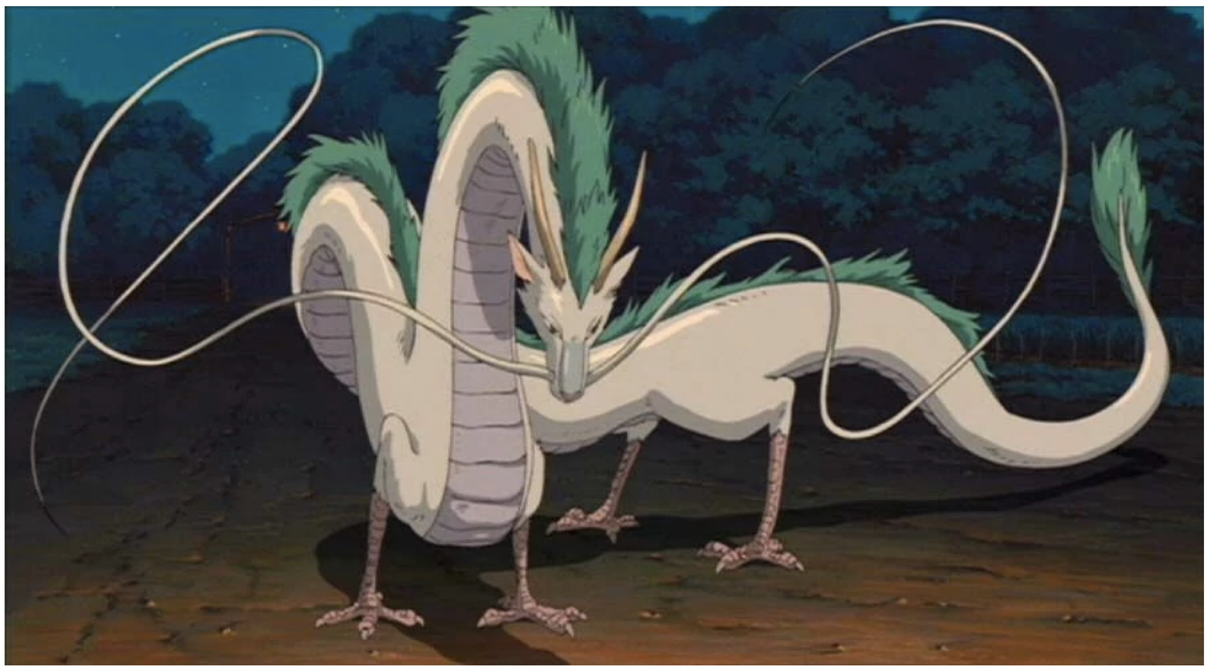
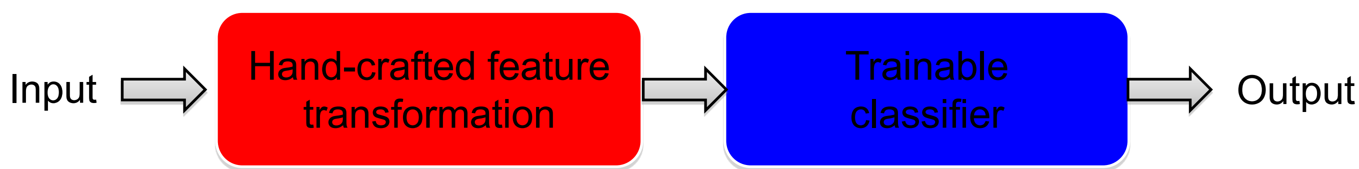
Multi-layer networks, hyperparameter search, validation

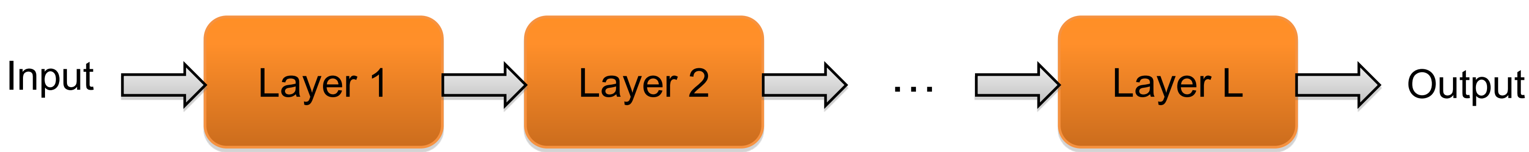


From linear to nonlinear classifiers

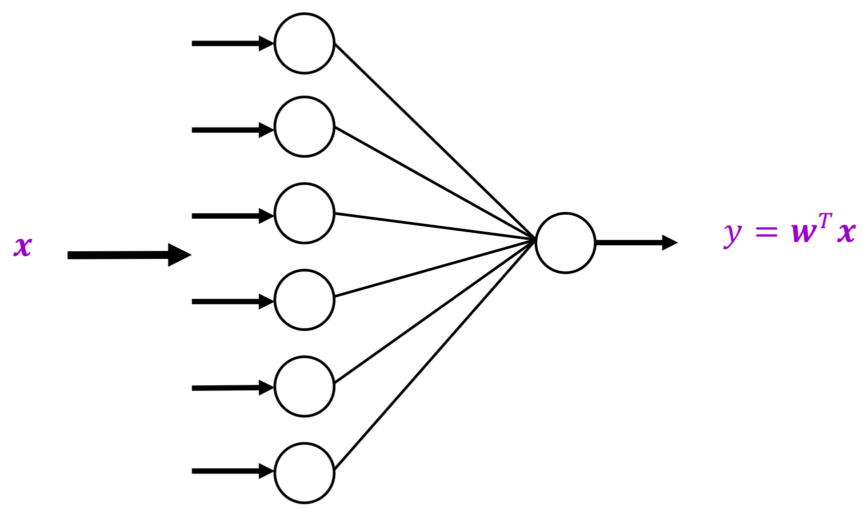
- To achieve good accuracy on challenging problems, we need to be able to train *nonlinear* models
- Two strategies for making nonlinear predictors out of linear ones:
  - **"Shallow" approach:** hand-crafted feature transformation (typically nonlinear, non-trainable) followed by trainable classifier (typically linear)



  - **"Deep" approach:** stack multiple layers of linear predictors (interspersed with nonlinearities)

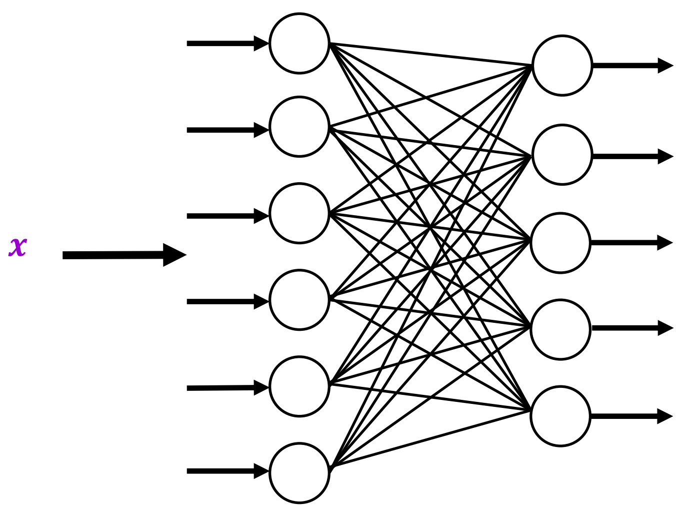


From linear classifiers to multi-layer networks



From linear classifiers to multi-layer networks

Linear layer

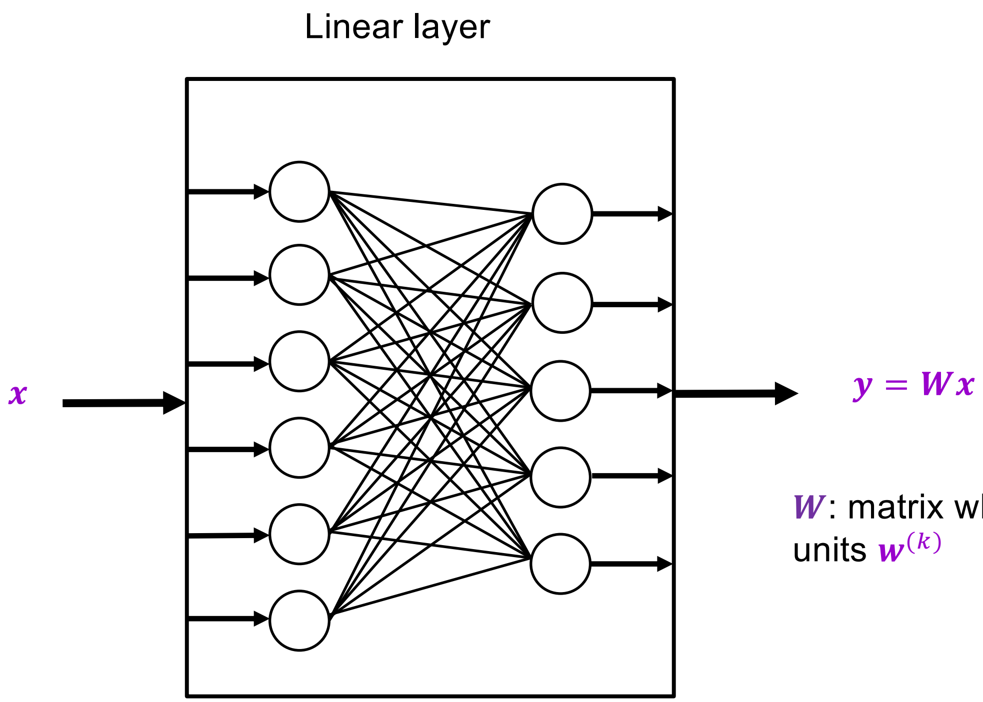


$$y^{(1)} = \boldsymbol{w}^{(1)} \cdot \boldsymbol{x}$$

$$y^{(2)} = \boldsymbol{w}^{(2)} \cdot \boldsymbol{x}$$

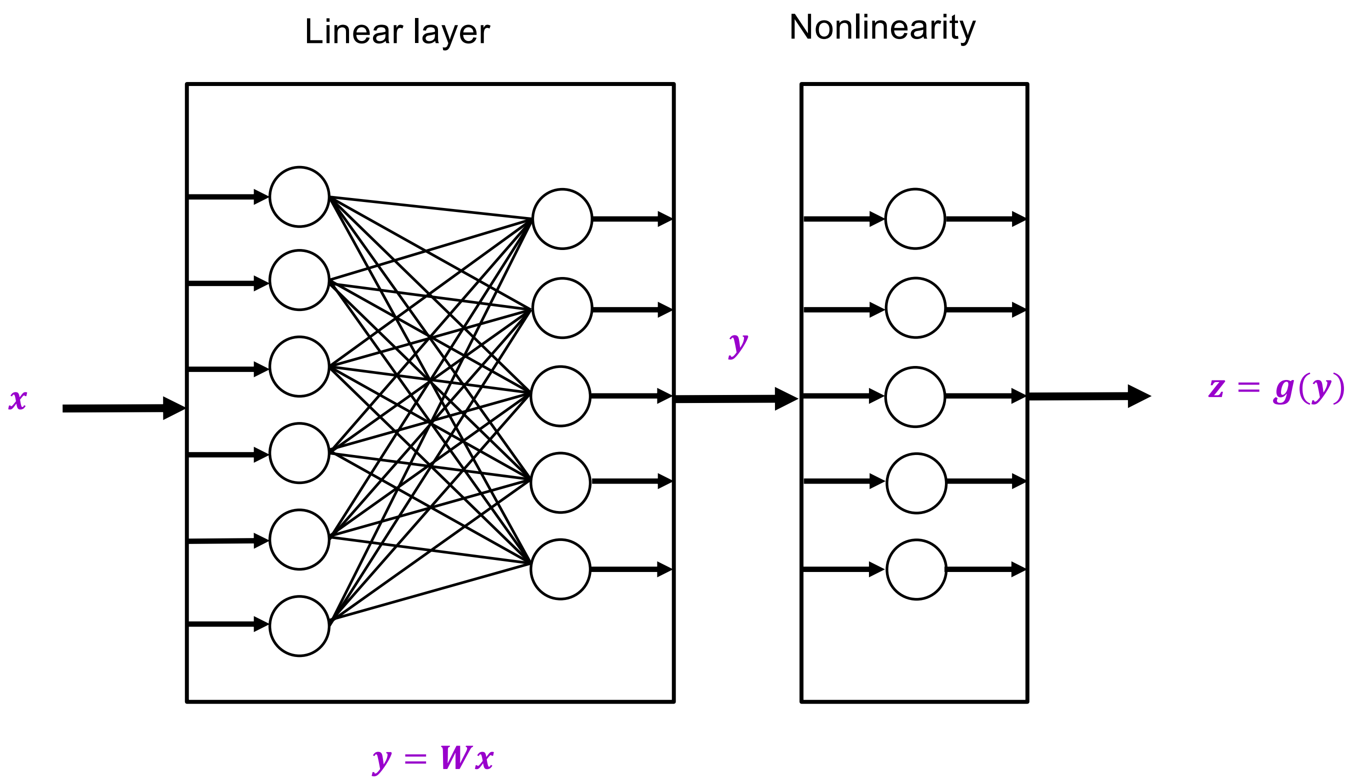
$$\vdots$$

From linear classifiers to multi-layer networks



W: matrix whose rows are weights of output units $w^{(k)}$

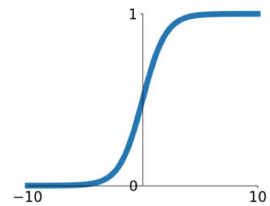
From linear classifiers to multi-layer networks



Common nonlinearities (or *activation functions*)

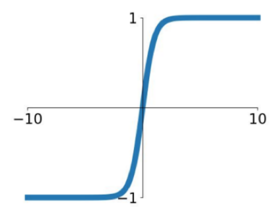
Sigmoid

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



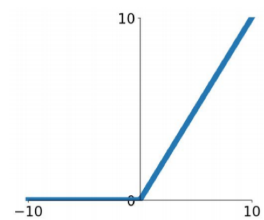
tanh

$\tanh(x)$



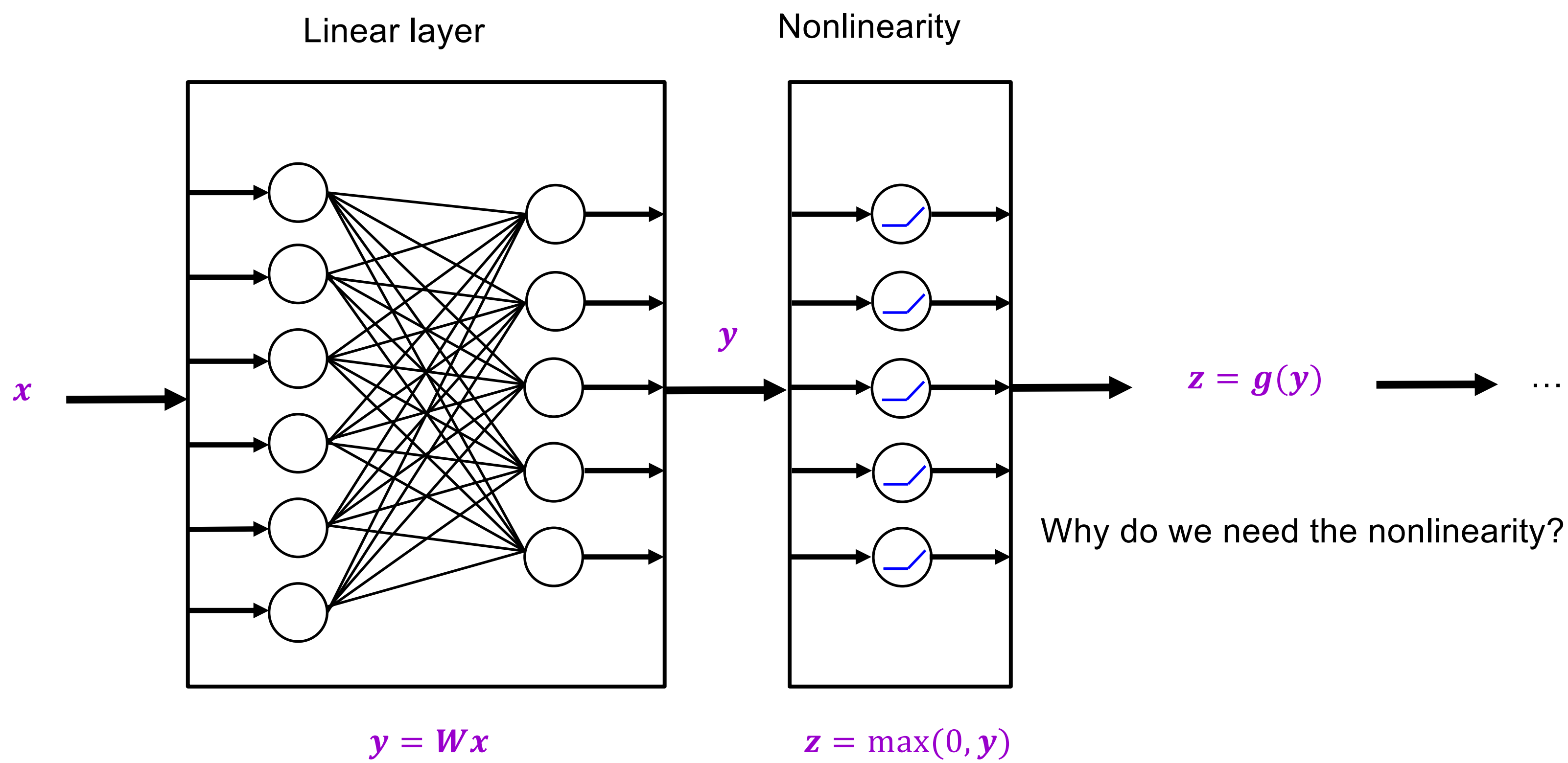
ReLU

$\max(0, x)$



Source: [Stanford 231n]

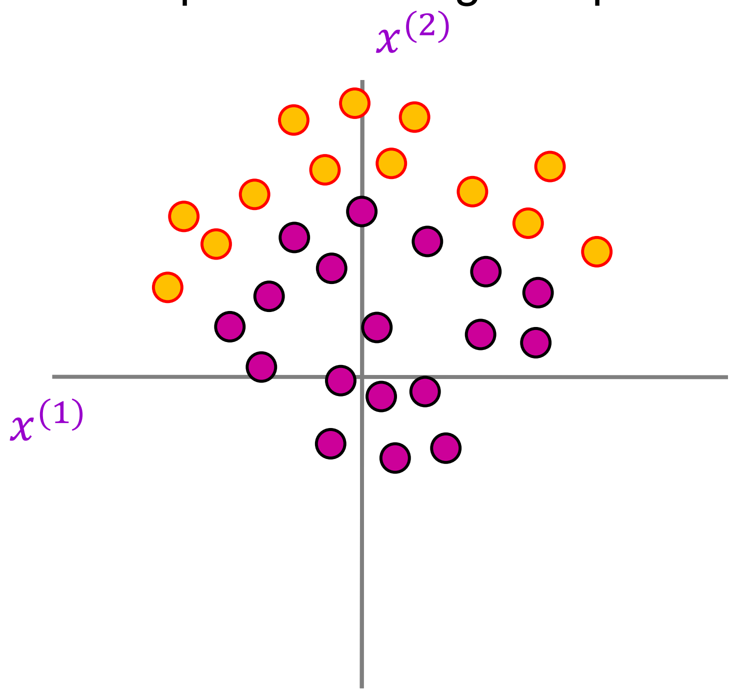
From linear classifiers to multi-layer networks



Why do we need the nonlinearity?

The power of nonlinearities

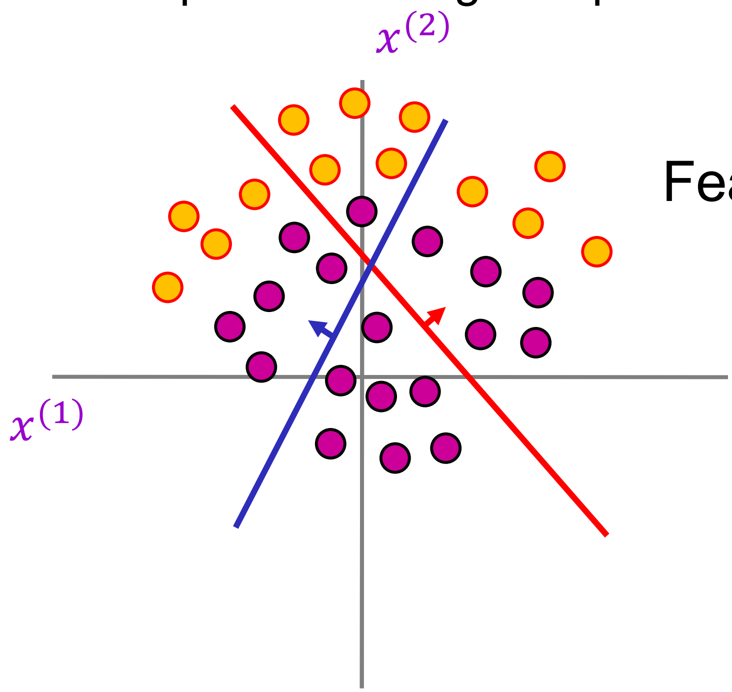
Points not linearly
separable in original space



Source: [J. Johnson]

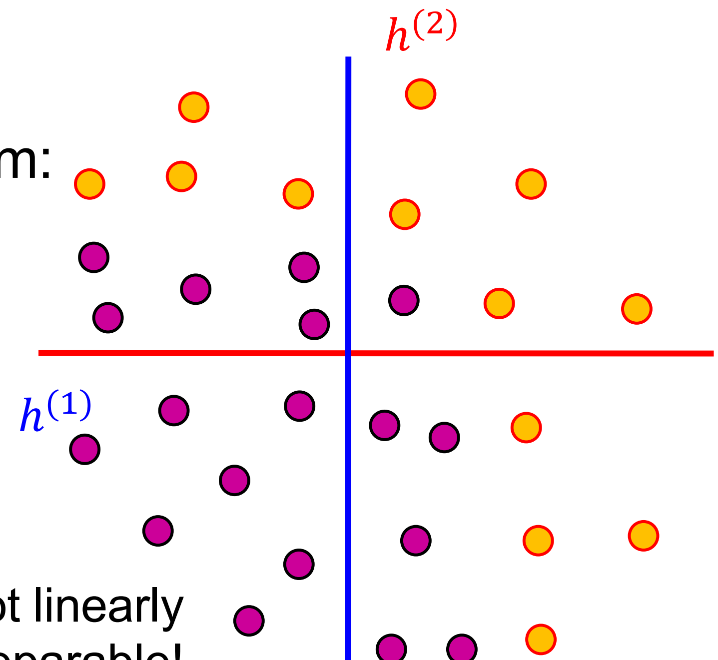
The power of nonlinearities

Points not linearly separable in original space



Consider a linear transform: $h = Wx + b$
Where x, h, b are 2-dimensional

Feature transform:
$h = Wx + b$



Still not linearly separable!

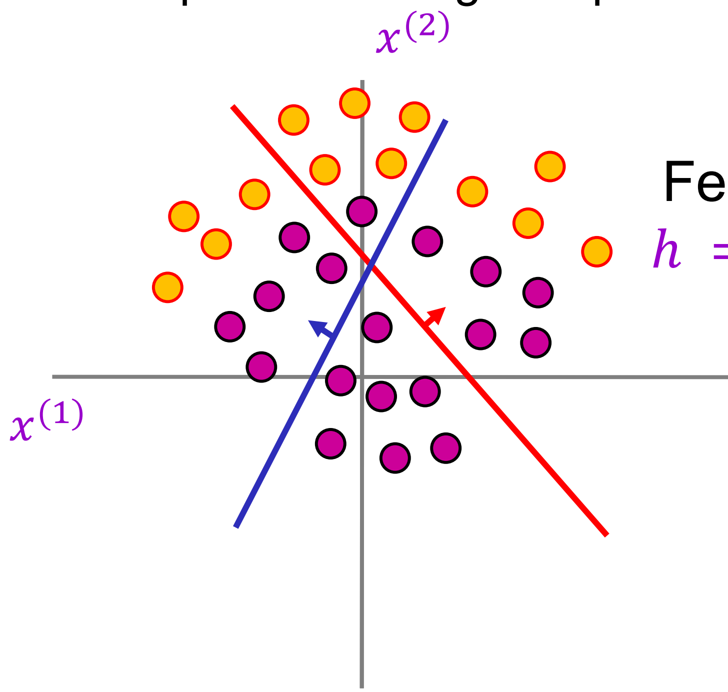
Source: [J. Johnson]

The power of nonlinearities

Points not linearly separable in original space

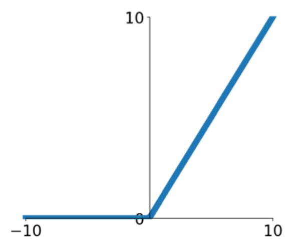
Let's add a nonlinearity:

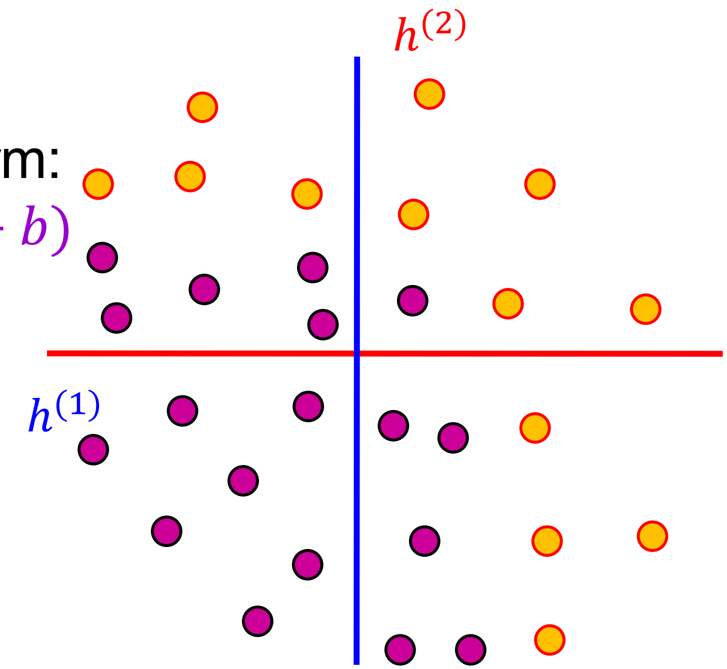
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$

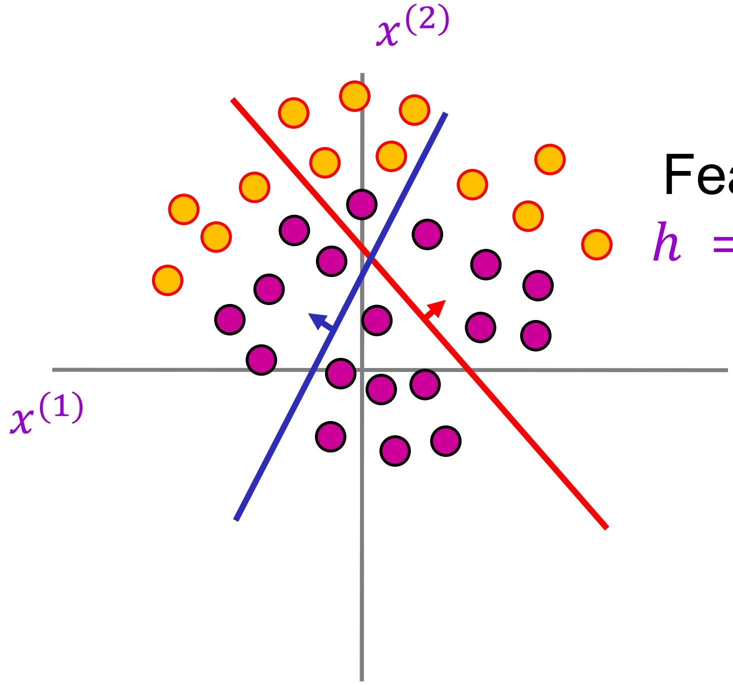




Source: [J. Johnson]

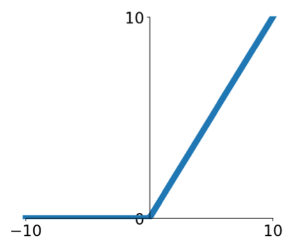
The power of nonlinearities

Points not linearly separable in original space



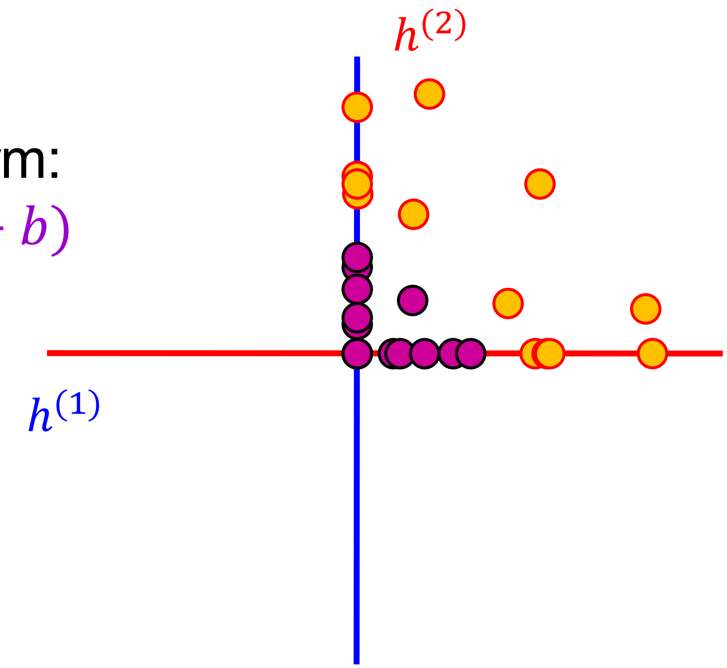
Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$



Let's add a nonlinearity:

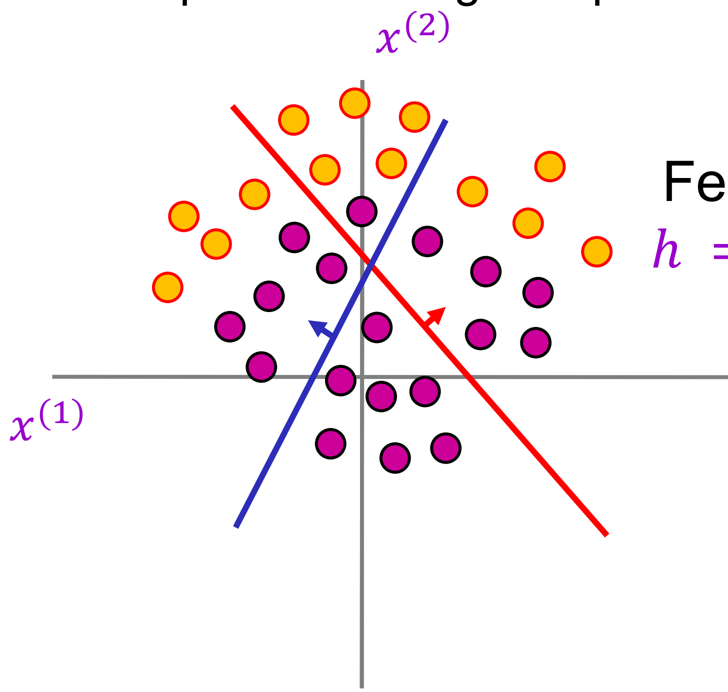
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Source: [J. Johnson]

The power of nonlinearities

Points not linearly separable in original space

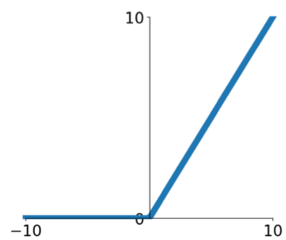


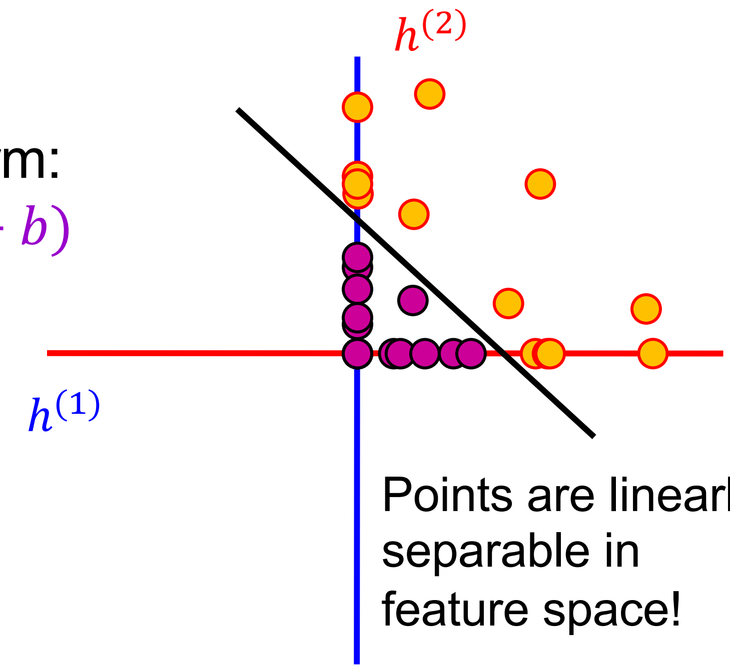
Let's add a nonlinearity:

$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$

Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$





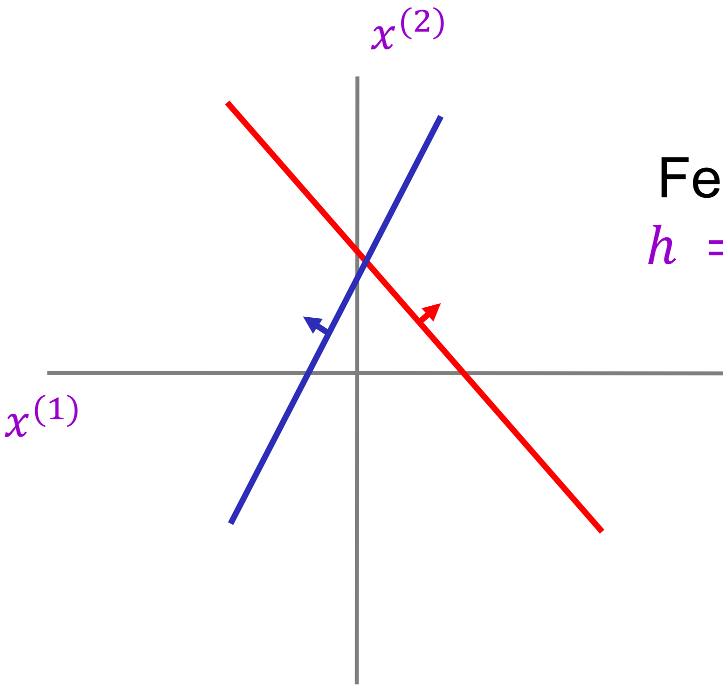
Points are linearly separable in feature space!

Source: [J. Johnson]

The power of nonlinearities

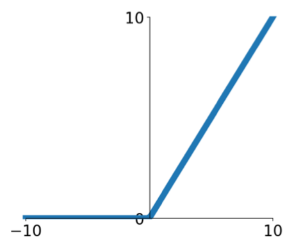
Let's add a nonlinearity:

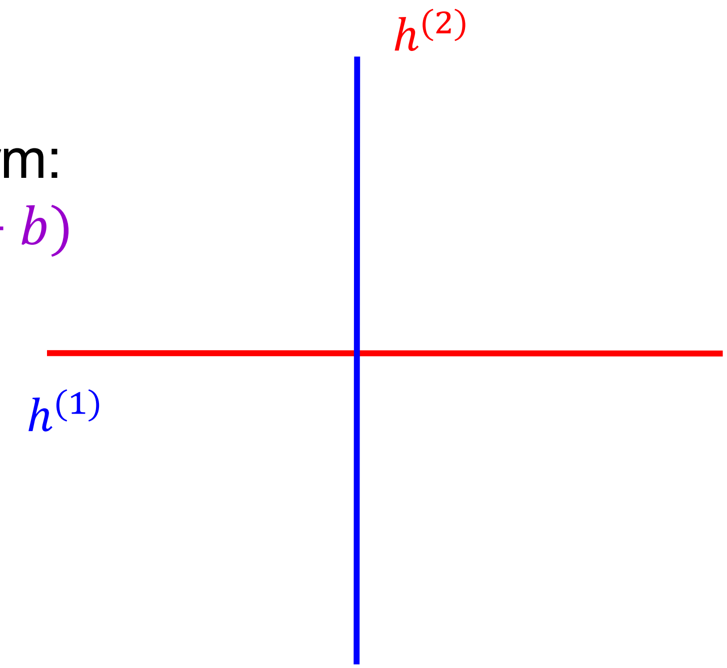
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$



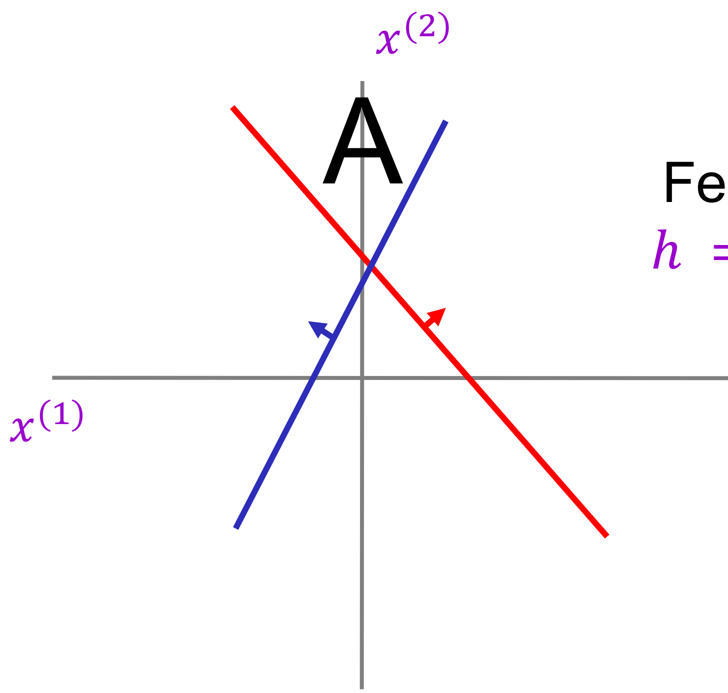


Source: [J. Johnson]

The power of nonlinearities

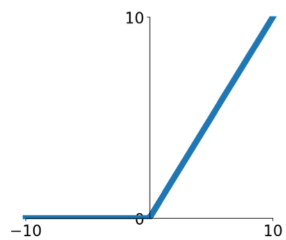
Let's add a nonlinearity:

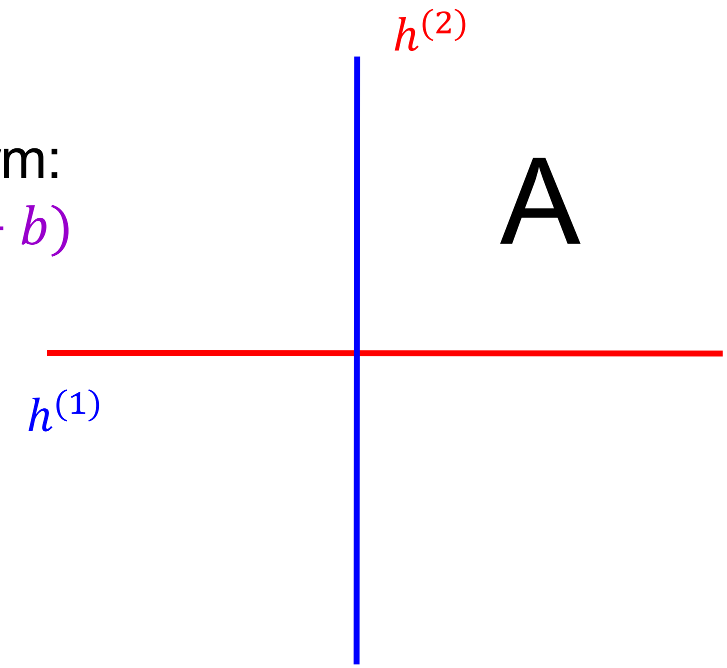
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$



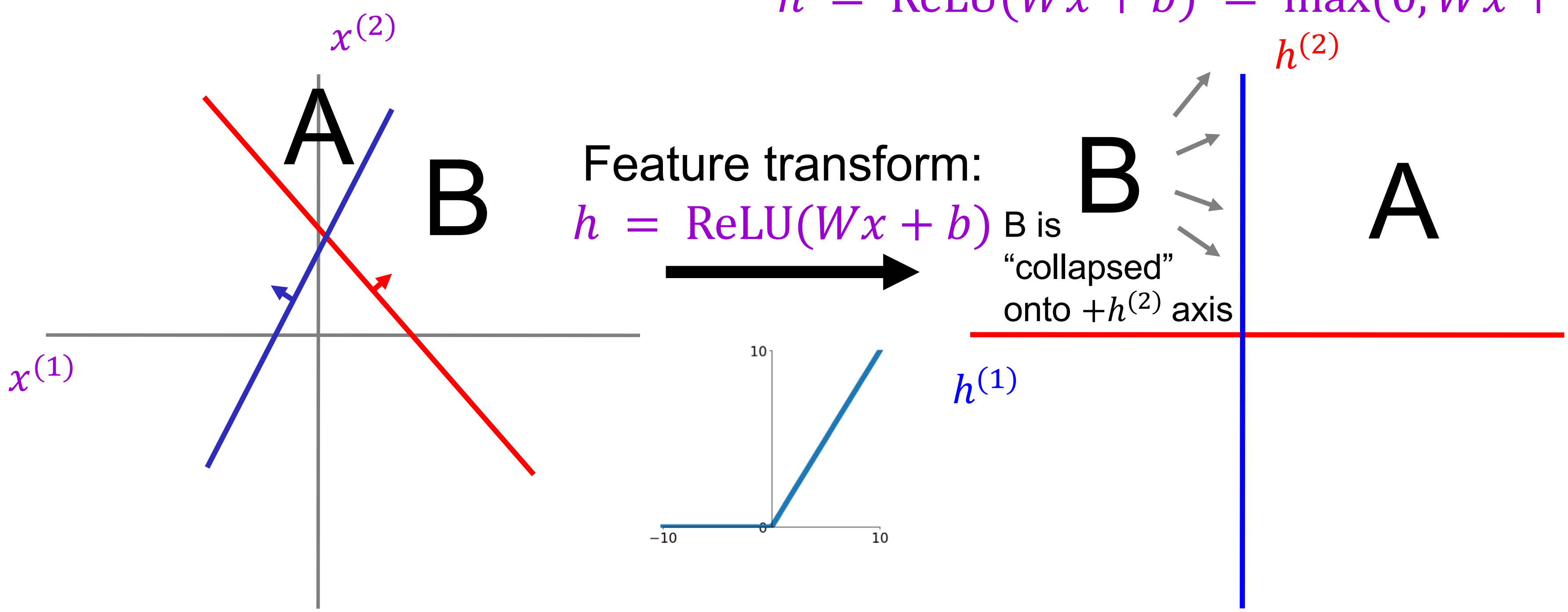


Source: [J. Johnson]

The power of nonlinearities

Let's add a nonlinearity:

$h = \mathrm{ReLU}(Wx+b) = \max(0, Wx+b)$

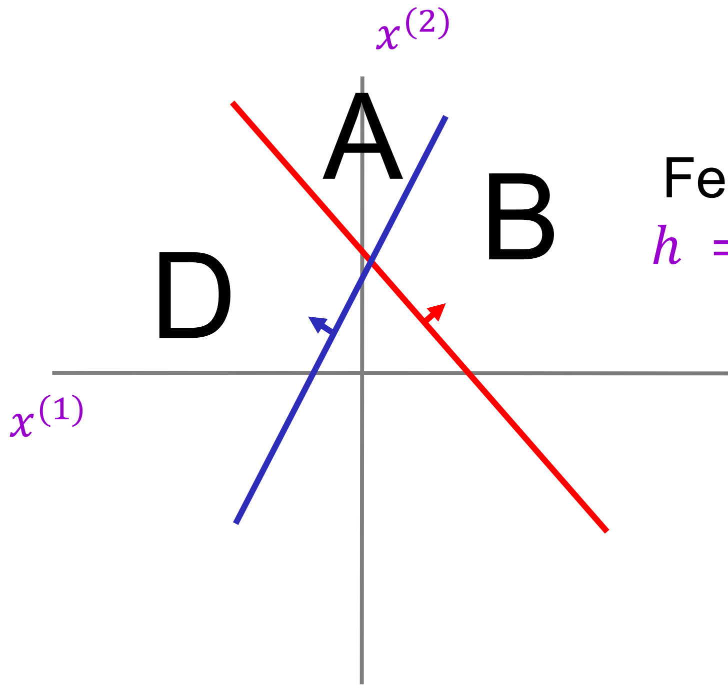


Source: [J. Johnson]

The power of nonlinearities

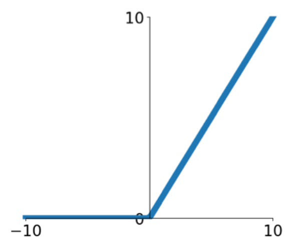
Let's add a nonlinearity:

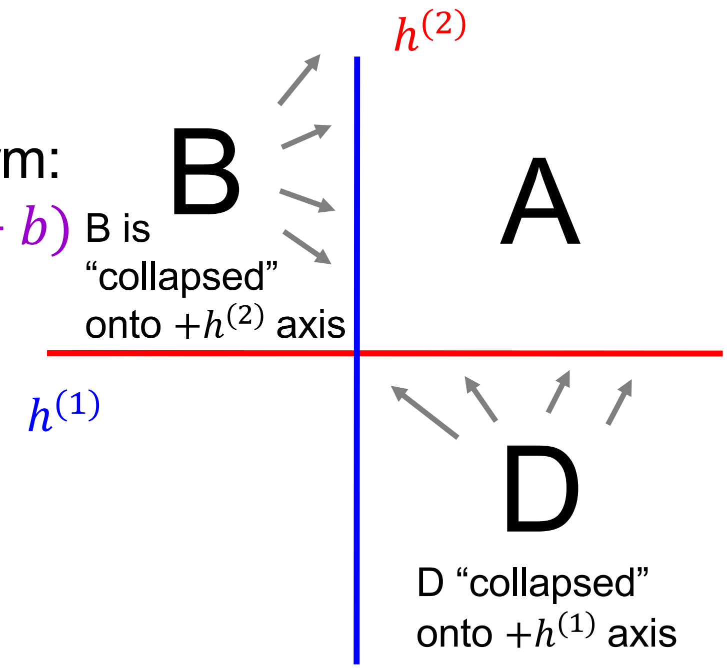
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$



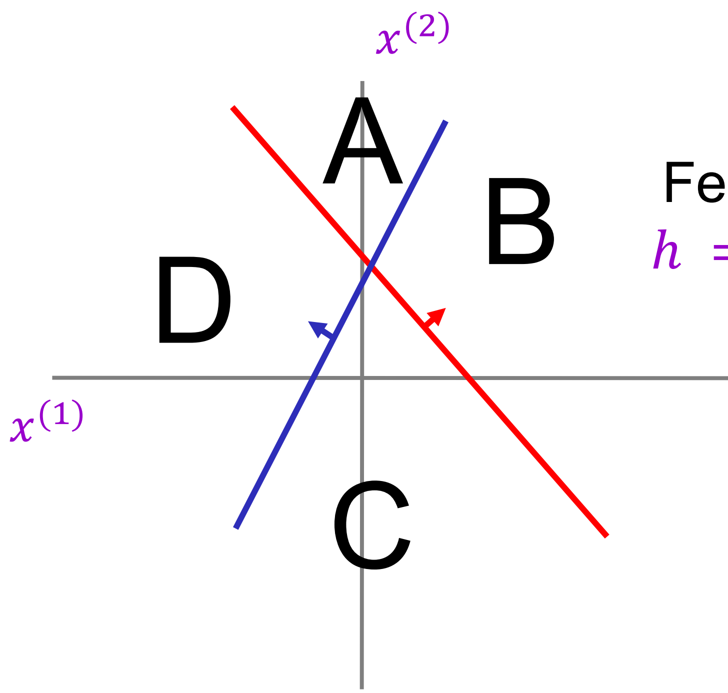


Source: [J. Johnson]

The power of nonlinearities

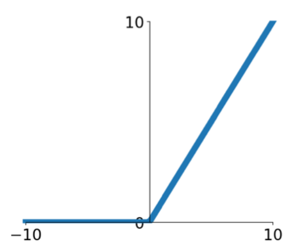
Let's add a nonlinearity:

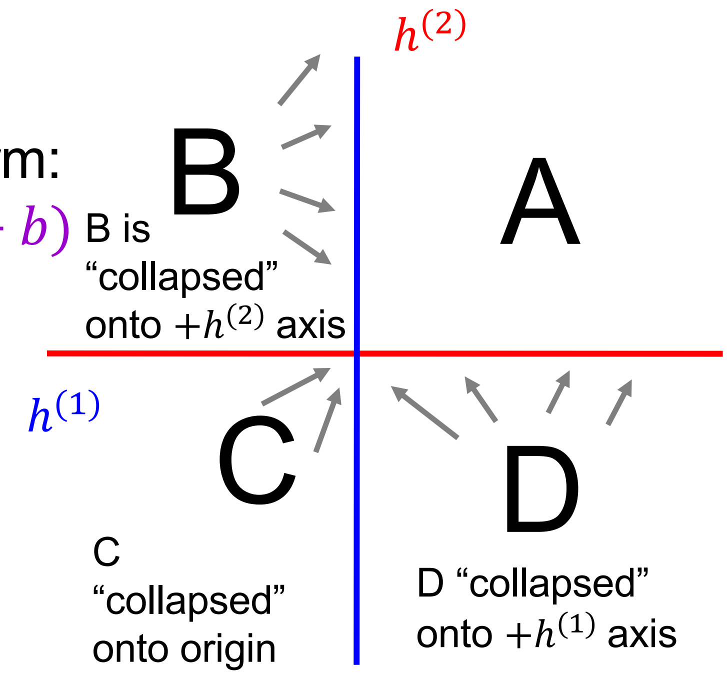
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$

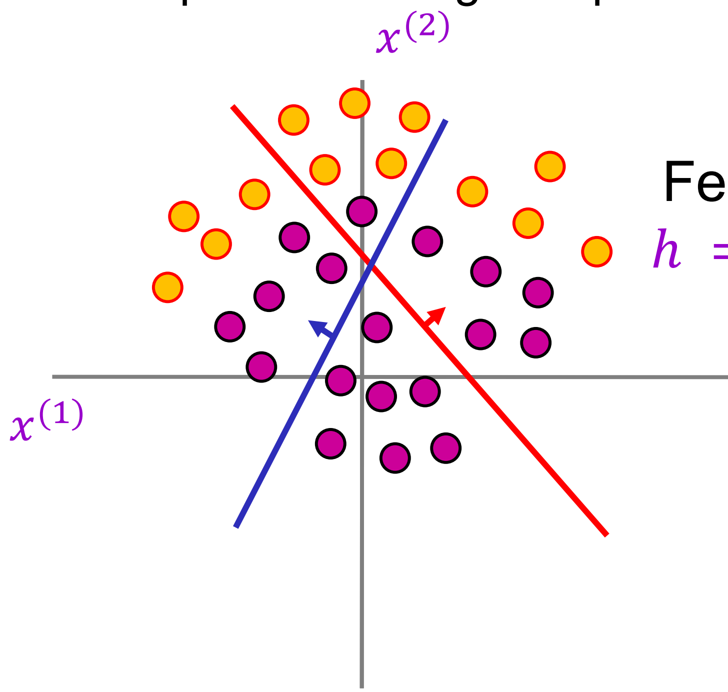




Source: [J. Johnson]

The power of nonlinearities

Points not linearly separable in original space

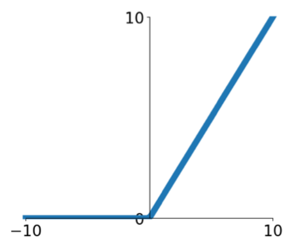


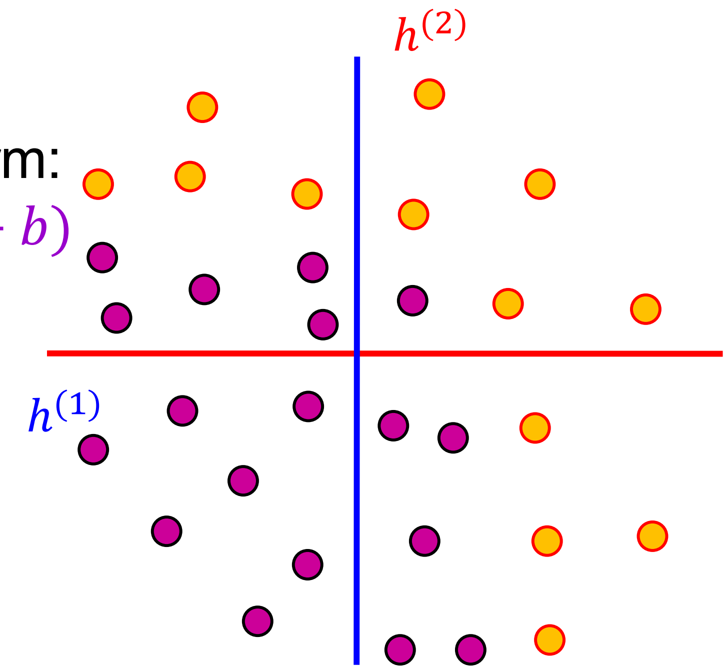
Let's add a nonlinearity:

$$h = \mathrm{ReLU}(Wx+b) = \max(0, Wx+b)$$

Feature transform:

$$h = \mathrm{ReLU}(Wx+b)$$

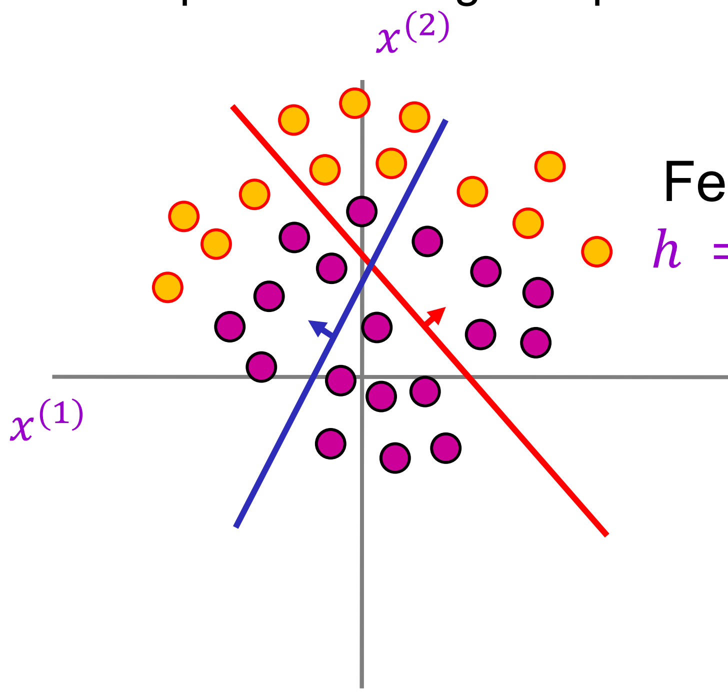




Source: J. Johnson

The power of nonlinearities

Points not linearly separable in original space

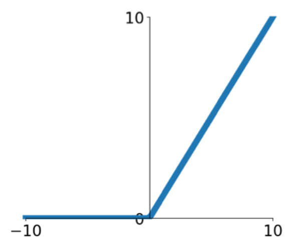


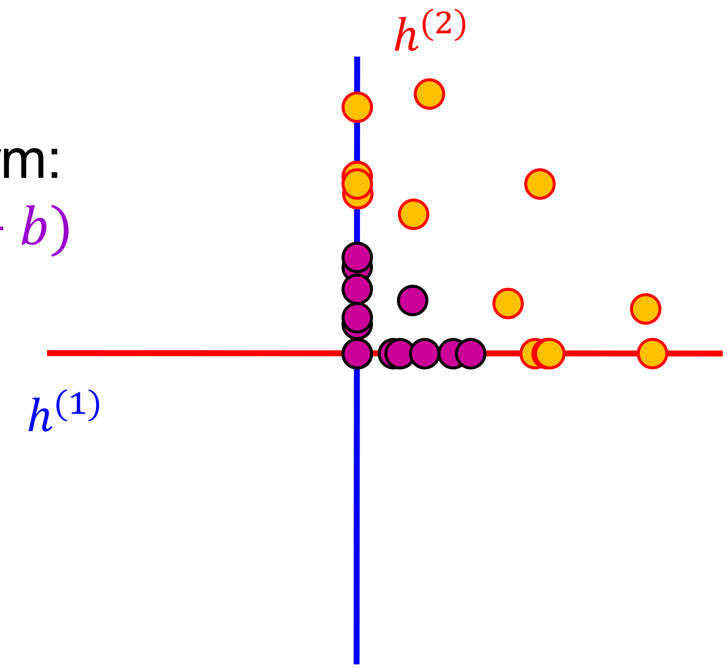
Let's add a nonlinearity:

$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$

Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$

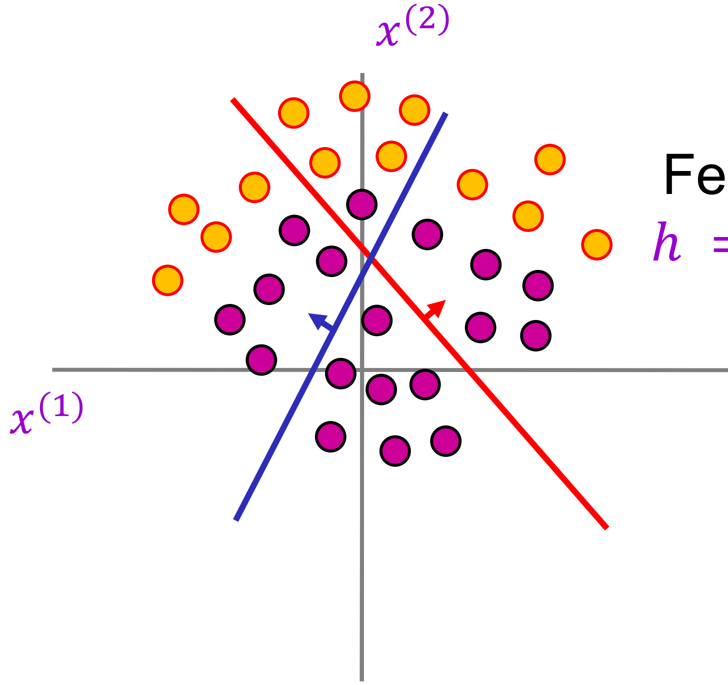




Source: [J. Johnson]

The power of nonlinearities

Points not linearly separable in original space

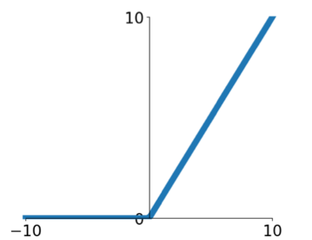


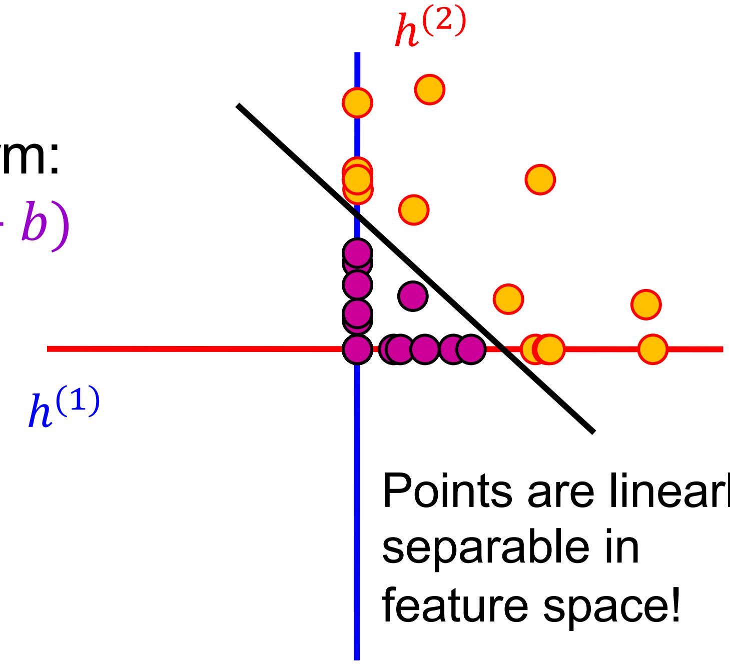
Let's add a nonlinearity:

$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$

Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$





Points are linearly separable in feature space!

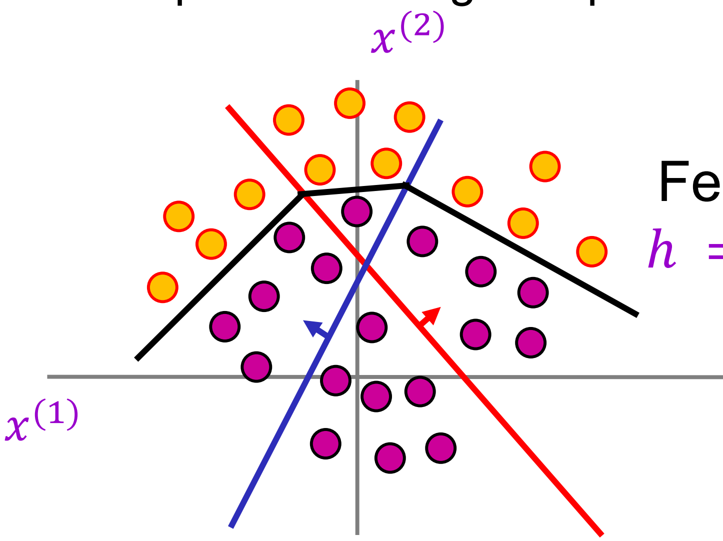
Source: [J. Johnson]

The power of nonlinearities

Points not linearly separable in original space

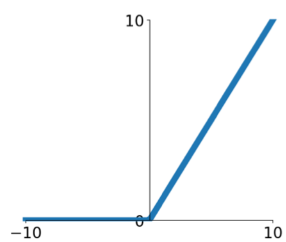
Let’s add a nonlinearity:

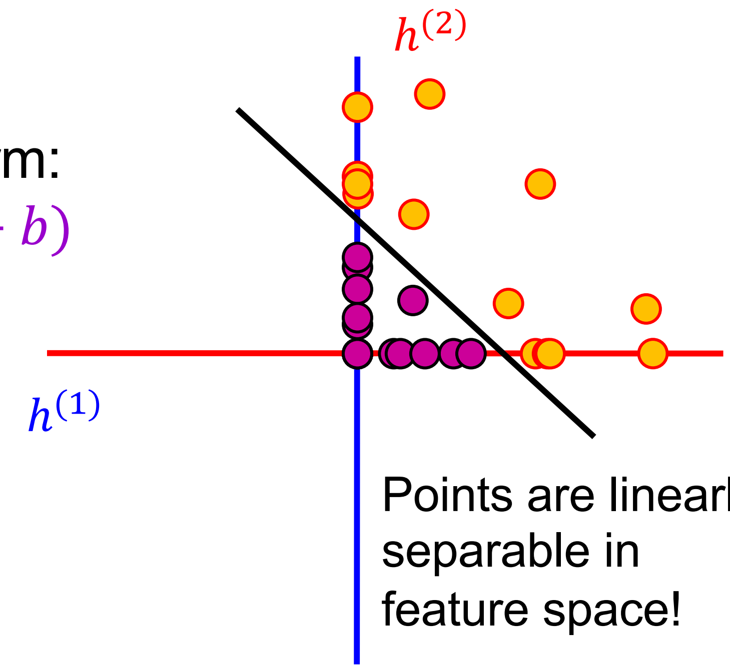
$$h = \mathrm{ReLU}(Wx + b) = \max(0, Wx + b)$$



Feature transform:

$$h = \mathrm{ReLU}(Wx + b)$$



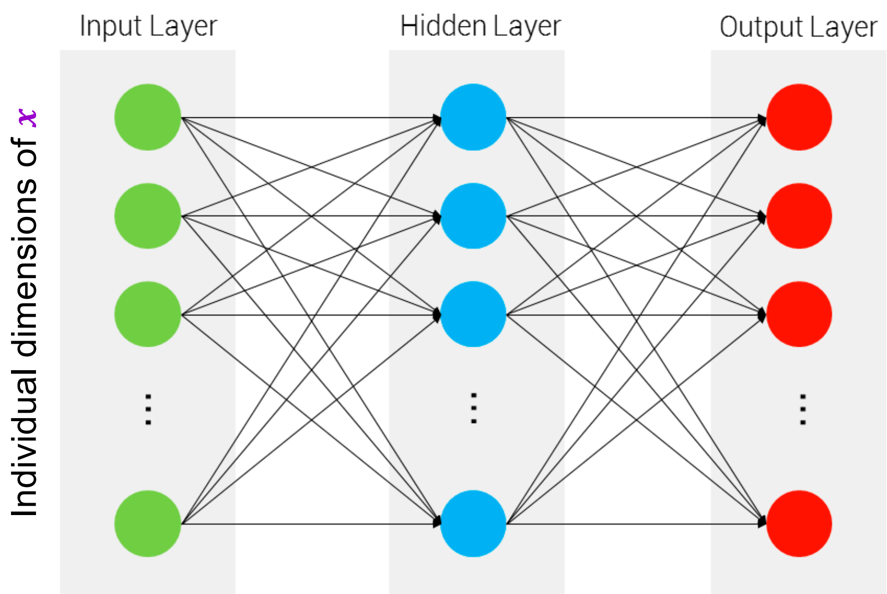


Linear classifier in feature space gives nonlinear classifier in original space

Points are linearly separable in feature space!

Source: [J. Johnson]

Two-layer neural network



Output of hidden layer: $\boldsymbol{g(W_1 x)}$ Final output: $\boldsymbol{g(W_2 g(W_1 x))}$

Image source

Two-layer networks as combinations of templates

Linear classifier: One template per class

Two-layer networks as combinations of templates

First layer: bank of templates
Second layer: recombines templates

Source: J. Johnson

Two-layer networks as combinations of templates

First layer: bank of templates
Second layer: recombines templates

Can use different templates to cover multiple *modes* of a class

Source: J. Johnson

Two-layer networks as combinations of templates

First layer: bank of templates
Second layer: recombines templates

It's a "distributed" representation:
Most templates are not interpretable

Source: J. Johnson

Expressiveness of two-layer networks

- How complex can we make the decision boundary in a two-layer network?
- The bigger the hidden layer, the more complex the model
- A two-layer network is a *universal function approximator*
  - But the hidden layer may need to be huge

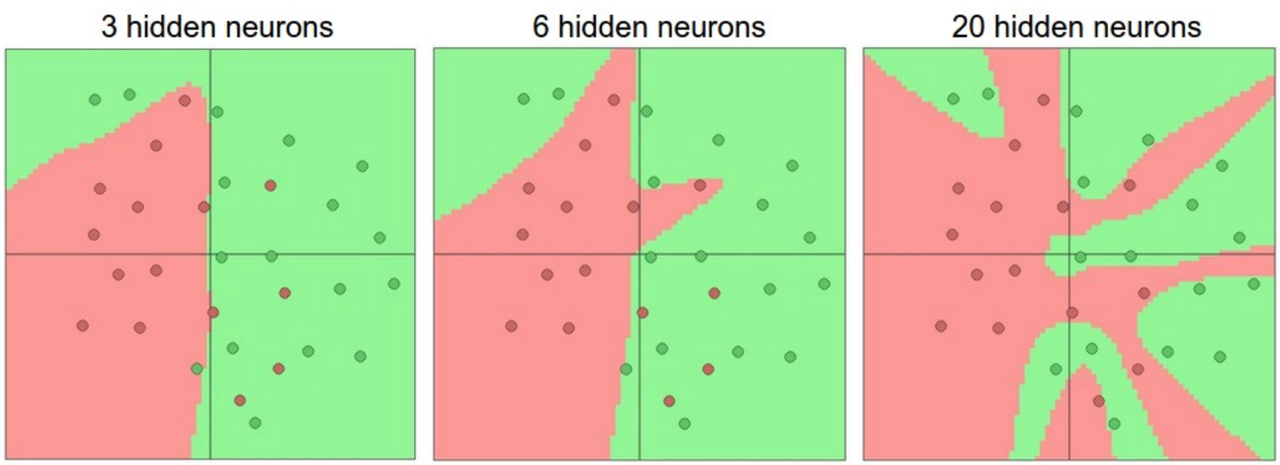
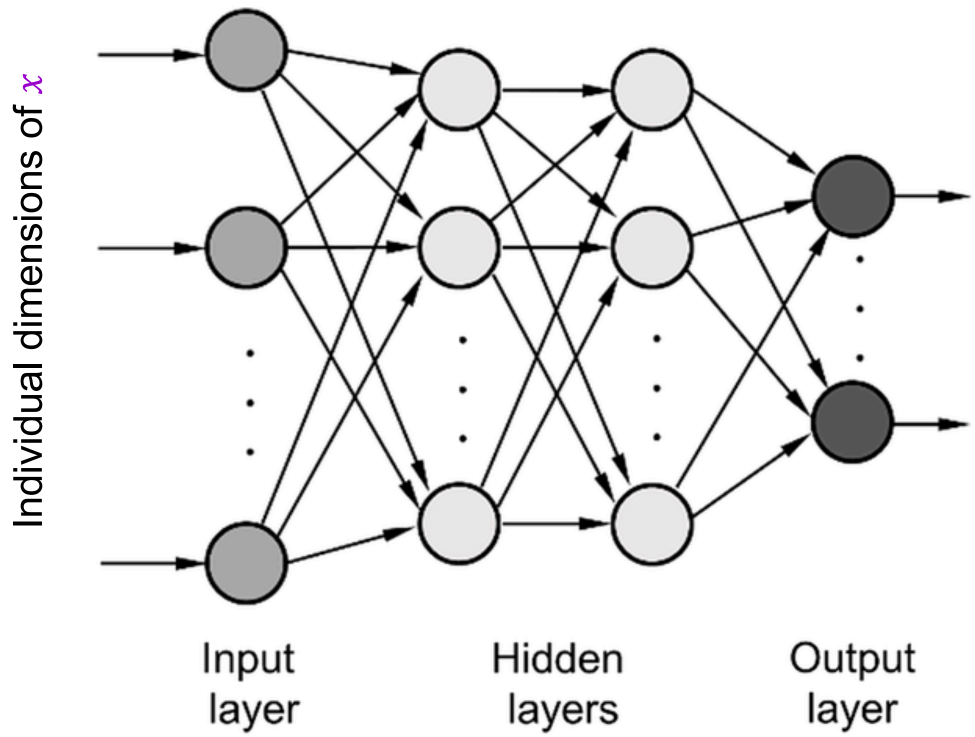


Figure source

Neural networks beyond two layers

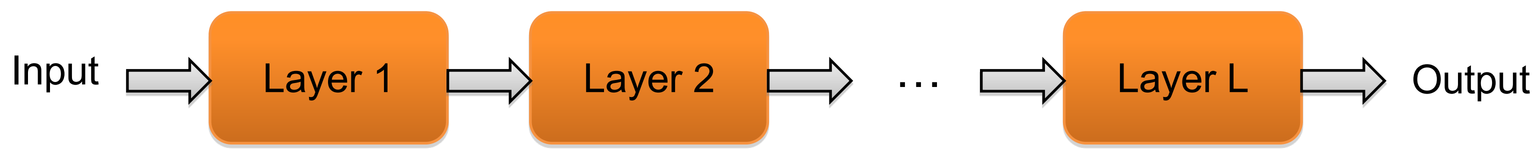


Output:

$$g_L(W_L \ldots g_2(W_2\, g_1(W_1 x)) \ldots)$$

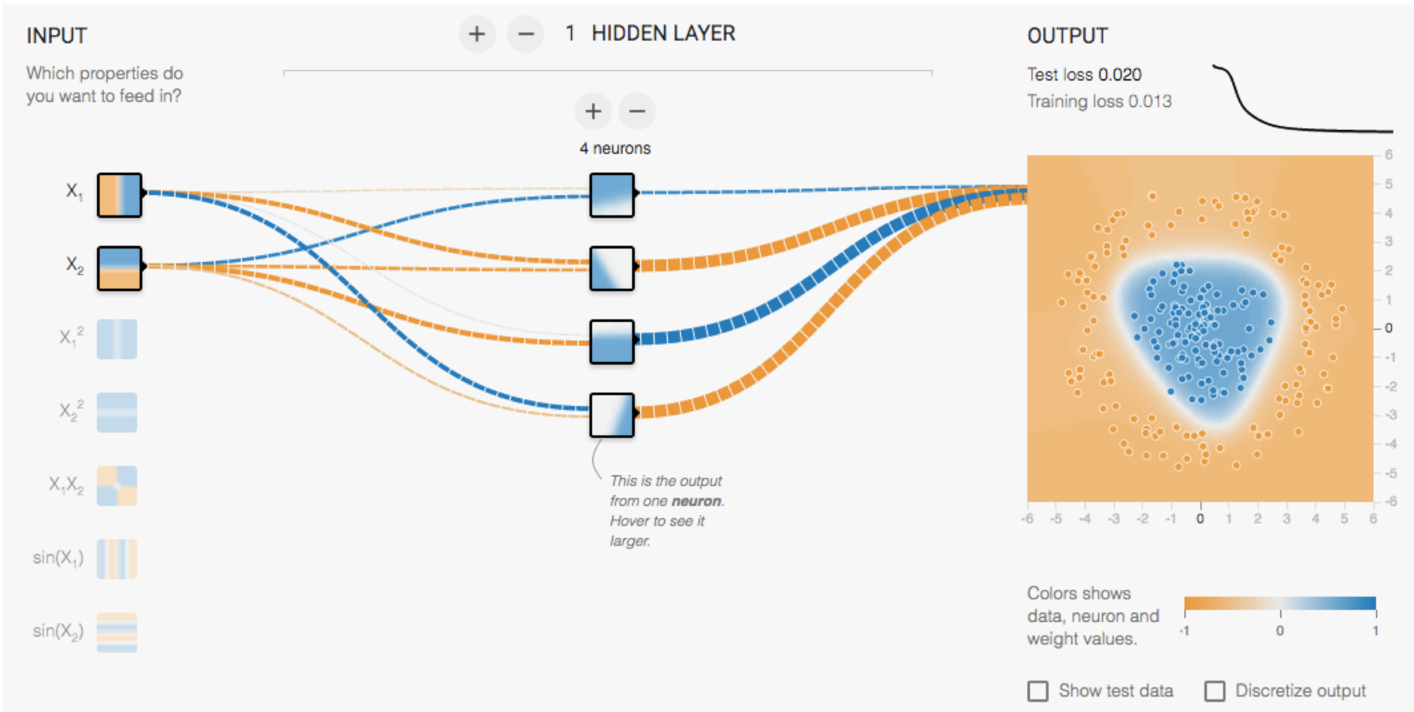
[Image source]()

“Deep” pipeline



- Learn a *feature hierarchy*
- Each layer extracts features from the output of previous layer
- All layers are trained jointly

Multi-Layer network demo



http://playground.tensorflow.org/

Overview

- Multi-layer networks
- Hyperparameter search, validation

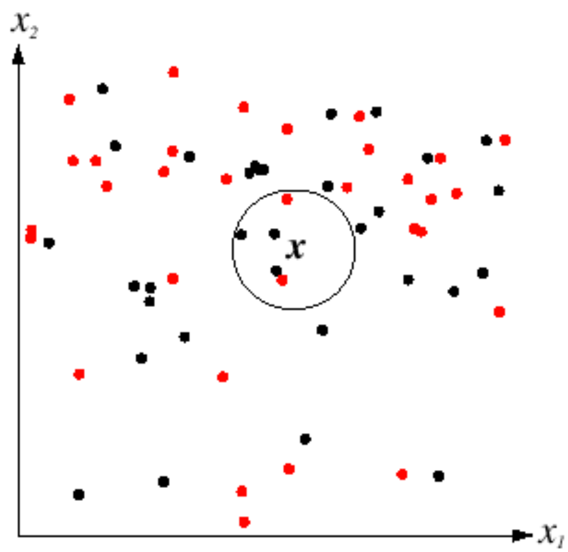
Supervised learning outline revisited

1. **Collect data and labels**
2. **Specify model:** select model class and loss function
3. **Train model:** find the parameters of the model that minimize the empirical loss on the training data

This involves *hyperparameters* that affect the generalization ability of the trained model

Hyperparameters

- K in K-nearest-neighbor
  - What if K is too large?
  - What if K is too small?



Hyperparameters

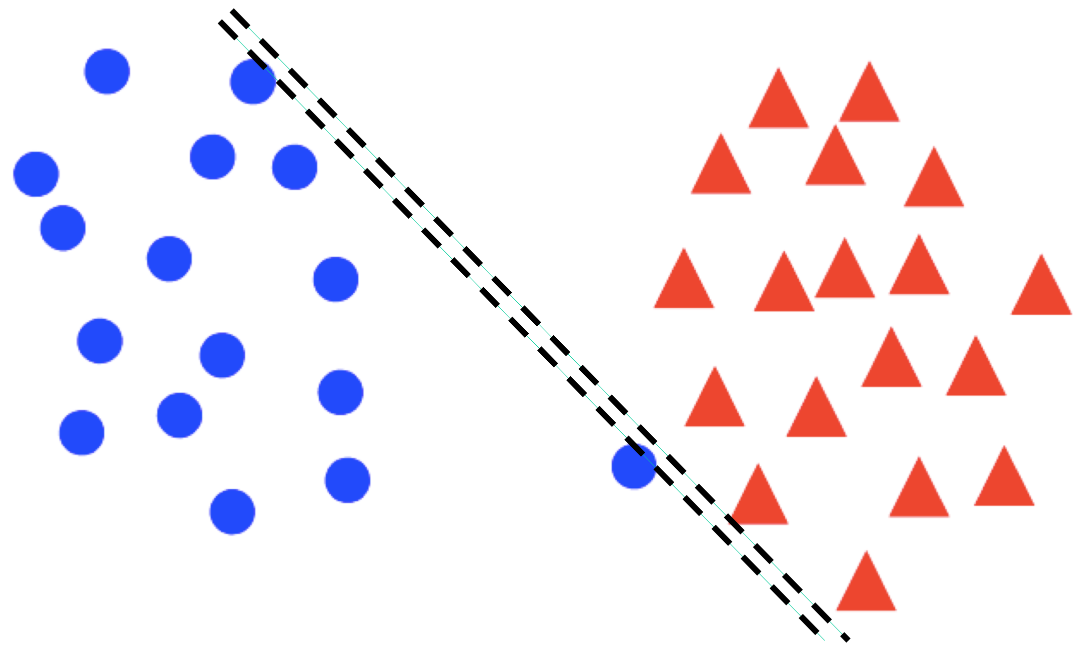
- Regularization constant λ
  - Recall: SVM optimization

$$\min_w \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$

  - What if λ is too large?
  - What if λ is too small?

Hyperparameters

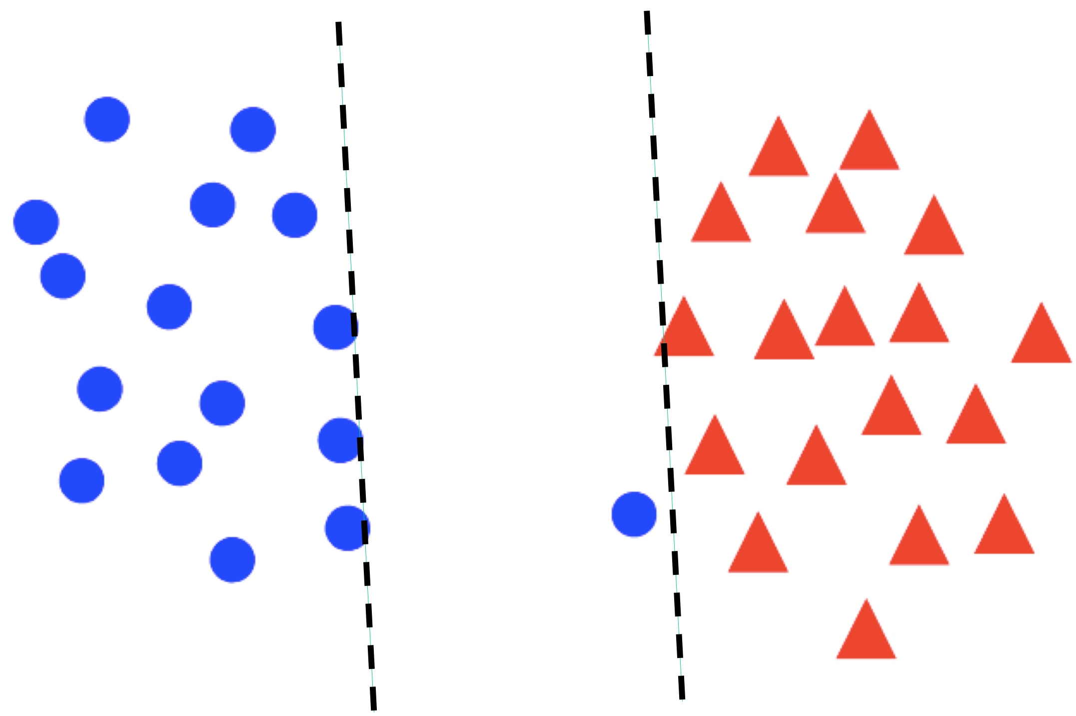
- Regularization constant λ
  - Tradeoff between margin and classification errors



Source

Hyperparameters

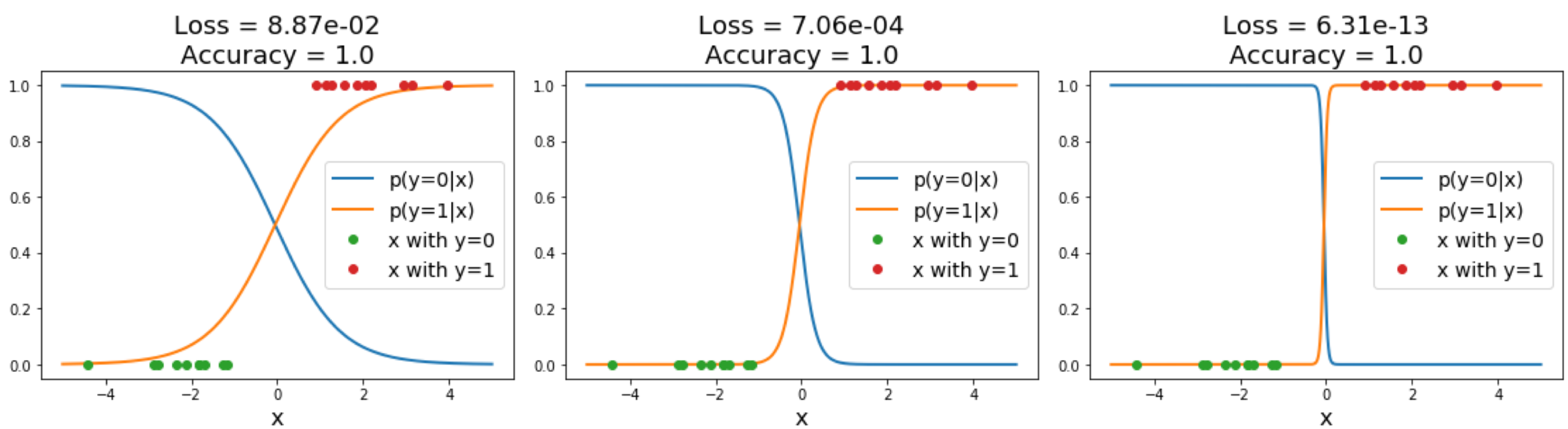
- Regularization constant λ
  - Tradeoff between margin and classification errors



Source

Hyperparameters

- Regularization constant λ
  - Related: preventing the classifier from getting over-confident

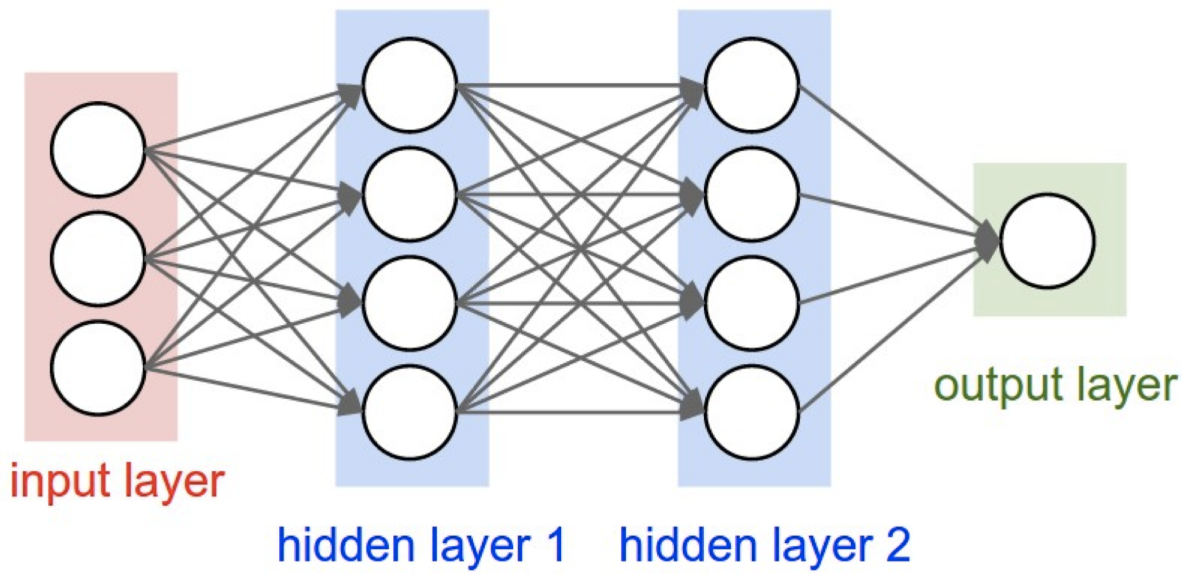


Sigmoid classifier, logistic loss

Source: [J. Johnson]

Hyperparameters in multi-layer networks

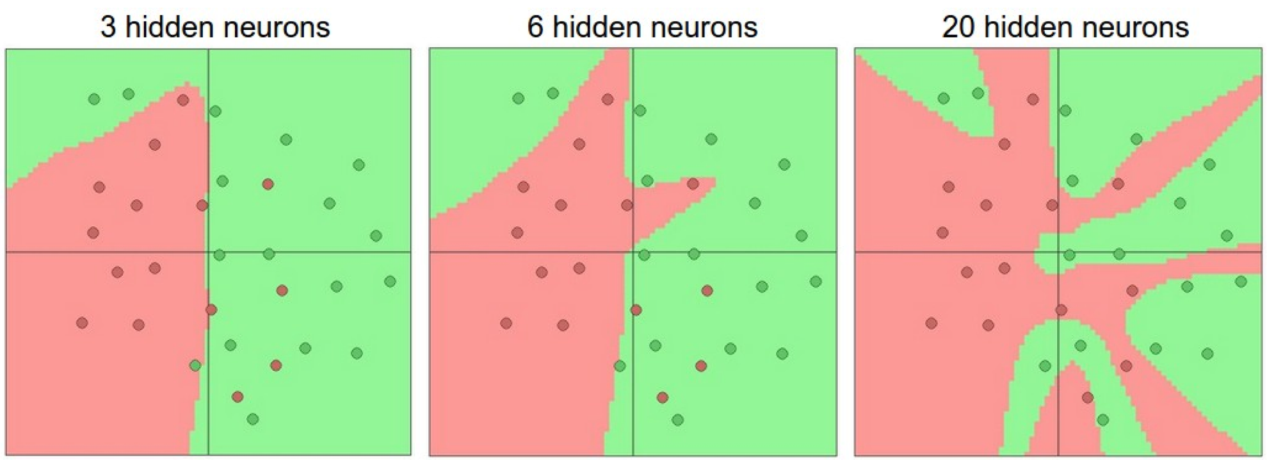
- Number of layers, number of units per layer



Source: [Stanford 231n]

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer

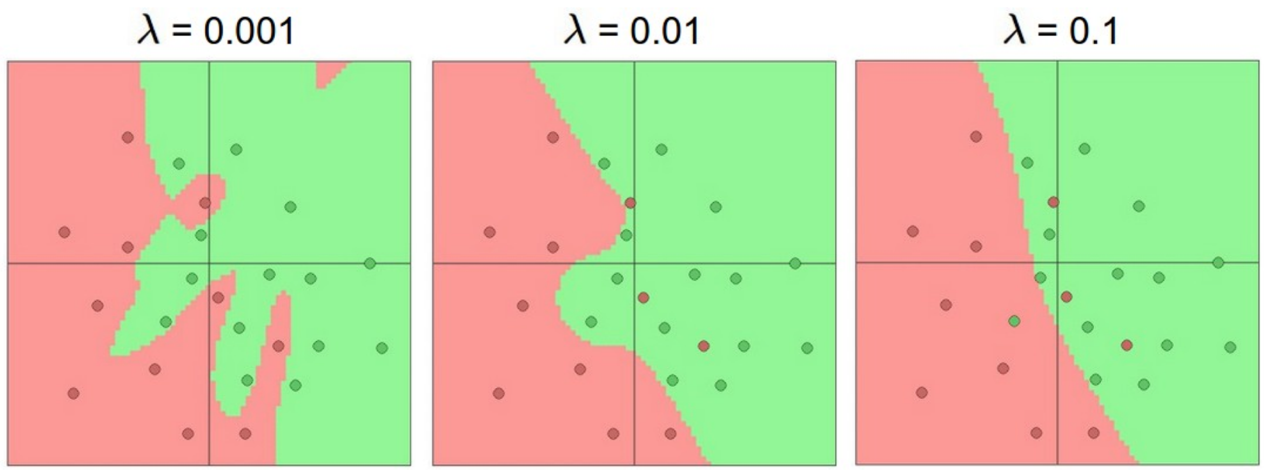


Number of hidden units in a two-layer network

Source: [Stanford 231n]

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Type of nonlinearity
- Type of loss function
- Regularization constant



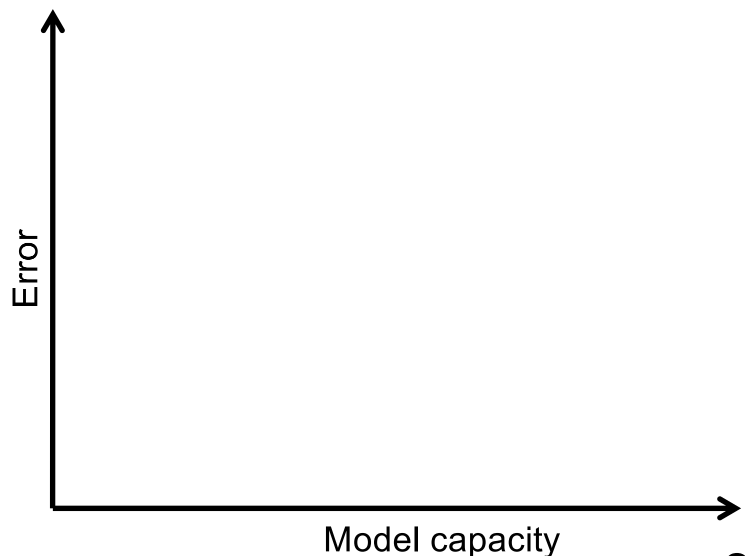
Source: [Stanford 231n]

Hyperparameters in multi-layer networks

- Number of layers, number of units per layer
- Type of nonlinearity
- Type of loss function
- Regularization constant
- SGD settings: learning rate schedule, number of epochs, minibatch size, etc.

- Our hyperparameter choices affect the “capacity” of the model and its ability to generalize to new data
  - But first: how do we measure the generalization ability of our model?
  - Need to measure both *training* and *test error*

Behavior of training and test error



Underfitting: training and test error are both *high*

Overfitting: training error is *low* but test error is *high*

Source: [D. Hoiem]

Underfitting and overfitting: The classical view

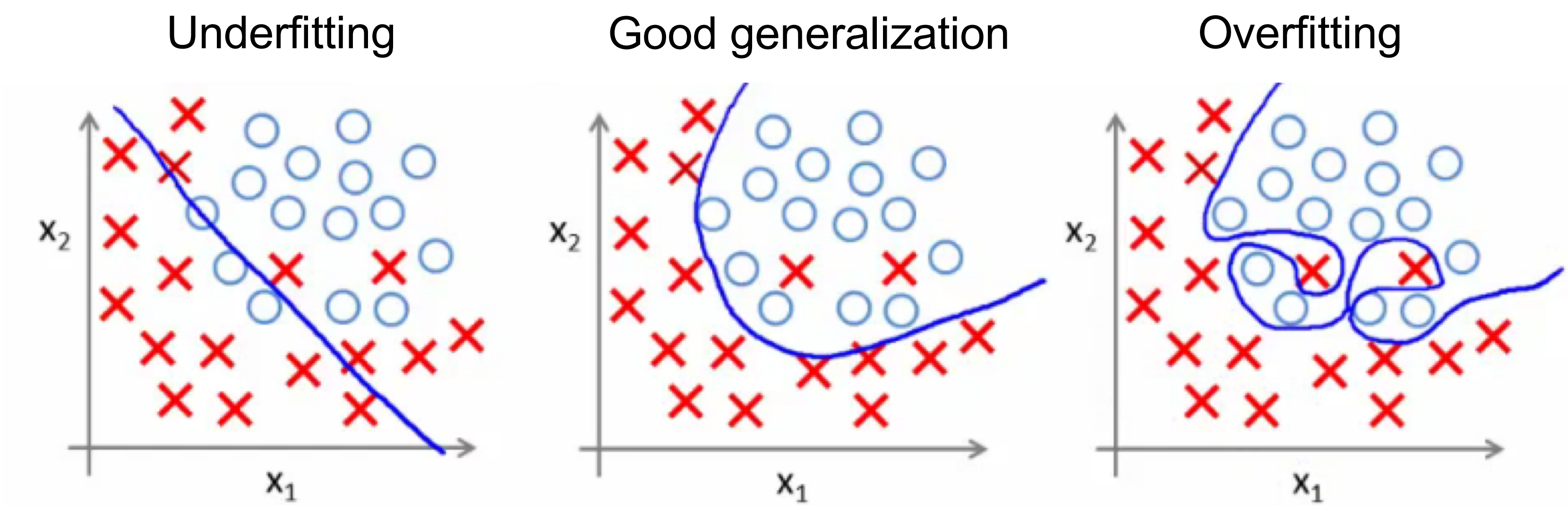
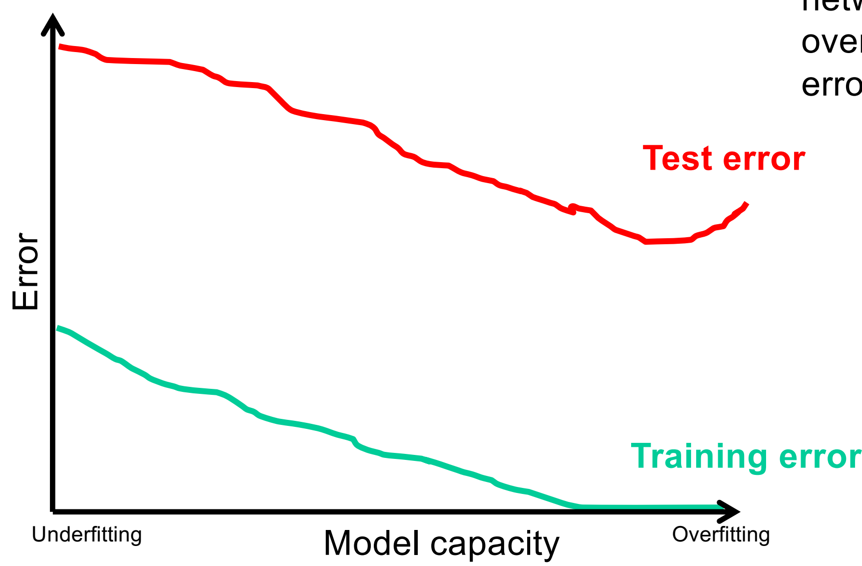


Figure source

Behavior of training and test error

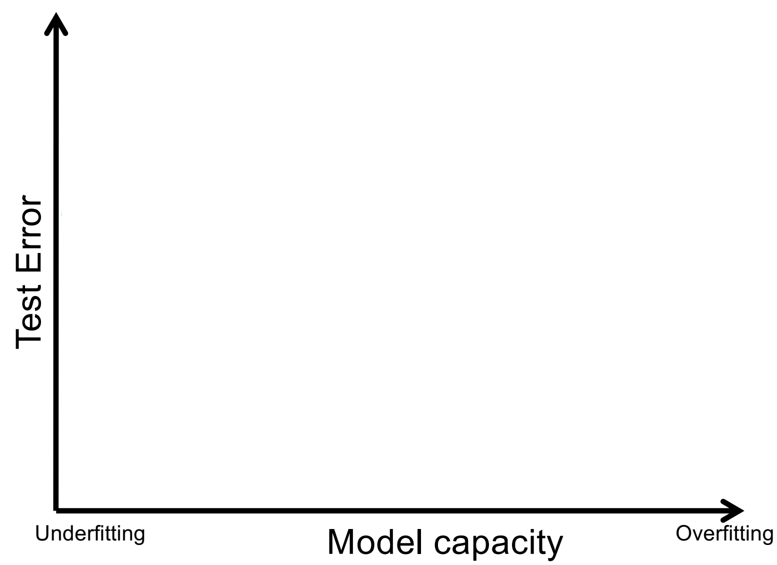


In practice, overparameterized neural networks tend to be resistant to overfitting, even after the training error goes to zero

In neural networks, we also observe this kind of behavior for a fixed model as a function of *training time*!

Source: [D. Hoiem]

Generalization and training set size

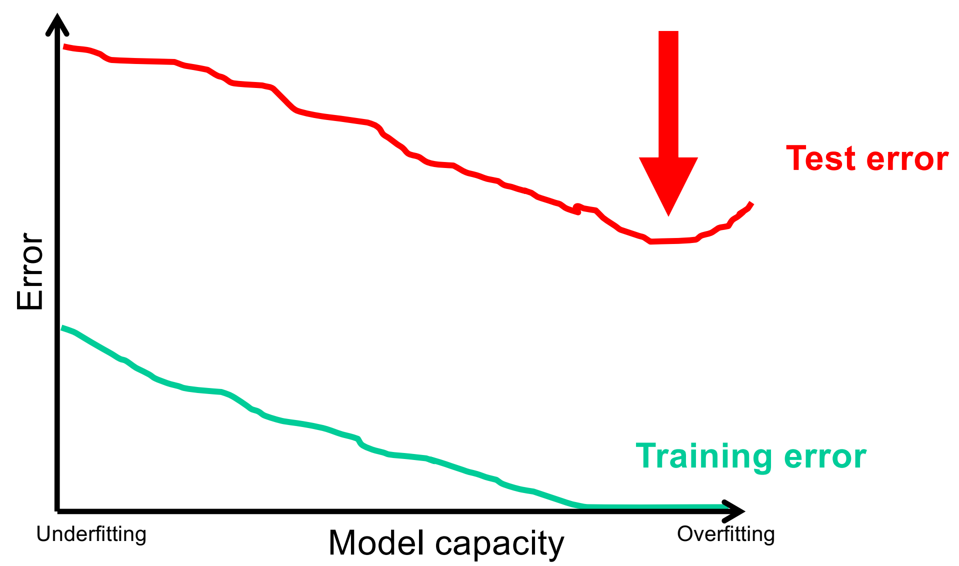


Bottom line: more training data is *always* good!

Source: [D. Hoiem]

Hyperparameter search in practice

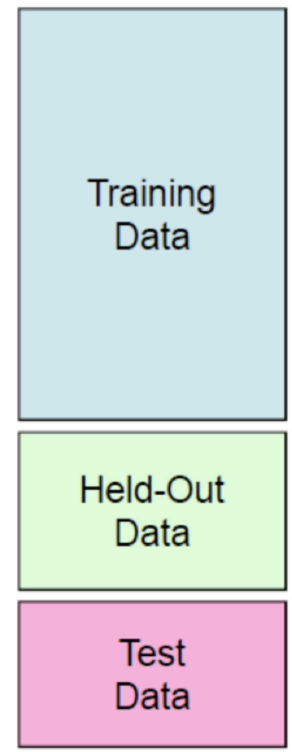
- Given a fixed dataset, you have to find the hyperparameter settings that give the best generalization performance



Source: [D. Hoiem]

Hyperparameter search in practice

- For a range of hyperparameter choices, iterate:
  - Learn parameters on the ***training data***
  - Measure accuracy on the ***held-out*** or ***validation data***
- Finally, measure accuracy on the ***test data***
- You should avoid peeking at the test set during hyperparameter search since it is supposed to represent *never before seen data*



The mysteries of generalization

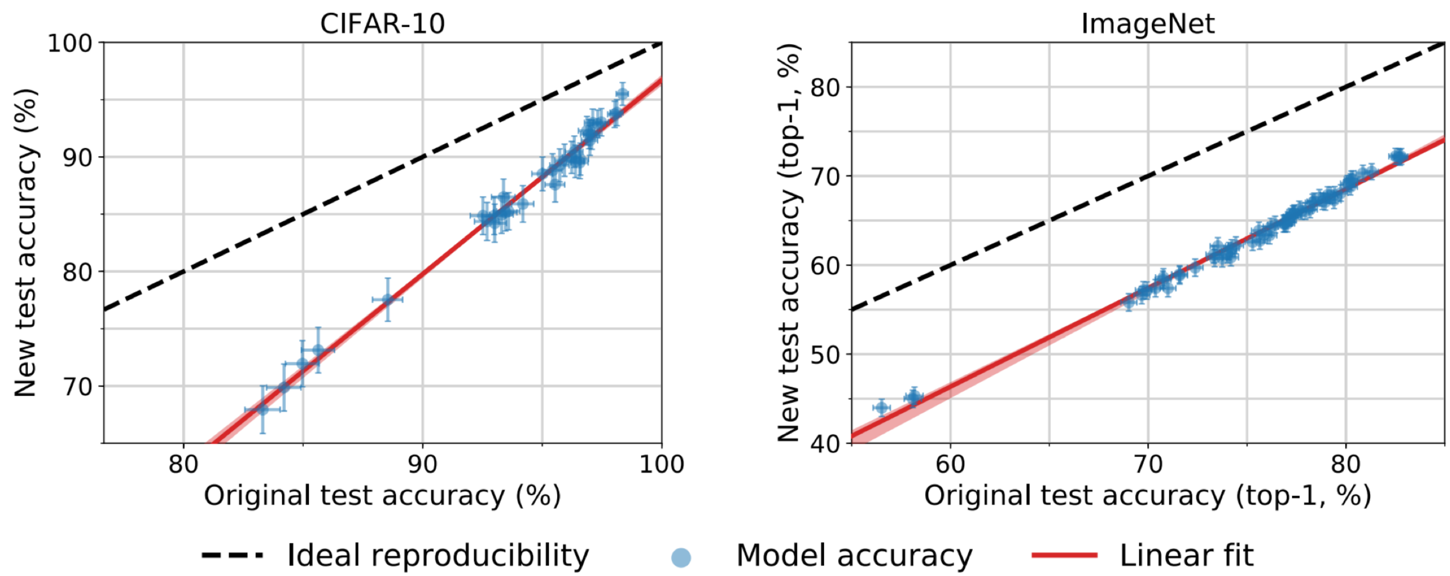


Figure 1. Model accuracy on the original test sets vs. our new test sets. Each data point corresponds to one model in our testbed (shown with 95% Clopper-Pearson confidence intervals). The plots reveal two main phenomena: (i) There is a significant drop in accuracy from the original to the new test sets. (ii) The model accuracies closely follow a linear function with slope *greater* than 1 (1.7 for CIFAR-10 and 1.1 for ImageNet). This means that every percentage point of progress on the original test set translates into more than one percentage point on the new test set. The two plots are drawn so that their aspect ratio is the same, i.e., the slopes of the lines are visually comparable. The red shaded region is a 95% confidence region for the linear fit from 100,000 bootstrap samples.

B. Recht et al. [Do ImageNet classifiers generalize to ImageNet?]() ICML 2019