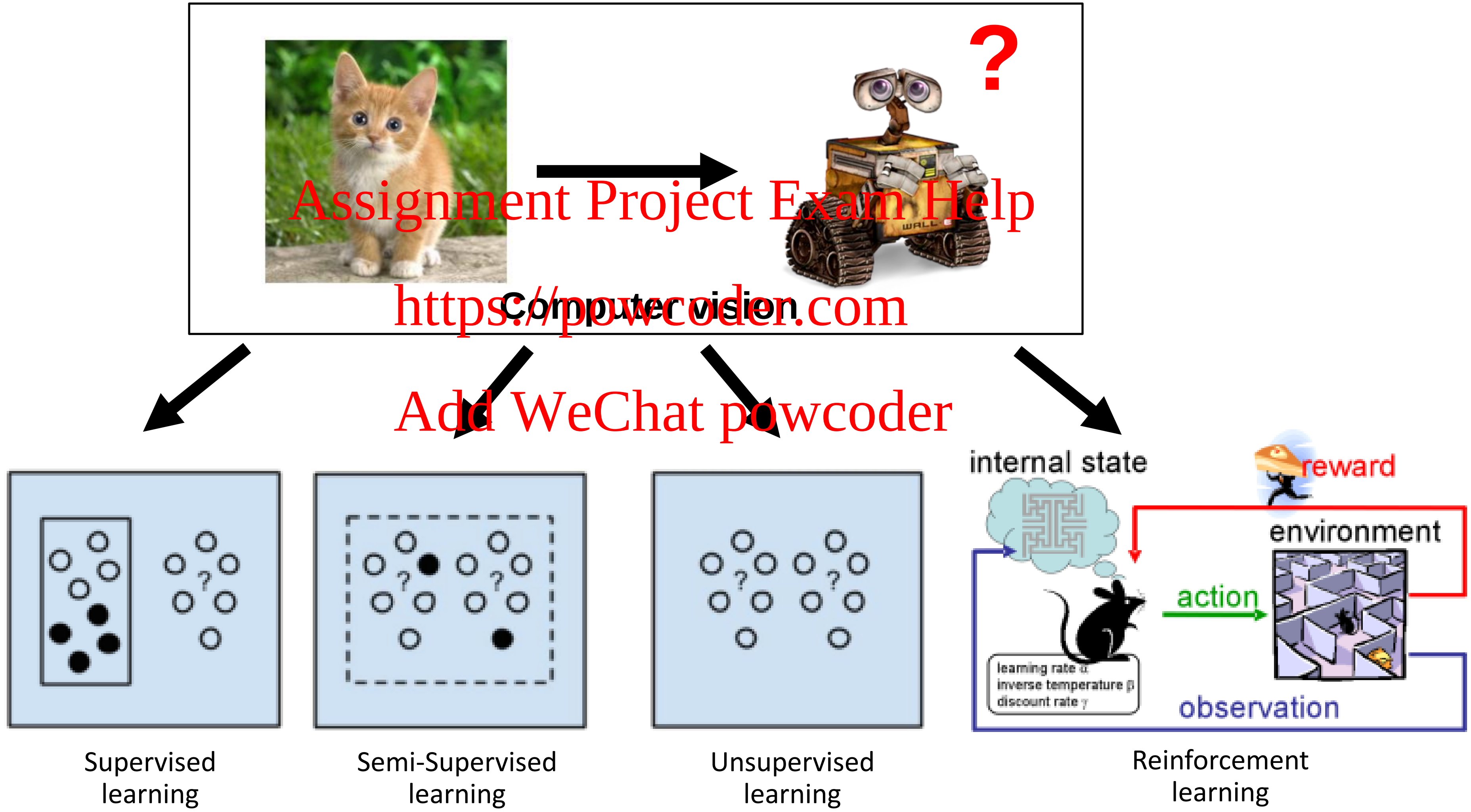
Training Model: under-fitting vs. over-fitting

- 1^{st} and 2^{nd} order model show poor performance for both the training and the test set.
- 12^{th} order model shows high performance in training set, but low performance in test set. → low generalization ability
- 3^{rd} and 4^{th} order model are lower than the 12^{th} order model for the training set, but the test set has high performance → higher generalization capability



1^{st} order 2^{nd} order 3^{rd} order 4^{th} order 12^{th} order

Spectrum of supervision



Spectrum of supervision

- **Supervised learning**
  - Both the feature vector $\mathbb{X}$ and the output $\mathbb{Y}$ are given.
  - Regression and classification problem

- **Unsupervised learning**

  - The feature vector $\mathbb{X}$ is given, but the output $\mathbb{Y}$ is not given.
  - Ex) Clustering, density estimation, feature space conversion

Spectrum of supervision

- **Reinforcement learning**
  - The output is given, but it is different from supervised learning.
  - Ex) Go
    - Once the game is over, you get a point (credit).
      If you win, get 1, and -1 otherwise.
    - The credit should be distributed to each sample of the game.

- **Semi-supervised Learning**
  - Some of data have both $\mathbb{X}$ and $\mathbb{Y}$, but others have only $\mathbb{X}$.
  - It is becoming important, since it is easy to collect $\mathbb{X}$, but $\mathbb{Y}$ requires manual tasks.

Introduction to Deep Learning

From Wiki

- **Deep learning**
  - is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations.

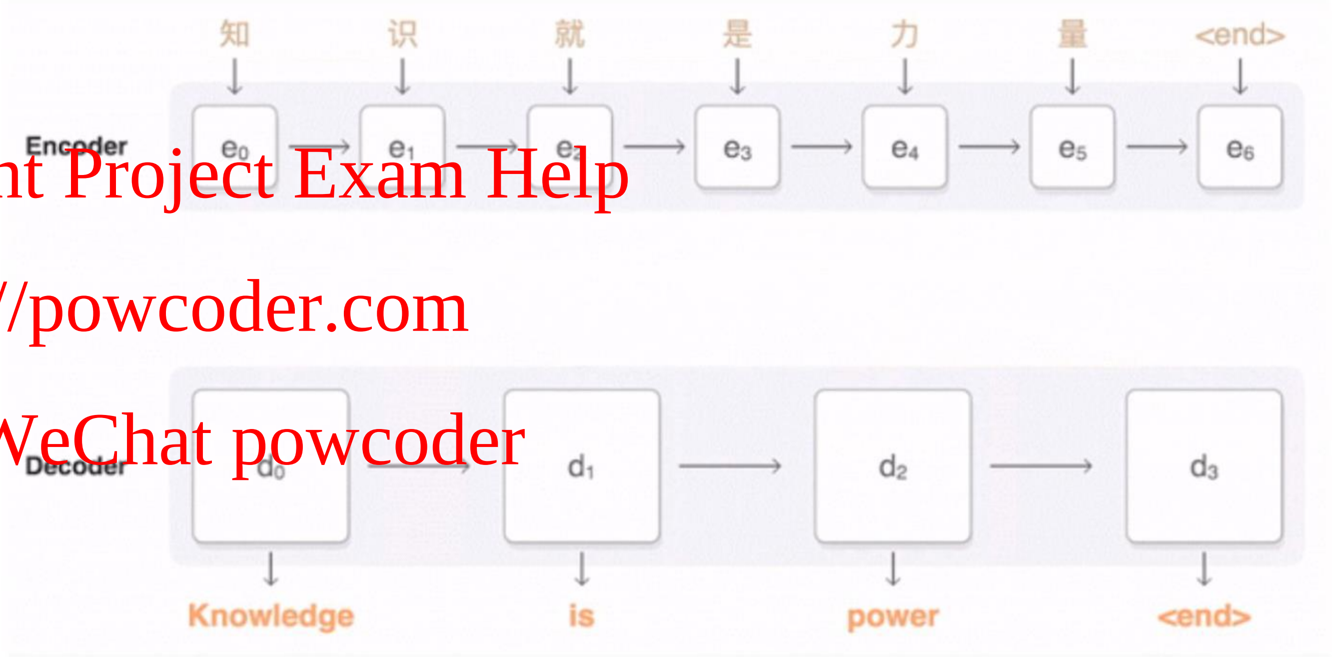
Deep learning success

- **Image classification**
- Machine translation
- Speech recognition
- Speech synthesis
- Game playing

- .. and many, many more

Deep learning success

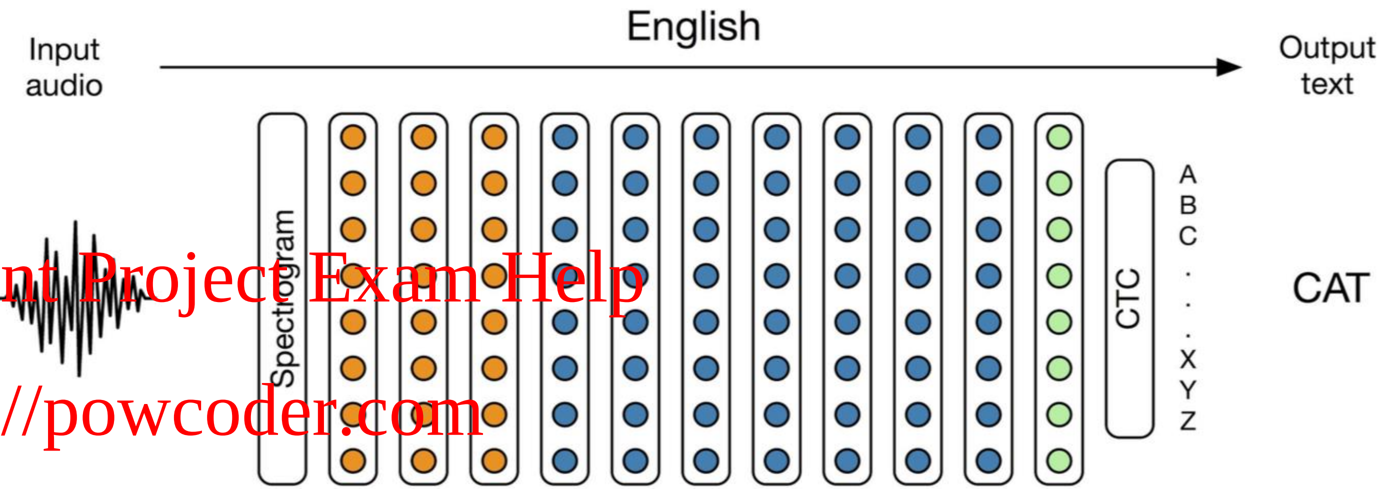
- Image classification
- **Machine translation**
- Speech recognition
- Speech synthesis
- Game playing



- .. and many, many more

Deep learning success

- Image classification
- Machine translation
- **Speech recognition**
- Speech synthesis
- Game playing

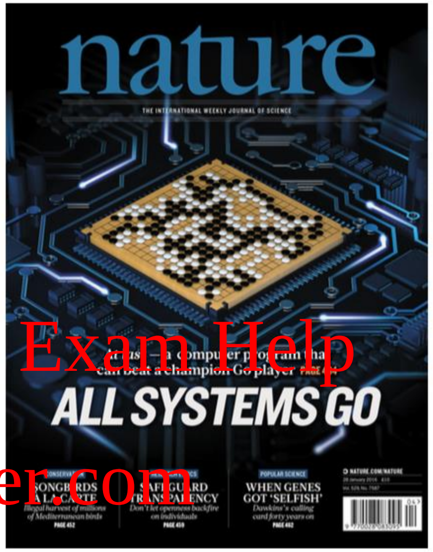


- .. and many, many more

Deep learning success

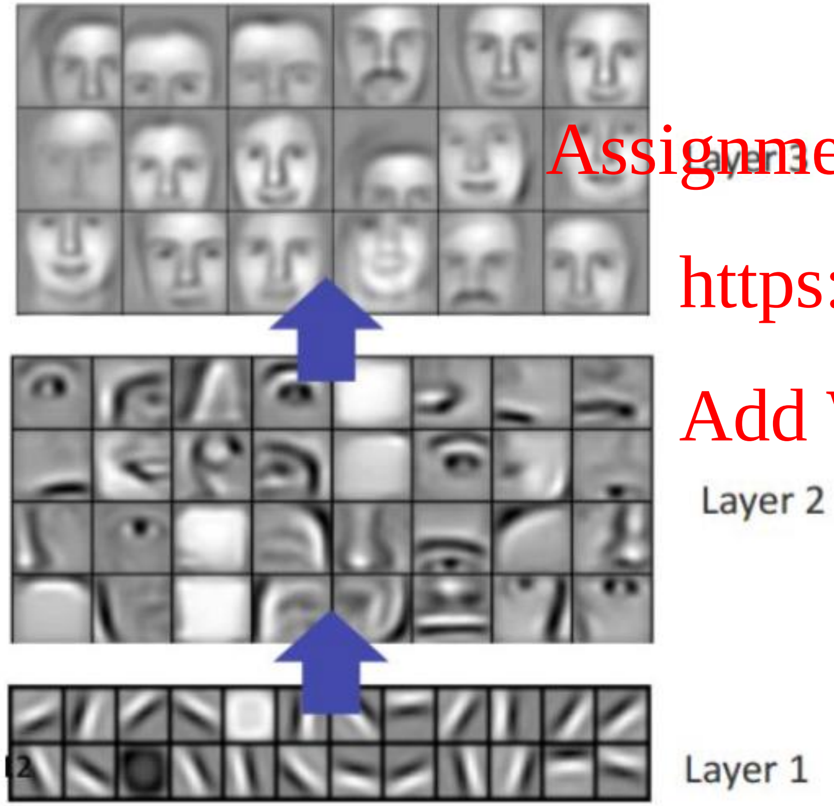
- Image classification
- Machine translation
- Speech recognition
- **Game playing**

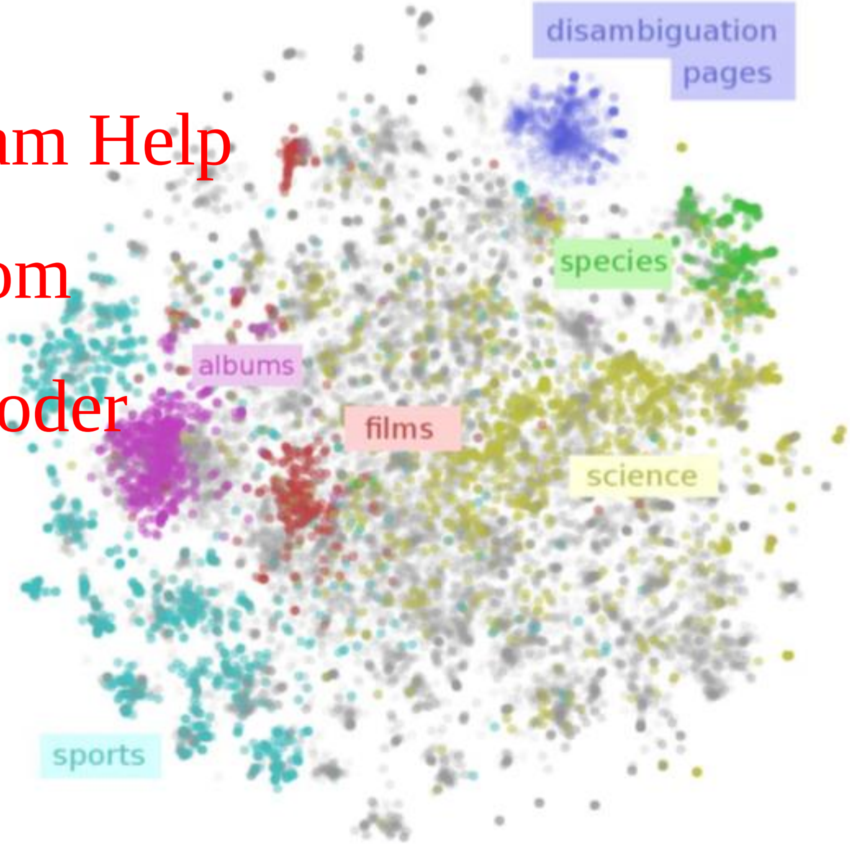
- .. and many, many more



Why deep learning?

- **Hand-crafted features vs. Learned features**



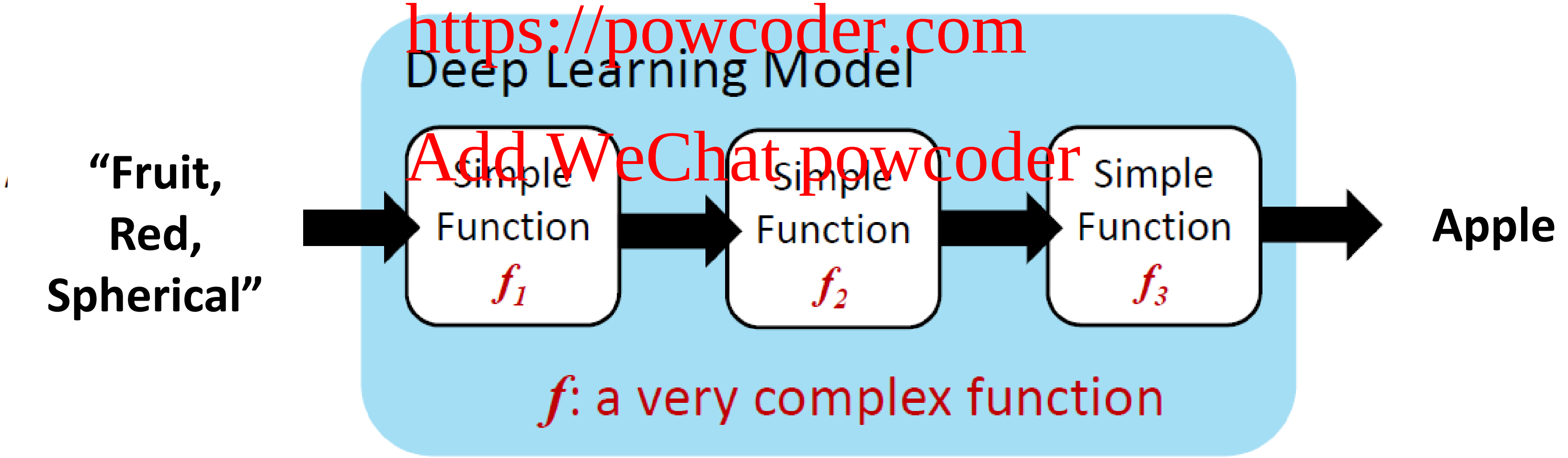


Why now?

- Large datasets
- GPU hardware advances + Price decreases
- Improved techniques (algorithm)

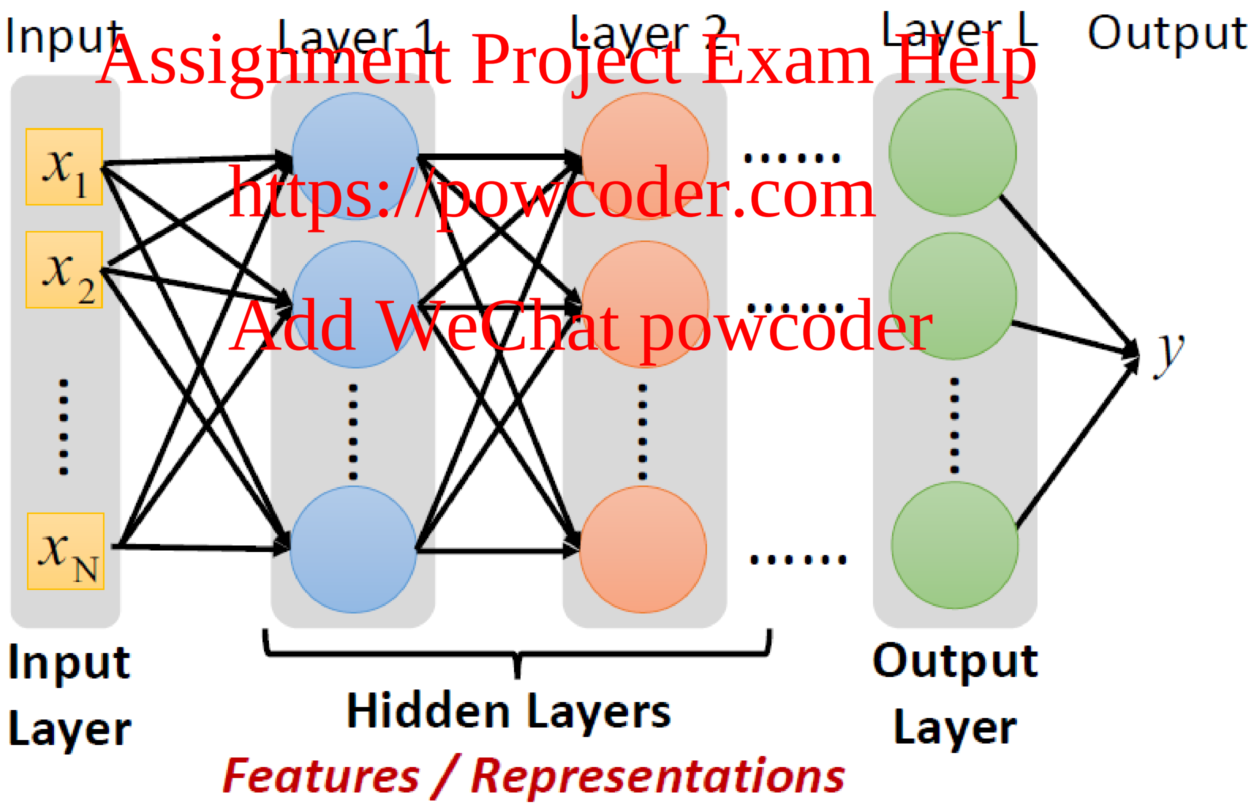
What is Deep Learning?

- **Stacked Functions Learned by Machine**
  - **End-to-end training**: what each function should do is learned automatically
  - Deep learning usually refers to neural network based model



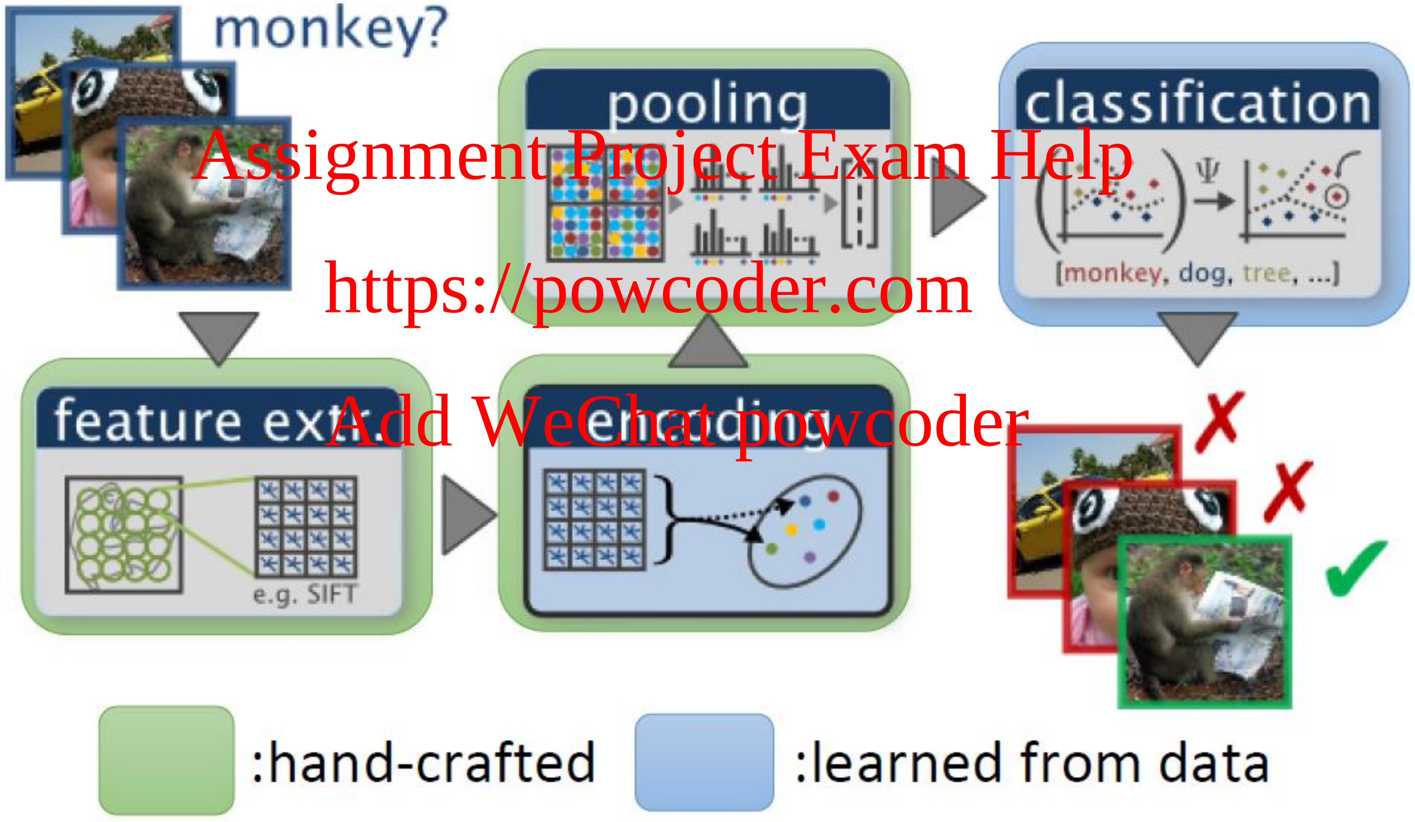
What is Deep Learning?

- **Stacked Functions Learned by Machine**
  - **Representation Learning:** learning features/representations
  - **Deep Learning:** learning (multi-level) features and an output



Machine Learning vs. Deep Learning

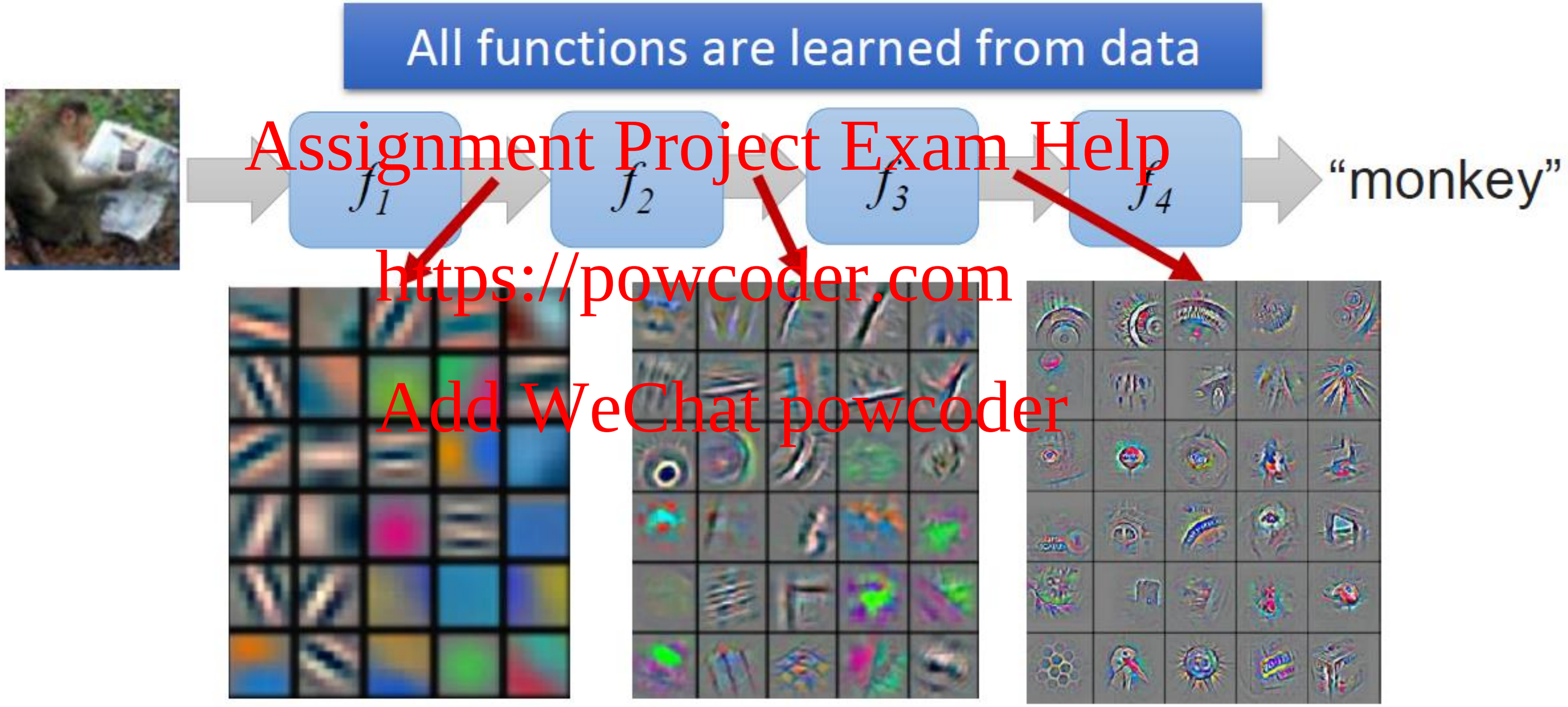
- **Deep vs Shallow: Image Recognition**
  - Shallow model using machine learning



http://www.slideshare.net/ckmarkohchang

Machine Learning vs. Deep Learning

- **Deep vs Shallow: Image Recognition**
  - Deep model using deep learning



Features / Representations

http://www.slideshare.net/ckmarkohchang

Machine Learning vs. Deep Learning

- **Machine Learning vs. Deep Learning**

Machine Learning = Feature descriptor + Classifier

Feature: hand-crafted domain-specific knowledge
Describing your data with features that computer can understand
Ex) SIFT, Bag-of-Words (BoW), Histogram of Oriented Gradient (HOG)

Optimizing the classifier weights on features
Ex) Nearest Neighbor (NN), Support Vector Machine (SVM), Random Forest (RF)

Machine Learning vs. Deep Learning

- **Machine Learning vs. Deep Learning**

Deep Learning = Feature descriptor + Classifier

Feature: Representation learned by machine
Automatically learned internal knowledge

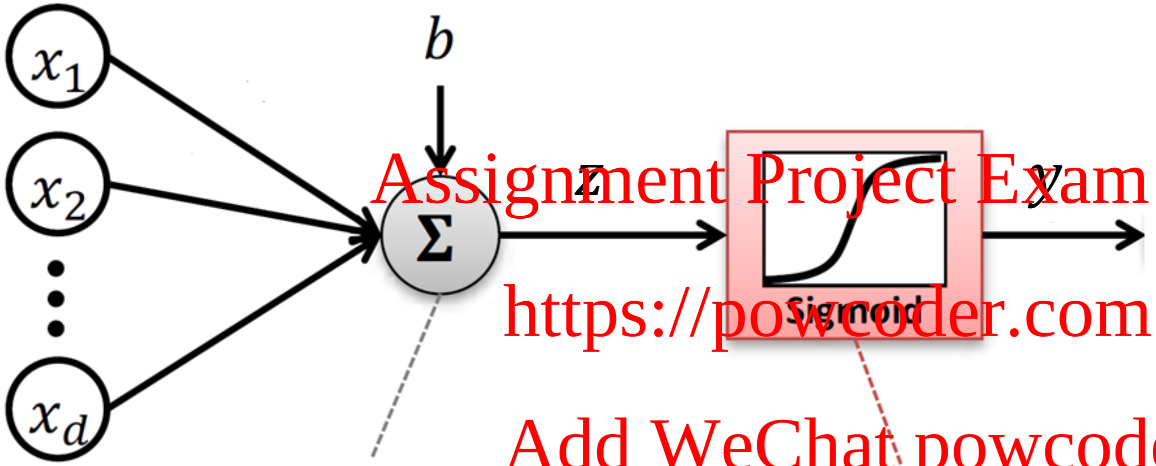
Optimizing the classifier weights on features

Neural network based model

A series of linear classifiers and non-linear activations + Loss function

Deep Learning

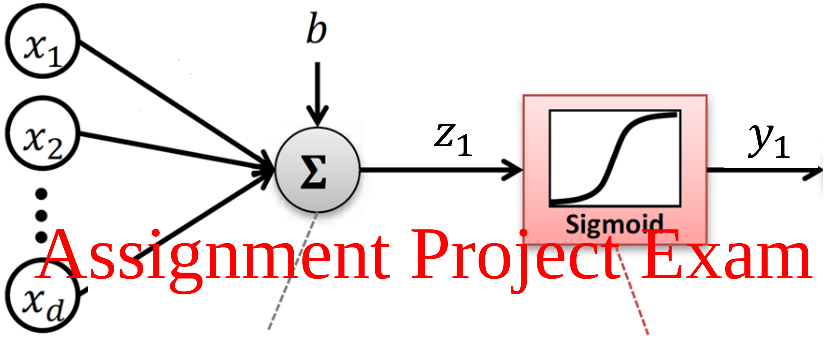
- **A single neuron**



$$z = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = \sum_{k=1}^{d} w_k x_k + b \qquad y = \frac{1}{1 + e^{-z}}$$

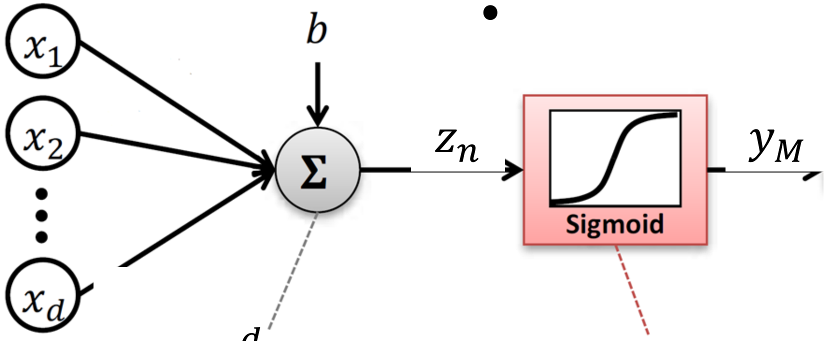
Deep Learning

- **A single layer with multiple neurons**



$$z_1 = \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 = \sum_{k=1}^{d} w_{1k}x_k + b_1 \quad y_1 = \frac{1}{1+e^{-z_1}}$$

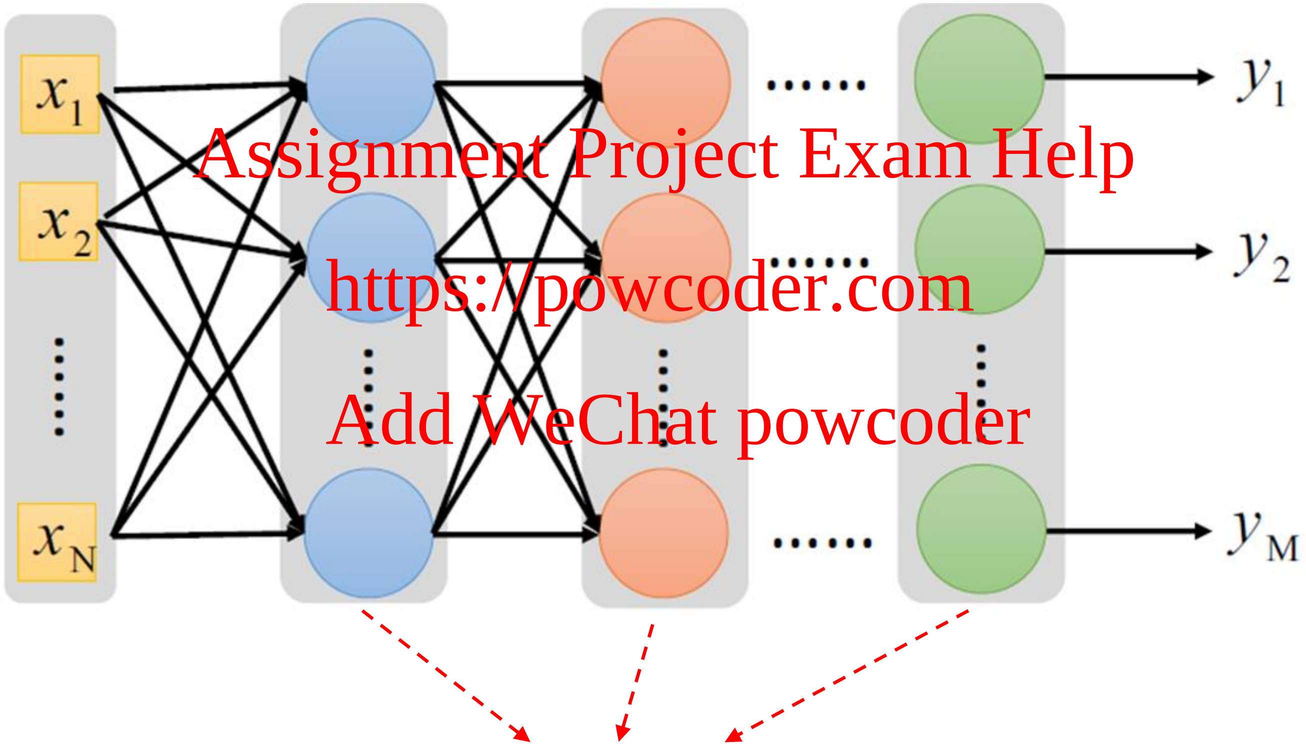
⋮



$$z_M = \boldsymbol{w}_M^{\mathrm{T}}\boldsymbol{x} + b_M = \sum_{k=1}^{d} w_{Mk}x_k + b_M \quad y_M = \frac{1}{1+e^{-z_n}}$$

Deep Learning

- **Deep Neural Network**
  - Cascading the neurons to form a neural network



Each layer consists of the linear classifier and activation function

Linear classifier

Neural Network

Linear classifiers

Parametric Approach

- (Review) Unit 3 ML basics and classification

Image

Array of **32x32x3** numbers
(3072 numbers total)

3072x1

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

10x1 10x3072

10x1

$f(\mathbf{x}, \mathbf{W})$

10 numbers giving class scores

W

parameters
(or weights)

Parametric Approach

- **(Review) Unit 3 recognition**

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

Stretch pixels into column

Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{array}{rl} \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} & \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array} \end{array}$$

What do we need now?

- **Functions to measuring the error between the output of a classifier and the given target value.**
  - Let's talk about designing error (a.k.a. loss) functions!

EBU7240

Computer Vision

- Loss functions -

Semester 1, 2021

Changjae Oh

Loss function

- **Loss function**
  - quantifies our unhappiness with the scores across the training data.

- **Type of loss function**
  - Hinge loss
  - Cross-entropy loss
  - Log likelihood loss
  - Regression loss

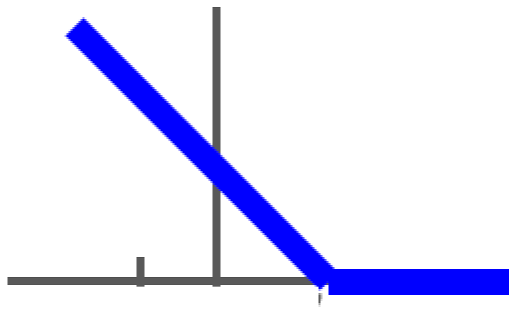
Loss Function: Hinge Loss

- **Binary hinge loss (=binary SVM loss)**

$$L_i = \max(0, 1 - y_i \cdot s)$$

$$s = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} + b$$

$y_i = \pm 1$ for positive/negative samples



- **Hinge loss (=multiclass SVM loss)**
  - C: The number of class (> 2)

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

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x}_i + \boldsymbol{b}$$

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_C^T \end{pmatrix} \qquad \boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_C \end{pmatrix}$$

$\boldsymbol{x}_i$: input data (e.g. image)
y_i: class label (integer, $1 \le y_i \le C$)

Loss Function: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(\boldsymbol{x}, \mathbf{W}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ are

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

Given a dataset of examples

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$$

$\boldsymbol{x}_i$: image

y_i: class label (integer)

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

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i(f(\boldsymbol{x}_i, \mathbf{W}), y_i)$$

Loss Function: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(\boldsymbol{x}, \mathbf{W}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ 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 (=hinge loss)

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

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

where score vector $\boldsymbol{s} = f(\boldsymbol{x}_i, \mathbf{W})$

Loss Function: Hinge Loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(\boldsymbol{x}, \mathbf{W}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ are

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Loss	2.9	0	12.9

Multiclass SVM loss (=hinge loss)

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

= 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

= 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

= 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

Loss over full dataset is average

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

Loss Function: Log Likelihood Loss

- **Log likelihood loss**

$$L_i = -\log p_j \text{ where } j \text{ satisfies } z_{ij} = 1$$

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_C \end{pmatrix}$$ probability for i^{th} image
(It is assumed to be *normalized*, i.e. $|\boldsymbol{p}| = 1$.)

$\boldsymbol{z_i}$: class label for i^{th} image
($C \times 1$ vector, $z_{ij} = 1$ when $j = y_i$ and 0 otherwise)

y_i: class label (integer, $1 \leq y_i \leq C$)

Example
Suppose i^{th} image belongs to class 2 and $C = 10$.

$$\boldsymbol{z_i} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \boldsymbol{p} = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix} \quad \Rightarrow \quad L_i = -\log 0.7$$

Loss Function: Cross-entropy Loss

- Cross-entropy loss

$$L_i = -\sum_{j=1}^{C}\left(z_{ij}\log p_j + (1 - z_{ij})\log(1 - p_j)\right)$$

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_C \end{pmatrix}$$

probability for i^{th} image
(It is assumed to be *normalized*, i.e. $|\boldsymbol{p}| = 1$.)

$\boldsymbol{z_i}$: class label for i^{th} image
($C \times 1$ vector, $z_{ij} = 1$ when $j = y_i$ and 0 otherwise)

y_i: class label (integer, $1 \leq y_i \leq C$)

Example
Suppose i^{th} image belongs to class 2 and $C = 10$.

$$\boldsymbol{z_i} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \boldsymbol{p} = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix} \quad \Rightarrow \quad L_i = -\log(1 - 0.1) - \log 0.7 - \log(1 - 0.2)$$

Softmax Activation Function

- **Softmax activation function**

scores = unnormalized log probabilities of the classes.

→ **Probability can be computed using scores as below.**

Probability of class label being k for an image $\boldsymbol{x}_i$

$$P(Y = k | X = \boldsymbol{x}_i) = p_k = \boxed{\frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}}}$$

Softmax activation function

unnormalized probabilities

$\boldsymbol{x}_i$: image

y_i: class label (integer, $1 \le y_i \le C$)

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

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x}_i + \boldsymbol{b}$$

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_C^T \end{pmatrix}$$

Softmax + Log Likelihood Loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}}\right)$$

unnormalized probabilities

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

→ L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

Softmax + Log likelihood loss:
is often called ‘softmax classifier’

Softmax + Cross-entropy Loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}}\right) - \sum_{k=1,k\neq y_i}^{C} \log\left(1 - \frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

L_i = -log(0.13)-log(1-0.87)-log(1-0.0)
= 0.89*2

Loss Function: Regression Loss

- **Regression loss**
  - Using L1 or L2 norms
  - Widely used in pixel-level prediction (e.g. image denoising)

$$L_i = |\boldsymbol{y}_i - \boldsymbol{s}_i|$$

$$L_i = (\boldsymbol{y}_i - \boldsymbol{s}_i)^2$$

$$\boldsymbol{y}_i = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \boldsymbol{s}_i = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix}$$

$$L_i = |\boldsymbol{y}_i - \boldsymbol{s}_i| = |0 - 0.1| + |1 - 0.7| + |0 - 0.2|$$

Regularization

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

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

No! 2W is also has L = 0!



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

Before:

= 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

With W twice as large:

= max(0, 2.6 - 9.8 + 1) + max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0 = 0

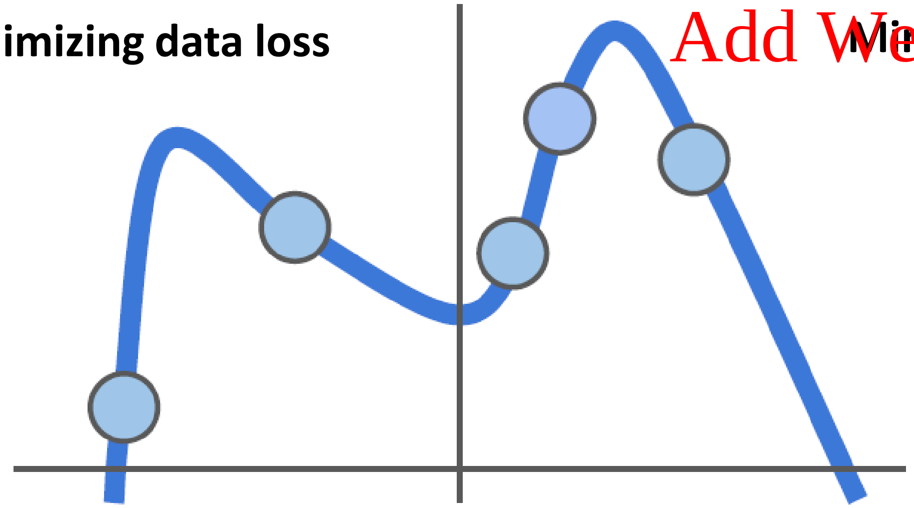
Regularization

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(\boldsymbol{x}_i, \mathbf{W}), y_i) + \lambda R(\mathbf{W})$$

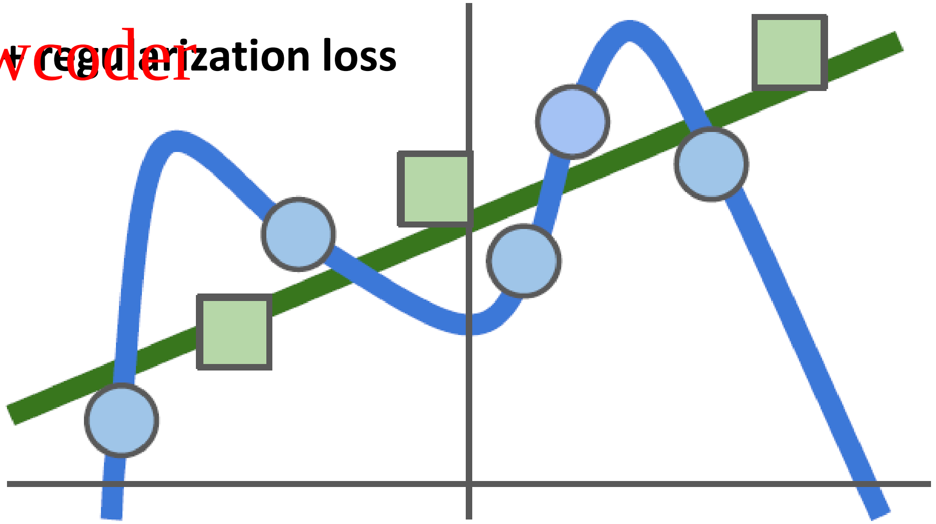
Data loss: Model predictions should match training data

Regularization: Model should be "simple" to avoid *overfitting*, so it works on test data

Minimizing data loss



Minimizing data + regularization loss



Regularization

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(\boldsymbol{x}_i, \mathbf{W}), y_i) + \lambda R(\mathbf{W})$$

λ: regularization strength (hyperparameter)

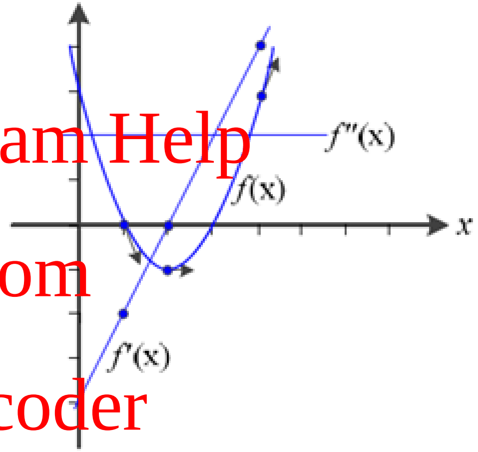
$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_C^T \end{pmatrix}$$

- L2 regularization: $R(\mathbf{W}) = \sum_{k,l} W_{k,l}^2$
- L1 regularization: $R(\mathbf{W}) = \sum_{k,l} |W_{k,l}|$
- Elastic net (L1 + L2): $R(\mathbf{W}) = \sum_{k,l} \beta W_{k,l}^2 + |W_{k,l}|$
- Max norm regularization: $|\boldsymbol{w}_j^T| < c$ **for all** j
- Dropout (will see later)
- Batch normalization, stochastic depth (will see later)

Optimization: Gradient Descent

- **Gradient Descent**
  - The simplest approach to minimizing a loss function

$$\mathbf{W}^{\mathrm{T}+1} = \mathbf{W}^{\mathrm{T}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{\mathrm{T}}}$$



$y = f(x) = x^2 - 4x + 3$

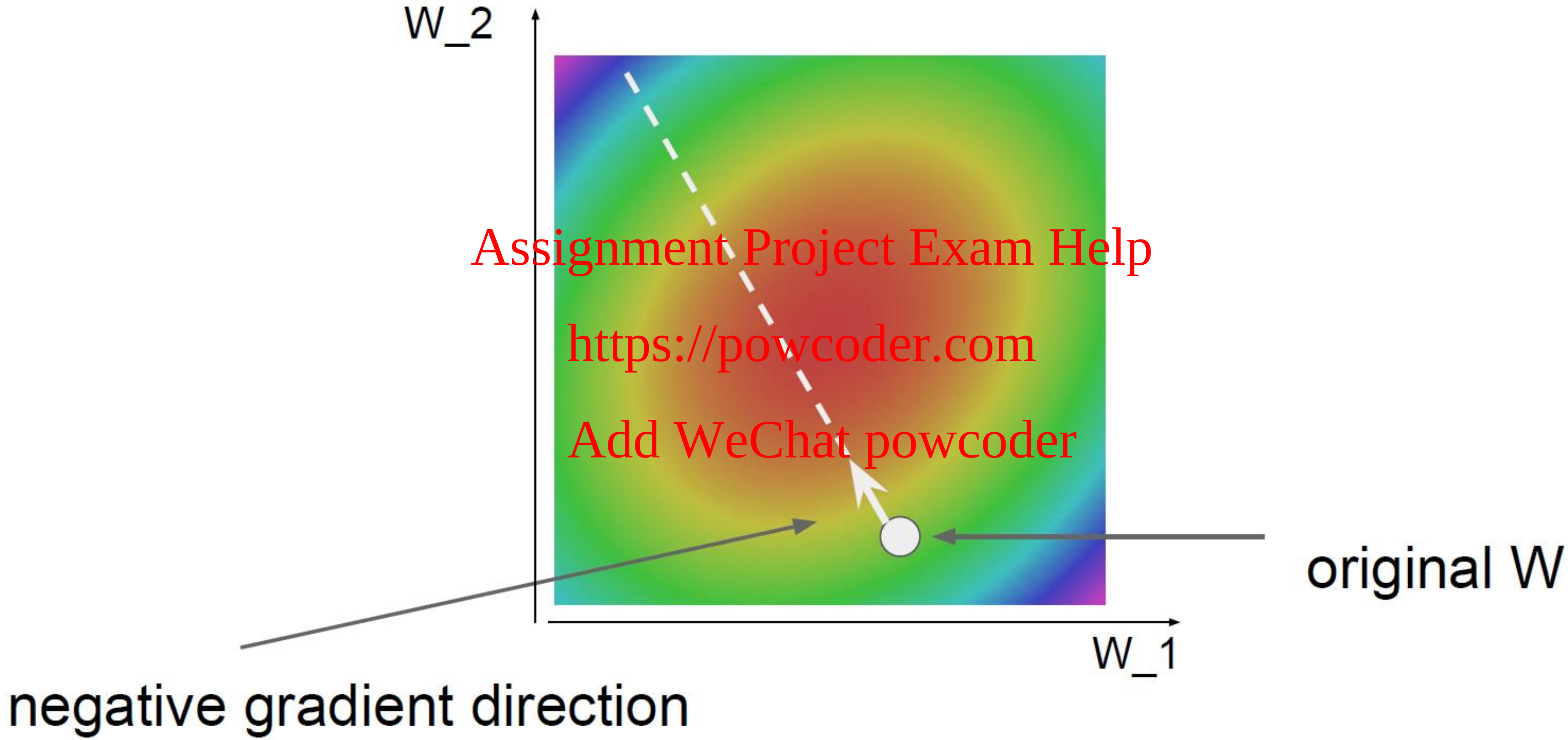
$y' = f'(x) = 2x - 4$

  - α: step size (a.k.a. learning rate)


```
# Vanilla Gradient Descent

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

Optimization: Gradient Descent



Optimization: Stochastic Gradient Descent (SGD)

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(\boldsymbol{x}_i, \mathbf{W}), y_i) + \lambda R(\mathbf{W})$$

Full sum is too expensive when N is large!

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial L_i(f(\boldsymbol{x}_i, \mathbf{W}), y_i)}{\partial \mathbf{W}} + \lambda \frac{\partial R(\mathbf{W})}{\partial \mathbf{W}}$$

Instead, approximating sum using a **minibatch** of 32 / 64 / 128/ 256 examples is 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
```

EBU7240

Computer Vision

- Backpropagation-

Semester 1, 2021

Changjae Oh

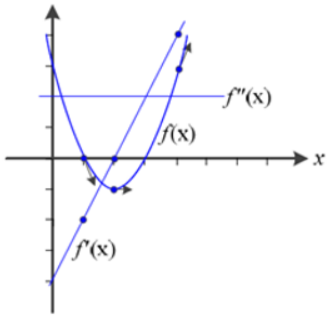
Backpropagation

- **A widely used algorithm for training feedforward neural networks.**
- **A way of computing gradients of expressions through recursive application of chain rule.**
  - Backpropagation computes the gradient of the loss function with respect to the weights of the network (model) for a single input–output example.

- **Gradient Descent**
  - The simplest approach to minimizing a loss function

$$\mathbf{W}^{\mathrm{T}+1} = \mathbf{W}^{\mathrm{T}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{\mathrm{T}}}$$

  - α: step size (a.k.a. learning rate)



$y = f(x) = x^2 - 4x + 3$

$y' = f'(x) = 2x - 4$

Derivative

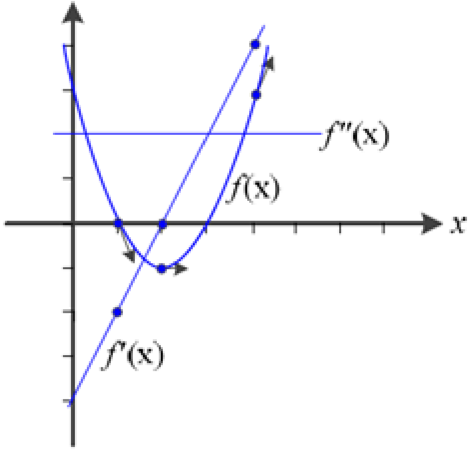
- **Optimization using derivative**
  - 1st order derivative

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

  - $f'(x)$: The slope of the function, indicating the direction in which the value increases

→ The minima of the objective function may exist in the direction of $-f'(x)$.

→ Gradient descent algorithm: $-f'(x)$



$y = f(x) = x^2 - 4x + 3$

$y' = f'(x) = 2x - 4$

$$\mathbf{W}^{\mathrm{T}+1} = \mathbf{W}^{\mathrm{T}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{\mathrm{T}}}$$

Derivative

- **Partial derivative**
  - Derivatives of functions with multiple variables
  - Gradient: the vector of the partial derivative

Ex) $\nabla f, \frac{\partial f}{\partial \mathbf{x}}, \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)^{\mathrm{T}}$

$$f(\mathbf{x}) = f(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

$$\nabla f = f'(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)^{\mathrm{T}} = (2x_1^5 - 8.4x_1^3 + 8x_1 + x_2, 16x_2^3 - 8x_2 + x_1)^{\mathrm{T}}$$

Derivative

- **Jacobian matrix**
  - 1^{st} order partial derivative matrix for $\mathbf{f}: \mathbb{R}^d \mapsto \mathbb{R}^m$

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_d} \end{pmatrix}$$

Ex) $\mathbf{f}: \mathbb{R}^2 \mapsto \mathbb{R}^3 \quad \mathbf{f}(\mathbf{x}) = (2x_1 + x_2^2, -x_1^2 + 3x_2, 4x_1x_2)^{\mathrm{T}}$

$$\mathbf{J} = \begin{pmatrix} 2 & 2x_2 \\ -2x_1 & 3 \\ 4x_2 & 4x_1 \end{pmatrix} \qquad \mathbf{J}|_{(2,1)^{\mathrm{T}}} = \begin{pmatrix} 2 & 2 \\ -4 & 3 \\ 4 & 8 \end{pmatrix}$$

- **Hessian matrix**

  - 2^{nd} order partial derivative matrix

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 x_1} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2 x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n x_n} \end{pmatrix}$$

Ex) $f(\mathbf{x}) = f(x_1, x_2)$

$$= \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

$$\mathbf{H} = \begin{pmatrix} 10x_1^4 - 25.2x_1^2 + 8 & 1 \\ 1 & 48x_2^2 - 8 \end{pmatrix}$$

$$\mathbf{H}|_{(0,1)^{\mathrm{T}}} = \begin{pmatrix} 8 & 1 \\ 1 & 40 \end{pmatrix}$$

Derivative

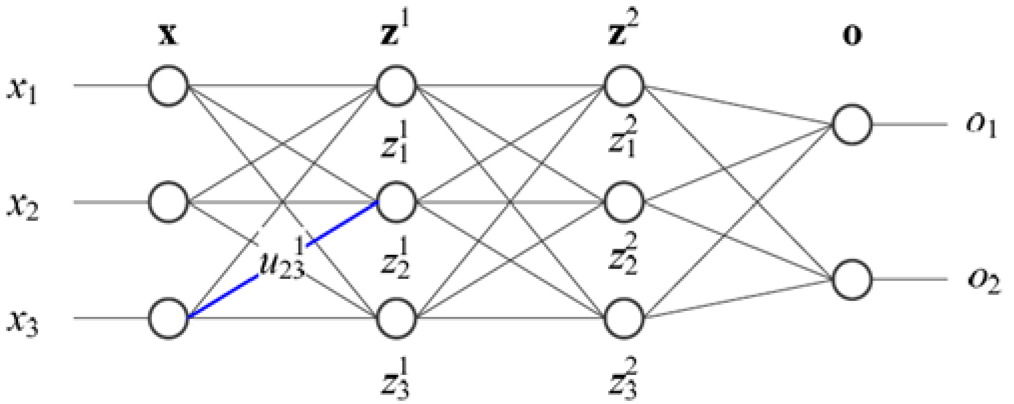
- **Chain rule**

$$f(x) = g(h(x)) \longrightarrow f'(x) = g'\big(h(x)\big)h'(x)$$

$$f(x) = g(h(i(x))) \qquad f'(x) = g'\big(h(i(x))\big)h'(i(x))i'(x)$$

Ex) $f(x) = 3(2x^2 - 1)^2 - 2(2x^2 - 1) + 5 \qquad h(x) = 2x^2 - 1$

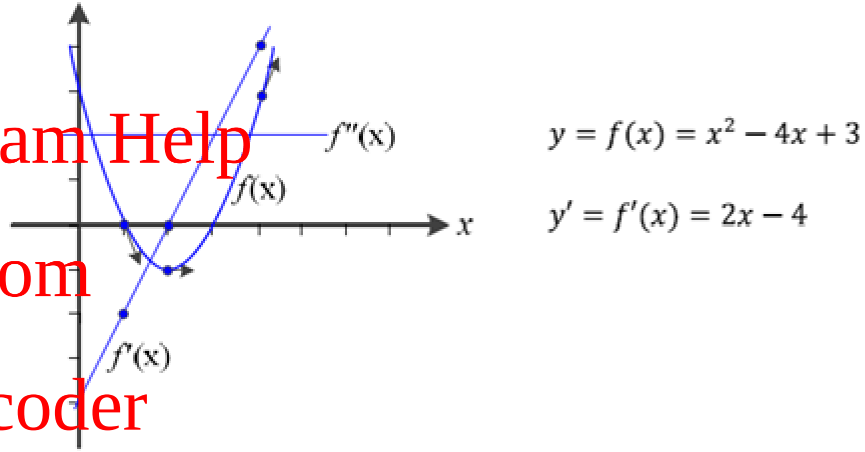
$$\longrightarrow \quad f'(x) = \underbrace{(3 * 2(2x^2 - 1) - 2)}_{g'(h(x))}\underbrace{(2 * 2x)}_{h'(x)} = 48x^3 - 32x$$



Why are we talking about derivatives?

- **Gradient Descent**
  - The simplest approach to minimizing a loss function

$$\mathbf{W}^{T+1} = \mathbf{W}^{T} - \alpha \frac{\partial L}{\partial \mathbf{W}^{T}}$$



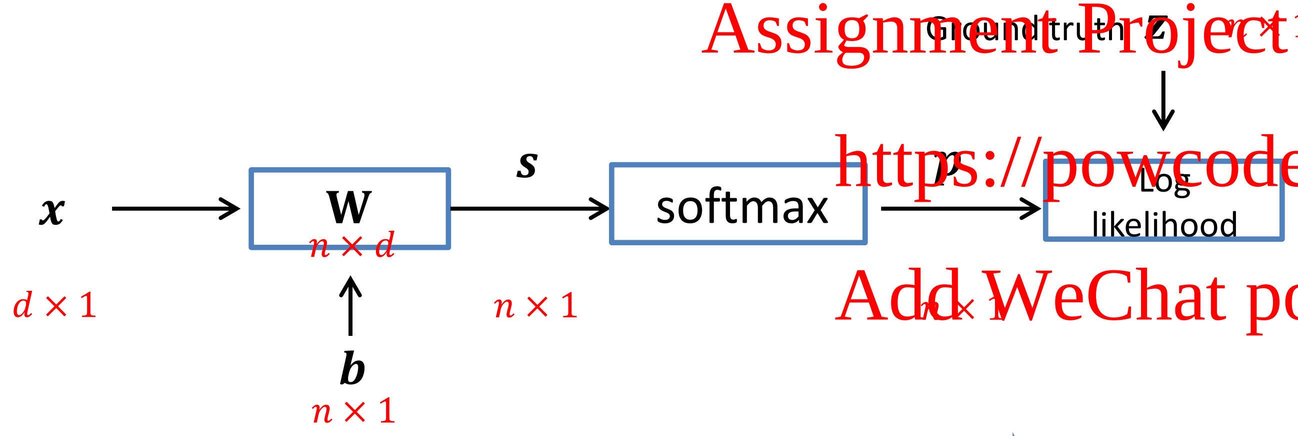
  - α: step size (a.k.a. learning rate)


```
# Vanilla Gradient Descent

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

Derivative

- **Example) Applying chain rule to single-layer perceptron**
  - Example of composite function
  - Back-propagation: use the chain rule to compute $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \boldsymbol{b}}$



$$\frac{\partial L}{\partial \boldsymbol{p}} \Rightarrow \frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} \Rightarrow \frac{\partial L}{\partial \boldsymbol{w}_j} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j} \frac{\partial L}{\partial \boldsymbol{s}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \boldsymbol{w}_1} \quad \frac{\partial L}{\partial \boldsymbol{w}_2} \quad \cdots \quad \frac{\partial L}{\partial \boldsymbol{w}_n} \right)^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} \frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

Analytic Gradient: Linear Equation

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \quad \longleftrightarrow \quad \begin{aligned} s_1 &= \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 \\ s_2 &= \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} + b_2 \\ &\vdots \\ s_n &= \boldsymbol{w}_n^{\mathrm{T}}\boldsymbol{x} + b_n \end{aligned}$$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\frac{\partial s_1}{\partial \boldsymbol{w}_1} = \boldsymbol{x}$$

$$\frac{\partial s_2}{\partial \boldsymbol{w}_1} = \mathbf{0}$$

$$\vdots$$

$$\frac{\partial s_n}{\partial \boldsymbol{w}_1} = \mathbf{0}$$

$$\Rightarrow \quad \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_1} = [\boldsymbol{x}\ \mathbf{0}\ \mathbf{0}\ \cdots \mathbf{0}] \in \Re^{d\times n}$$

j[th] column

$$\boxed{\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j} = [\mathbf{0}\ \mathbf{0}\ \boldsymbol{x}\ \cdots \mathbf{0}] \in \Re^{d\times n}}$$

$$\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} = \mathbf{I} \in \Re^{n\times n}$$

Analytic Gradient: Linear Equation

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \quad \longleftrightarrow \quad \begin{aligned} s_1 &= \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 \\ s_2 &= \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} + b_2 \\ &\vdots \\ s_n &= \boldsymbol{w}_n^{\mathrm{T}}\boldsymbol{x} + b_n \end{aligned}$$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\frac{\partial s_1}{\partial \boldsymbol{x}} = \boldsymbol{w}_1$$

$$\frac{\partial s_2}{\partial \boldsymbol{x}} = \boldsymbol{w}_2$$

$$\vdots$$

$$\frac{\partial s_n}{\partial \boldsymbol{x}} = \boldsymbol{w}_n$$

$$\Rightarrow \quad \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \cdots \boldsymbol{w}_n] = \mathbf{W}^{\mathrm{T}} \in \Re^{d\times n}$$

Analytic Gradient: Sigmoid Function

- **Sigmoid function**

For a scalar x

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \rightarrow \quad \frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{1+e^{-x}} \frac{1}{1+e^{-x}} = (1-\sigma(x))\sigma(x)$$

Similarly, for a vector $\boldsymbol{s} \in \Re^{n\times 1}$

$$\boldsymbol{p} = \sigma(\boldsymbol{s}) = \frac{1}{1+e^{-\boldsymbol{s}}} \quad \rightarrow \quad \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = diag\big((1-\sigma(s_j))\sigma(s_j)\big) = \begin{bmatrix} (1-\sigma(s_1))\sigma(s_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (1-\sigma(s_n))\sigma(s_n) \end{bmatrix}$$

for $j = 1, \ldots, n$

Analytic Gradient: Softmax Activation Function

- **Softmax function**

$$p_k = \frac{e^{s_k}}{\sum_{j=1}^{n} e^{s_j}} \quad \Rightarrow \quad \boldsymbol{p} = \frac{e^{\boldsymbol{s}}}{\sum_{j=1}^{n} e^{s_j}}$$ in vector form

score function $\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$

probability $\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}$

- **1st order derivative of softmax function**

$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \frac{diag(e^{\boldsymbol{s}}) \cdot \sum e^{s_j} - e^{\boldsymbol{s}}(e^{\boldsymbol{s}})^{\mathrm{T}}}{(\sum e^{s_j})^2} = \frac{1}{(\sum e^{s_j})^2} \left\{ \begin{pmatrix} e^{s_1}\sum e^{s_j} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{s_n}\sum e^{s_j} \end{pmatrix} - \begin{pmatrix} e^{s_1}e^{s_1} & \cdots & e^{s_1}e^{s_n} \\ \vdots & \ddots & \vdots \\ e^{s_n}e^{s_1} & \cdots & e^{s_n}e^{s_n} \end{pmatrix} \right\}$$

Analytic Gradient: Softmax Activation Function

$$\mathbf{D} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \frac{diag(e^{s}) \cdot \sum e^{s_j} - e^{s}(e^{s})^{\mathrm{T}}}{(\sum e^{s_j})^2} = \frac{1}{(\sum e^{s_j})^2}\left\{\begin{pmatrix} e^{s_1}\sum e^{s_j} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{s_n}\sum e^{s_j} \end{pmatrix} - \begin{pmatrix} e^{s_1}e^{s_1} & \cdots & e^{s_1}e^{s_n} \\ \vdots & \ddots & \vdots \\ e^{s_n}e^{s_1} & \cdots & e^{s_n}e^{s_n} \end{pmatrix}\right\}$$

For $a = b$

$$\frac{e^{s_a}(\sum e^{s_j} - e^{s_a})}{(\sum e^{s_j})^2} = p_a(1 - p_a)$$

For $a \neq b$

$$-\frac{e^{s_a}e^{s_b}}{(\sum e^{s_j})^2} = -p_a p_b$$

$$\Rightarrow \quad D_{ab} = p_a(\delta_{ab} - p_b)$$

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

Analytic Gradient: Hinge Loss

For simplicity of notation, the index of training image *i* is omitted here

- **1st order derivative of binary hinge loss**

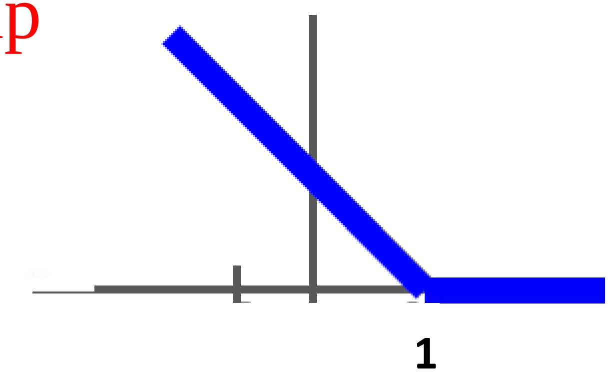
$$s = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$$

$y = \pm 1$ for positive/negative samples

$$L = \max(0, 1 - y \cdot s)$$

$$= \begin{cases} 1 - y \cdot s & \text{if } 1 - y \cdot s > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\Rightarrow \quad \frac{\partial L}{\partial s} = \begin{cases} -y & \text{if } 1 - y \cdot s > 0 \\ 0 & \text{otherwise} \end{cases}$$



Analytic Gradient: Hinge Loss

For simplicity of notation, the index of training image *i* is omitted here

- **1st order derivative of hinge loss**

$$\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$$

$$L = \sum_{j=1, j \neq y}^{n} \max(0, s_j - s_y + 1)$$

x: image

y: class label (integer, $1 \leq y \leq n$)

$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} \qquad \boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$$

$$\frac{\partial L}{\partial s_y} = -\sum_{j=1, j \neq y}^{n} 1(s_j - s_y + 1 > 0) \qquad \text{for } j = y$$

$$1(F) = \begin{cases} 1 & if\ F\ is\ true \\ 0 & otherwise \end{cases}$$

$$\frac{\partial L}{\partial s_j} = 1(s_j - s_y + 1 > 0) \qquad \text{for } j \neq y$$

Analytic Gradient: Log Likelihood Loss

For simplicity of notation, the index of training image *i* is omitted here

$L = -\log p_y$ where y satisfies $z_y = 1$

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}$$

probability for i^{th} image
(It is assumed to be *normalized*, i.e. $|\boldsymbol{p}| = 1$.)

$$\Rightarrow \quad \frac{\partial L}{\partial \boldsymbol{p}} = -\begin{array}{c} \\ \\ \\ \end{array}\begin{bmatrix} 0 \\ \vdots \\ 1/p_y \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \\ \\ \leftarrow y^{th} \text{ row} \\ \\ \\ \end{array}$$

y: class label for i^{th} image ($1 \le y \le n$)

$\boldsymbol{z}$: class probability for i^{th} image
$\boldsymbol{z} = (z_1\ z_2 \dots z_n)^T$, $z_y = 1$ and $z_{k \neq y} = 0$

Example
Suppose i^{th} image belongs to class 2 and $n = 10$. → $y = 2$

$$\boldsymbol{z} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \boldsymbol{p} = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix} \qquad \Rightarrow \qquad \frac{\partial L}{\partial \boldsymbol{p}} = -\begin{bmatrix} 0 \\ 1/0.7 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Analytic Gradient: Cross-entropy Loss

For simplicity of notation, the index of training image *i* is omitted here

$$L = -\sum_{j=1}^{n}\left(z_j \log p_j + (1 - z_j)\log(1 - p_j)\right)$$

$$\boldsymbol{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}$$

probability for i^{th} image
(It is assumed to be *normalized*, i.e. $|\boldsymbol{p}| = 1$.)

$$\Rightarrow \quad \frac{\partial L}{\partial \boldsymbol{p}} = -\begin{array}{c} \begin{bmatrix} 1/(1 - p_1) \\ \vdots \\ 1/p_y \\ \vdots \\ 1/(1 - p_n) \end{bmatrix} \end{array} \begin{array}{l} \\ \\ \leftarrow y^{th} \text{ row} \\ \\ \\ \end{array}$$

y: class label for i^{th} image ($1 \le y \le n$)

$\boldsymbol{z}$: class probability for i^{th} image

$\boldsymbol{z} = (z_1\ z_2 \ldots\ z_n)^T$, $z_y = 1$ and $z_{k \neq y} = 0$

Example

Suppose i^{th} image belongs to class 2 and $n = 10$. → $y = 2$

$$\boldsymbol{z} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \boldsymbol{p} = \begin{pmatrix} 0.1 \\ 0.7 \\ 0 \\ \vdots \\ 0.2 \end{pmatrix} \qquad \Rightarrow \qquad \frac{\partial L}{\partial \boldsymbol{p}} = -\begin{bmatrix} 1/(1 - 0.1) \\ 1/0.7 \\ 0 \\ \vdots \\ 1/(1 - 0.2) \end{bmatrix}$$

Analytic Gradient: Regression Loss

For simplicity of notation, the index of training image *i* is omitted here

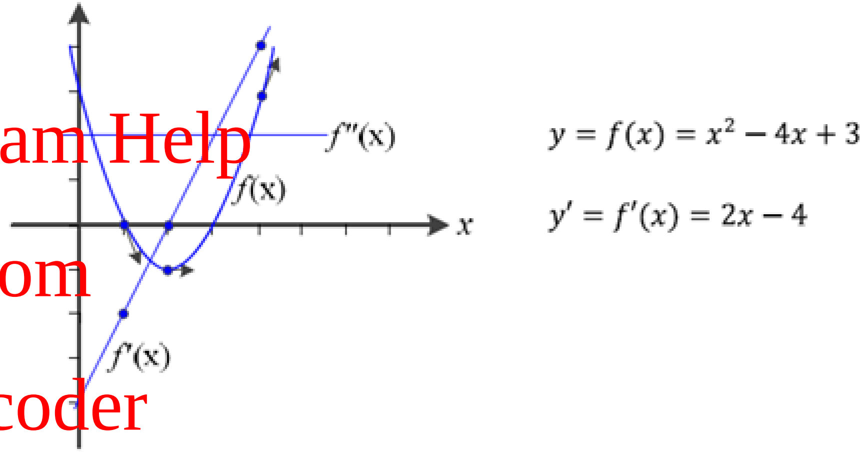
- Regression loss

$$L = (\boldsymbol{y} - \boldsymbol{s})^2 \quad \Rightarrow \quad \frac{\partial L}{\partial \boldsymbol{s}} = -2(\boldsymbol{y} - \boldsymbol{s}) \qquad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \qquad \boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$$

Why are we talking about derivatives?

- **Gradient Descent**
  - The simplest approach to minimizing a loss function

$$\mathbf{W}^{\mathrm{T}+1} = \mathbf{W}^{\mathrm{T}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{\mathrm{T}}}$$



  - α: step size (a.k.a. learning rate)


```
# Vanilla Gradient Descent

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