CS 577: NATURAL LANGUAGE PROCESSING

Abulhair Saparov

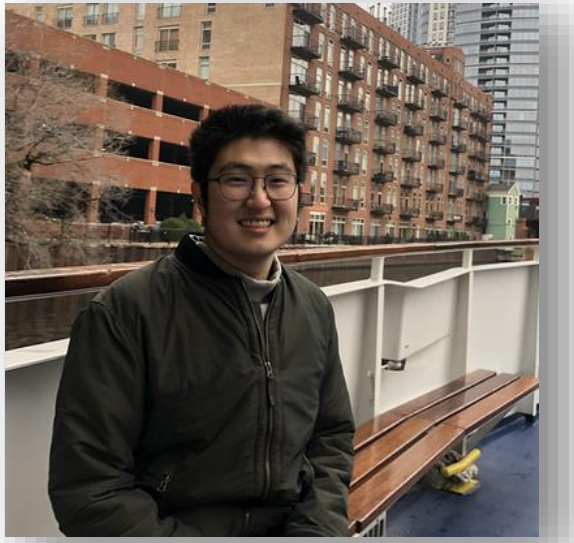
Lecture 2: Text Classification

TEACHING ASSISTANT: NATHANIEL GETACHEW

Office hours:
Wednesdays 4:30-5:30pm
DSAI B061

TEACHING ASSISTANT: YUNXIN SUN

Office hours:
- Thursdays 4:00-5:00pm
- DSAI B047



COURSE WEBSITE

asaparov.org/cs577_fall2025/

- (link will be shared in Brightspace and Ed shortly)
- Lecture slides will be uploaded there.
  - I will make an effort to upload them before class
- Contains information on office hours, grading, schedule, course policies.
  - Schedule is subject to change throughout semester.

TASK: SPAM DETECTION

Dear customer,

Your Subscription was successfully completed today, and your account will be credited with $400.99.Within the next 24 hours, the transaction will show up in your account statement. Please get in touch with our billing department right once if you think this transaction was not authorized or if you want to terminate your membership.

Customer Id	SDAF2354W76TER
Invoice Number	9187248935EW
Customer-Care No	+1 (951)-(262)-(3062)

Dear Abulhair,

Thanks for your order. And can we just say — excellent choice. Read on to see all the details.

Estimated delivery window Dec 19, 2024 – Jan 13, 2025.

Reflect on your own thinking: What are you looking at to determine whether the email is spam or not?

TASK: SPAM DETECTION

Dear customer,

Your Subscription was successfully completed today, and your account will be credited with $400.99.Within the next 24 hours, the transaction will show up in your account statement. Please get in touch with our billing department right once if you think this transaction was not authorized or if you want to terminate your membership.

Customer Id	SDAF2354W76TER
Invoice Number	9187248935EW
Customer-Care No	+1 (951)-(262)-(3062)

SPAM

Dear Abulhair,

Thanks for your order. And can we just say — excellent choice. Read on to see all the details.

Estimated delivery window Dec 19, 2024 – Jan 13, 2025.

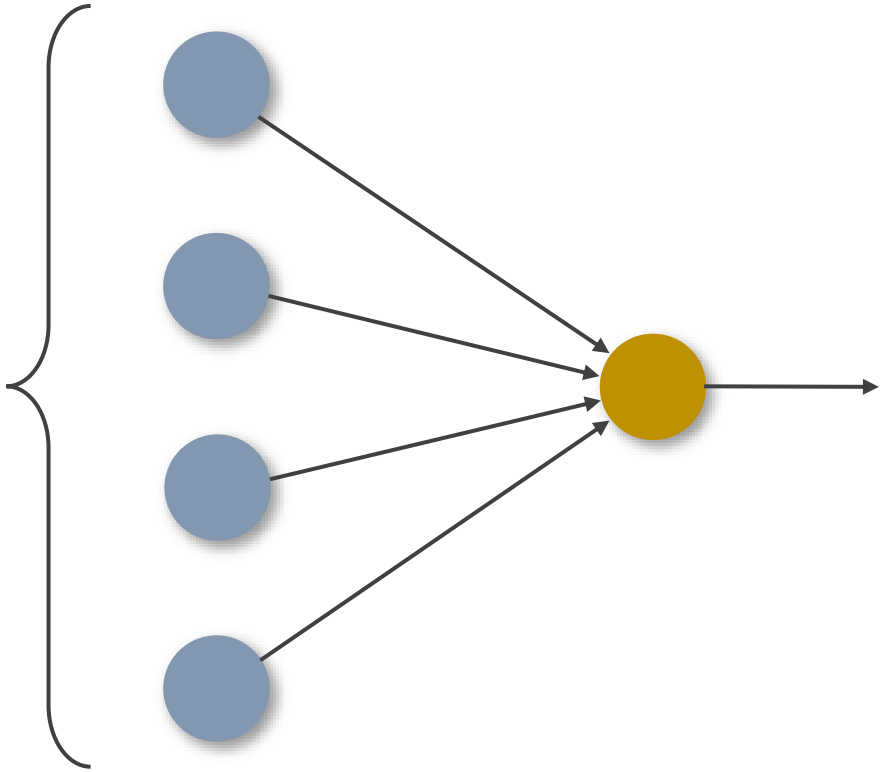
NOT SPAM

HOW TO SOLVE THIS TASK?

- Suppose we have a large labeled dataset:
  - Thousands of emails, each labeled as SPAM or NOT SPAM.
- This is a supervised learning problem.
  - If we had the email data, but no labels, then it would be unsupervised.
- This is a binary classification problem since the output space is a discrete set of 2 elements.
- How do we learn a function whose input is a new email,
- And whose (hopefully accurate) output is SPAM or NOT SPAM?

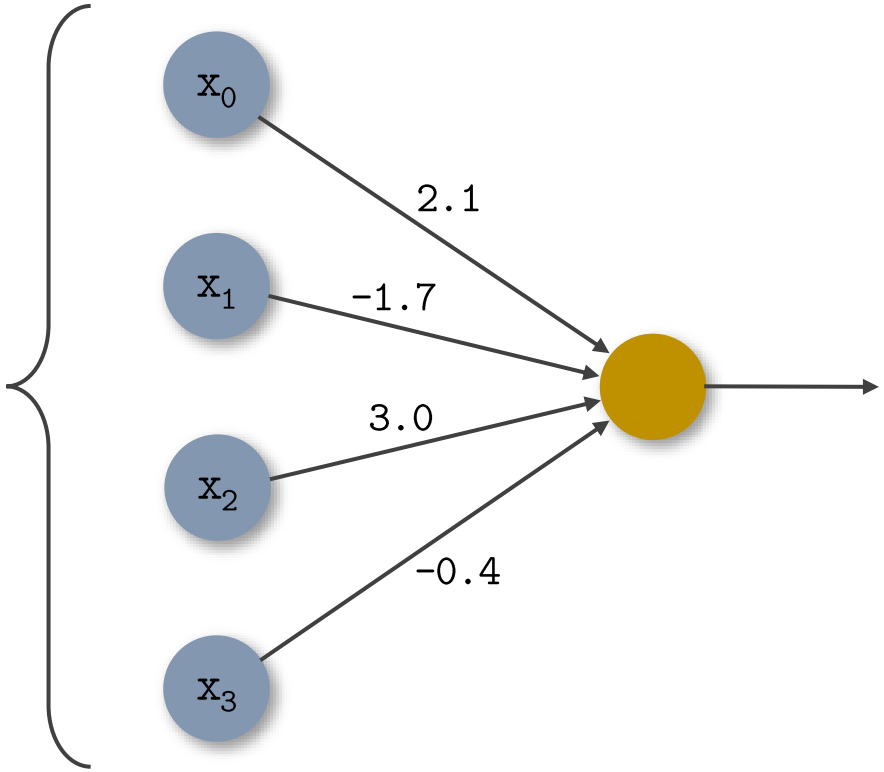
SIMPLE METHOD: PERCEPTRON

input neurons:
a real value is
assigned to each



SIMPLE METHOD: PERCEPTRON

input neurons:
a real value is
assigned to each

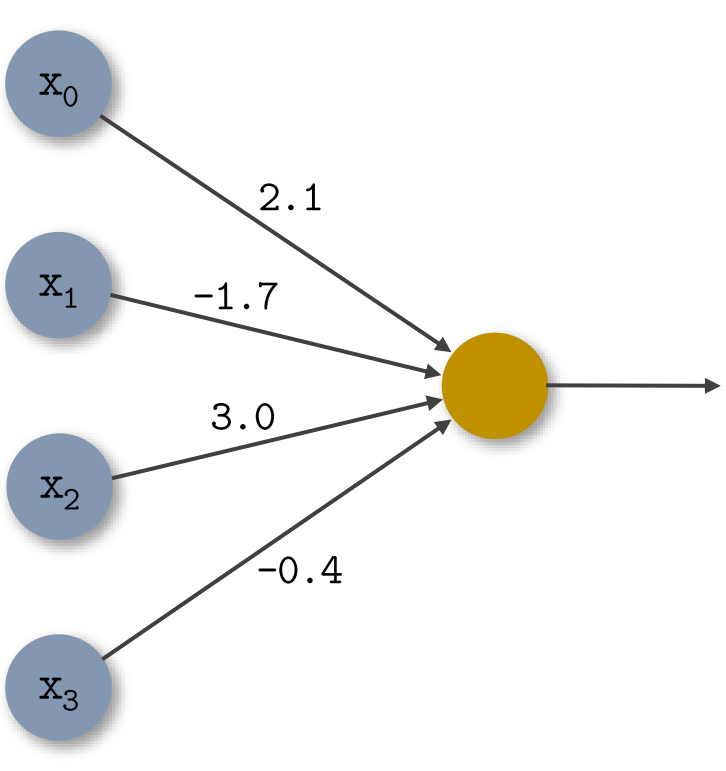


single output neuron:

$$\text{sign}\{2.1(x_0)-1.7(x_1)+3.0(x_2)-0.4(x_3)\}$$

SIMPLE METHOD: PERCEPTRON

input neurons:
a real value is
assigned to each



Let:

$\mathbf{x} = [x_0, x_1, x_2, x_3]$

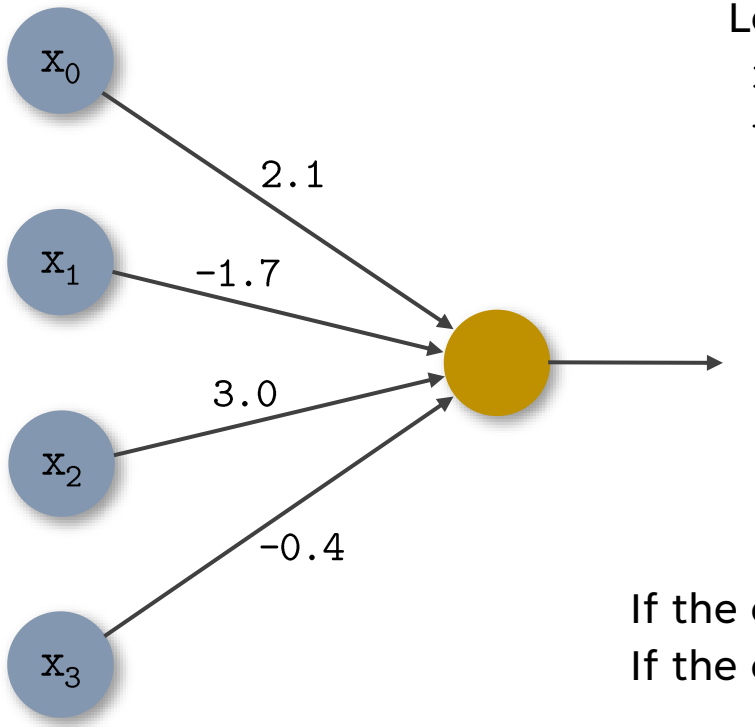
$\mathbf{w} = [2.1, -1.7, 3.0, -0.4]$

single output neuron:

$\text{sign}\{2.1(x_0)-1.7(x_1)+3.0(x_2)-0.4(x_3)\}$

SIMPLE METHOD: PERCEPTRON

input neurons:
a real value is
assigned to each



Let:

$\mathbf{x} = [x_0, x_1, x_2, x_3]$

$\mathbf{w} = [2.1, -1.7, 3.0, -0.4]$

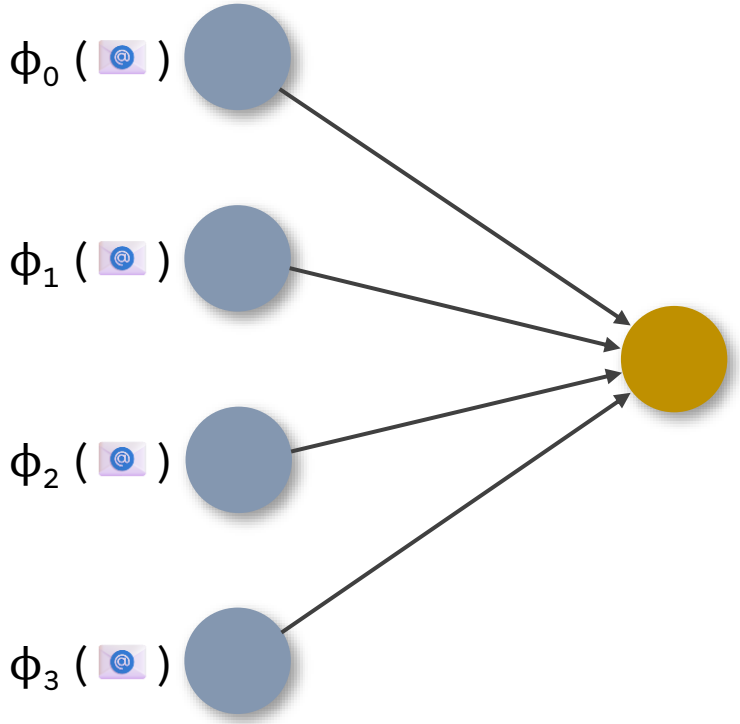
single output neuron:

$\text{sign}\{\mathbf{w}^T\mathbf{x}\}$

If the output is +1, we can say the input is SPAM.
If the output is −1, the input is NOT SPAM.

WHAT ABOUT THE INPUT?

- How do we convert an email into a real-valued vector?
- One idea is to use feature functions:
  - Let's say we have feature functions ϕ_0, ϕ_1, ϕ_2, ϕ_3.
  - Each feature function converts a sequence of text into a real number:
  - ϕ_i: @ -> $\mathbb{R}$
- For spam detection, some ideas for feature functions:
  - ϕ_i(@) = number of spelling mistakes
  - ϕ_i(@) = number of phone numbers
  - ϕ_i(@) = does the user's name appear?



$= \text{sign}\{w^T \Phi(\ @\ )\}$

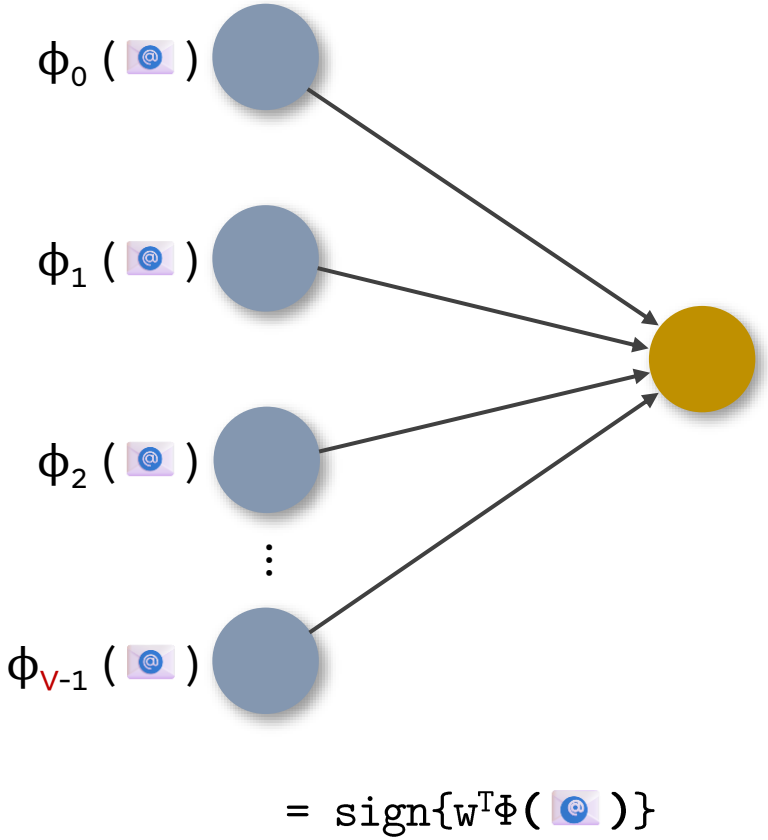
WHAT ABOUT THE INPUT?

- How do we convert an email into a real-valued vector?
- One well-known set of feature functions is bag-of-words.
  - We have V feature functions where V is the size of vocabulary.
  - ϕ_i counts how many times the i-th vocab item appears in the input.
  - This effectively discards all word order information.

  - This can work well in many cases, but not all cases:
  - E.g., suppose task is question answering and we are given a math word problem to solve.

    "Alice gave 10 apples to Bob."

    "Bob gave 10 apples to Alice."
  - These two sentences have the same bag-of-words representation.



WHAT ABOUT THE INPUT?

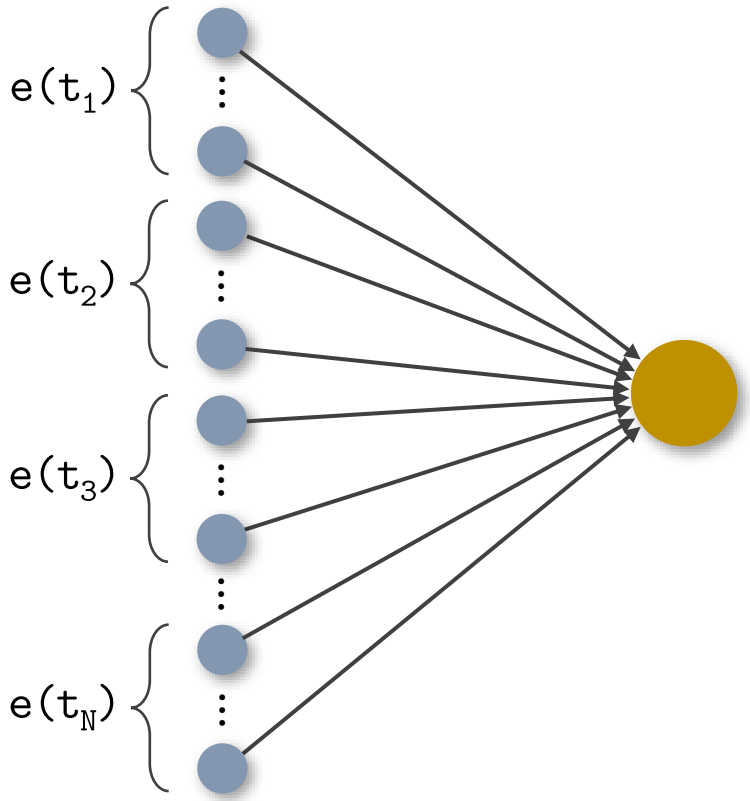
- How do we convert an email into a real-valued vector?
- Another idea is to use an embedding:
  - A word embedding is a function that converts a word into a vector.
  - Suppose the email consists of the words:

$$t_1,\ t_2,\ t_3,\ \ldots\ ,\ t_N$$

  - Then, an embedding function e can map each into an E-dimensional real-valued vector, where E is the embedding dimension.

$$e(t_1),\ e(t_2),\ e(t_3),\ \ldots\ ,\ e(t_N)$$

- An example of a simple embedding is one-hot embedding.
  - E = vocabulary size
  - $e(t_i)$ = 1 at the index corresponding to the word t_i
    = 0 everywhere else.



$$= \text{sign}\{w^T e(t)\}$$

TRAINING

- How do we learn the weights **w**?
- We can pose the learning problem as an optimization problem.
- Let $x_1, x_2, \ldots, x_N$ be the set of emails in the training dataset.
- Let $y_1, y_2, \ldots, y_N$ be the set of labels,

  where $y_i = +1$ if the i-th email is SPAM,

  and $y_i = -1$ if the i-th email is NOT SPAM.
- To form an optimization problem, we need a loss function.
  - The loss function measures the "distance" from the current model's predictions and the ground truth.
- One common loss function is mean-squared error (MSE):

$$f_w(x_i) = w^T \Phi(x_i)$$

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f_w(x_i))^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T \Phi(x_i))^2$$

TRAINING

- Now we have an objective function for the perceptron:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - w^T\Phi(x_i))^2$$

- We want to find the value of w that minimizes $L(w)$.
- There are many optimization algorithms.
- Gradient descent:

  Start with an initial guess for w.

  Repeat:

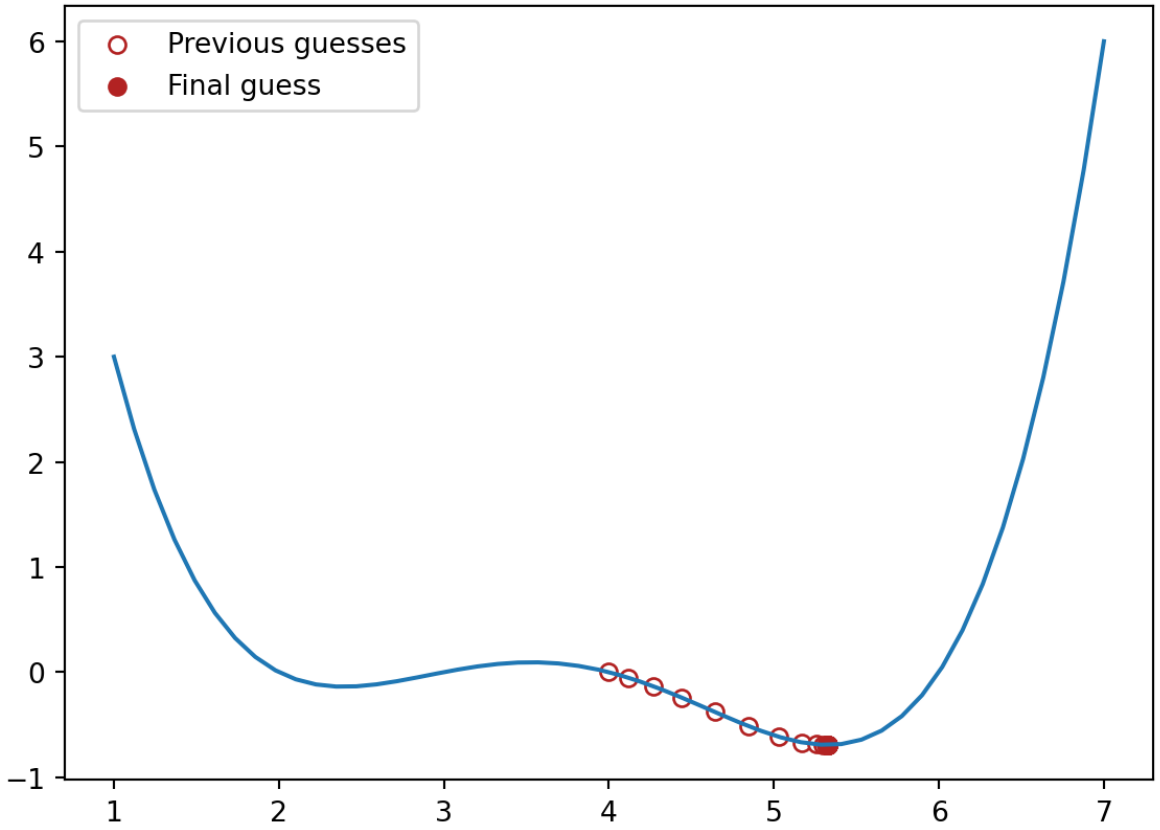
  Compute the gradient of the loss: $\nabla_w L(w)$

  Take a step in the direction of the negative gradient: $w_{new} \leftarrow w - \eta \cdot \nabla_w L(w)$

  } One full pass over the training data is called an epoch.

$$\nabla_w L(w) = \frac{1}{N}\sum_{i=1}^{N}\nabla_w(y_i - w^T\Phi(x_i))^2 = -\frac{2}{N}\cdot\sum_{i=1}^{N}\Phi(x_i)(y_i - w^T\Phi(x_i))$$

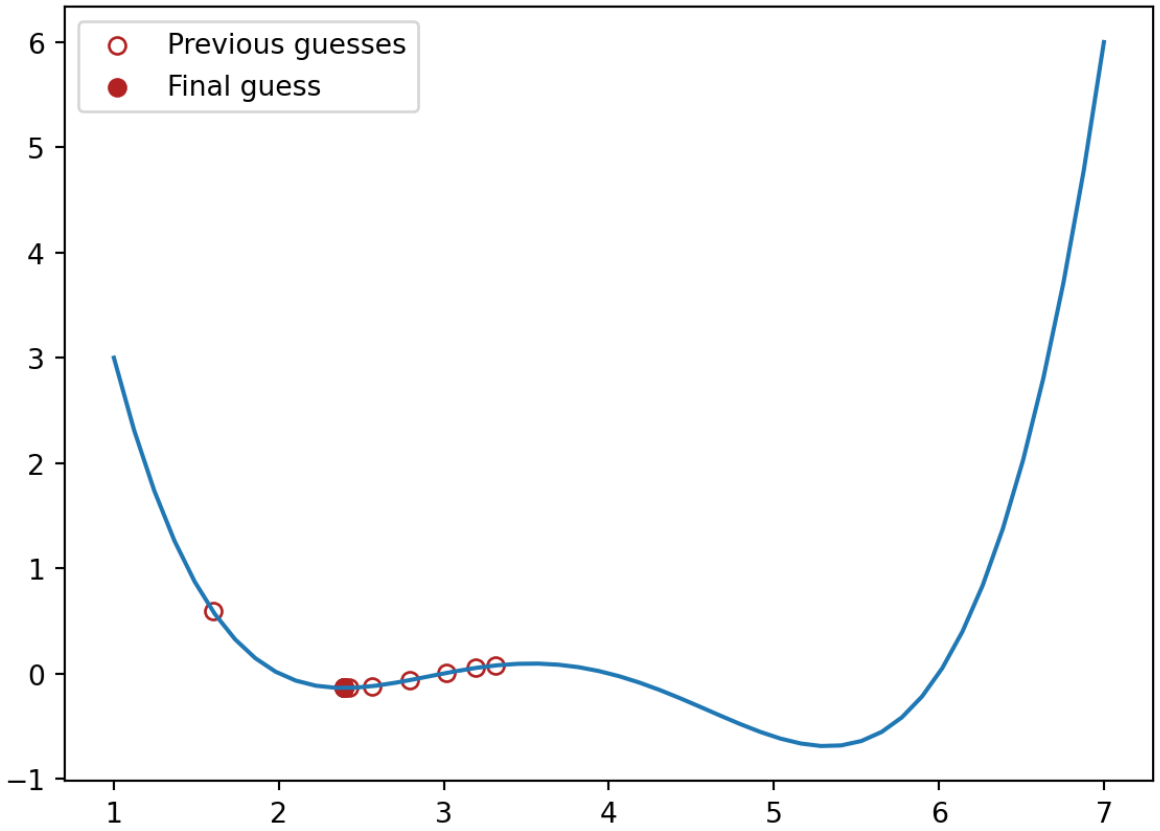
GRADIENT DESCENT



The learning rate η must be carefully set.

[ds100.org/course-notes-su2]

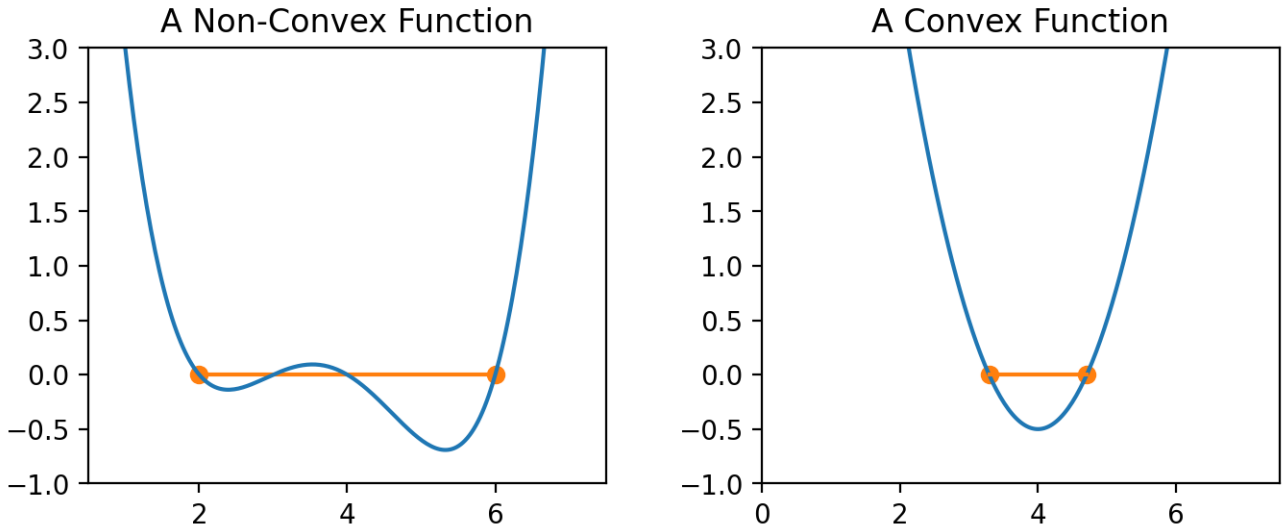
GRADIENT DESCENT



Gradient descent may find a local minimum.

[ds100.org/course-notes-su2]

CONVEXITY



Many real-world functions are not convex.
For example, neural network training objectives are non-convex.

[ds100.org/course-notes-su2]

TRAINING

- So now we have all the ingredients to train a perceptron spam classifier.
- What if the training set is very large? (i.e., we have > millions of emails)
  - The training set will not fit in memory.
- Computing the gradient requires iterating over the full dataset.

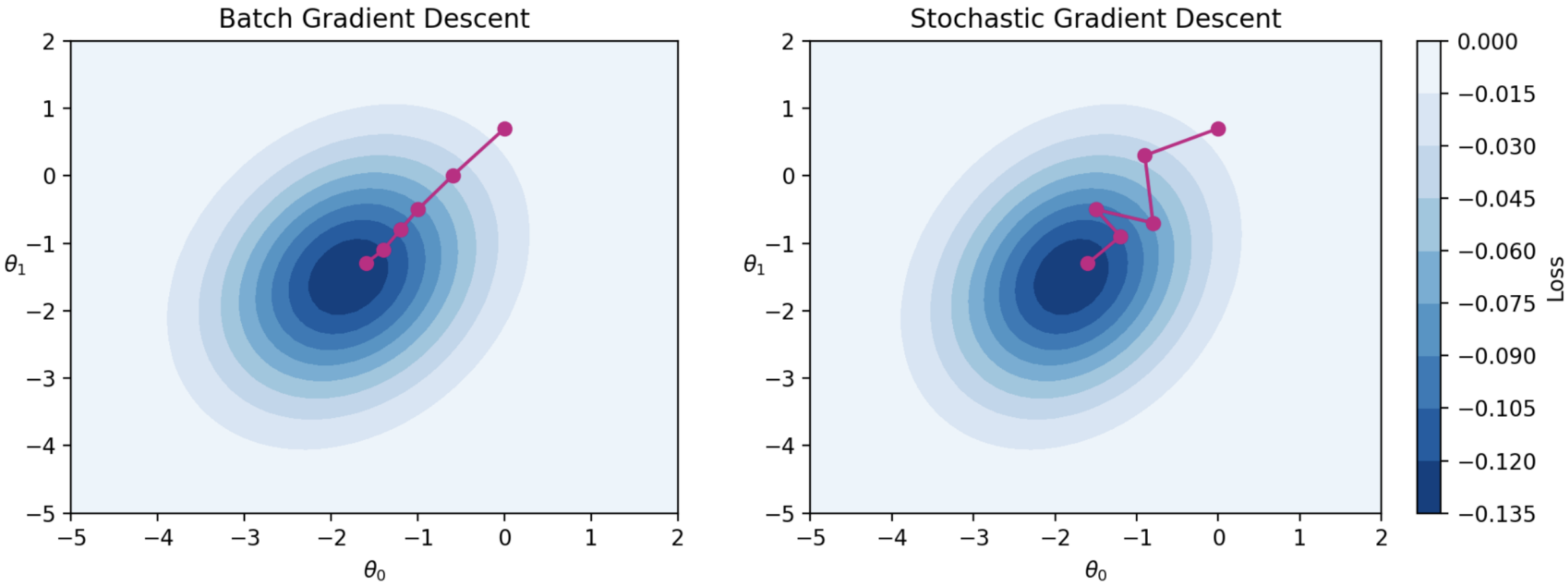
$$\nabla_w L(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla_w (y_i - f(x_i))^2$$

- Instead, we can estimate the gradient by randomly sampling one example at each iteration:

$$\nabla_w L(w) \approx \nabla_w (y_i - f(x_i))^2$$

- This is a noisy estimate of the gradient.
- This optimization algorithm is called stochastic gradient descent (SGD).

STOCHASTIC GRADIENT DESCENT



MINI-BATCH GRADIENT DESCENT

- Estimating the gradient with a single training example is too noisy.
- Instead, for each iteration, we can sample a batch of B random training examples.
  - B is the batch size.
  - Let b_1, …, b_B, be the indices of the batch:

$$\nabla_w L(w) \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_w \left(y_{b_i} - f(x_{b_i})\right)^2$$

- This is also oftentimes called stochastic gradient descent.

- Gradient descent methods only use the first derivative.
- There are other optimization algorithms that use the second derivative (Hessian).
- But they are more expensive, especially if we have many parameters (w is very large).
- There are algorithms that approximate parts of the Hessian using the gradient.
  - E.g., Adam, Sophia

EVALUATION

- Now we have a trained model.
- How do we evaluate it?
- An easy approach:
  - Split the data into two parts: the training set, and the test set.
  - Train on the training set, and measure accuracy on the test set.

$$\text{accuracy} = \frac{\text{\# of correctly-labeled examples in test set}}{\text{total \# of examples in test set}}$$

- But consider the spam detection task:
  - Suppose 99% of emails are truly not spam.
  - A classifier that always predicts `NOT SPAM` will get 99% accuracy.

EVALUATION

- We must be careful not to over- or under-estimate model performance.

true positives = # of emails we predicted to be SPAM, and are actually SPAM

false positives = # of emails we predicted to be SPAM, and are actually NOT SPAM

false negatives = # of emails we predicted to be NOT SPAM, and are actually SPAM

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \qquad \text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- Precision: Out of all the examples we predicted to be SPAM, how many did we get right?
- Recall: Out of all the examples that are actually SPAM, how many did we get right?

EVALUATION

- F1 score = 2PR/(P + R) (harmonic mean of precision and recall)
  - This is better than taking the average (P + R)/2 (arithmetic mean of precision and recall).
  - Consider a classifier that always predicts SPAM, but 99 out of 100 of emails are NOT SPAM.

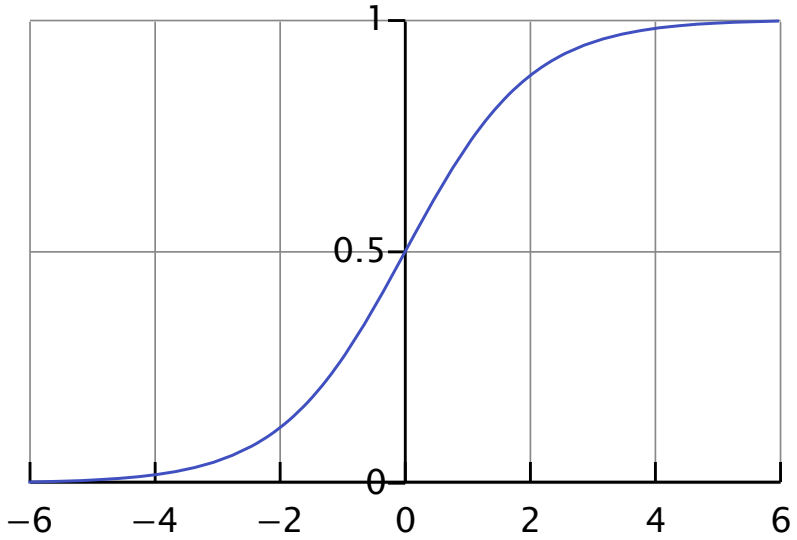
$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \qquad \text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

  - True positives is 1, false positives is 99, so precision is 0.01.
  - False negatives is 0, so recall is 1.0.
  - So the average score is: (0.01 + 1.0)/2 = 0.505.
  - The F1 score is 2(0.01)/(0.01 + 1.0) = 0.02.
  - F1 is more appropriate here since this classifier is very bad.

OTHER MACHINE LEARNING METHODS

- The perceptron is a very simple model: $f_w(x) = \text{sign}\{w^T\Phi(x)\}$
- We can replace it other models:
  - Another well-known classification model is logistic regression.

$$f_w(x) = \frac{\exp(w^T\Phi(x))}{1 + \exp(w^T\Phi(x))} = \sigma(w^T\Phi(x))$$



- The output of the logistic function is between 0 and 1.
- We can let 1 represent SPAM, and 0 represent NOT SPAM.
- This function is commonly used to model probabilities.

- We can think of the model predicting the probability that the input is SPAM.
- Logistic regression is a probabilistic or statistical model.

OTHER MACHINE LEARNING METHODS

- Both the perceptron and logistic regression are linear classifiers.
  - They have linear classification boundaries.
  - We can determine the classification boundary of the perceptron by inspecting the values of x where the sign of $f_w(x) = w^T x$ flips. (for simplicity, assume $\Phi(x) = x$)
  - The sign flips when $w^T x = 0$,
  - This is the equation of a hyperplane.
- Logistic regression:
  - The boundary is where the logistic function is 0.5.

$$\frac{1}{1 + \exp(w^T x)} = \frac{1}{2}$$

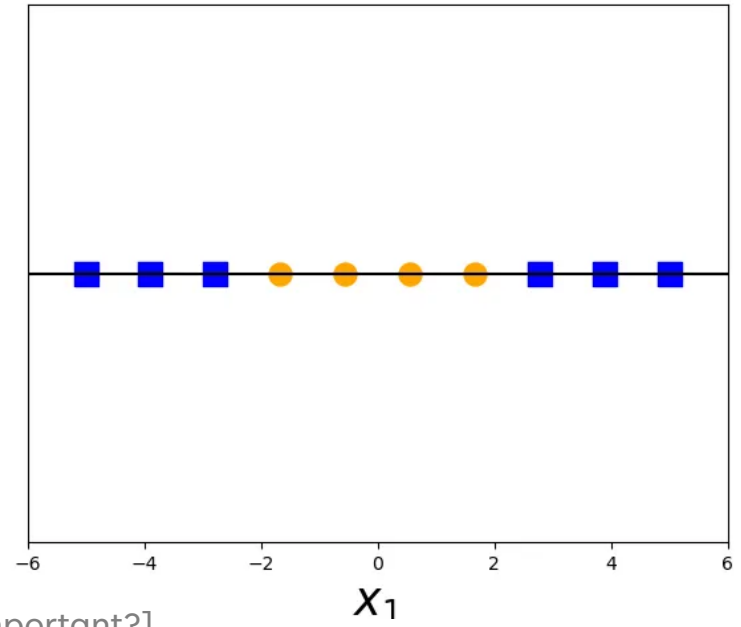
$$1 + \exp(w^T x) = 2$$

$$\exp(w^T x) = 1$$

$$w^T x = 0$$

LINEAR CLASSIFIERS

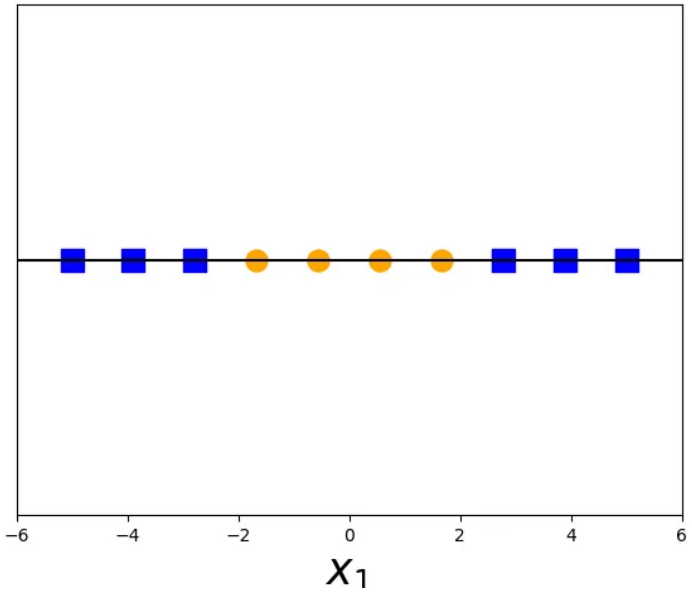
- Linear classifiers are limited in their expressiveness.
- There are certain kinds of data that they simply can't learn.
  - Specifically, linear classifiers can only learn data that is linearly separable.

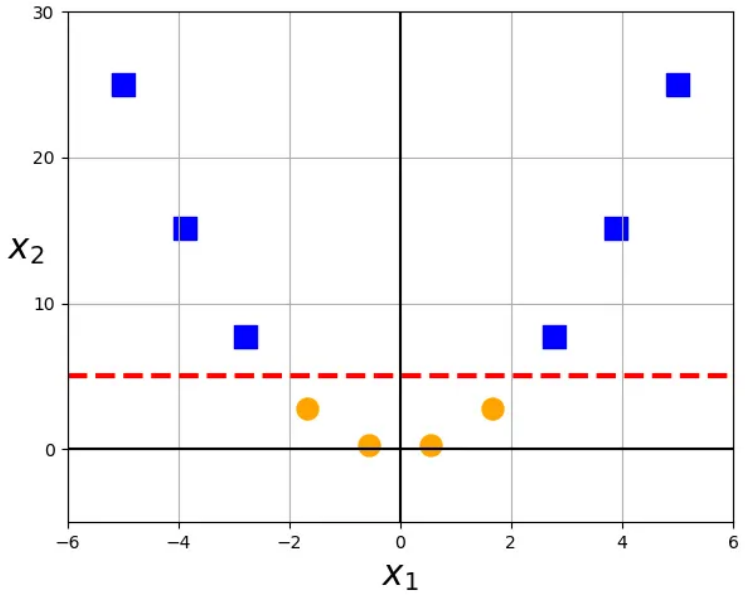


[Grace Zhang, What is the kernel trick? Why is it important?]
[Drew Wilimitis, The Kernel Trick in Support Vector Classification]

LINEAR CLASSIFIERS

- But we can use feature functions to map each point into a higher dimension.
  - The points can become linearly separable in higher dimensions.
  - More generally: Mapping into higher dimensions can make the data easier to learn.



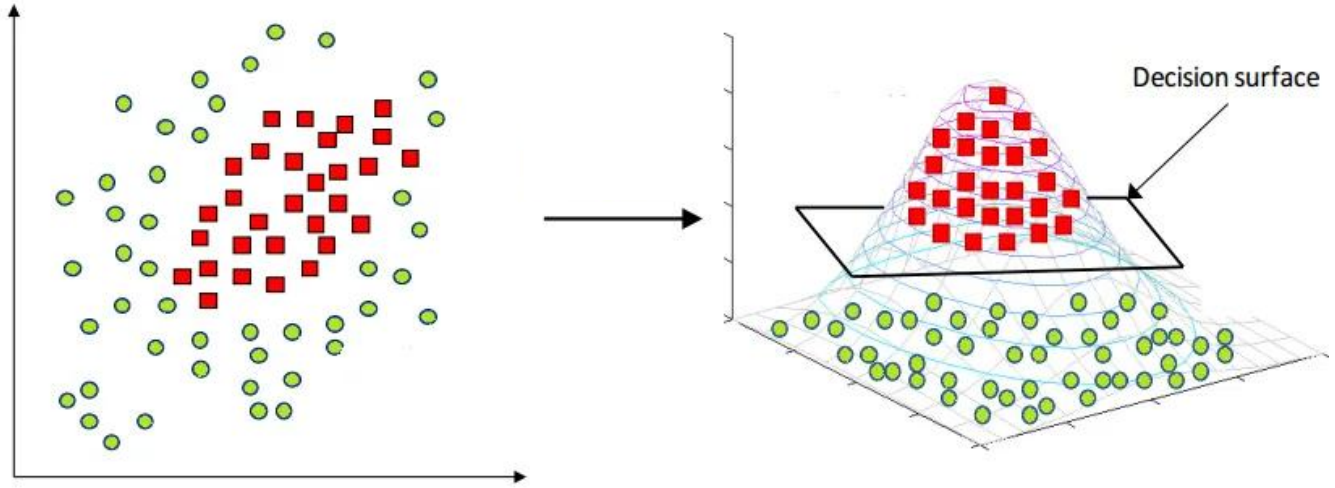


$\phi(x) = x^2$

[Grace Zhang, What is the kernel trick? Why is it important?]
[Drew Wilimitis, The Kernel Trick in Support Vector Classification]

LINEAR CLASSIFIERS

- But we can use feature functions to map each point into a higher dimension.
  - The points can become linearly separable in higher dimensions.
  - More generally: Mapping into higher dimensions can make the data easier to learn.



- More expressive machine learning models typically have more parameters, and are easier to overfit.

[Grace Zhang, What is the kernel trick? Why is it important?]
[Drew Wilimitis, The Kernel Trick in Support Vector Classification]

MULTI-CLASS CLASSIFICATION

- So far, we only considered binary classification.
  - Where there are only two output classes.
- What about sentiment analysis?
  - Given a user review of a product, the task is to classify whether the review is positive, negative, or neutral.
- Logistic regression can be extended to the multi-class setting:
  - Suppose we have K output classes.
  - The probability that the output y is k, given the input x, is:

$$p(y = k) = \frac{\exp(w_k^T \Phi(x))}{\sum_{j=1}^{K} \exp(w_j^T \Phi(x))}$$

  - Notice that now we have K weight vectors.
  - We compute the above probability for all k. The output is now a k-dimensional vector that sums to 1.
  - Note that $f(\alpha) = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp(\alpha_j)}$ is also known as the softmax function.

MULTI-CLASS CLASSIFICATION

- How do we train multi-class logistic regression?
- For probabilistic models, like logistic regression, we can use cross-entropy loss:

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p(y_i=k)\log f_w(x_i)_k$$

- Since we know the ground truth labels y_i, $p(y_i=k) = 1$ if the ground truth label for the i-th example is k, and 0 otherwise.

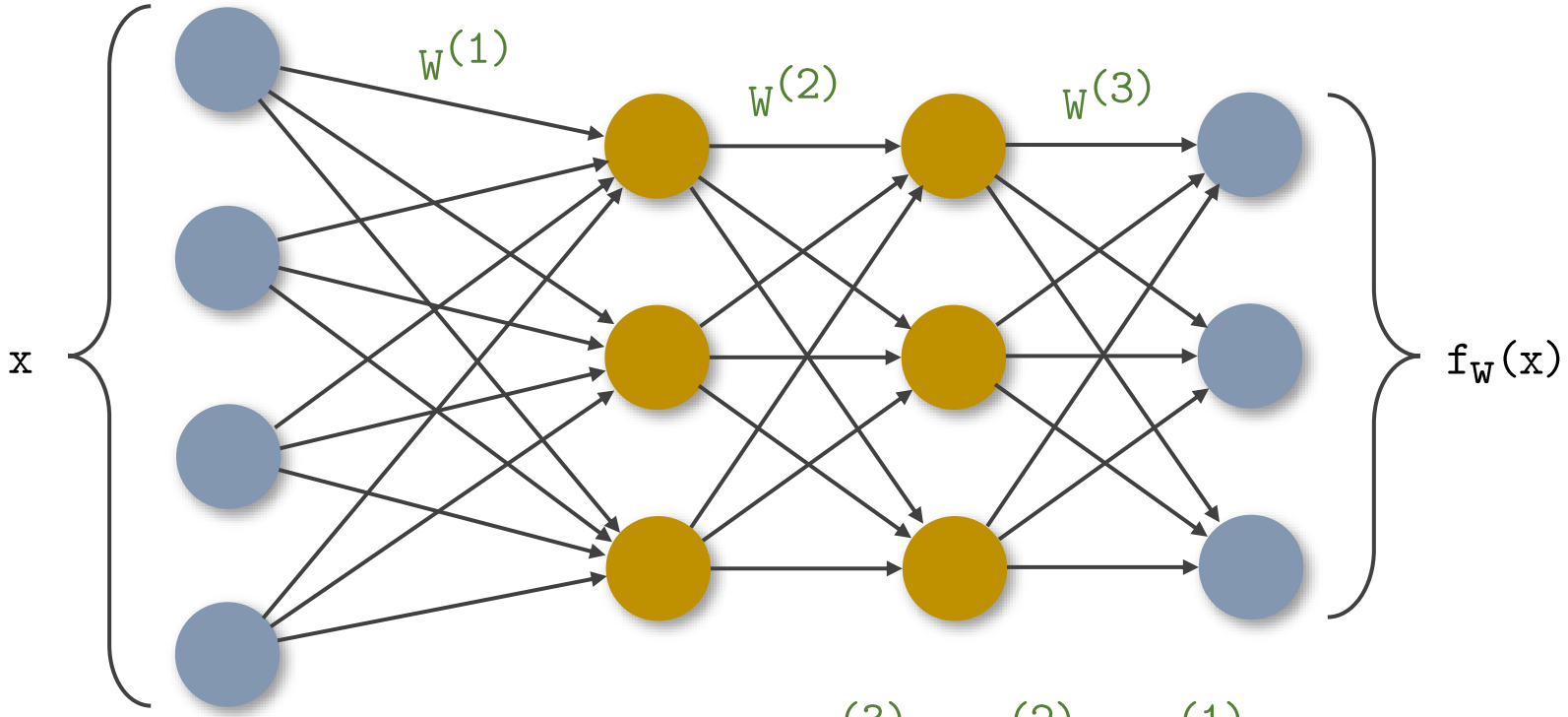
$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}\log f_w(x_i)_{y_i}$$

convex!

- Note that $f_w(x_i)_{y_i}$ is the model's prediction of the probability that the i-th example has label y_i.
  - This is the likelihood of the i-th example.
  - When learning, we are minimizing the loss, which is equivalent the negative log likelihood.
  - Therefore, minimizing the cross-entropy loss is equivalent to maximizing the likelihood.

MULTI-LAYER PERCEPTRON

- We can swap f with other machine learning models.



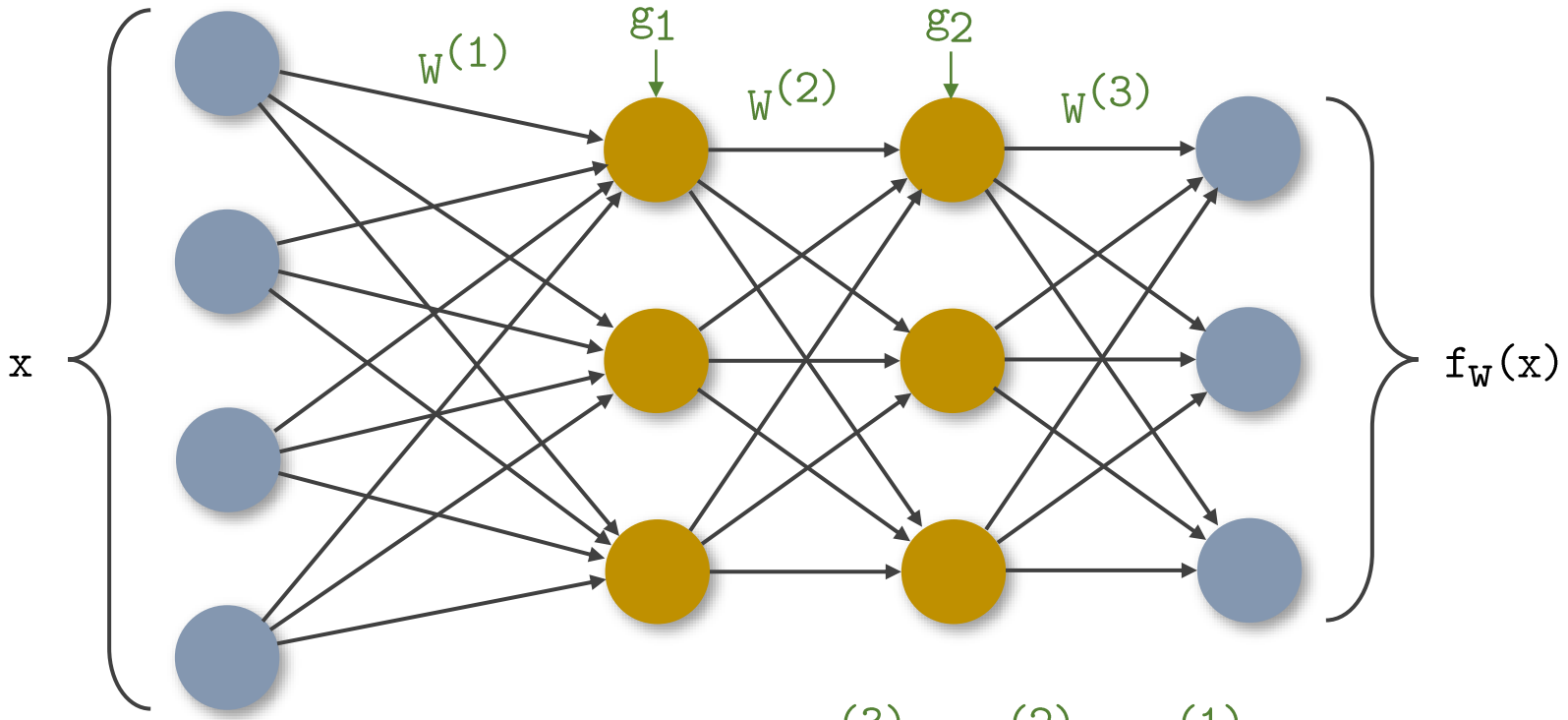
$$f_W(x) = W^{(3)} g_2(W^{(2)} g_1(W^{(1)} x))$$

$W^{(1)}$ are the connection weights in the first layer.

$W^{(1)}$ is a matrix:
Number of rows is the number of neurons in the next layer.
Number of columns is the number of neurons in the previous layer.

$W^{(1)}_{j,i}$ is the connection weight from neuron i in the previous layer to neuron j in the next.

MULTI-LAYER PERCEPTRON



$$f_W(x) = W^{(3)} g_2(W^{(2)} g_1(W^{(1)} x))$$

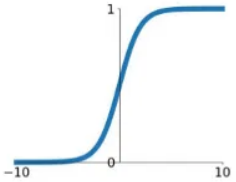
g_1 and g_2 are activation functions.
They must be non-linear since otherwise, adjacent layers would collapse into a single linear transformation.
(compositions of linear functions are linear)

MULTI-LAYER PERCEPTRON

Activation Functions

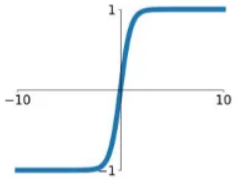
Sigmoid

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



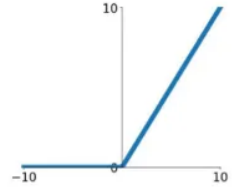
tanh

$$\tanh(x)$$



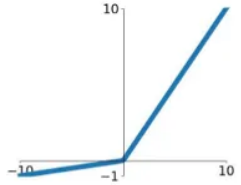
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

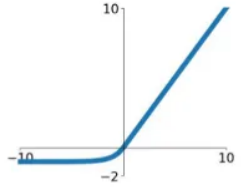


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



[Shruti Jadon, Introduction to Different Activation Functions for Deep Learning]

MULTI-LAYER PERCEPTRON

- If we want the output to be a probability distribution, we can add a `softmax` at the end.

$$f_W(x) = \mathrm{softmax}(W^{(3)} g_2(W^{(2)} g_1(W^{(1)} x)))$$

- Multi-layer perceptrons (MLPs) are also called fully-connected feed-forward (FF) networks.
- We can increase the number of layers, and/or the number of neurons in the hidden layers, to increase the complexity and expressiveness of the model.

TRAINING THE MLP

- How do we learn the weight matrices W, given a training dataset?
- We can use gradient descent!
- There is an efficient algorithm for computing the gradients in neural networks, called backpropagation (or backprop).
  - Not that difficult to derive. I encourage you to try.
  - Essentially repeated use of the chain rule.
- MLPs, like all neural networks, can learn nonlinear decision boundaries, and can be very expressive (especially if there are many neurons/layers).
- But they need more data to train (and to avoid overfitting) than simpler models.

EXPRESSIVENESS OF MLPS

- Universal approximation theorem: MLPs with one hidden layer and non-polynomial activation functions can approximate any function, with sufficiently many neurons in the hidden layer.
- There are similar theorems that look at the arbitrary-depth case.
- But keep in mind expressiveness doesn't imply learnability.
  - Just because a machine learning model can express a function does not mean that it can easily learn it from data.

QUESTIONS?