CS 480/680
Introduction to Machine Learning

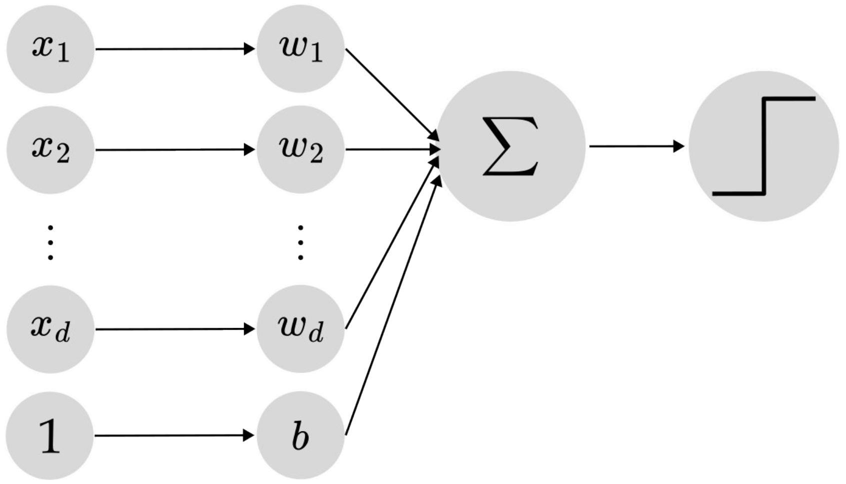
Lecture 13
Multilayer Perceptrons and Deep Learning

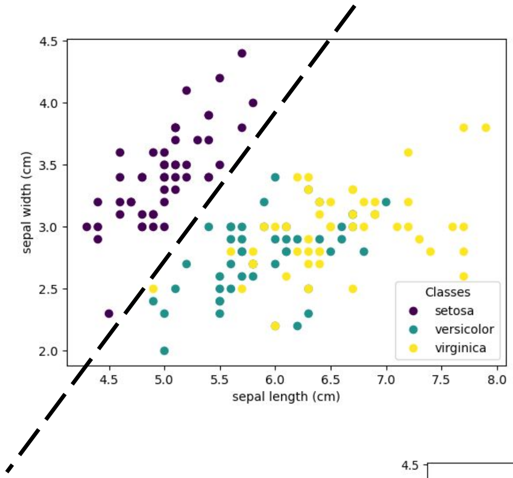
Kathryn Simone

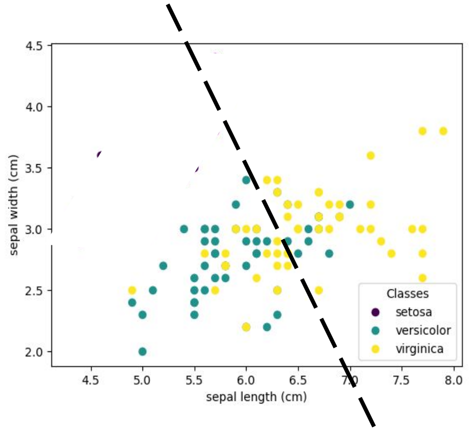
31 October 2024

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

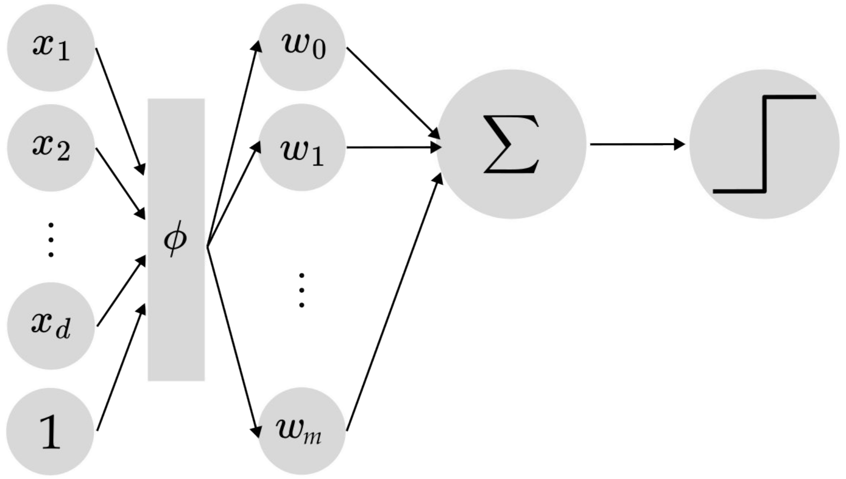
Learning functions from linear combinations of inputs

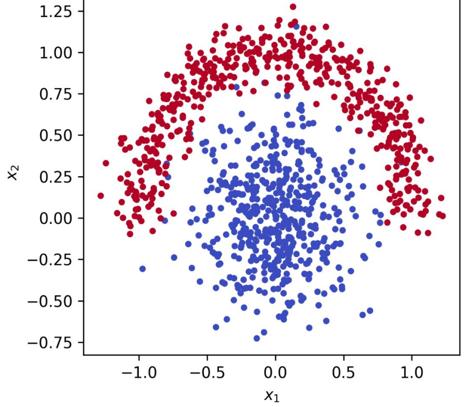


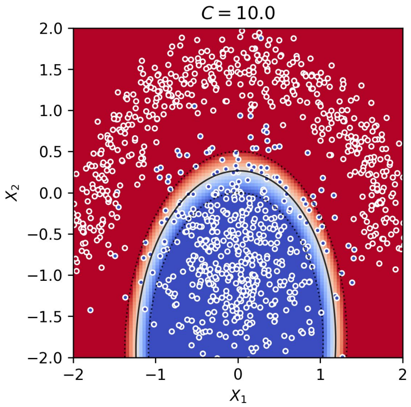




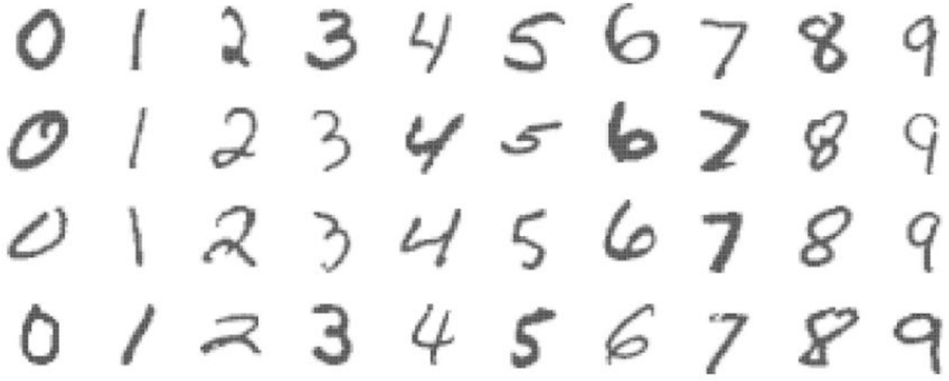
Learning functions from nonlinear features of inputs

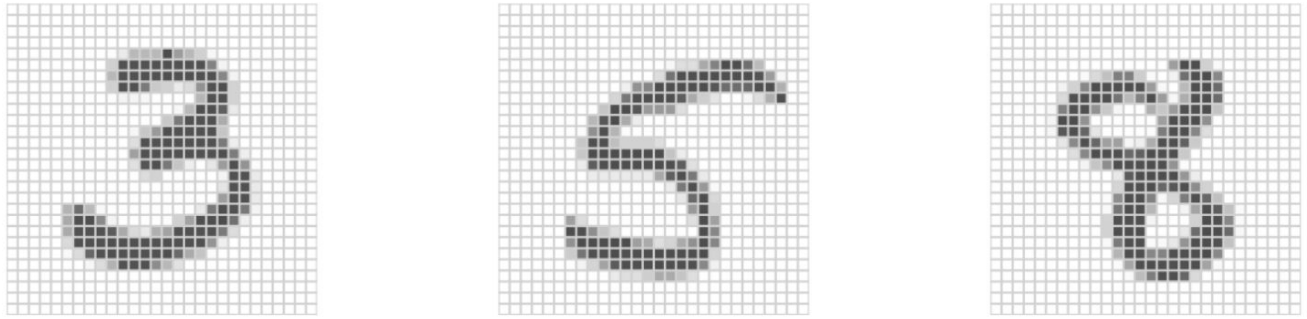






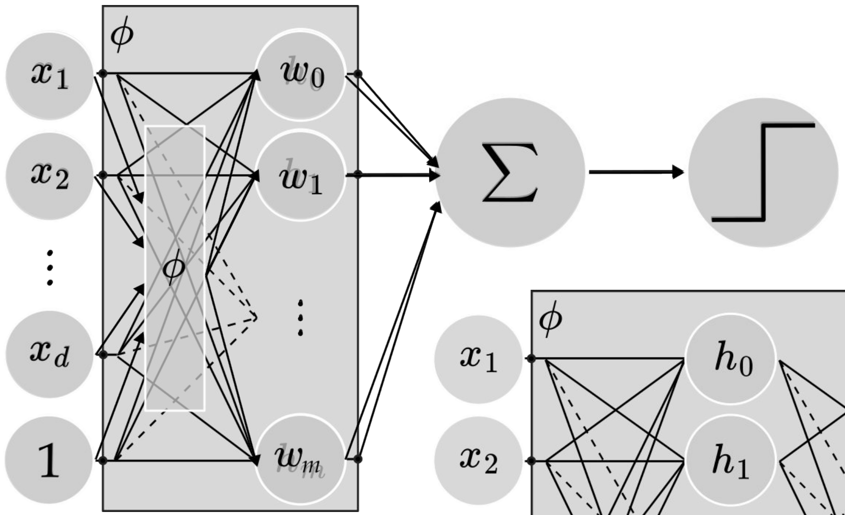
What if you don't know what the features should be?

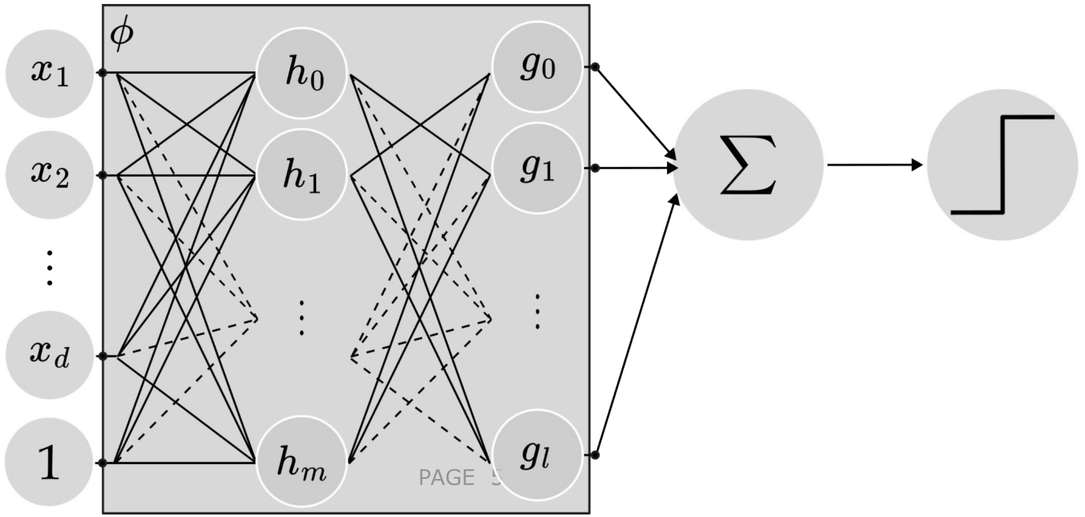




ESL Section 10.3

Neural Networks: Learn the features of the data from the data





Key questions

I. What do we mean by useful features?

II. How to fit the parameters?

III. How to prevent overfitting?

Key questions

I. **What do we mean by useful features?**

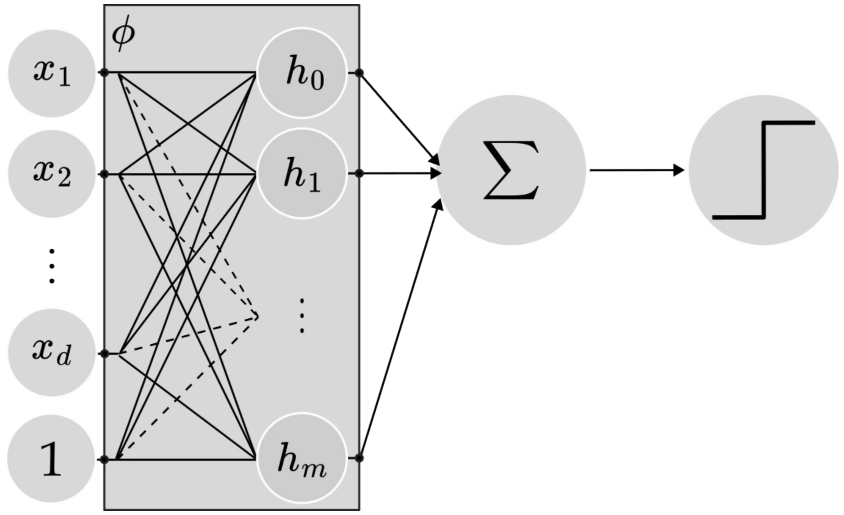
II. How to fit the parameters?

III. How to prevent overfitting?

How can we represent useful features?

Suppose we create a feature-map $\phi : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}^{\mathrm{m}}$ $x \in \mathbb{R}^{\mathrm{d}}$ using a *projection matrix* $V \in \mathbb{R}^{\mathrm{d}\times\mathrm{m}}$, $V_{ij} \in \mathbb{R}$

$$\phi(x) = \langle x, V \rangle$$



$$V = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1m} \\ v_{21} & v_{22} & \dots & v_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{d1} & v_{d2} & \dots & v_{dm} \end{bmatrix}$$

How can we approximate features?

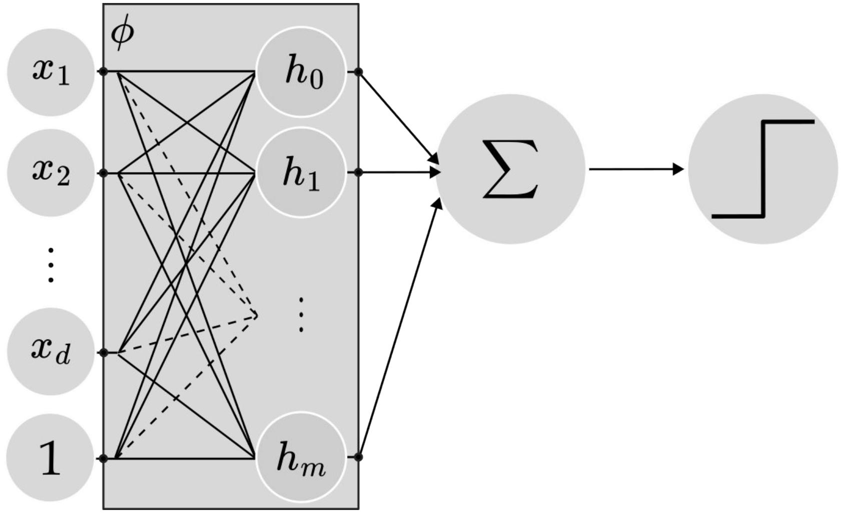
$$\phi(x) = \langle x, V \rangle$$

$$\phi(x) = x^T V$$

$$= \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1m} \\ v_{21} & v_{22} & \dots & v_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{d1} & v_{d2} & \dots & v_{dm} \end{bmatrix}$$

$$= \begin{bmatrix} x_1 v_{11} + x_2 v_{21} + \cdots + v_{d1} x_d \\ x_1 v_{12} + x_2 v_{22} + \cdots + v_{d2} x_d \\ \vdots \\ x_1 v_{1m} + x_2 v_{2m} + \cdots + v_{dm} x_d \end{bmatrix}^T$$

$$= \begin{bmatrix} \phi_1(x) = \sum_{i=1}^{d} v_{i1} x_i \\ \phi_2(x) = \sum_{i=1}^{d} v_{i2} x_i \\ \vdots \\ \phi_m(x) = \sum_{i=1}^{d} v_{im} x_i \end{bmatrix}^T$$



Linear combinations of input features are not more expressive

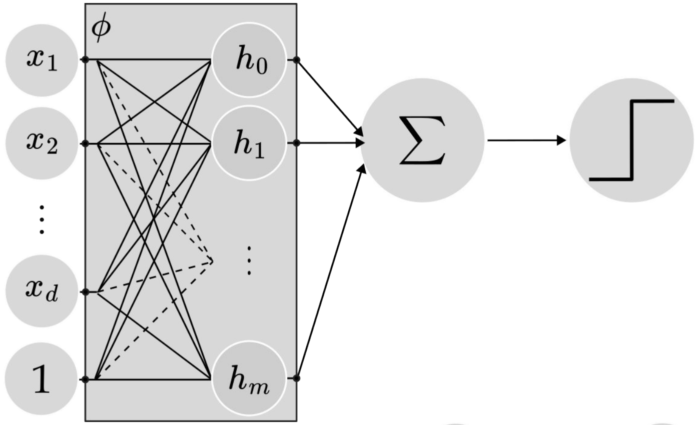
$$\hat{y} = \sum_{j=1}^{m} w_j \phi_j(x)$$

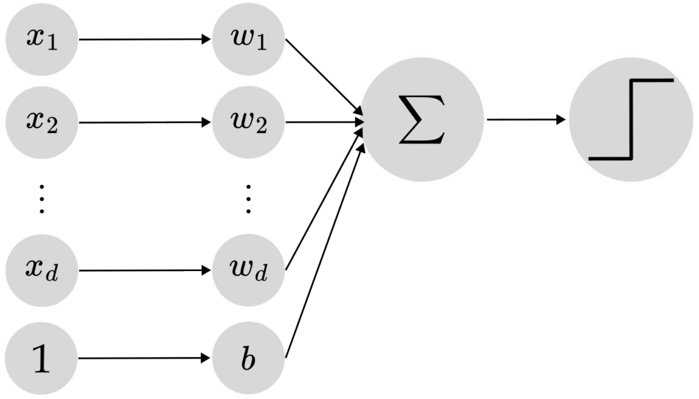
$$= \sum_{j=1}^{m} w_j \sum_{i=1}^{d} v_{ij} x_i$$

$$= \sum_{j=1}^{m} \sum_{i=1}^{d} w_j v_{ij} x_i$$

If we define $u_i = \sum_{j=1}^{m} w_j v_{ij}$, then

$$\hat{y} = \sum_{i=1}^{d} u_i x_i$$





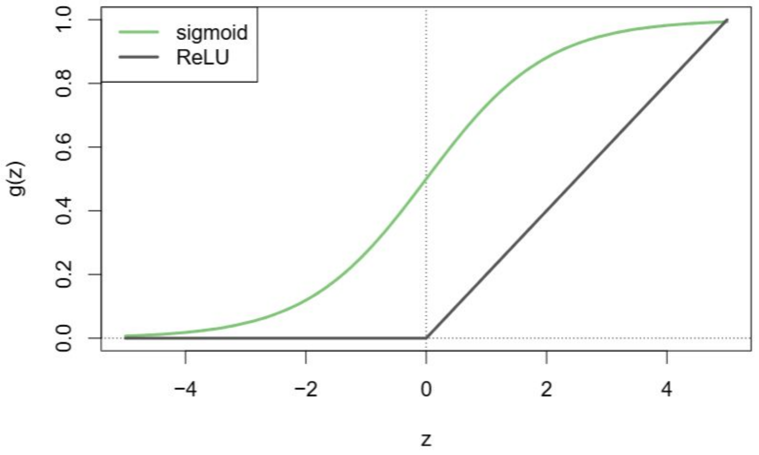
Solution: Introduce a nonlinearity on hidden units

$$\{\phi_j(x)\} = \left\{ f\left(\sum_{i=1}^{d} v_{im} x_i \right) \right\} \quad \text{for } j = 1, 2, \ldots, m$$

$$\text{Sigmoidal} : \sigma(t) = \frac{1}{1+e^{-t}}$$

$$\text{Tanh} : \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$

$$\begin{aligned} \text{Rectified Linear Unit} : \text{ReLU}(t) &= \max(0, t) \\ &= \max(0, t+b) \end{aligned}$$

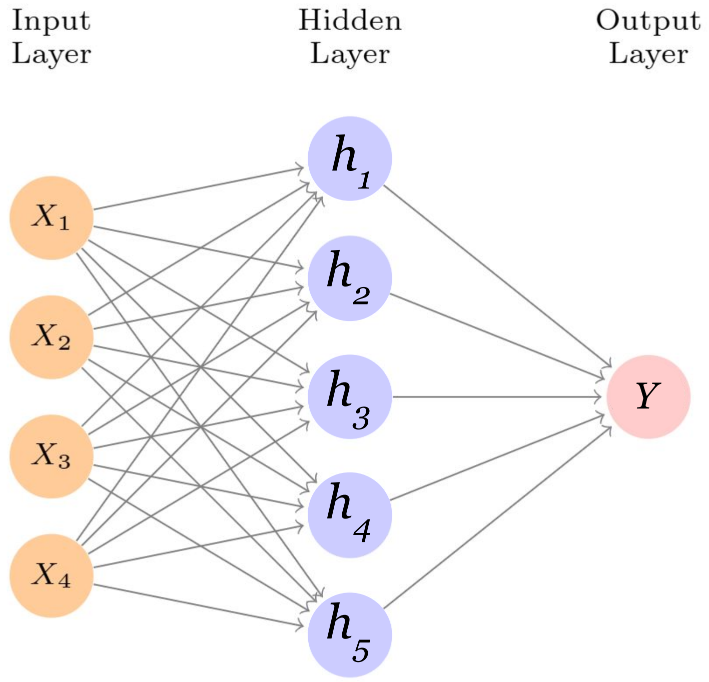


ESL Section 10.1

Universal Approximation Theorem (Hornik 1989, Cybenko 1989)

Universal Approximation Theorem:
A feedforward network with a linear output layer and at least one hidden layer can approximate any continuous function $f : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}^{\mathrm{k}}$ on a closed and bounded subspace of $\mathbb{R}^{\mathrm{d}}$, provided:

- ▷ The hidden layer is sufficiently wide
- ▷ The activation function is nonlinear and continuous.



Adapted from ESL Section 10.1

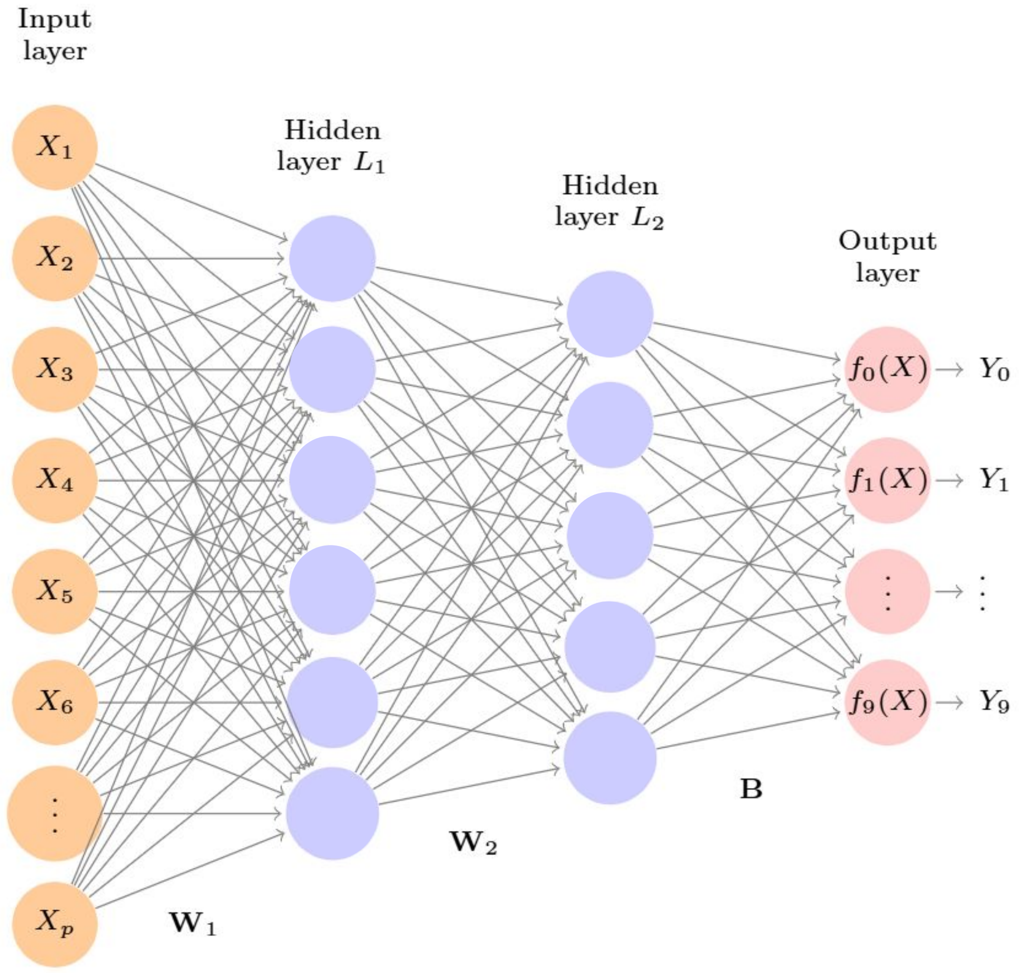
General Architecture of a Multilayer Perceptron

Architecture Design:

- Depth: # of hidden layers
- Width: # units per hidden layer

Functions on output layer

- Typically add a learnable bias
- Sigmoid for classification
- Softmax for multi-class classification
- Could approximate multiple functions at once



Adapted from ESL Section 10.1

Key questions

I. What do we mean by useful features?

II. How to fit the parameters?

III. How to prevent overfitting?

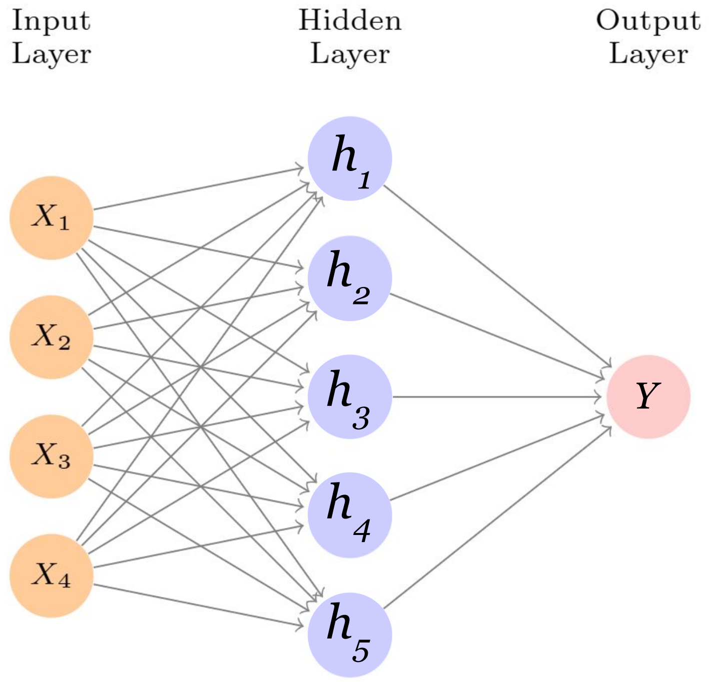
How to fit the parameters?

Objective:

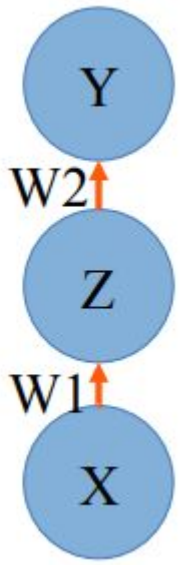
$$\text{argmin}_\theta \mathbb{E}[l_\theta(x, y)]$$

# trainable parameters:

- Input to Hidden Layer:
  4 x 5 weights/neuron
- Hidden Layer:
  Assuming ReLU, 1 bias/neuron
- Hidden to Output:
  5 weights + 1 bias
- Total: 31 parameters



Do MLPs have convex objectives?



$$Y = W_1 W_2 X$$

Trained to compute the identity function with squared loss, given a single example pair $(x_i, y_i) = (1,\ 1)$

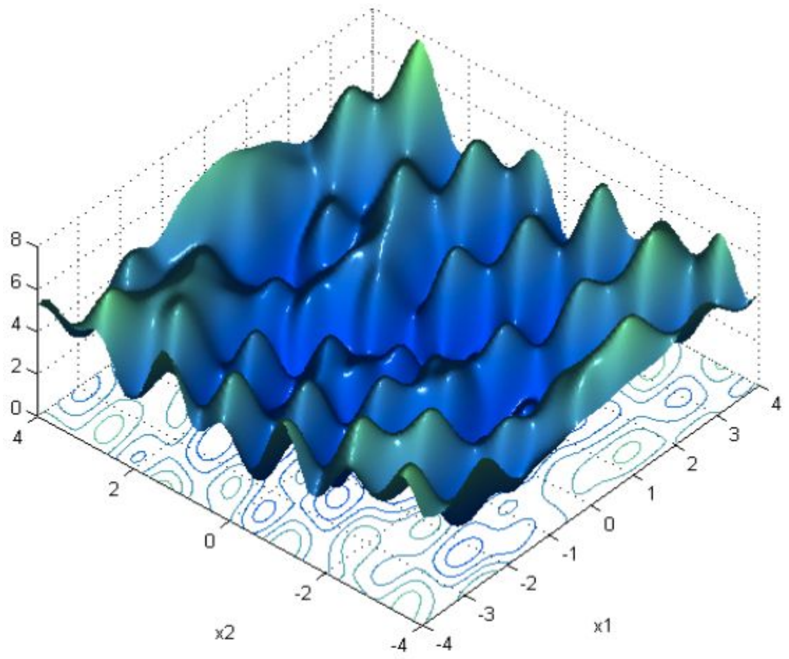
$$L(W) = (1 - W_1 W_2)^2$$

$$\implies W_1 W_2 = 1$$

$$\implies W_2 = \frac{1}{W_1}$$

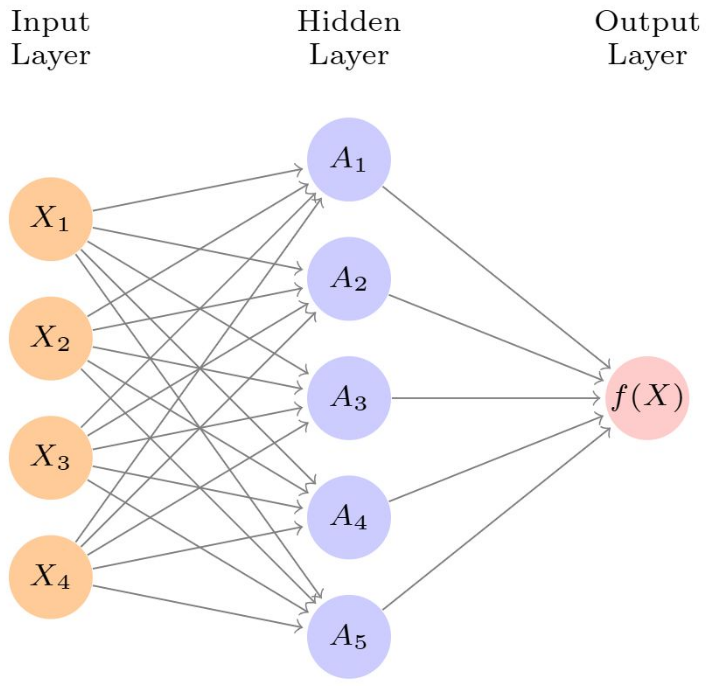
https://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf

MLPs may have many local minima that are close to global minima



https://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf

How to compute the gradient of the loss for an MLP?



Computing the gradients of an MLP (linear output)

$$L = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = z$$

$$z = w_1 X_1 + w_2 X_2 + b_{\text{out}}$$

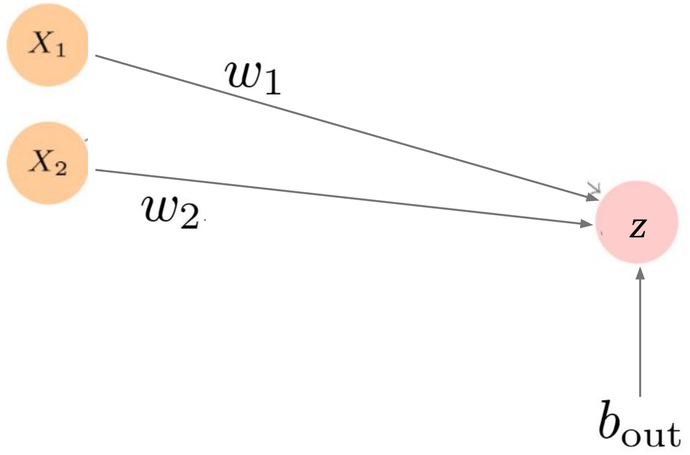
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{d}{d\hat{y}}\left(\frac{1}{2}(\hat{y} - y)^2\right) = (\hat{y} - y)(1) = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{d}{dz}(z) = 1$$

$$\frac{\partial z}{\partial w_1} = \frac{d}{dw_1}\left(w_1 X_1 + w_2 X_2 + b_{\text{out}}\right)$$

$$\frac{\partial L}{\partial w_1} = (\hat{y} - y) \cdot 1 \cdot X_1 \ = (w_1 X_1 + w_2 X_2 + b_{\text{out}} - y) X_1$$



Computing the gradients of an MLP (nonlinear output)

$$L = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = f(z)$$

$$z = w_1 X_1 + w_2 X_2 + b_{\text{out}}$$

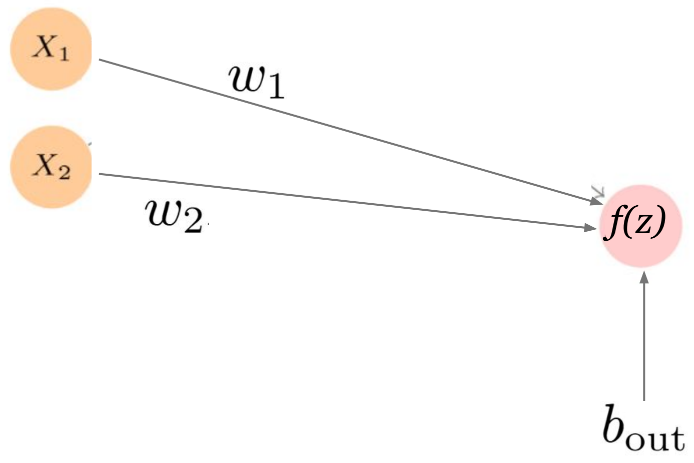
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{d}{d\hat{y}}\left(\frac{1}{2}(\hat{y} - y)^2\right) = (\hat{y} - y)(1) = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{d}{dz} f(z) = f'(z)$$

$$\frac{\partial z}{\partial w_1} = \frac{d}{dw_1}\left(w_1 X_1 + w_2 X_2 + b_{\text{out}}\right) = X_1$$

$$\frac{\partial L}{\partial w_1} = (\hat{y} - y) \cdot f'(z) \cdot X_1$$



Computing the gradients of an MLP (hidden layer)

$$L = \frac{1}{2}(\hat{y} - y)^2 \qquad \hat{y} = f(z)$$

$$z = w_{out}a + b_{\text{out}}$$

$$a = h(t)$$

$$t = w_1X_1 + w_2X_2 + b_1$$

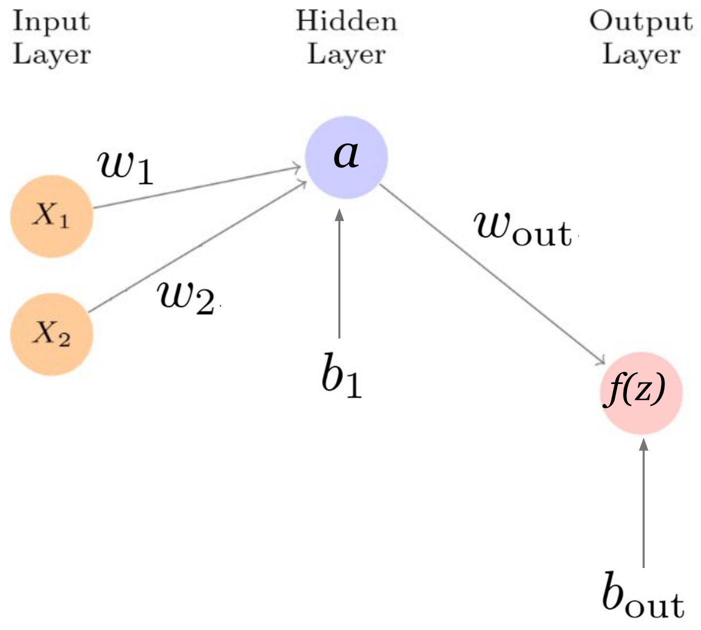
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial a}\frac{\partial a}{\partial t}\frac{\partial t}{\partial w_1}$$

$$\frac{\partial z}{\partial a} = \frac{d}{da}(w_{out}a + b_{out}) = w_{out}$$

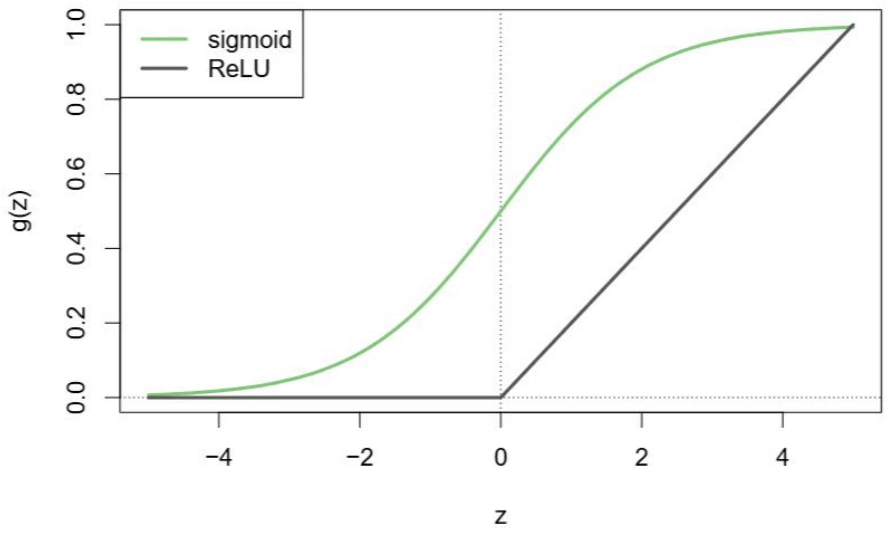
$$\frac{\partial a}{\partial t} = \frac{d}{dt}(h(t)) = h'(t)$$

$$\frac{\partial t}{\partial w_1} = \frac{d}{dw_1}(w_1X_1 + w_2X_2 + b_1) = X_1$$

$$\implies \frac{\partial L}{\partial w_1} = (\hat{y} - y)\cdot f'(z)\cdot w_{out}\cdot h'(t)\cdot X_1$$



ReLU activation avoids vanishing gradient



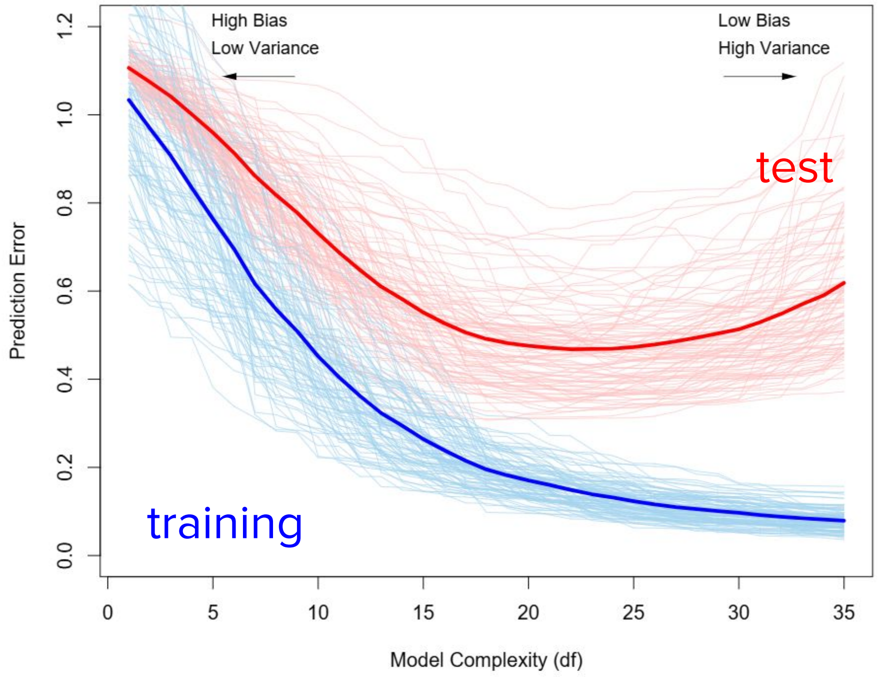
Key questions

I. What do we mean by useful features?

II. How to fit the parameters?

III. **How to prevent overfitting?**

Deep neural networks are susceptible to overfitting



$\boldsymbol{p}$ parameters, $\boldsymbol{n}$ training samples, $\boldsymbol{d}$ features

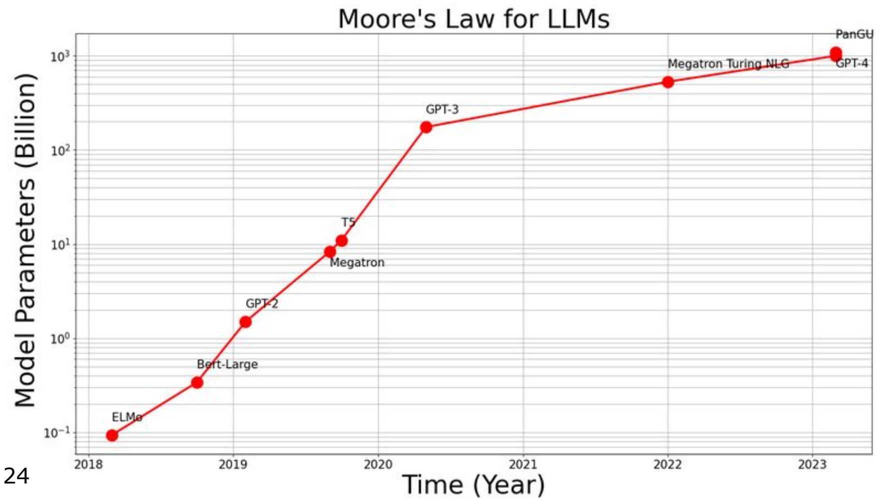
- Classical ML:

$$\boldsymbol{p} \risingdotseq \boldsymbol{d}; \ll \boldsymbol{n}$$

- Modern Neural Networks:

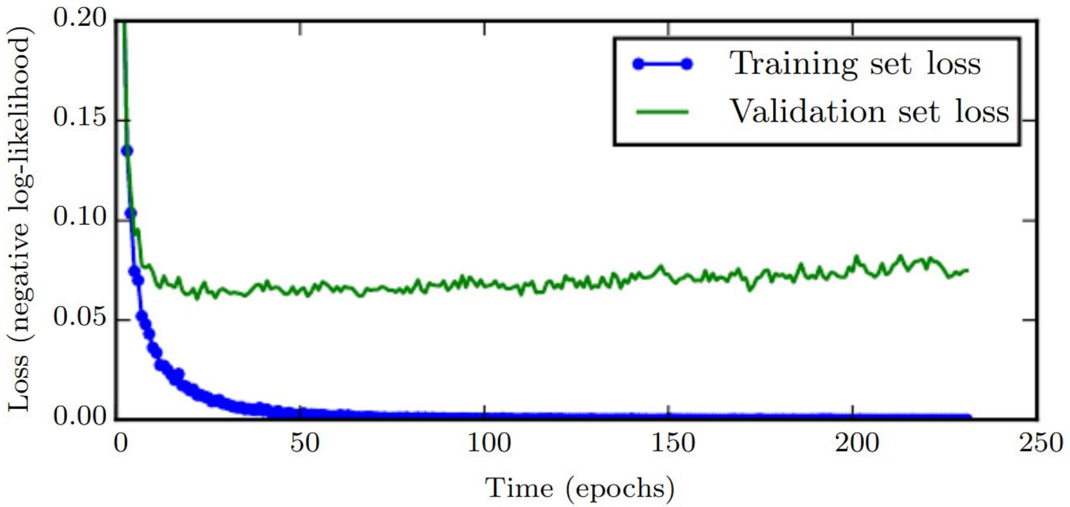
$$\boldsymbol{p} \gg \boldsymbol{nd}$$

Have sufficient representational capacity to memorize the training dataset



Regularization Technique #1: Early stopping

- Periodically evaluate the model on a validation set
    - Save a copy of the model if error on the validation set improves
    - Return to the last model checkpoint
- `patience` is number of times to observe worsening validation set error before giving up
- Can think of number of training steps as a hyperparameter



DL Section 7.8

Regularization Technique #2: Weight decay

$$L = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} L_w(x_i, y_i)$$

$$\nabla_w L = \lambda w + \frac{1}{n}\sum_{i=1}^{n} \nabla_w L_w(x_i, y_i)$$

$$w_t = w_{t-1} - \eta\left(\lambda w_{t-1} + \frac{1}{n}\sum_{i=1}^{n} \nabla_{w_{t-1}} L_{w_{t-1}}(x_i, y_i)\right)$$

$$w_t = (1 - \eta\lambda)w_{t-1} - \eta\frac{1}{n}\sum_{i=1}^{n} \nabla_{w_{t-1}} L_{w_{t-1}}(x_i, y_i)$$

Regularization Technique #3: Data Augmentation

Applications

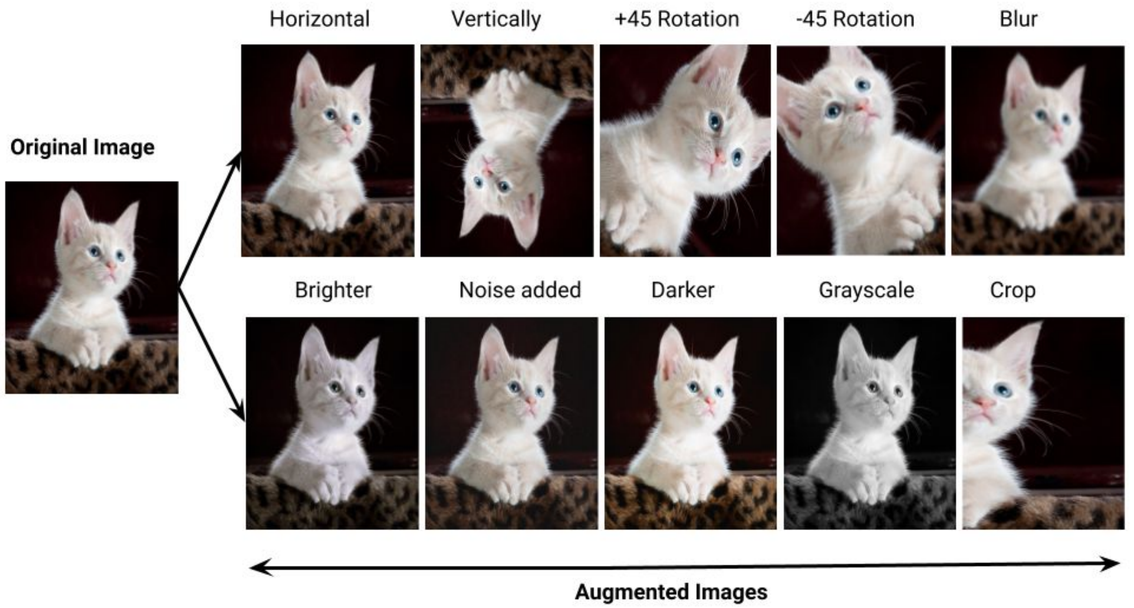
- Classification, object recognition

Must be used wisely

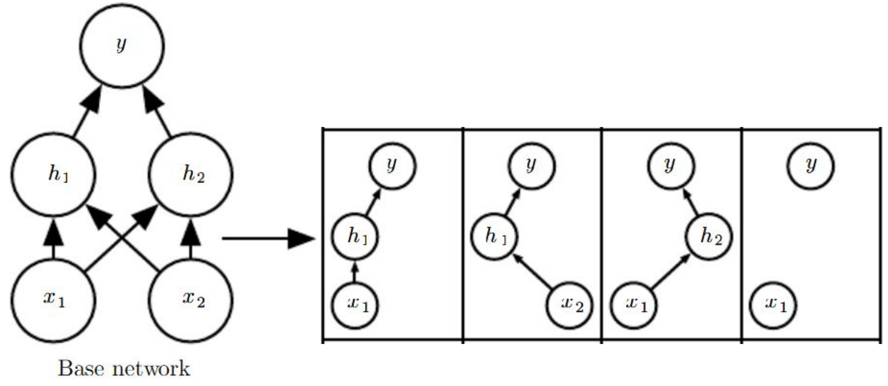
- Certain transformations may change the correct class
- For example, in optical character recognition (OCR):

  "6" vs "9" , "b" vs. "d"

Unclear how to apply to tasks like density estimation



Regularization Technique #4: Dropout (Srivastava 2014)



Standard drop-down regularization:

- Remove some fraction of p the nodes in each layer during training
- Applied to each training point separately

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

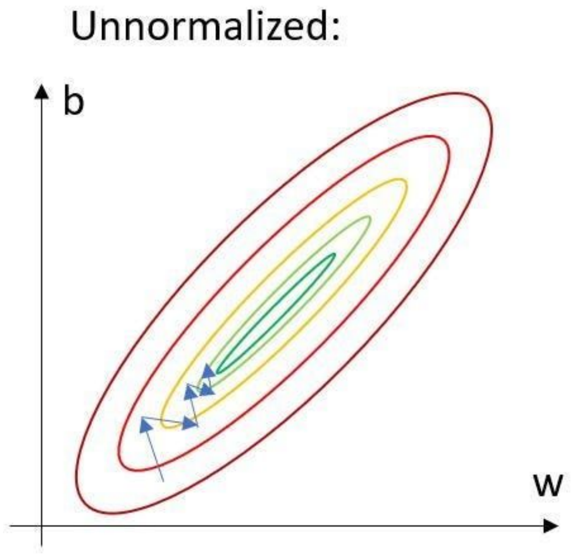
$$E[h'] = h$$

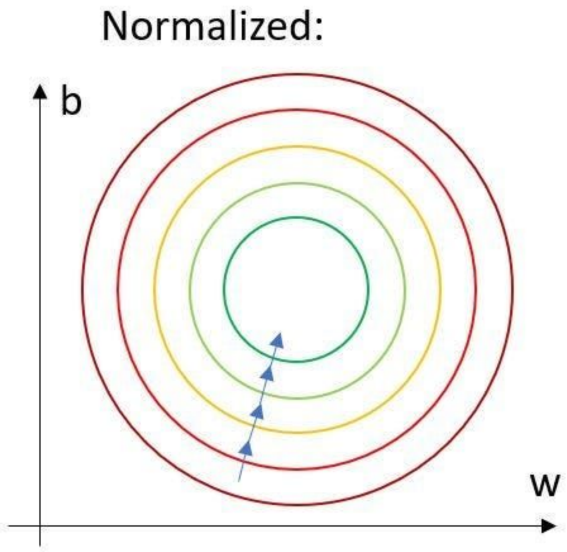
Advantages

- Computationally cheap
  - Generate n binary numbers and multiply them by the state
  - $O(n)$ computation/sample/update
  - At inference, only apply the scale
- Can apply different p to different layers
- Compatible with most architectures, training procedures
- Implemented in most libraries
- Better than weight decay

D2L Section 5.6, DL Section 7.12

Regularization (kindof) Technique #5: Normalization





https://medium.com/@ngneha090/batch-normalization-in-deep-learning-5f200f6f7733

Regularization (kindof) Technique #5: Normalization

$$\mathrm{BN}(\boldsymbol{x}) = \gamma \frac{\boldsymbol{x} - \hat{\mu}_{\mathcal{B}}}{\hat{\sigma}_{\mathcal{B}}} + \boldsymbol{\beta}$$

Where:

$$\hat{\mu}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x} \in \mathcal{B}} \boldsymbol{x} \text{ (Batch mean)}$$

$$\hat{\sigma}_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x} \in \mathcal{B}} (\boldsymbol{x} - \hat{\mu}_{\mathcal{B}})^2 + \epsilon \text{ (Batch variance)}$$

$\boldsymbol{\gamma}, \boldsymbol{\beta}$: learnable parameters, and
ϵ: hyperparameter to clip variance estimate.

Benefits
- Improve optimization by create a consistent scale across features
- Provides a form of regularization by exposing the network to different distributions across batches

Considerations
- Requires reasonable batch size
- Distributions should not vary significantly between batches (e.g. in reinforcement learning)
- Should be used in combination with other techniques

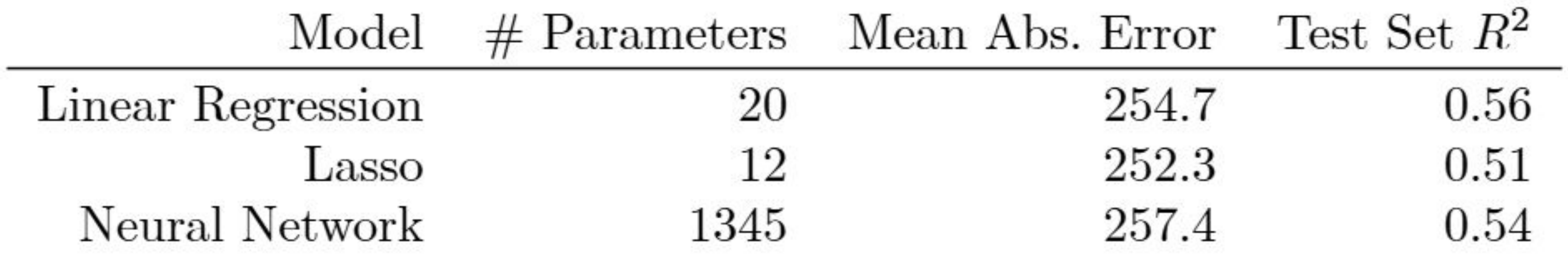
Applications to Deep Learning
- Apply to each neuron independently
- Apply to each layer independently

D2L Section 8.5

When is all this effort worth it? A case study

Baseball Player Salary Dataset

- 263 players
- 19 variables
- 176/87 train/test split
- Goal: Predict player salary



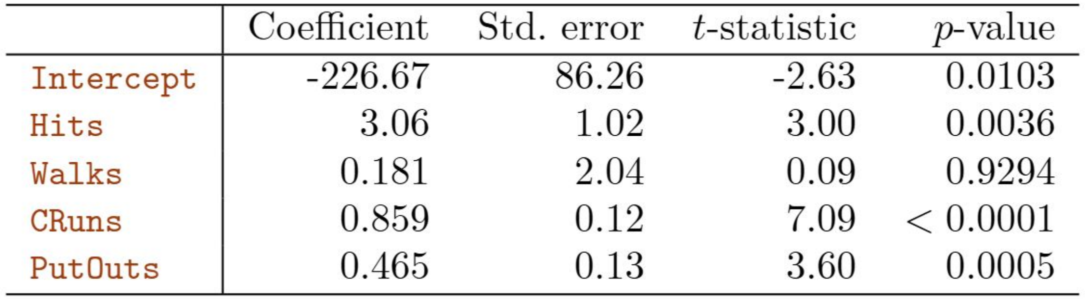
Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1345	257.4	0.54

ISL Section 10.6

Attempt neural network after exhausting simpler options

In this case, the Neural Network…

- Achieved similar accuracy
- Required configuring hyperparameters
- Unlike linear methods, parameter values do not suggest a direct interpretation



	Coefficient	Std. error	t-statistic	p-value
`Intercept`	-226.67	86.26	-2.63	0.0103
`Hits`	3.06	1.02	3.00	0.0036
`Walks`	0.181	2.04	0.09	0.9294
`CRuns`	0.859	0.12	7.09	< 0.0001
`PutOuts`	0.465	0.13	3.60	0.0005

Now that we're at the end of the lecture, you should be able to…

- ★ Motivate the need for **learned features**.
- ★ State the **universal approximation theorem** of a multilayer perceptron (MLP) including the the conditions (**nonlinear activation, sufficient depth**) and caveats (**potentially infinite width**) to achieve the associated guarantee.
- ★ Use the **chain rule to compute the gradient** with respect to the parameters.
- ★ List and describe theoretical issues (**vanishing gradients, model complexity**) that must be addressed to apply an MLP to real data.
- ★ Describe the **architecture** and **parameterization** of a MLP.
- ★ List, describe, and apply **strategies to reduce overfitting** such as **early stopping**, **weight decay**, **data augmentation**, **drop-out**, as well as **batch/layer normalization**.
- ★ Recommend either a **traditional ML algorithm** or a **neural network** depending on the nature of the problem, the data, and specific goals.