



Computer
Science

1

CSC380: Principles of Data Science

Nonlinear Models

Kyoungseok Jang

- Synchronized D2L and Gradescope
 - Don't be freaked out – we have more than 50% left.
 - Final 20% Final Project 14% HW 5, 6, 7 each 6%, attendance 10%
 - In 'Final Calculated Score' I applied your curved score
 - 'Midterm' tab will look the same
- Final project & HW7: April 13th
 - We delayed HW5 one week
 - I recommend you finish HW7 first and try to solve the final project
 - Group submission is not allowed for the final project
 - Deadline: April 21st for HW7 and May 5th for the final project
 - No extension, so I recommend you finish it before the deadline.

- New TA: Tugay Bilgis
 - Office hour: Thursday 10 – 11 am, Gould-Simpson 934

Question: Lasso vs Ridge

4

- Lasso
 - Good at eliminating features
 - Strong in a sparse environment: only a few features actually affect the result
 - It can be a drawback when 1) high-dim data with few samples or 2) when a group of variables is correlated but each variable has its own meaning.
 - No closed-form solution
 - Needs an iterative method, which is more expensive than a closed-form
 - For theoreticians like me: hard to find the theoretical properties of w^{L1}
- Ridge
 - Closed-form solution
 - Easy to find out, computationally cheap, easy to interpret theoretically
 - Almost never eliminates any feature
 - Strong when all your features are meaningful.
- Check 'Elastic net' if you are interested

- Example: suppose you have three data points
 - $(x,y) = (0,1), (1,4), (2,9)$
- True relationship: $y = (x + 1)^2 = x^2 + 2x + 1$
- Linear regression cannot catch this relationship perfectly.
- Instead, create additional 'features' $x_0 = 1, x_2 = x^2$
- Now your dataset changes to
 - $(x_0, x, x_2, y) = (1,0,0,1), (1,1,1,4), (1,2,4,9)$
- Linear regression $y = w^T \tilde{x} = w_0x_0 + w_1x + w_2x_2$
- Your conclusion $y = w_0 + w_1x + w_2x^2$

Review: Basis Functions

6

- A **basis function** can be any function of the input features **X**
- Define a set of B basis functions $\phi_1(x), \dots, \phi_B(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{b=1}^B w_b \phi_b(x) = w^T \phi(x)$$

notation:
 $\phi(x) := [\phi_1(x), \dots, \phi_B(x)]$

- The model is *linear in the transformed basis/induced features* $\phi(x)$.
 - You can use any linear method on this transformed features
- The model is *nonlinear in the data* **X**
 - The resulting model will be nonlinear eventually.

Review question: final exam from the last year

7

20. Answer true or false for each of the following.

(a): k -nearest neighbor is easier to interpret than decision tree in general because it is based on the Euclidean distance.

(b): The main reason why people often prefer $L1$ regularizer over $L2$ regularizer is computational efficiency.

(d): Basis function can be combined with any linear classifiers to build a nonlinear classifier.

Answer: F, F, T

- If we have time left, we can check these:

12. In a Bernoulli Naive Bayes model with k classes and v binary features, how many parameters are needed? Clarification: each feature is modeled by separate Bernoulli models (i.e., not a shared model).

k -sided die for $p(y)$ which has $k - 1$ parameters due to normalization, v coins for $p(x_i | y)$ for each class y . The total is $(k - 1) + vk$. We also accept $k + vk = (v + 1)k$.

11. You have a data set with n items and you want to evaluate neural network's performance. For each of the following methods, how many neural networks do you need to train, and how many training data points will each neural network be trained on?

1. Split data into 70% training and 30% test.

2. K fold cross validation.

3. Leave-one-out.

1) 1, $0.7n$. 2) K , $\frac{K-1}{K}n$, 3) n , $n-1$

Linear Regression

9

Recall the ordinary least squares solution is given by,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m1} & \dots & x_{mD} \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \quad w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Design Matrix
(each training input on a column)

Vector of
Training labels

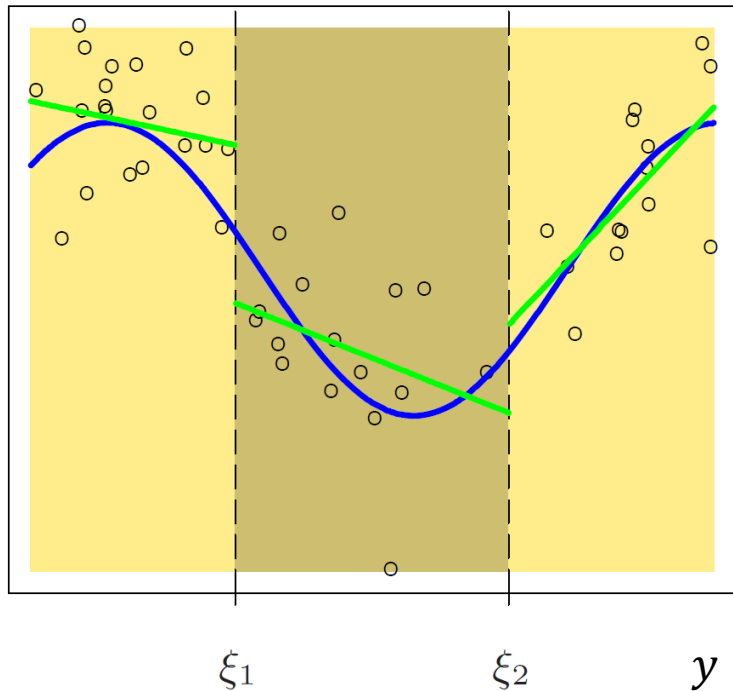
Can similarly solve in terms of basis functions,

$$\Phi = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_B(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_B(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_m) & \dots & \phi_B(x_m) \end{pmatrix} \quad w^{\text{OLS}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

Example: Piecewise Linear Regression

10

[Source: Hastie et al. (2001)]



Regression lines are discontinuous at boundary points

Decompose the input space into 3 regions with indicator basis functions,

$$\begin{aligned}\phi_1(x) &= x \cdot I\{x < \xi_1\} & \phi_4(x) &= I\{x < \xi_1\} \\ \phi_2(x) &= x \cdot I\{\xi_1 \leq x < \xi_2\} & \phi_5(x) &= I\{\xi_1 \leq x < \xi_2\} \\ \phi_3(x) &= x \cdot I\{\xi_2 \leq x\} & \phi_6(x) &= I\{\xi_2 \leq x\}\end{aligned}$$

Fit linear regression model,

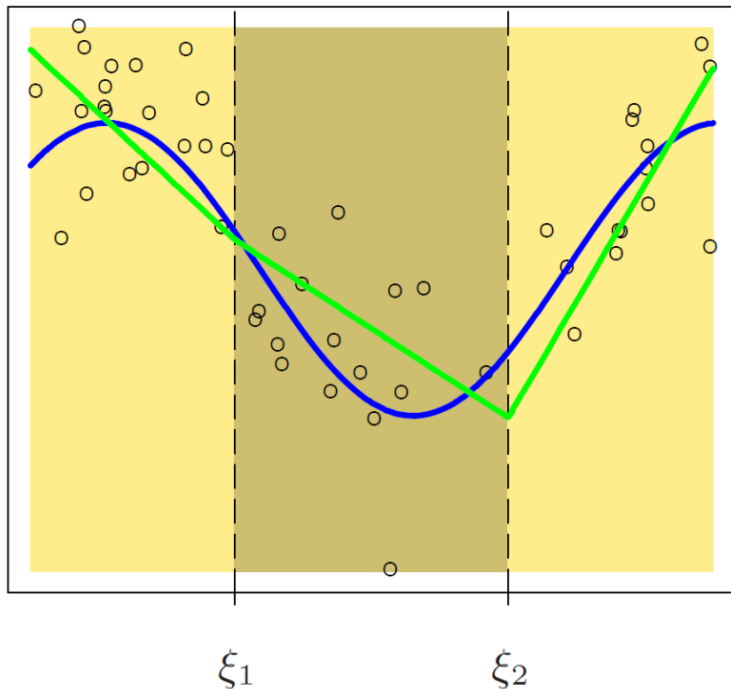
$$y = w^T \phi(x) = \sum_{i=1}^6 w_i \phi_i(x) = \overbrace{(w_1 x + w_2) I\{x < \xi_1\}}^{\text{First line segment}} + \dots$$

Effectively fits 3 linear regressions independently to data in each region

Example: Piecewise Linear Regression

11

[Source: Hastie et al. (2001)]



Enforce constraint that lines agree at boundary points,

$$\phi_1(x) = 1$$

$$\phi_2(x) = x$$

$$\phi_3(x) = (x - \xi_1)_+ \quad \text{<: activated only after } x \geq \xi_1$$

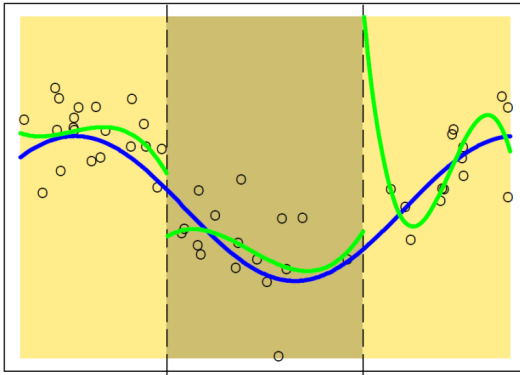
$$\phi_4(x) = (x - \xi_2)_+ \quad \text{<: activated only after } x \geq \xi_2$$

Where $(z)_+ := \max\{z, 0\}$.
I.e., the positive part of z

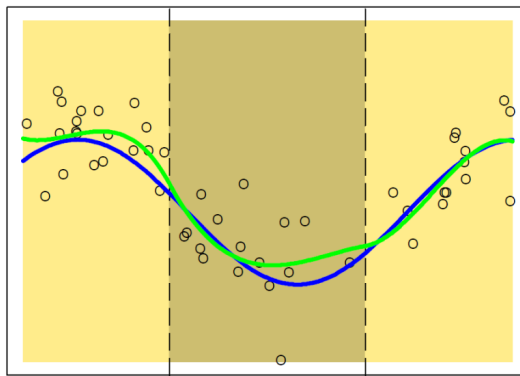
An improvement, but generally prefer *smoother* functions...

[Source: Hastie et al. (2001)]

Discontinuous

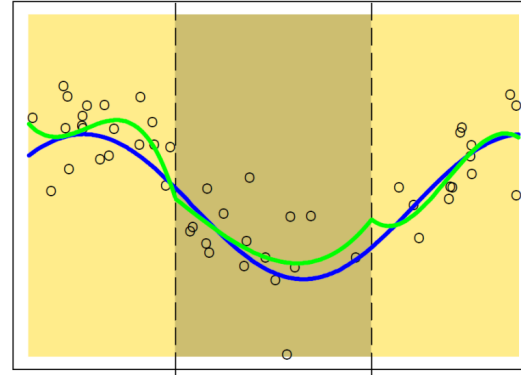


Continuous First Derivative

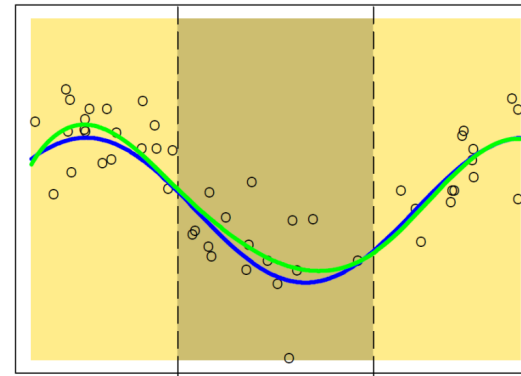


ξ_1 ξ_2

Continuous



Continuous Second Derivative



ξ_1 ξ_2

Replace linear basis functions with polynomial,

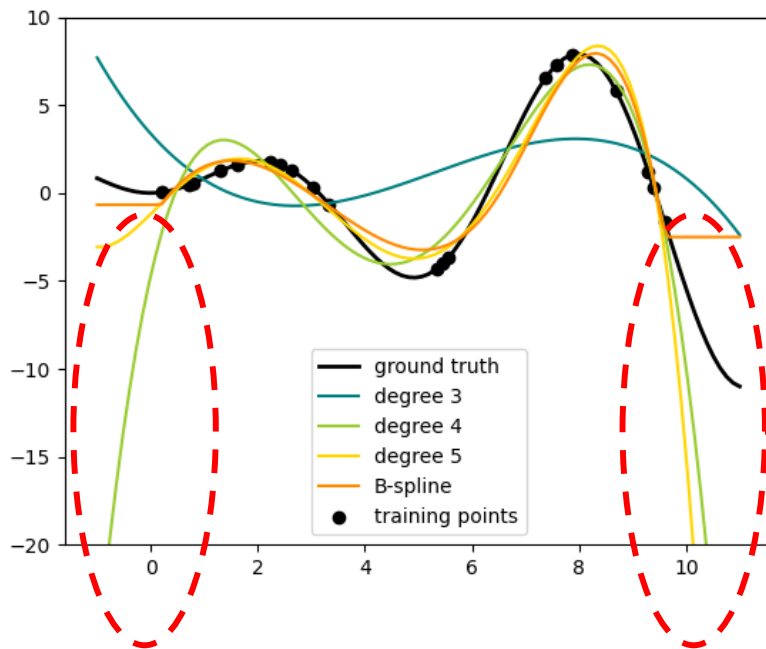
$$\phi_1(x) = 1 \quad \phi_2(x) = x$$

$$\phi_3(x) = x^2 \quad \phi_4(x) = x^3$$

$$\phi_5(x) = (x - \xi_1)_+^3$$

$$\phi_6(x) = (x - \xi_2)_+^3$$

Additional constraints ensure smooth 1st and 2nd derivatives at boundaries



These piecewise regression functions are called *splines*

Supported in Scikit-Learn
`preprocessing.SplineTransformer`

Caution Polynomial basis functions often yield poor out-of-sample predictions with higher order producing more extreme predictions

- Generally the first step in data science involves *preprocessing* or transforming data in some way
 - Filling in missing values (imputation)
 - Centering / normalizing / standardizing
 - Etc.
- We then fit our models to this preprocessed data
- One way to view preprocessing is simply as computing some basis function $\phi(x)$, nothing more

PROs

- More flexible modeling that is nonlinear in the original data
- Increases model expressivity

CONs

- Typically requires **more parameters** to be learned
- More sensitive to **overfitting** training data (due to expressivity)
- Requires more **regularization** to avoid overfitting
- Need to find *good* basis functions (feature engineering)

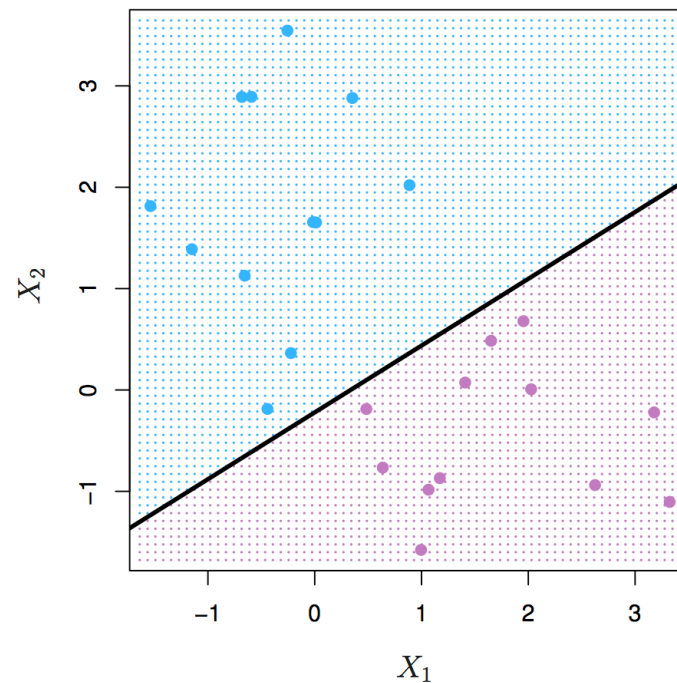
- Basis Functions
- **Support Vector Machine Classifier**
- Kernels
- Neural Networks

Linear Decision Boundary

17

Forget about the 'regression' point of view for now..

At the end of the day, we just want a line that separates the two classes well.

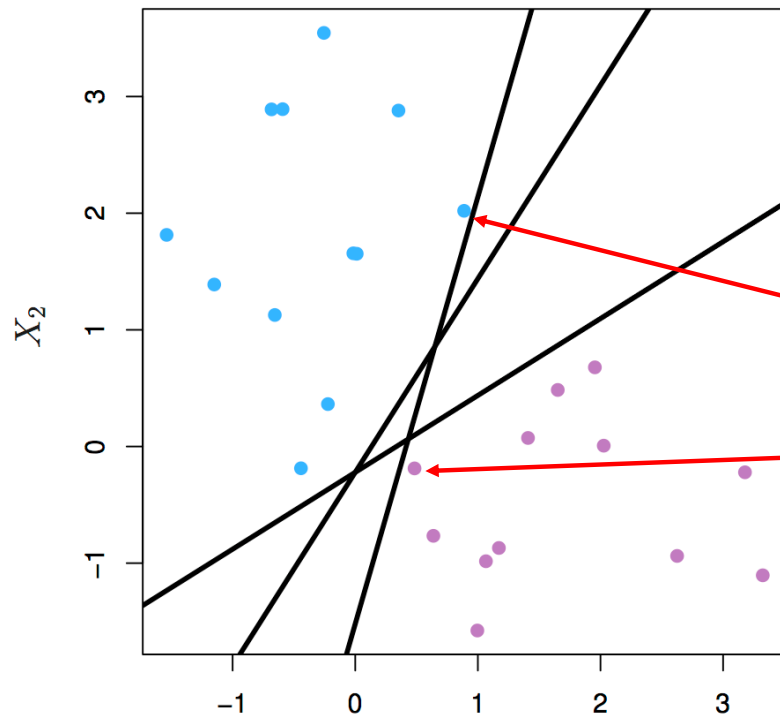


[Source: <http://www-bcf.usc.edu/~gareth/ISL/>]

Linear Decision Boundary

18

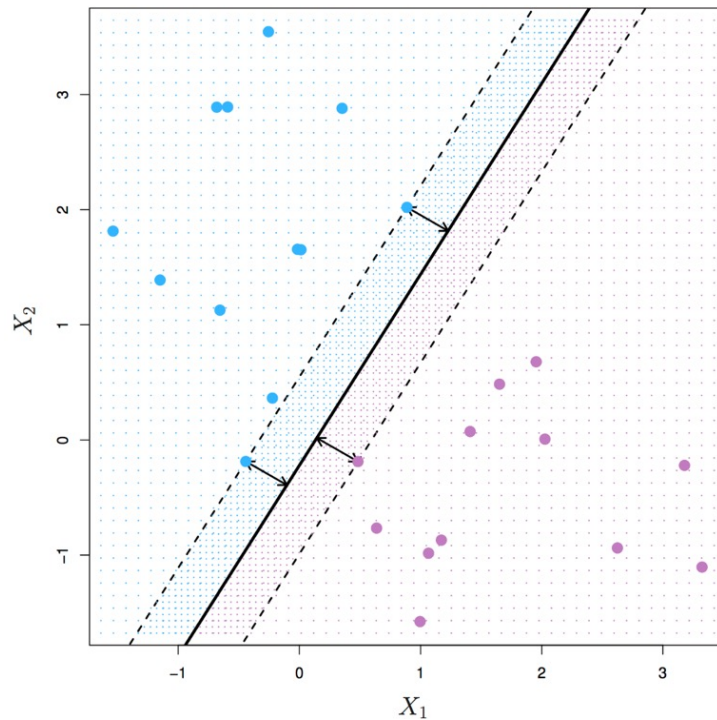
Note: Any boundary that separates classes is equivalently good on training data



Q: but if you have to choose one, which one will you choose?

1) Believing that these points are 'lucky' points that is barely out of the boundary.

2) Think most of your data are general and try to give enough space.



The **margin** measures minimum distance between each class and the decision boundary

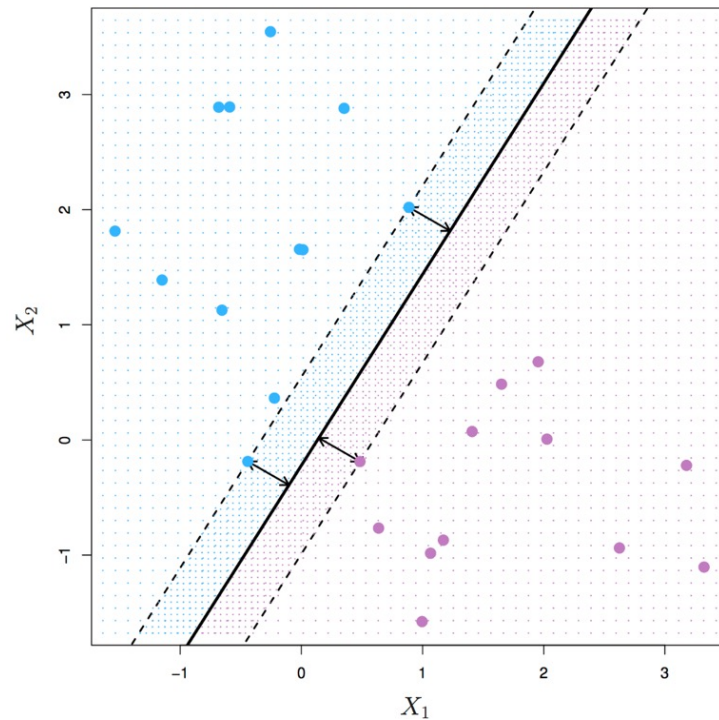
Observation Decision boundaries with larger margins are more likely to generalize to unseen data

Idea Learn the classifier with the largest margin that still separates the data...

...we call this a **max-margin classifier**

Max-Margin Classifier (Linear Seperable Case)

20



For now, let's focus on the case where the data is **linearly separable**

(Otherwise, there is no margin to talk about!)

Max-Margin Classifier (Linear Separable Case)

21

Recall that the linear model is given by

$$f(x) = w^T x + b$$

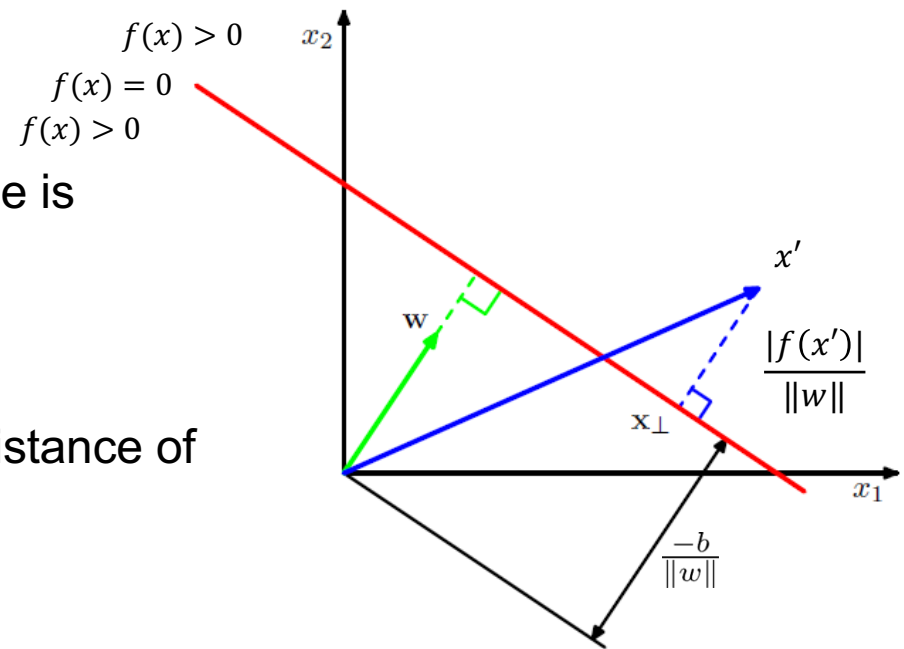
Let classes be $\{-1, 1\}$ so classification rule is

$$\text{Class} = \begin{cases} -1 & \text{if } f(x) < 0 \\ 1 & \text{if } f(x) \geq 0 \end{cases}$$

Decision boundary is now at $f(x) = 0$ and distance of x' to the decision boundary (margin) is

$$\frac{|f(x')|}{\|w\|}$$

where $\|w\| = \sqrt{w^T w} = \sqrt{\sum_i w_i^2}$



Distance from a point to a plane equation:

[wiki/Distance_from_a_point_to_a_plane](https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_plane)

Max-Margin Classifier (Linear Separable Case)

22

For training data $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$, a classifier $f(x) = w^\top x + b$ with 0 train error will satisfy

$$y^{(i)} f(x^{(i)}) = y^{(i)} (w^\top x^{(i)} + b) > 0$$

↓ negative margin
when misclassifying it!

The margin for $(x^{(i)}, y^{(i)})$ is given by,

$$\frac{y^{(i)} (w^\top x^{(i)} + b)}{\|w\|}$$

The margin of a classifier $f(x)$ is

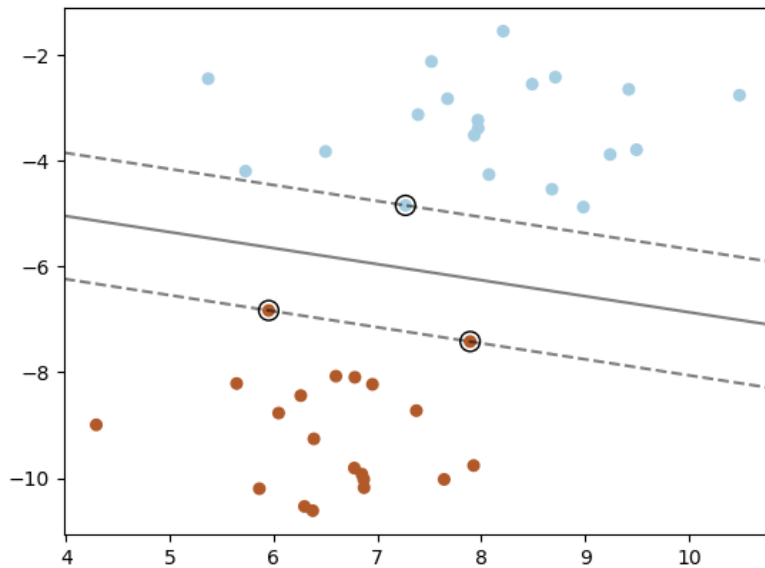
$$\min_i \frac{y^{(i)} (w^\top x^{(i)} + b)}{\|w\|}$$

Find f that maximize margin

$$\arg \max_{w,b} \min_i \frac{y^{(i)} (w^\top x^{(i)} + b)}{\|w\|}$$

Max-Margin Classifier (Linear Separable Case)

23



Maximize the
minimum margin

$$\arg \max_{w,b} \min_i \frac{y^{(i)} (w^\top x^{(i)} + b)}{\|w\|}$$

Minimum margin over
all training data

Find the parameters (w,b) that **maximize** the **smallest margin** over all the training data

Max-Margin Classifier (Linear Separable Case)

24

Learning objective is hard to solve in this form...

$$\arg \max_{w,b} \min_i \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|} \quad \leftarrow (*)$$

But we can scale parameters $w \rightarrow \alpha w$ and $b \rightarrow \alpha b$ without changing the margin

\Rightarrow But then, there exists an infinitely many solution

\Rightarrow Optimization packages will do a bad job.

...we can pick the nearest point j to the margin and restrict $y^{(j)}(w^\top x^{(j)} + b) = 1$

This means 1) $y^{(i)}(w^\top x^{(i)} + b) \geq 1, \forall i$

$$2) (*) = \arg \max_{w,b} \frac{1}{\|w\|} = \arg \min_{w,b} \|w\| \quad \Rightarrow \min_i y^{(i)}(w^\top x^{(i)} + b) = 1 \quad !!!$$

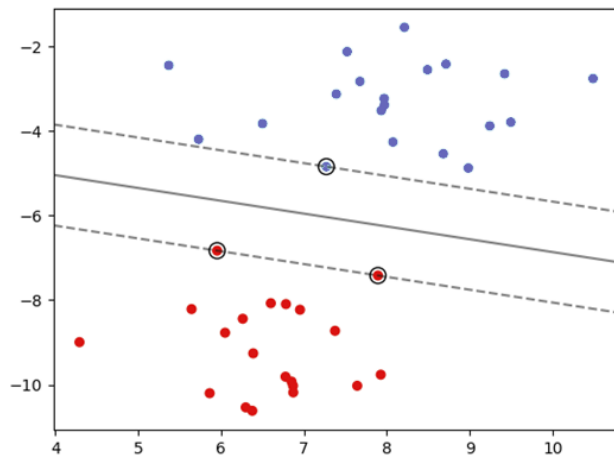
with above constraint!

We now just need to choose (w,b) that minimizes $\|w\|$ under this constraint!

Support Vector Machine (Hard Margin)

25

... it leads to



$$\min_{w,b} \frac{1}{2} \|w\|^2$$

subject to

$$y^{(i)} (w^\top x^{(i)} + b) \geq 1 \quad \text{for } i = 1, \dots, m$$

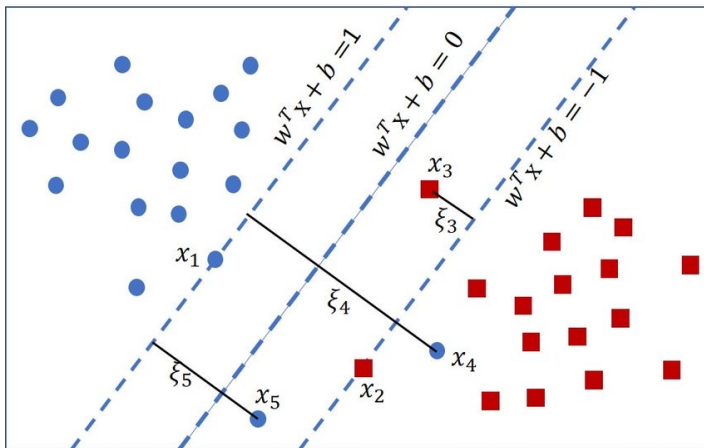
This is a convex (quadratic) optimization problem that can be solved efficiently

- Data are D-dimensional *vectors*
- Margins determined by nearest data points called *support vectors*
- We call this a *support vector machine* (SVM)

Support Vector Machine (Soft Margin)

26

If the data is linearly not separable,



$$\min_{w, b, \xi_{\{1:m\}}} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i$$

subject to

$$y^{(i)} (w^\top x^{(i)} + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, m$$

auxiliary 'slack' variable

C: tradeoff between margin and the slack!

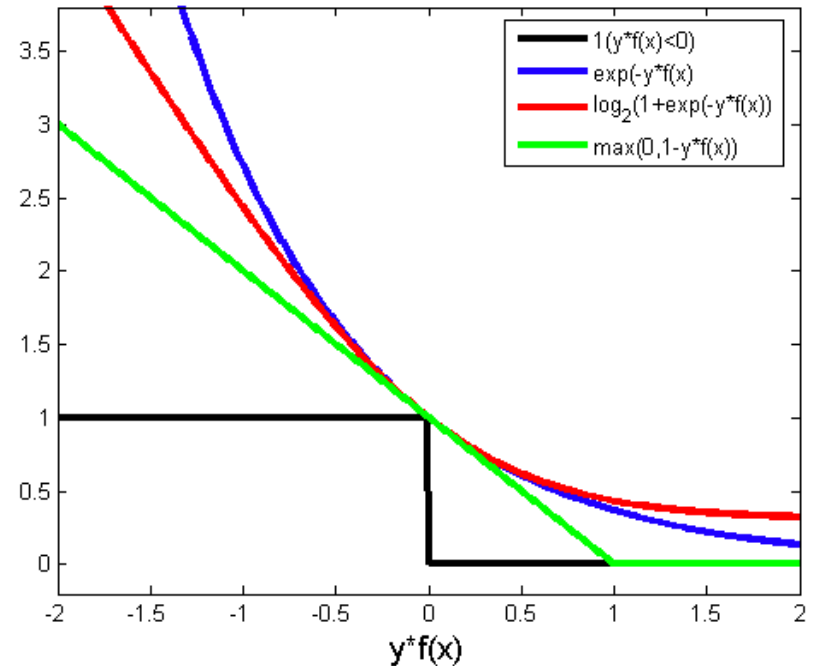
Support Vector Machine (Loss function perspective) 27

$$\begin{aligned} & \text{minimize } \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^m \xi_i \\ & \text{subject to} \\ & y^{(i)} (w^\top x^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \text{for } i = 1, \dots, m \end{aligned}$$

Equivalent formulation

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (1 - y^{(i)} (w^\top x^{(i)} + b))_+$$

$$\ell(f; x^{(i)}, y^{(i)}) = (1 - y^{(i)} f(x^{(i)}))_+$$



$$(X)_+ := \max(X, 0)$$

- Eventually, we only care about classification result, not the likeliness level in classification problems.
- $err = \frac{1}{m} \sum_{i=1}^m I(f(x^{(i)}) \neq y^{(i)})$ (Train set error, black line)
- Therefore, it's better to minimize this zero-one loss. In our real life, algorithms will be scored based on this zero-one loss.
- However, it is hard to use optimizations with this error, we use alternatives.
 - Known to be NP-hard
 - Therefore, it is better to be closer to the zero-one loss.

General Principle

29

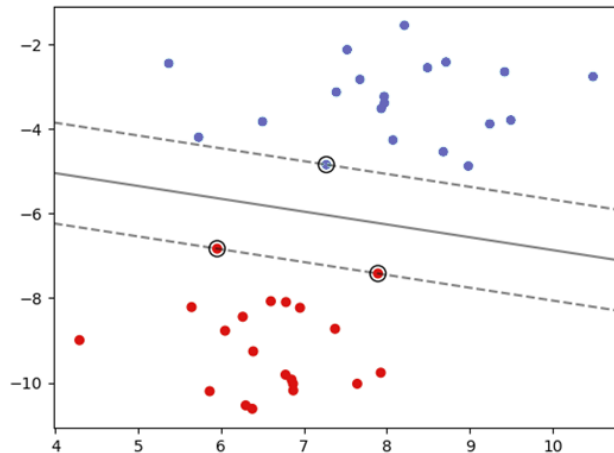
$$\arg \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (1 - y^{(i)}(w^\top x^{(i)} + b))_+$$

=> by setting $C = 1/\lambda$, it's equivalent to solve

$$\arg \min_{w,b} \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^m (1 - y^{(i)}(w^\top x^{(i)} + b))_+$$

SVM belongs to the general loss-oriented formulation!

$$\text{Model} = \arg \min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$



$$\min_{w, b} \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^m \xi_i$$

subject to

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, m$$

Those data points achieving equality $y^{(i)}(w^\top x^{(i)} + b) = 1 - \xi_i$ are called **support vectors**.

Turns out, if you knew support vectors already, solving the optimization problem above with **just the support vectors as train set** leads to the same solution.

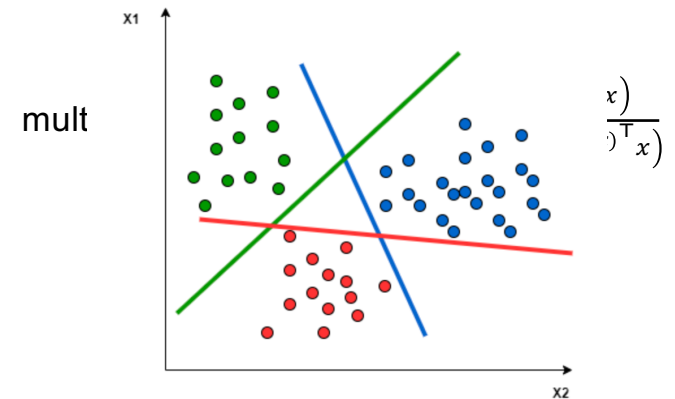
⇒ Leave-one-out cross validation can be done fast!

- Recall: logistic regression had a very natural extension to multi-class.
- What about SVM?

... Researchers have found a few, but it was not any better than a simple trick below.

[One-vs-the-rest trick]

- Given: dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^m$
- For each class $c \in \{1, \dots, C\}$
 - Define label $z^{(i)} \in \{-1, 1\}$ where 1 for class c and -1 for other classes, for all $i=1, \dots, m$.
 - Train a classifier f_c with $\{(x^{(i)}, z^{(i)})\}_{i=1}^m$
- To classify x^* , compute $\hat{y} = \arg \max_{c \in \{1, \dots, C\}} \text{decision_value}(f_c(x^*))$



decision value in our case: proportional to the signed distance from the boundary

SVM with linear decision boundaries,

[`sklearn.svm.LinearSVC`](#)

Call options include...

penalty : {'l1', 'l2'}, default='l2'

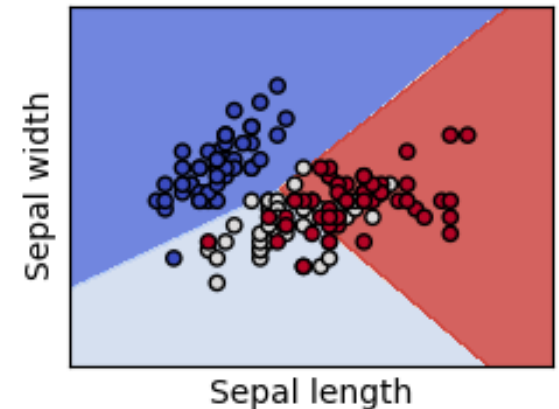
Specifies the norm used in the penalization. The 'l2' penalty is the standard used in SVC. The 'l1' leads to `coef_` vectors that are sparse.

dual : bool, default=True

Select the algorithm to either solve the dual or primal optimization problem. Prefer `dual=False` when `n_samples > n_features`.

C : float, default=1.0

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.



**Only showing linear
for a reason that will
be clear soon...**

Other options for controlling optimizer (e.g. convergence tolerance 'tol')

- Suppose that $w = (1,1)$, $b = 2$ in our SVM classifier, and your dataset was linearly separable.
- Q1) What is the prediction of your classifier when your input is
 - $x^* = (0,0)$?
 - $x^* = (3, -6)$?
- Q2) Find out a support vector from these candidates (hint: $y^{(i)} = \pm 1$)
 - (1) $x^{(1)} = (1,1)$
 - (2) $x^{(2)} = (1, -3)$
 - (3) $x^{(3)} = (4, -4)$

- Basis Functions
- Support Vector Machine Classifier
- **Kernels**
- Neural Networks

Support Vector Machine (Dual)

35

(Theorem) SVM can be trained in two equivalent ways. (not important for your score)

Primal

D+1 variables

$$\min_{w,b} \|w\|^2 + C \sum_{i=1}^m \xi_i$$

$$\text{s.t. } y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i, \forall i$$

Dual

$\alpha \in \mathbb{R}^m$

$$\max_{0 \leq \alpha \leq C} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)})^\top x^{(j)}$$

$$\text{s.t. } \sum_i \alpha_i y^{(i)} = 0$$

important

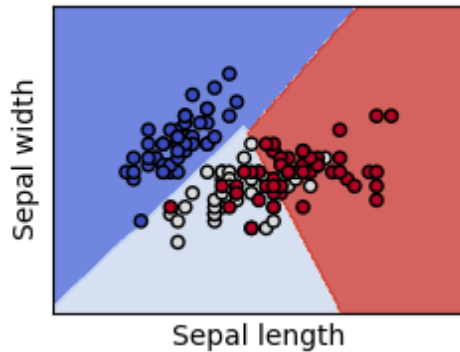
Then, the learned function is $f(x) = \sum_{i=1}^m \alpha_i y^{(i)} (x^{(i)})^\top x$

Suppose we use the basis expansion: $x^{(i)} \rightarrow \phi(x^{(i)})$

Key observation: all the operations can be done as long as we have a magic function $\kappa(x, x')$ that evaluates $\langle \phi(x), \phi(x') \rangle$.

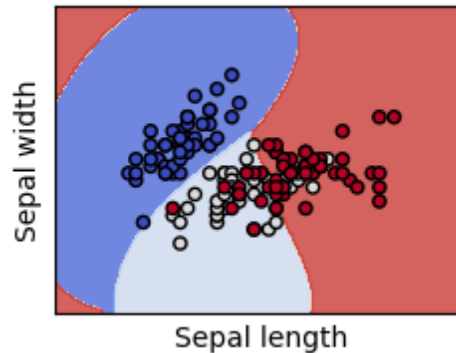
called 'kernel' function

SVC with linear kernel



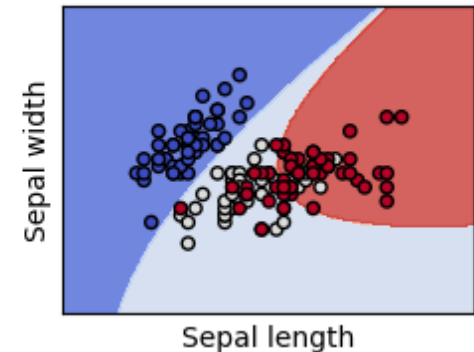
$$\kappa(x, x') = x^T x'$$

SVC with RBF kernel



$$\kappa(x, x') = \exp(-\gamma \|x - x'\|^2)$$

SVC with polynomial (degree 3) kernel

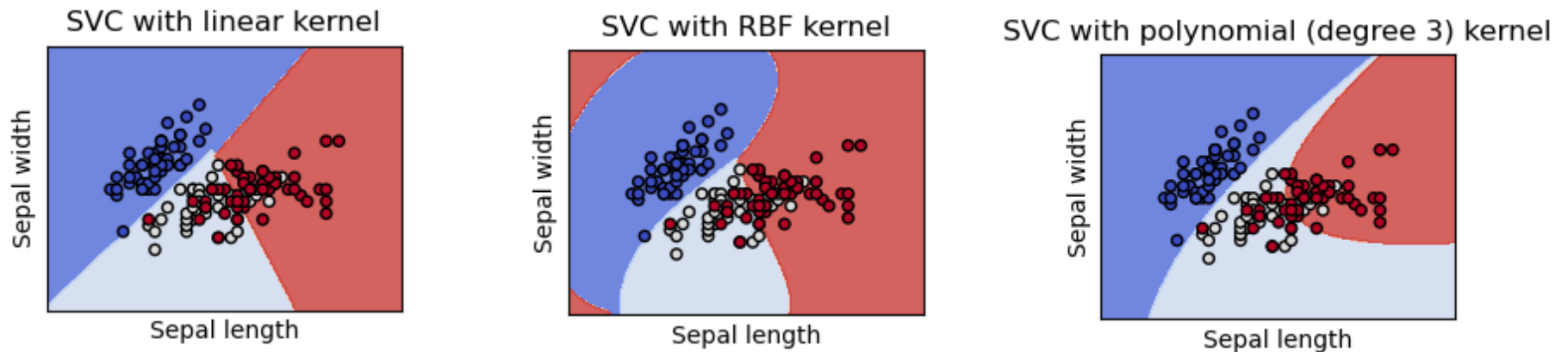


$$\kappa(x, x') = (x^T x' + c)^3$$

← **Note: basis function is infinite dim.**

(Theorem) If the kernel function $\kappa(x, x')$ satisfies certain condition, there exists a basis function $\phi(x)$ for which $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$!

- In fact, $\phi(x)$ could even be infinite dimensional: $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^\infty$



$$\kappa(x, x') = x^T x'$$

$$\kappa(x, x') = \exp(-\gamma \|x - x'\|^2)$$

$$\kappa(x, x') = (x^T x' + c)^3$$

- General kernel-based SVM lives in:

[`sklearn.svm.SVC\(kernel='kernel name'\)`](#)

- Supports most major kernel types

The learned function is $f(x) = \sum_{i=1}^m \alpha_i y^{(i)} \left(\phi(x^{(i)}) \right)^\top \phi(x) = \sum_{i=1}^m \alpha_i y^{(i)} \kappa(x^{(i)}, x)$

Note:

- We need to store training data for making prediction.
- Fortunately, $\alpha_i = 0$ for non-support vectors.
- So, we only need to store support vectors!

Q: Which ML algorithms did we have to do the same?

turns out, you can use kernels for logistic regression, but there is **no such thing as support vectors** in there!

=> SVM trains classifiers that require less storage!

kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

gamma : {'scale', 'auto'} or float, default='scale'

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

for RBF,

small γ : complex decision boundary

large γ : more like linear decision boundary

- if `gamma='scale'` (default) is passed then it uses $1 / (n_features * X.var())$ as value of gamma,
- if 'auto', uses $1 / n_features$.

max_iter : int, default=-1

Hard limit on iterations within solver, or -1 for no limit.

verbose : bool, default=False

Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

class_weight : dict or 'balanced', default=None

Set the parameter C of class i to `class_weight[i]*C` for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

Example: Fisher's Iris Dataset

40

Classify among 3 species of Iris flowers...



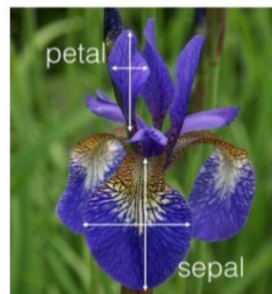
Iris setosa



Iris versicolor



Iris virginica

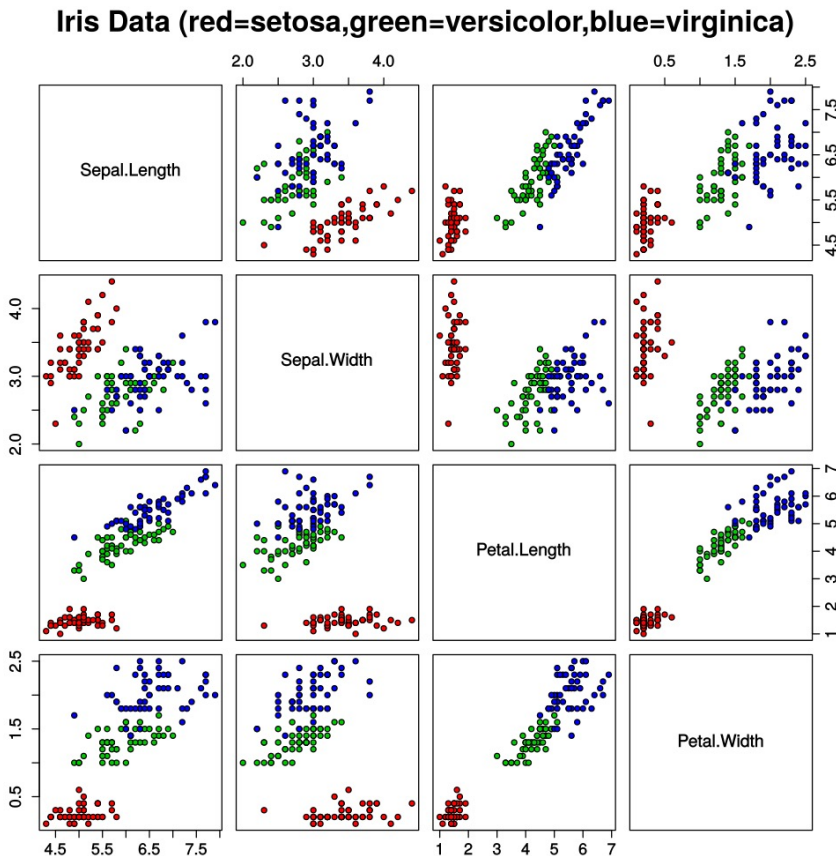


Four features (in centimeters)

- Petal length / width
- Sepal length / width

Example: Fisher's Iris Dataset

41



*Fairly easy to separate
setosa from others using a
linear classifier*

*Need to use nonlinear basis /
kernel representation to
better separate other classes*

Example: Fisher's Iris Dataset

42

Train 8-degree polynomial kernel SVM classifier,

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='poly', degree=8)
svclassifier.fit(X_train, y_train)
```

Generate predictions on held-out test data,

```
y_pred = svclassifier.predict(X_test)
```

Show confusion matrix and classification accuracy,

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
avg / total	0.97	0.97	0.97	30

[Source: <https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/>]

Kernel Ridge Regression

43

Recall the solution of L2-regularized linear regression (ridge regression),

$$\Phi = \begin{pmatrix} \phi_1(x^{(1)}) & \dots & \phi_B(x^{(1)}) \\ \phi_1(x^{(2)}) & \dots & \phi_B(x^{(2)}) \\ \vdots & \vdots & \vdots \\ \phi_1(x^{(m)}) & \dots & \phi_B(x^{(m)}) \end{pmatrix}$$

m by B

$$w^{\text{ridge}} = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{y}$$

B by B

= with some algebra..

$$= \Phi^T (\Phi \Phi^T + \lambda I)^{-1} \mathbf{y}$$

B by m m by m

$$\Phi \Phi^T = \begin{pmatrix} \kappa(x^{(1)}, x^{(1)}) & \kappa(x^{(1)}, x^{(2)}) & \dots & \kappa(x^{(1)}, x^{(m)}) \\ \kappa(x^{(2)}, x^{(1)}) & \kappa(x^{(2)}, x^{(2)}) & \dots & \kappa(x^{(2)}, x^{(m)}) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x^{(m)}, x^{(1)}) & \kappa(x^{(m)}, x^{(2)}) & \dots & \kappa(x^{(m)}, x^{(m)}) \end{pmatrix}$$

Kernel Ridge Regression

44

Once we learn w , then the prediction for x is

$$f(x) = w^T \phi(x)$$

Solution to ridge regression $= [(\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{y}]^T \phi(x)$

previous slide $= \mathbf{y}^T (\Phi \Phi^T + \lambda I)^{-1} \underbrace{\Phi \cdot \phi(x)}$

m by m matrix computable by $\kappa(x, x')$

m by 1 vector computable by

$$\begin{pmatrix} \kappa(x^{(1)}, x) \\ \vdots \\ \kappa(x^{(m)}, x) \end{pmatrix}$$

Can now express regression without explicitly specifying basis functions

Primal

$$\Phi = \begin{pmatrix} \phi_1(x^{(1)}) & \dots & \phi_B(x^{(1)}) \\ \phi_1(x^{(2)}) & \dots & \phi_B(x^{(2)}) \\ \vdots & \vdots & \vdots \\ \phi_1(x^{(m)}) & \dots & \phi_B(x^{(m)}) \end{pmatrix}$$

$$\Phi\Phi^T =$$

Dual

$$\begin{pmatrix} \kappa(x^{(1)}, x^{(1)}) & \kappa(x^{(1)}, x^{(2)}) & \dots & \kappa(x^{(1)}, x^{(m)}) \\ \kappa(x^{(2)}, x^{(1)}) & \kappa(x^{(2)}, x^{(2)}) & \dots & \kappa(x^{(2)}, x^{(m)}) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x^{(m)}, x^{(1)}) & \kappa(x^{(m)}, x^{(2)}) & \dots & \kappa(x^{(m)}, x^{(m)}) \end{pmatrix}$$

$$w = (\underbrace{\Phi^T \Phi + \lambda I}_{\text{B by B Matrix Inversion: } O(B^3)})^{-1} \Phi^T y$$

$$w = \Phi^T (\underbrace{\Phi\Phi^T + \lambda I}_{\text{m by m Matrix Inversion } O(m^3)})^{-1} y$$

B by B Matrix Inversion: $O(B^3)$

m by m Matrix Inversion $O(m^3)$

- If $B \ll m$, use primal. If $B \gg m$, use dual.
- If $B = \infty$, then w cannot be computed anyways (why?)
 - But we can still make predictions with $f(x) = y^T (\Phi\Phi^T + \lambda I)^{-1} \Phi \cdot \phi(x)$!!

alpha : float or array-like of shape (n_targets,), default=1.0

Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to $1 / (2C)$ in other linear models such as `LogisticRegression` or `LinearSVC`. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number. See [Ridge regression and classification](#) for formula.

kernel : str or callable, default="linear"

Kernel mapping used internally. This parameter is directly passed to `pairwise_kernel`. If `kernel` is a string, it must be one of the metrics in `pairwise.PAIRWISE_KERNEL_FUNCTIONS`. If `kernel` is "precomputed", X is assumed to be a kernel matrix. Alternatively, if `kernel` is a callable function, it is called on each pair of instances (rows) and the resulting value recorded. The callable should take two rows from X as input and return the corresponding kernel value as a single number. This means that callables from `sklearn.metrics.pairwise` are not allowed, as they operate on matrices, not single samples. Use the string identifying the kernel instead.

gamma : float, default=None

Gamma parameter for the RBF, laplacian, polynomial, exponential chi2 and sigmoid kernels. Interpretation of the default value is left to the kernel; see the documentation for `sklearn.metrics.pairwise`. Ignored by other kernels.

Example: Kernel Ridge Regression

47

Generate some sinusoidal (periodic) data,

```
X = 15 * rng.rand(100, 1)
y = np.sin(X).ravel()
y += 3 * (0.5 - rng.rand(X.shape[0])) # add noise
```

Define an exponentiated sinusoidal kernel,

```
from sklearn.gaussian_process.kernels import ExpSineSquared
kernel = ExpSineSquared(length_scale=4.64, periodicity=12.9)
```

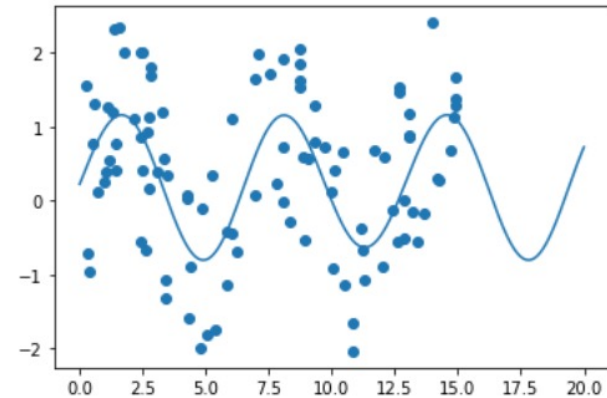
$$\exp\left(-\frac{2 \sin^2(\pi d(x_i, x_j)/p)}{l^2}\right)$$

Fit kernel ridge regression,

```
from sklearn.kernel_ridge import KernelRidge
kr = KernelRidge(kernel=kernel, alpha=0.001).fit(X, y)
```

Plot results,

```
X_plot = np.linspace(0, 20, 10000)[: , None]
y_kr = kr.predict(X_plot)
plt.scatter(X, y)
plt.plot(X_plot, y_kr)
plt.show()
```



- Final exam question in last year

15. We are given a train set $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$. With the SVM dual formulation's solution $\alpha \in \mathbb{R}^m$, the learned function can be written as $f(x) = \sum_{i=1}^m \alpha_i y^{(i)} (x^{(i)})^\top x$ for a test point x . Let s be the number of support vectors. Say we would like to train a nonlinear SVM (dual formulation) with a basis function $\phi(x) \in \mathbb{R}^\infty$ for which there exists a known kernel function $k(x, x')$.

(i) Write down the learned function $f(x)$ based on the kernel function.

(ii) To evaluate $f(x)$ for a test point x , how many evaluations of kernel function do we need to perform?

(i) $f(x) = \sum_{i=1}^m \alpha_i y^{(i)} k(x^{(i)}, x)$

(ii) s



CSC380: Principles of Data Science

Nonlinear Models 2

Kyoungseok Jang

- HW6 deadline: April 12th
- Office hours on Thursday: 10 – 11 am, GS942
 - Previously made a wrong announcement (GS934)

- Some students asked me whether r^2 score can be negative
 - Yes, when your model is so poor (worse than constant model)
 - You will get negative r^2 score if you naively use `cv=5` on your cross-val-score.
 - It happens because our prostate cancer dataset has some structure.
- For HW6, if you got a negative r^2 score, it's fine, keep going.
- If you don't like it, you can use `cv=KFold(n_splits=5, shuffle=True)` instead.

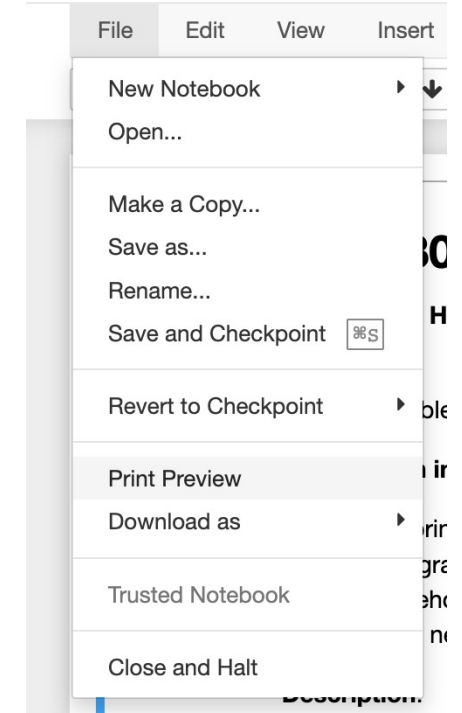
- Fit_transform: fit + transform
 - Therefore, using the fit_transform function on your **TEST SET** will change your transformation criterion **based on your test set.**
 - It also includes label_encoder: they **automatically create a mapping between feature and number.** If you use fit_transform on your test set, it may be possible that the mapping might be different.
 - e.g.) train: male → 0, female → 1, test: male → 1, female → 0
- Transform only outputs 'transformed input', not actually transform your input.
 - E.g.) `ss.transform(X_train)` will not change `X_train`
 - To transform your train set, do `X_train = ss.transform(X_train)`

- F1_macro is different from F1.
 - F1, precision, accuracy are the scores for the ‘positive class’
 - Precision: $TP/(TP+FP)$, Accuracy: $TP/(TP+FN)$ – all focus on positive set
 - Because we are usually interested in the positive set (e.g. COVID test positive)
 - F1_macro: F1 score for multi-class
 - For multiclass, there’s no major difference between classes (e.g. cat vs dog)
 - measures F1 for each class, and average
 - So, if you use F1_macro instead of F1 in HW5, it means $\frac{F_1(pos)+F_1(neg)}{2}$
 - What we want is F1, or F1(pos).
 - Same for accuracy, precision.

- Please use File – print to print your homework as one PDF
 - On your upper left corner, you can find the ‘File’ tab.

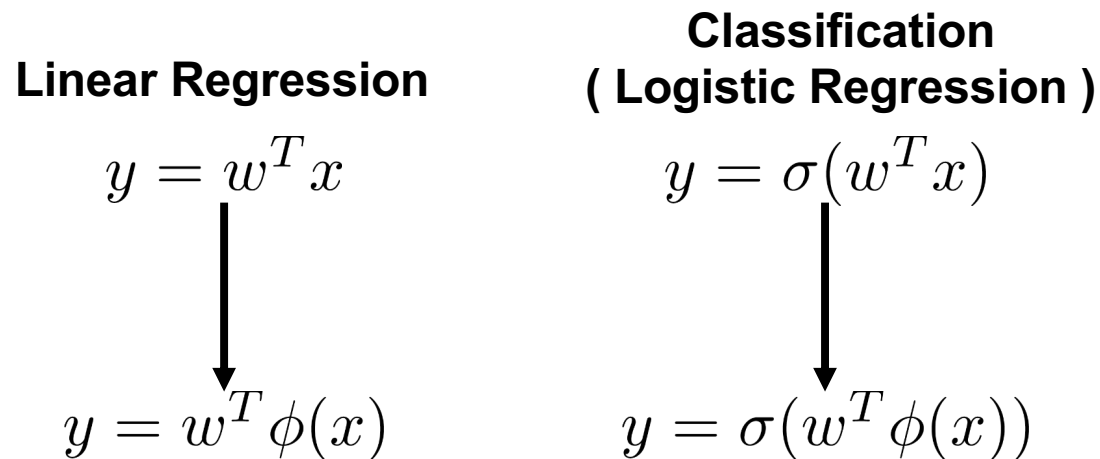
 jupyter hw06_sol L

- Try to merge your PDFs into one file.



- Basis Functions
- Support Vector Machine Classifier
- Kernels
- **Neural Networks**

Basis functions transform linear models into nonlinear ones...

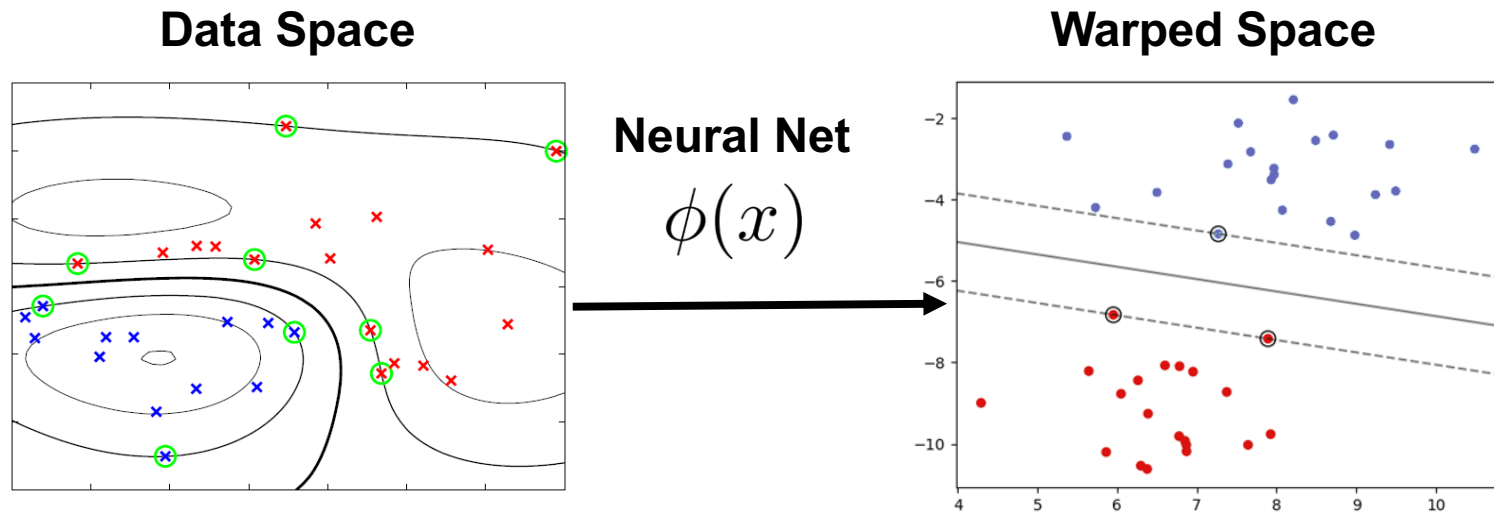


...but it is often difficult to find a good basis transformation

Learning Basis Functions

57

What if we could learn a basis function so that a simple linear model performs well...

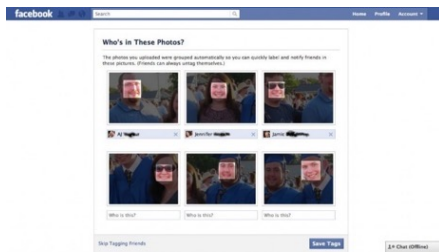


Ignore the circled points...I reused these from the SVM slides

...this is essentially what standard neural networks do...

- Flexible nonlinear transformations of data
- Efficient learning procedure scales to massive data (compared to other nonlinear models)
 - Q: k-NN's space complexity with m data points and D features $O(md)$
- Apply to many Machine Learning / Data Science problems
 - Regression
 - Classification
 - Dimensionality reduction
 - Function approximation
 - Many application-specific problems
 - And exceptional performance on image/voice/natural language

Forms of NNs are used all over the place nowadays...



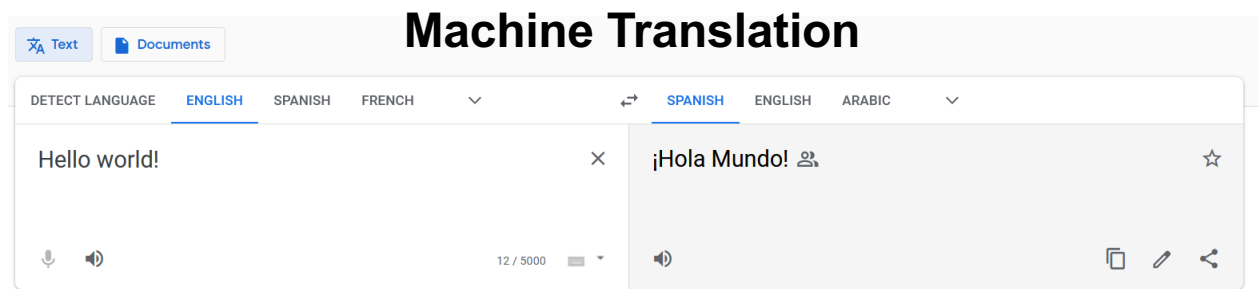
FB Auto Tagging



Self-Driving Cars



Creepy Robots



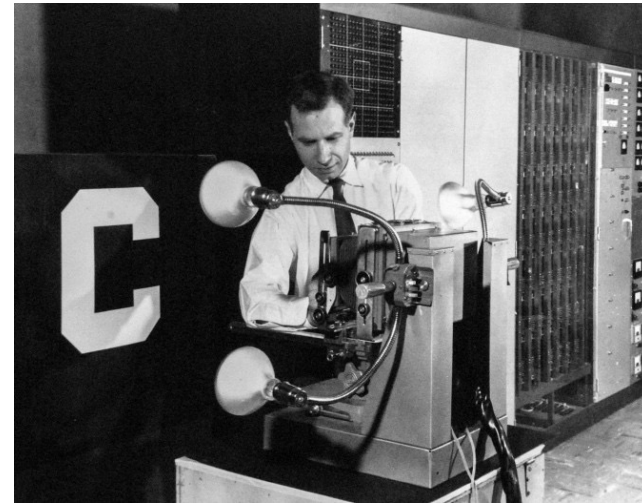
Send feedback

Rosenblatt's Perceptron

60

Despite recent attention, neural networks are fairly old

In 1957 Frank Rosenblatt constructed the first (single layer) neural network known as a “perceptron”



He demonstrated that it is capable of recognizing characters projected onto a 20x20 “pixel” array of photosensors

FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

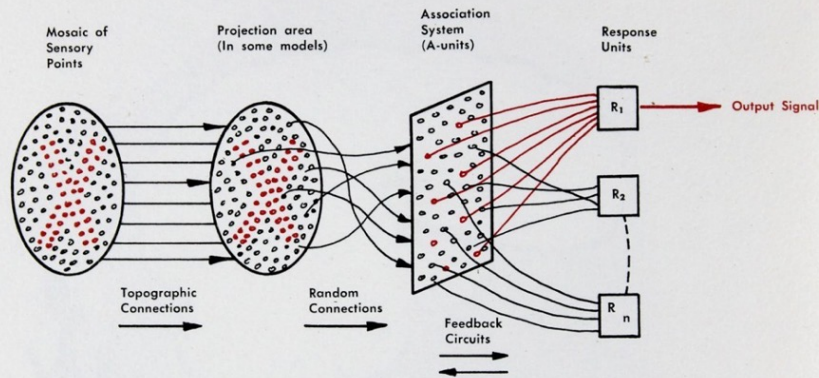
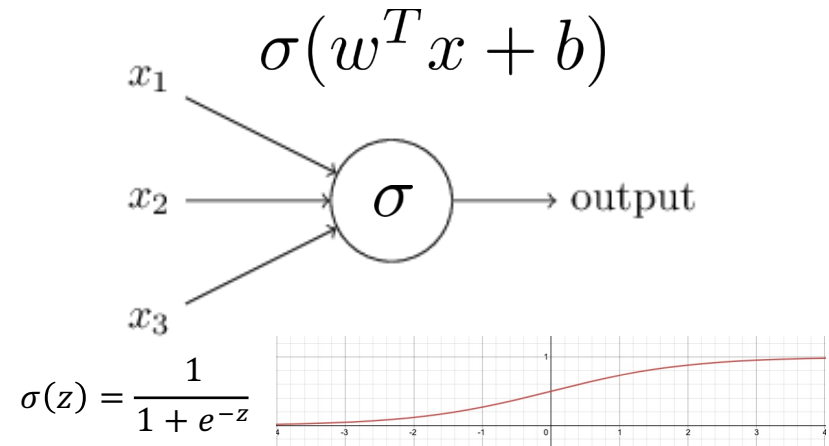


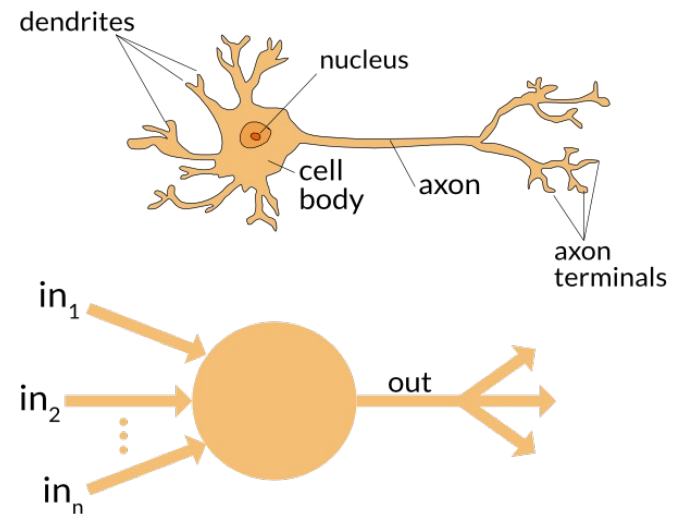
FIG. 2 — Organization of a perceptron.

Perceptron



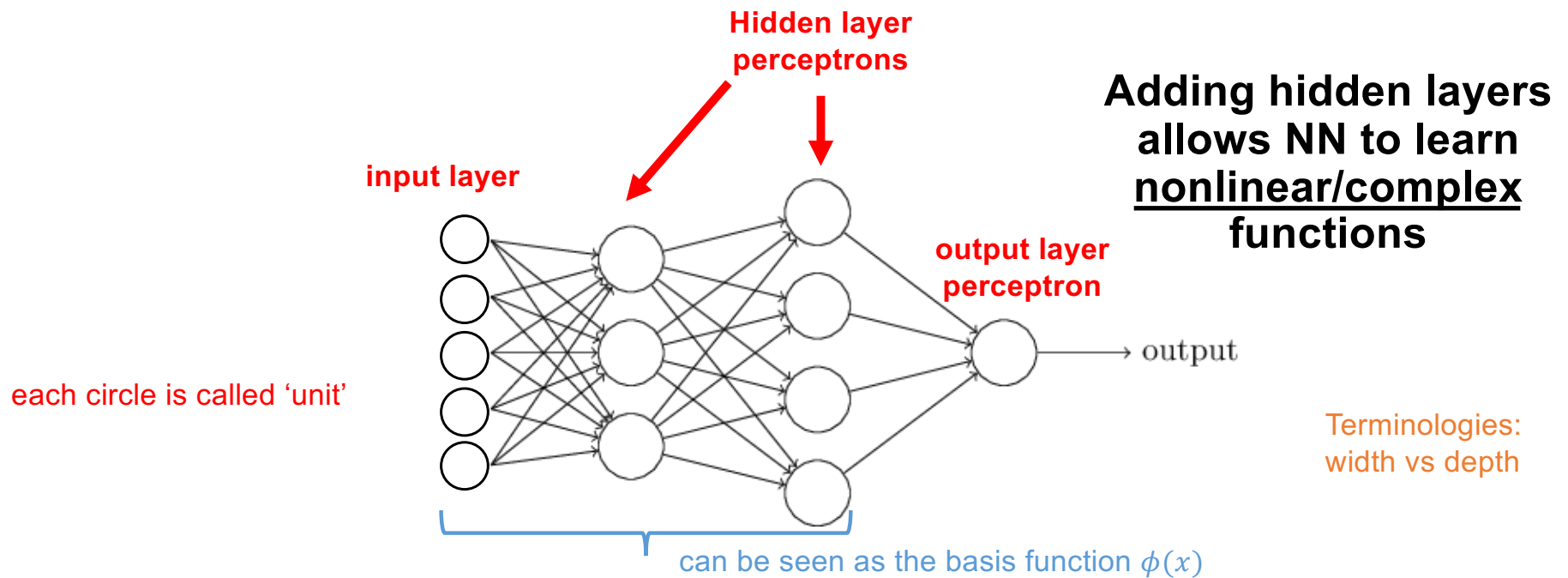
- “Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanics” (1962)
- Criticized by Marvin Minsky in book “Perceptrons” since can only learn linearly-separable functions
- **The perceptron is just logistic regression in disguise**

- It mimics a 'neuron', a unit in our neural system.
 - Takes electric signals from dendrites
 - Doing some processes in the cell body
 - Deliver the electric signal through the axon
- However, one neuron is not enough!
 - Brain is strong since it is a huge network.



Multilayer Perceptron

63



This is the quintessential (*Artificial*) *Neural Network*...

... the image above is a special case called *Feed Forward Neural Net*

feed forward: no backward connection

[Source: <http://neuralnetworksanddeeplearning.com>]

Side note on the history of AI

64

Ongoing battle between: bottom-up vs top-down

- Bottom-up: Connectionist. Mimic the biology of humans/animals.
- Top-down: Symbolic/Logical approach. Mimic how humans reason.

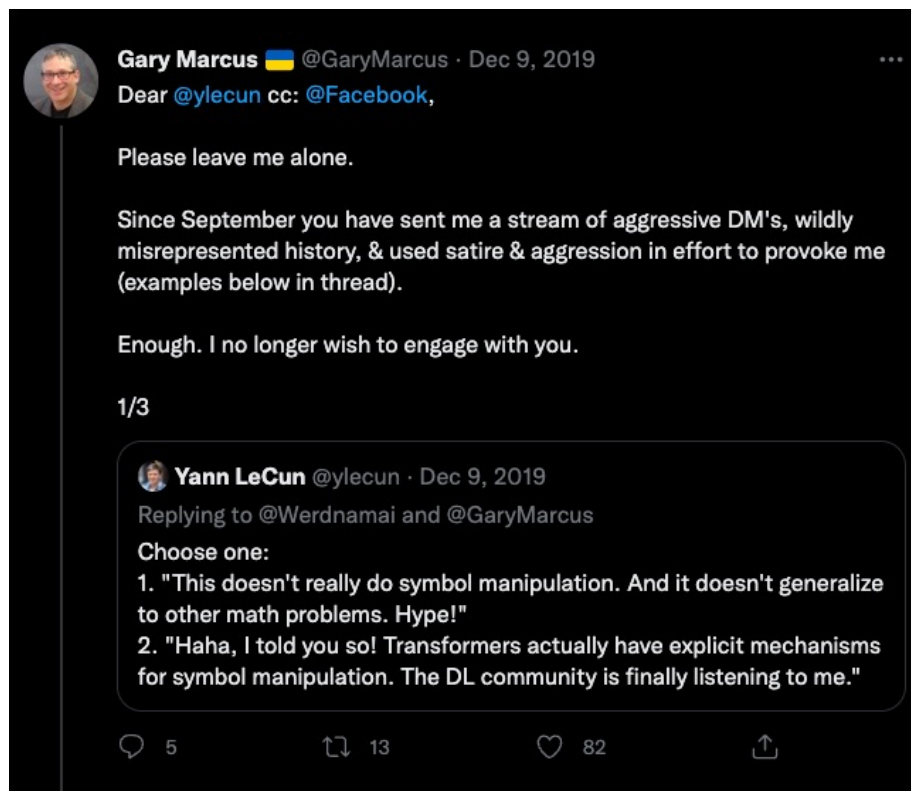
[Bottom-up is the winner nowadays](#); I personally believe that the reasons include computational complexity and availability of data in modern era.

Some battles are still going on, occasionally

65

Yann Lecun: famous neural network researcher

Gary Marcus: Likes symbolic systems. AI scientist, author, and entrepreneur (unclear about his standing as a researcher)



Gary Marcus @GaryMarcus · Dec 9, 2019
Dear @ylecun cc: @Facebook,

Please leave me alone.

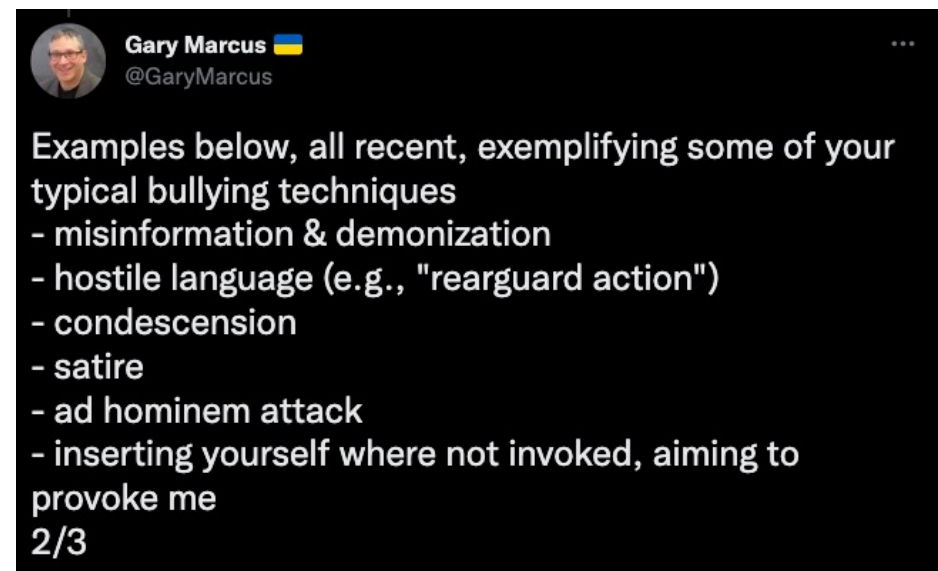
Since September you have sent me a stream of aggressive DM's, wildly misrepresented history, & used satire & aggression in effort to provoke me (examples below in thread).

Enough. I no longer wish to engage with you.

1/3

Yann LeCun @ylecun · Dec 9, 2019
Replying to @Werdnamai and @GaryMarcus
Choose one:
1. "This doesn't really do symbol manipulation. And it doesn't generalize to other math problems. Hype!"
2. "Haha, I told you so! Transformers actually have explicit mechanisms for symbol manipulation. The DL community is finally listening to me."

5 13 82



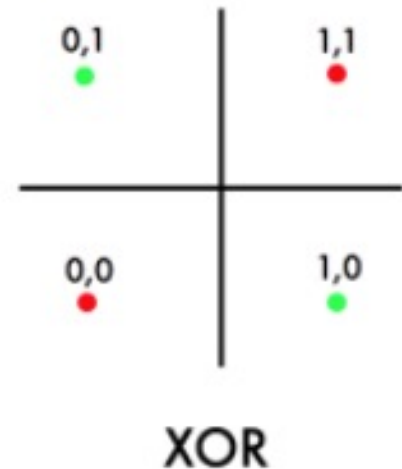
Gary Marcus @GaryMarcus

Examples below, all recent, exemplifying some of your typical bullying techniques

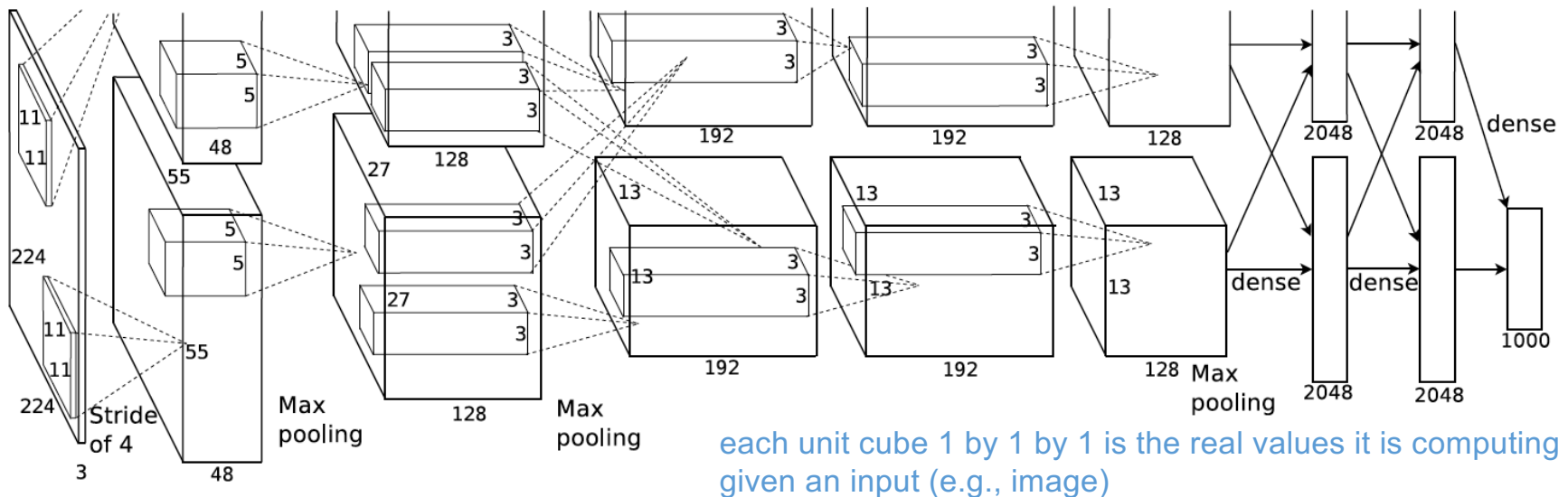
- misinformation & demonization
- hostile language (e.g., "rearguard action")
- condescension
- satire
- ad hominem attack
- inserting yourself where not invoked, aiming to provoke me

2/3

- 60's: early interest in perceptron, but the XOR problem..!
(time for the symbolic camp to laugh at connectionists)
- MLP was a way to get around, but people did not know how to train it
- Werbos'74 breakthrough: **backpropagation** (but still hard to get people back) for training MLP.
- NN became popular again in '86 with McClelland, Rumelhart, and Hinton on training large-scale neural nets.
- Around '97 or so to '12 is a dark time for neural nets; probabilistic models and SVM dominated.
- Circa 2012, neural nets came back as 'deep neural networks'.



Modern *Deep Neural networks* add many hidden layers



...and have many millions of parameters (=weights/biases) to learn

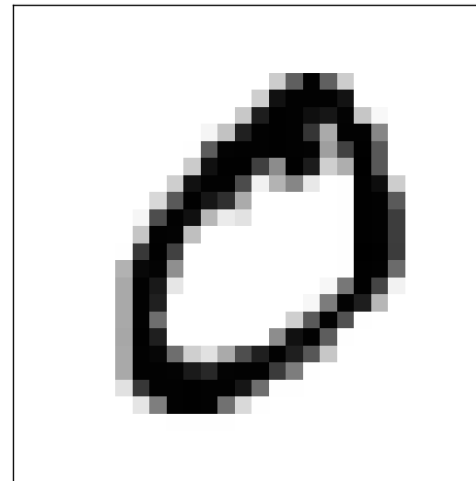
[Source: Krizhevsky et al. (NIPS 2012)]

Classifying handwritten digits is the “Hello World” of NNs



Modified National Institute of Standards and Technology (MNIST) database contains 60k training and 10k test images

Each character is centered in a 28x28=784 pixel grayscale image



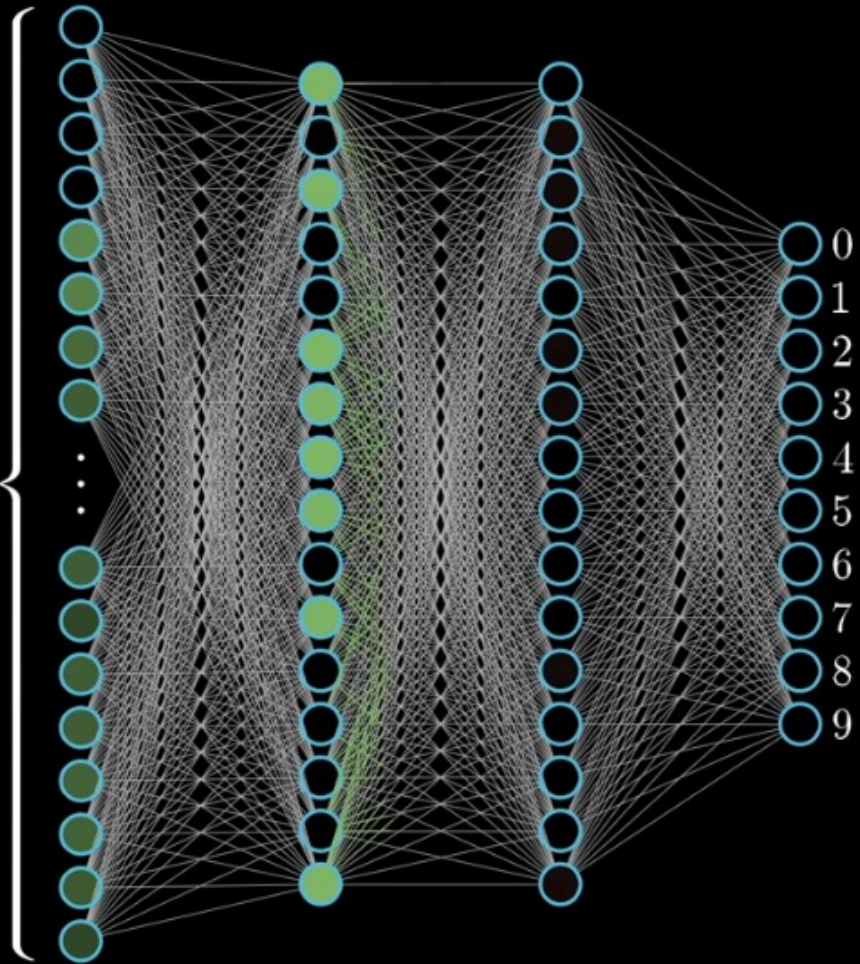
Fun fact: Kernel Ridge Regression with RBF kernel with no regularization gives 1.2% test error rate.

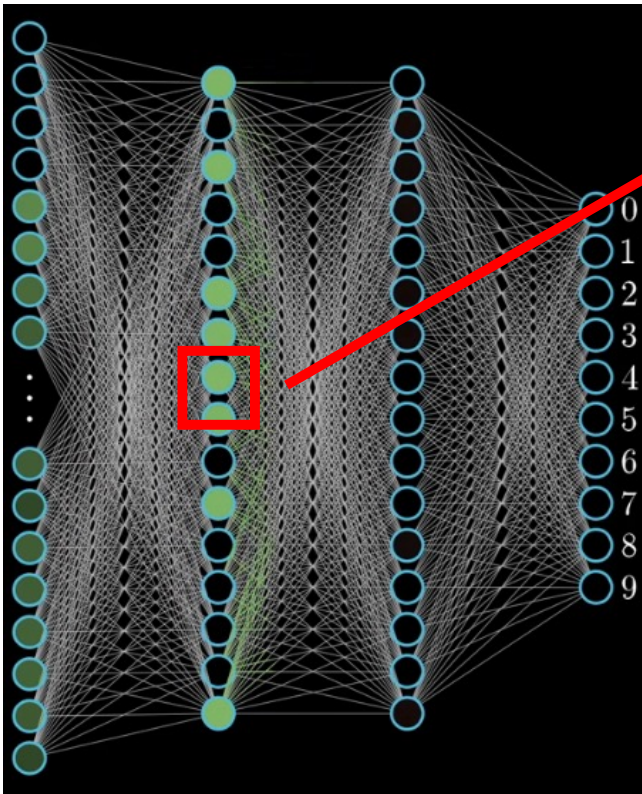
[Source : 3Blue1Brown : <https://www.youtube.com/watch?v=aircAruvNjI>]



784

Each image pixel is a number in $[0,1]$ indicated by highlighted color





Each node computes a *weighted combination* of nodes at the previous layer...

$$w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Then applies a *nonlinear function* to the result

$$\sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n + b)$$

Usually, we also introduce a constant *bias* parameter (usually hidden when we visualize the network)

We call this an *activation function* and typically write it in vector form,

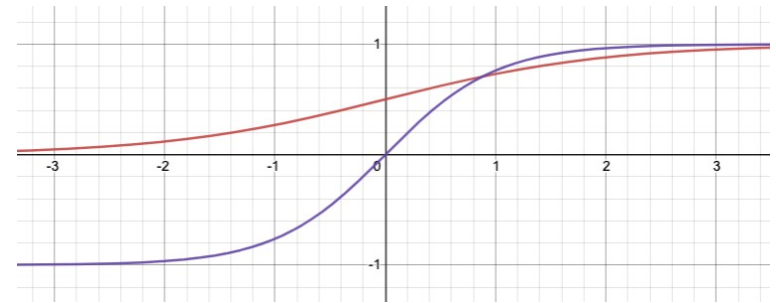
$$\sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n + b) = \sigma(w^T x + b)$$



An early choice was the *logistic function*,

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

Later, people found that a scaled version called **tanh** trains faster (=converges faster)

$$\tanh(z) = 2\sigma(2z) - 1$$



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

Nonlinear Activation functions

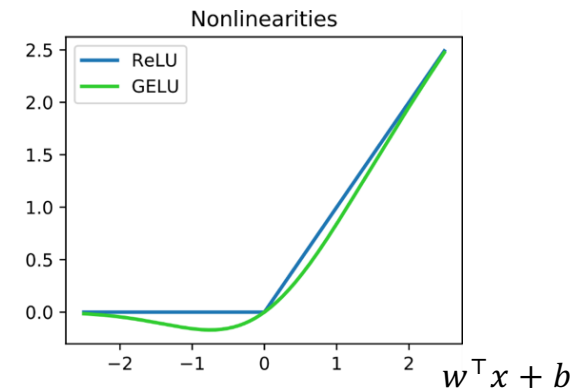
72

Another choice that is found to work even better is the *rectified linear unit (ReLU)*,

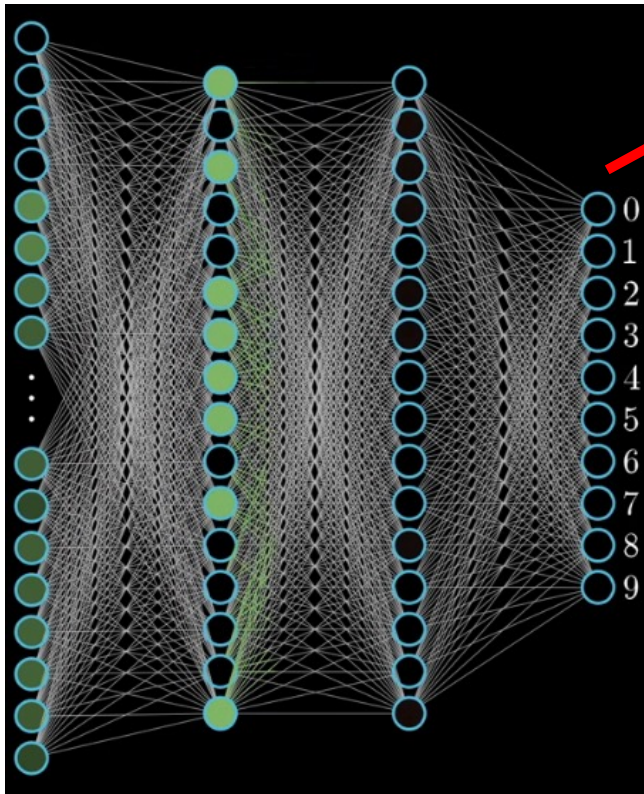
$$\sigma(w^T x + b) = \max(0, w^T x + b)$$

Or the smooth *Gaussian error linear unit (GeLU)*,

$$v = w^T x + b \quad \sigma(v) = v\Phi(v) \quad \leftarrow \text{Gaussian CDF}$$



Generic recommendation: Go with ReLU



Final layer is a linear model...
for classification this is a logistic regression

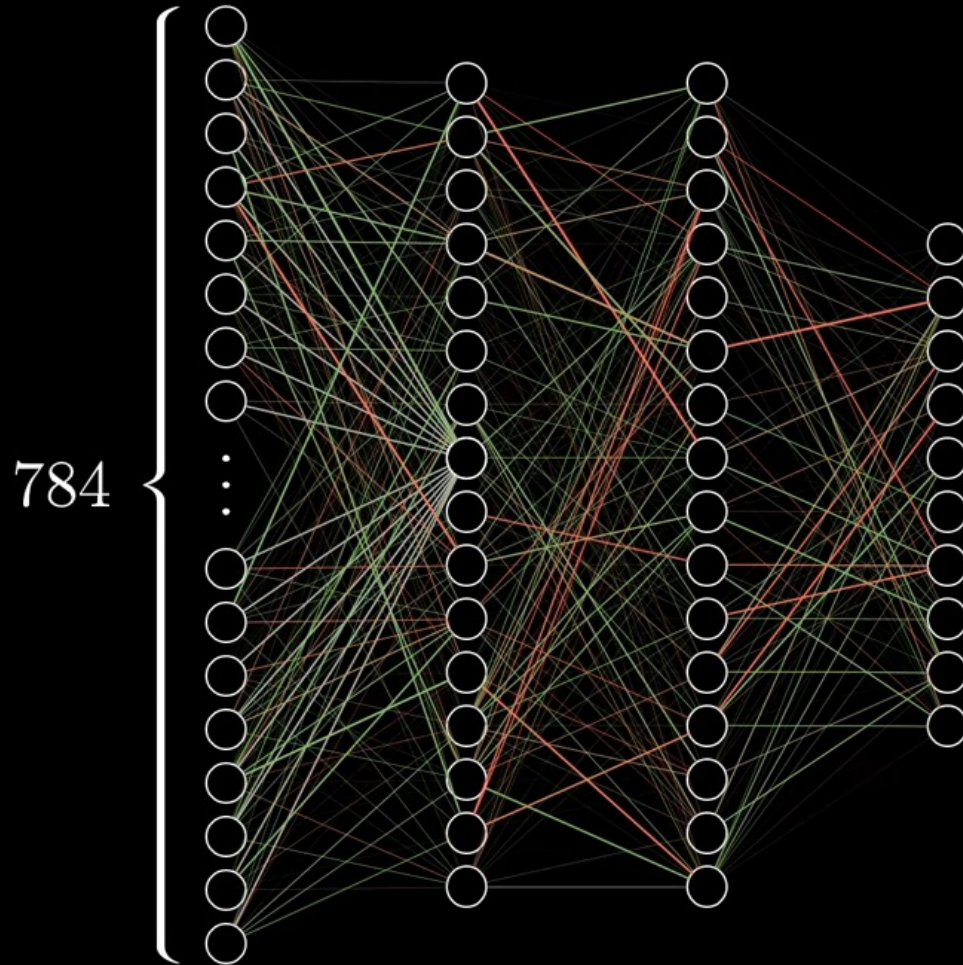
$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

**Vector of activations from
previous layer**

Note: we don't use ReLU for the last layer

[Source : 3Blue1Brown : <https://www.youtube.com/watch?v=aircAruv4j4>]

74




$784 \times 16 + 16 \times 16 + 16 \times 10$
weights

$16 + 16 + 10$
biases

13,002

Each parameter has some impact on the output...need to train all these parameters simultaneously to have a good prediction accuracy

17. A fully-connected feedforward neural network has input $x \in \mathbb{R}^{10}$, a first hidden ReLU layer with 6 units, a second hidden ReLU layer with 5 units, and a single sigmoid output unit. How many parameters are there in the neural network? Don't forget the bias parameters.



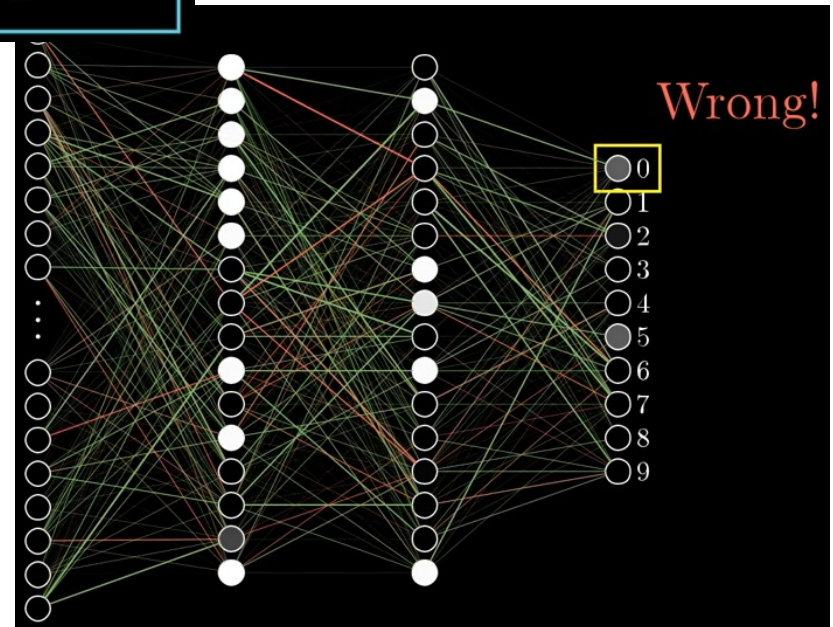
Training Multilayer Perceptron

76

$$X^{\text{Train}} = \begin{matrix} 0 & 4 & 1 & 9 & 2 & 1 & 3 & 1 & 4 & 3 \\ 5 & 3 & 6 & 1 & 7 & 2 & 8 & 6 & 9 & 4 \\ 0 & 9 & 7 & 1 & 2 & 4 & 3 & 2 & 7 & 3 \\ 8 & 6 & 9 & 0 & 5 & 6 & 0 & 7 & 6 & 1 \\ 8 & 7 & 9 & 3 & 9 & 8 & 5 & 9 & 3 & 3 \\ 0 & 7 & 4 & 9 & 8 & 0 & 9 & 4 & 7 & 4 \\ 4 & 6 & 0 & 4 & 5 & 6 & 1 & 0 & 0 & 1 \\ 7 & 1 & 6 & 3 & 0 & 2 & 7 & 7 & 7 & 9 \\ 0 & 2 & 6 & 7 & 8 & 3 & 9 & 0 & 4 & 6 \\ 7 & 4 & 6 & 8 & 0 & 7 & 8 & 3 & 7 & 5 \end{matrix}$$
$$Y^{\text{Train}} = \begin{pmatrix} 0 & 4 & 1 & \dots & 3 \\ 5 & 3 & 6 & \dots & 4 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 7 & 4 & 6 & \dots & 5 \end{pmatrix}$$



For each training example, predict label and adjust weights...



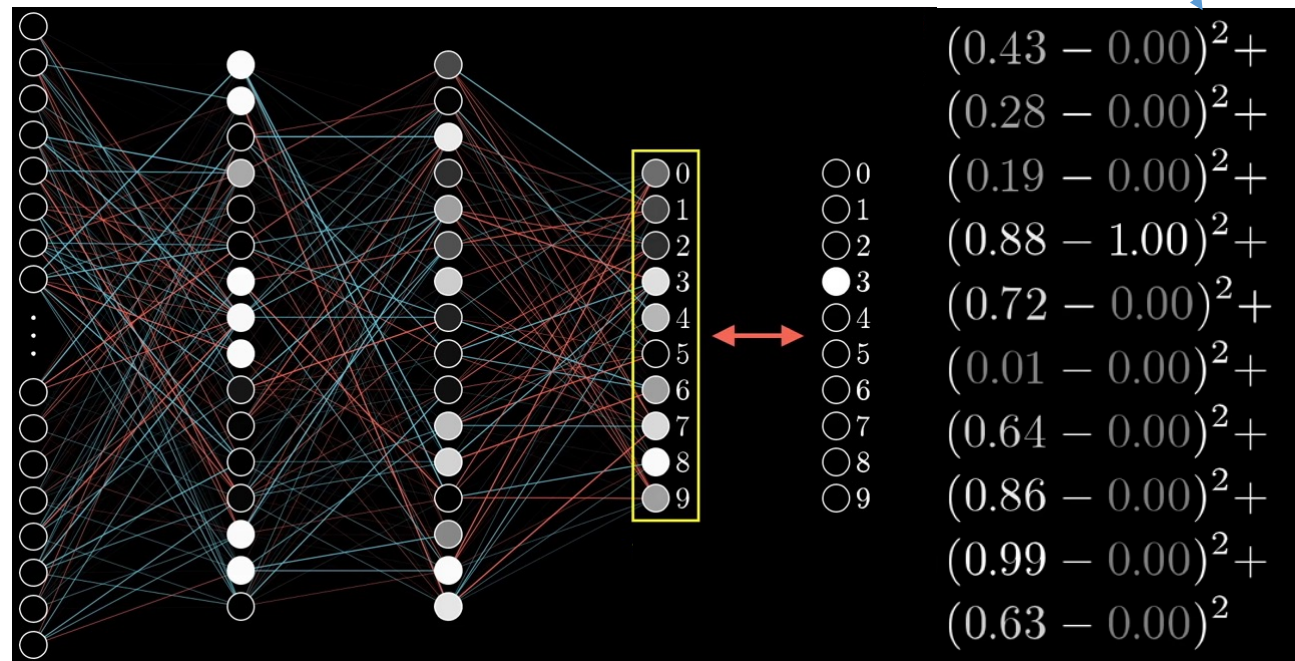
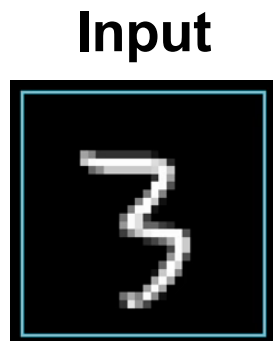
- How to score final layer output?
- How to adjust weights?

Training Multilayer Perceptron

77


Score based on difference between final layer and one-hot vector of true class...

for each of exposition, 3blue1brown uses squared loss, but it should really be logistic loss here.



[Source : 3Blue1Brown : <https://www.youtube.com/watch?v=aircAruvnKk>]

Our cost function for i^{th} input is error in terms of weights / biases...

$$\text{Cost}_i(w_1, \dots, w_n, b_1, \dots, b_n)$$


13,002 Parameters
in this network

...minimize cost over all training data...

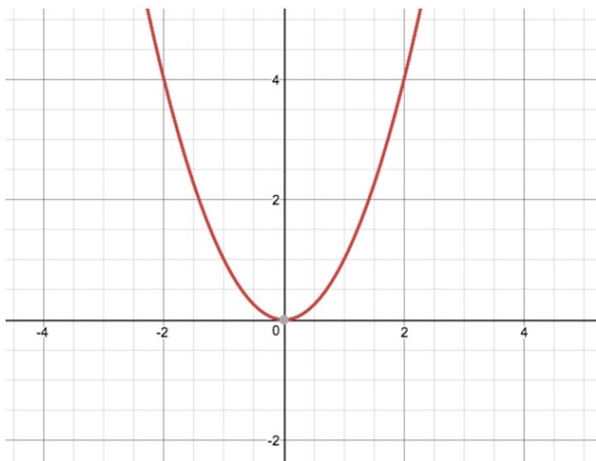
$$\min_{w,b} \mathcal{L}(w, b) = \sum_i \text{Cost}_i(w_1, \dots, w_n, b_1, \dots, b_n)$$

This is a super high-dimensional optimization (13,002 dimensions in this example)...how do we solve it?

Gradient descent!

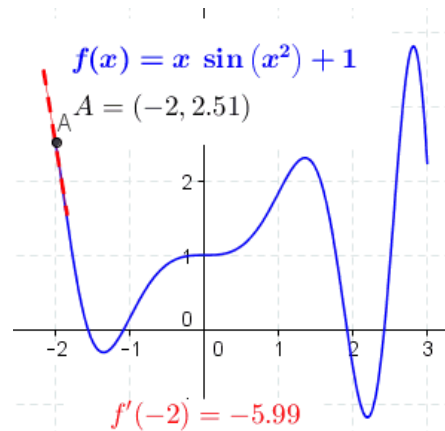
Need to find zero derivative (gradient) solution...

Convex Cost Function



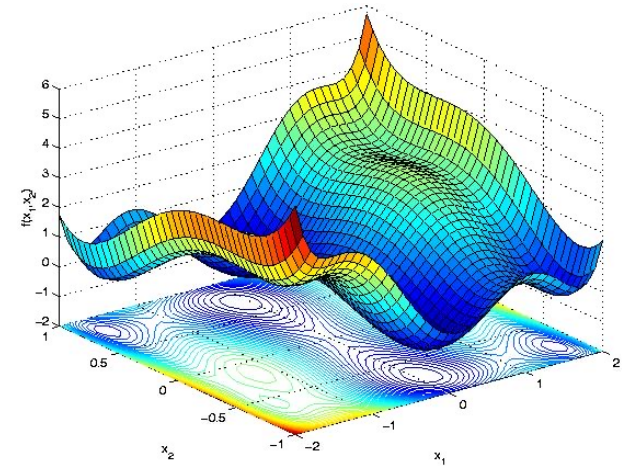
YAY!

Non-convex Cost Function



Boo!

High-Dimensional Non-convex

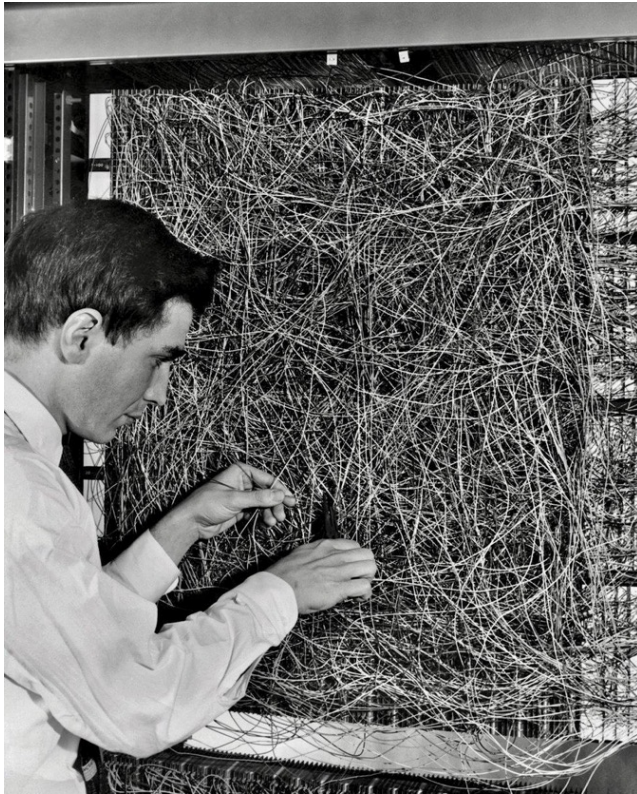


Super Boo!

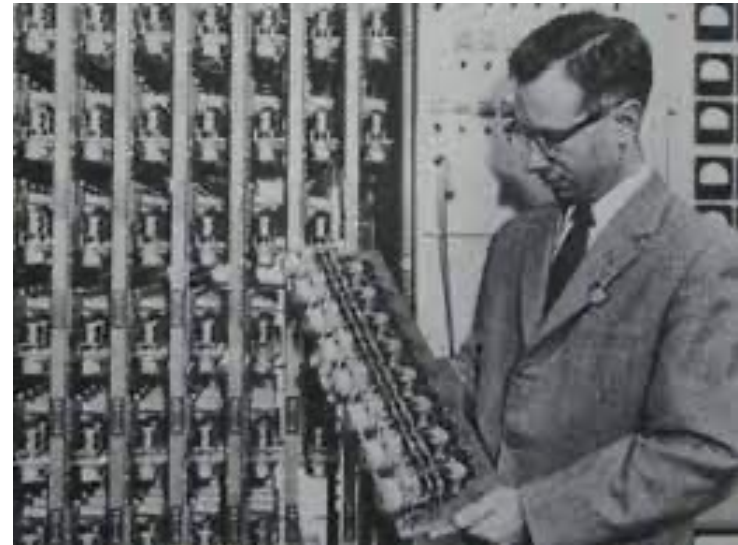
Actually, the situation is much worse, since the cost is super (13,002) high dimensional...

Training the Multilayer Perceptron

80



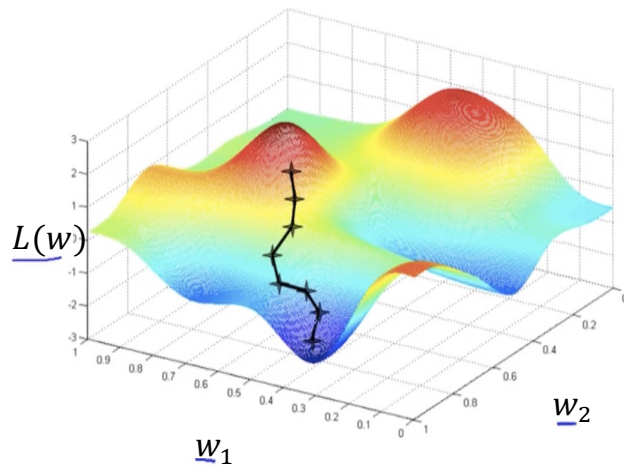
Training the MLP is
challenging...but it's much easier
than how Rosenblatt did it



- In this part we will talk a lot about calculus, and maybe linear algebra.
- These will not be included in your final exam or any evaluations, so please listen casually.

How to minimize a function?

$$\arg \min_w L(w)$$



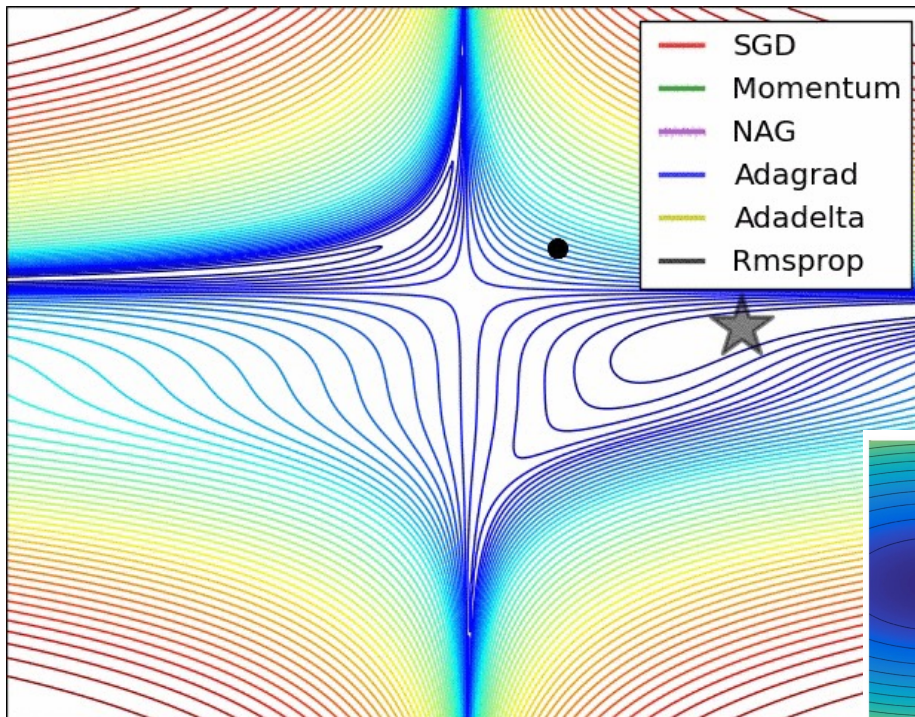
Randomly start from some $w^{(1)} \in \mathbb{R}^d$

For $t = 1, 2, \dots$

- Compute the gradient $g_t \in \mathbb{R}^d$ at the location $w^{(t)}$
- Move to that direction:
$$w^{(t+1)} = w^{(t)} - \eta_t \cdot g^{(t)}$$
where $\eta_t > 0$ is a stepsize parameter.
- If $L(w^{(t+1)}) \approx L(w^{(t)})$, stop.

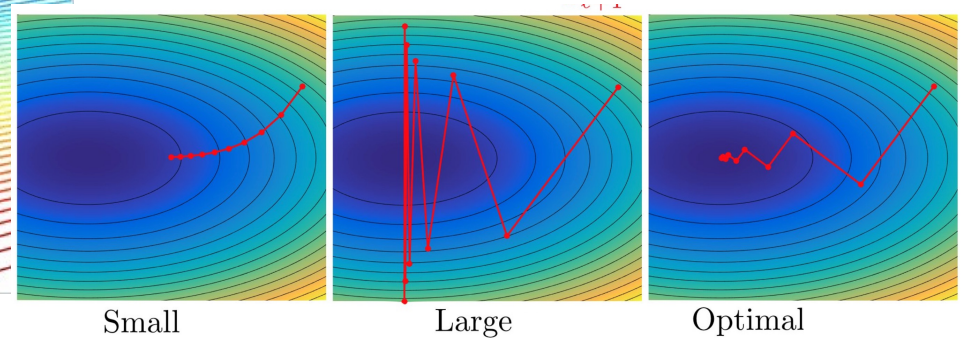
The choice of η_t matters! (default: $\eta_t = 0.01$)

The Importance of Stepsize



each algorithm has a different way to set the stepsize η_t !

SGD: $\eta_t = \text{some constant}$.



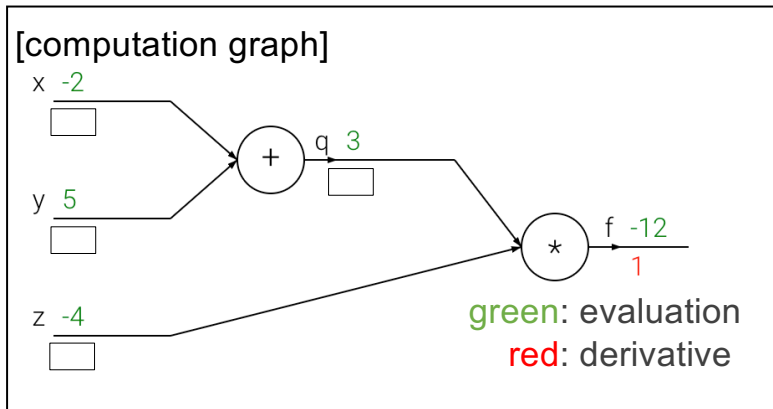
<https://www.datasciencecentral.com/profiles/blogs/an-overview-of-gradient-descent-optimization-algorithms>
<https://twitter.com/gabrielpeyre/status/1233270607518683136/photo/1>

Computing Gradients for Neural Nets

Consider a simpler function.

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

Let's evaluate the gradient at $x = -2, y = 5, z = -4$



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

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

$$\frac{\partial f}{\partial z} = \boxed{}$$

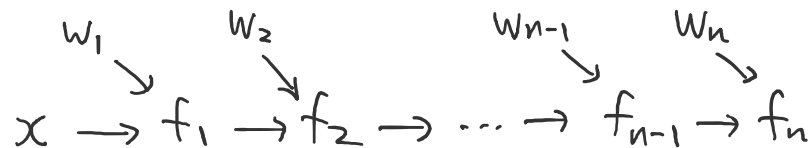
$$\frac{\partial f}{\partial q} = \boxed{} \quad \frac{\partial f}{\partial z} = \boxed{} \quad \frac{\partial q}{\partial x} = \boxed{} \quad \frac{\partial q}{\partial y} = \boxed{}$$

General strategy: 1. forward pass to compute values at each node => we get the value of $f(x,y,z)$,
2. backward pass => we get the gradient at x,y,z

Review: Chain rule

85

General strategy: 1. forward pass to evaluate values at each node => we get the value of $f(x,y,z)$,
2. backward pass => we get the gradient at x,y,z



$$\frac{\partial f_n}{\partial w_1} = \frac{\partial f_n}{\partial f_{n-1}} \frac{\partial f_{n-1}}{\partial f_{n-2}} \dots \frac{\partial f_2}{\partial f_1} \frac{\partial f_1}{\partial w_1}$$

$\underbrace{\hspace{10em}}_{=:\delta(f_1)}$

$$\implies \frac{\partial f_n}{\partial w_1} = \delta(f_1) \frac{\partial f_1}{\partial w_1}$$

Generalize it: $\delta(f_i) = \delta(f_{i+1}) \frac{\partial f_{i+1}}{\partial f_i}, \forall i \in [n-1]$

define $\delta(f_n) = 1$ and compute $\delta(f_{n-1})$, and so on.

- Every function can be written as a computation graph.
- Chain rule says gradient computation can be decomposed into intermediate ones.
 - Essentially, '**divide-and-conquer**' strategy.
- You can implement it with **recursion**.

Python has an implementation of autodiff. E.g., pytorch

Q: Why don't you want to work out gradients by yourself and hard-code it in python?

- It's just... annoying
- Time-consuming
- **Error prone** (so, risk of getting stuck for a long time!)

before autodiff was popular, 80% of the reason why gradient descent does not converge was: mistakes in math!

... but of course, hardcoded ones are faster.

Autodiff example

87

```
# https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html
```

```
import torch
```

```
#- turn on the gradient tracking
```

```
x = torch.tensor([[1.0,2],[3,4]],requires_grad=True)
# requires_grad is False by default
```

```
print(x)
```

np.array, with extra features

```
#-
```

```
y = x + 2
```

```
print(y) #- because we did `requires_grad`,
#-it tracks who created it.
```

```
z = y * y * 3
```

```
out = z.mean()
```

```
print(z)
```

```
print(out)
```

```
z.retain_grad()
```

```
out.backward() # execute backward pass
```

```
print("#- grads")
```

```
print(y.grad)
```

```
print(z.grad)
```

```
print(x.grad)
```

```
tensor([[1., 2.],
        [3., 4.]], requires_grad=True)
```

```
tensor([[3., 4.],
        [5., 6.]], grad_fn=<AddBackward0>)
```

```
tensor([[ 27.,  48.],
        [ 75., 108.]], grad_fn=<MulBackward0>)
```

```
tensor(64.5000, grad_fn=<MeanBackward0>)
```

```
#- grads
```

```
None
```

```
tensor([[0.2500, 0.2500],
        [0.2500, 0.2500]])
```

```
tensor([[4.5000, 6.0000],
        [7.5000, 9.0000]])
```

<ipython-input-40-a3156942a32d>:22: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor is being accessed. Its .grad attribute won't be populated during autograd.backward(). If you indeed want the gradient for a non-leaf Tensor, use .retain_grad() on the non-leaf Tensor. If you access the non-leaf Tensor by mistake, make sure you access the leaf Tensor instead. See github.com/pytorch/pytorch/pull/30531 for more informations.

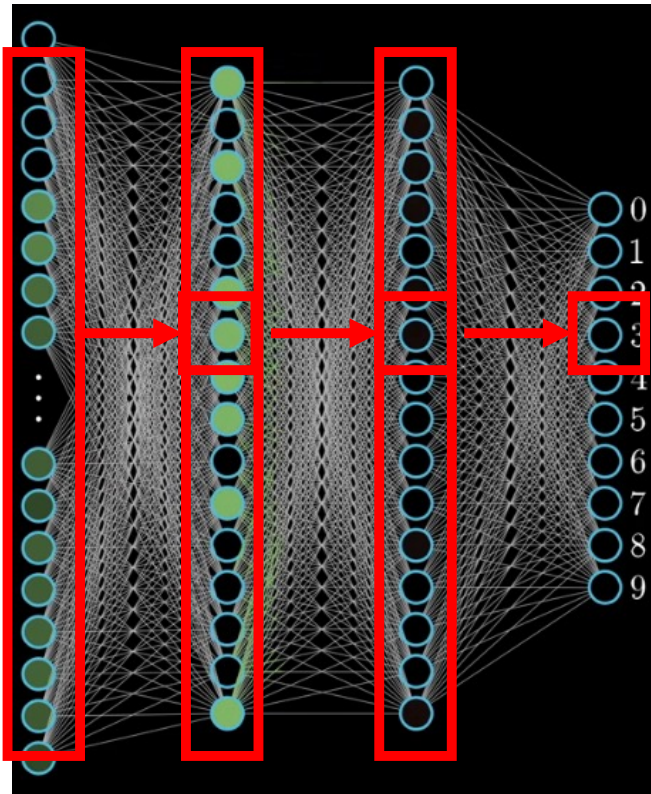
```
print(y.grad)
```

pytorch is optimized for obtaining the gradient of f w.r.t. the input only. If you want to obtain intermediate gradients, you need to use retain_grad().

Backpropagation

88

[Source : 3Blue1Brown : <https://www.youtube.com/watch?v=aircAruvnKk>]



Randomly initialize $\{w^{(u)}\}_{u \in \text{units in neural net}}$

For $i \in \{1, \dots, n_epochs\}$

- For (x,y) in train set:
 - Forward pass:
 - evaluate the neural net output
 - measure the loss
 - Backward pass: compute the gradients.
 - Take the gradient step to update the weights $\{w^{(u)}\}$

Dependencies between layers.

No dependencies between units at the same layer.
⇒ Many GPU supported libraries available.

Play with a small multilayer perceptron on a binary classification task...

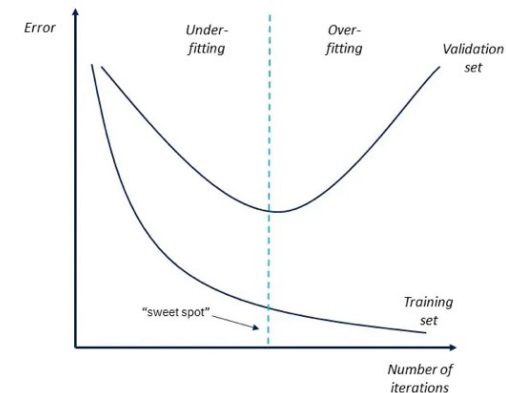
<https://playground.tensorflow.org/>

With four parameters I can fit an elephant. With five I can make him wiggle his trunk. - John von Neumann

$$w = \arg \min_w \text{Cost}(w) + \alpha \cdot \text{Regularizer}(\text{Model})$$

Our example model has 13,002 parameters...that's a lot of elephants!
Regularization is critical to avoid overfitting...

...numerous regularization schemes are used in training neural networks.
but the standard is of course, $\sum_i w_i^2$



hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*

The *i*th element represents the number of neurons in the *i*th hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Activation function for the hidden layer.

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*

The solver for weight optimization.

alpha : *float, default=0.0001*

L2 penalty (regularization term) parameter.

learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*

Learning rate schedule for weight updates.

early_stopping : *bool, default=False*

Whether to use early stopping to terminate training when validation score is not improving. If set to true,

Fetch MNIST data from www.openml.org :

```
X, y = fetch_openml("mnist_784", version=1, return_X_y=True)
X = X / 255.0
```

Train test split (60k / 10k),

```
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Create MLP classifier instance,

- Single hidden layer (50 nodes)
- Use stochastic gradient descent
- Maximum of 10 learning iterations
- Small L2 regularization $\alpha=1e-4$

```
mlp = MLPClassifier(
    hidden_layer_sizes=(50,),
    max_iter=10,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate_init=0.1,
)
```


Fit the MLP and print stuff...

```
mlp.fit(X_train, y_train)
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

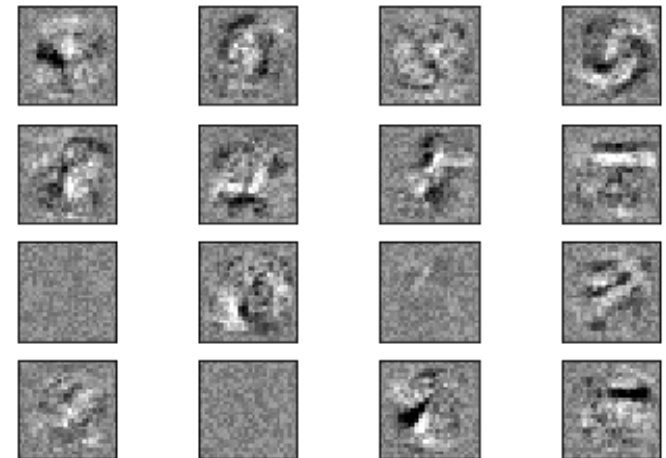
```
Iteration 1, loss = 0.32009978
Iteration 2, loss = 0.15347534
Iteration 3, loss = 0.11544755
Iteration 4, loss = 0.09279764
Iteration 5, loss = 0.07889367
Iteration 6, loss = 0.07170497
Iteration 7, loss = 0.06282111
Iteration 8, loss = 0.05530788
Iteration 9, loss = 0.04960484
Iteration 10, loss = 0.04645355
Training set score: 0.986800
Test set score: 0.970000
```

Visualize the weights for each node...

`coefs_[i]`: $n_{\text{input}} \times n_{\text{output}}$ matrix of weights for layer i

```
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray,
               |vmin=0.5 * vmin, vmax=0.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())
```

...magnitude of weights indicates which input features are important in prediction



Many other NN architectures exist beyond MLP

- **Convolutional NN (CNN)** For image processing / computer viz.
- **Recurrent NN (RNN)** For sequence data (e.g. acoustic signals, video, etc.) , long short-term memory (LSTM) is popular
- **Generative Adversarial Nets (GANs)** For generating creepy deepfakes
- **Restricted Boltzmann Machine (RBM)** Another generative model

Many open areas being researched

- More reliable uncertainty estimates
- Robustness to exploits
- Interpretability
- Better scalability



There are **tons** of excellent resources for learning about neural networks online...here are two quick ones:

3Blue1Brown Youtube channel has a nice four-part intro:

<https://www.youtube.com/watch?v=aircAruvnKk>

Free book by Michael Nielson uses MNIST example in Python:

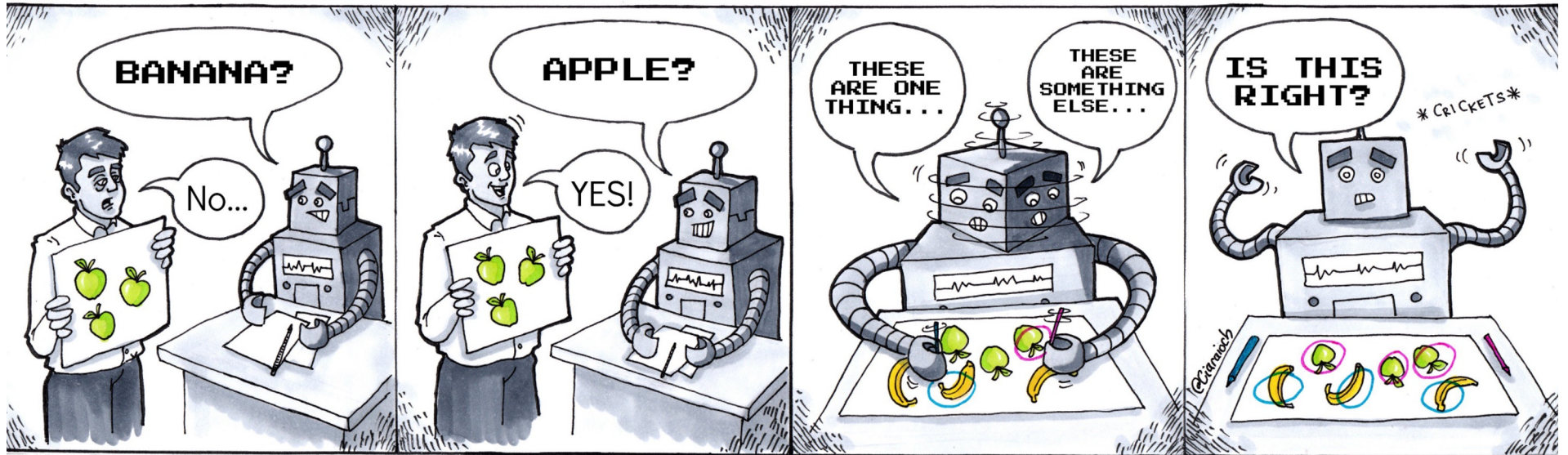
<http://neuralnetworksanddeeplearning.com/>

Prof. Steven Bethard often teaches an excellent class:

ISTA 457 / INFO 557

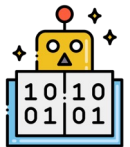
- Raise your hand if you prefer ‘the earlier upload of HW7’
 - The deadline will be the same (April 21st)
 - If you prefer, I will upload HW7 tonight.

- Learning with unlabeled data
- What can we expect to learn?
 - **Clustering**: obtain partition of the data that are well-separated.
 - can be viewed as a preliminary classification without predefined class labels.
 - **Components**: extract common components that compose data points.
 - e.g., topic modeling given a set of articles: each article talks about a few topics => extract the topics that appear frequently.
- Use
 - As a summary of the data
 - **Exploratory data analysis**: what are the **patterns** we can get even without labels?
 - Often used as a ‘preprocessing techniques’
 - e.g., extract useful **features** using soft clustering assignments (e.g., “gaussian mixture model”)



Supervised Learning

Unsupervised Learning



Task 1 : Group These Set of Document into 3 Groups based on meaning

Doc1 : Health , Medicine, Doctor

Doc 2 : Machine Learning, Computer

Doc 3 : Environment, Planet

Doc 4 : Pollution, Climate Crisis

Doc 5 : Covid, Health , Doctor



Task 1 : Group These Set of Document into 3 Groups based on meaning

Doc1 : Health , Medicine, Doctor

Doc 2 : Machine Learning, Computer

Doc 3 : Environment, Planet

Doc 4 : Pollution, Climate Crisis

Doc 5 : Covid, Health , Doctor



Task 1 : Group These Set of Document into 3 Groups based on meaning

Doc1 : Health , Medicine, Doctor

Doc 5 : Covid, Health , Doctor

Doc 3 : Environment,
Planet

Doc 4 : Pollution, Climate
Crisis








Doc 2 : Machine
Learning, Computer

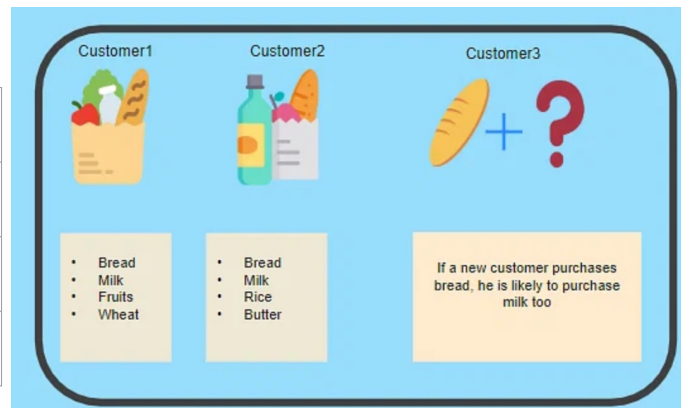


Task 2: Recommendation

10
2

- Discover the probability of the co-occurrence of items in a collection
 - Market basket analysis
 - Semantic clustering (Topic modeling)
 - Movie recommendation

	 Harry Potter	 The Triplets of Belleville	 Shrek	 The Dark Knight Rises	 Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
			?	✓	✓



From: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/supervised-and-unsupervised-learning>
 And <https://developers.google.com/machine-learning/recommendation/collaborative/basics>