Fundamentals of Machine Learning and Neural Networks

AI in Medicine

Ricardo Henao

What is happening today?

- This class is focused on “Deep Learning”
  - What it is
  - Some of its many applications
  - How to *build your own* deep networks
- These algorithms *learn* from data to automate tasks and make predictions
- Today, you will learn about what we mean by “deep learning” and *build* your first deep network

Deep Learning is state-of-the-art for *many* applications

- Deep Learning is **not new**—many of the techniques go back decades
- Recent resurgence due to *amazing* performance on benchmark tasks
- One key task was the ImageNet Challenge
  - Want to recognize what is in an image (1 of 1000 categories)
  - Have ~1 million example images
  - Very relevant for things such as image search
- Example images are shown on the right, with predicted categories beneath each image

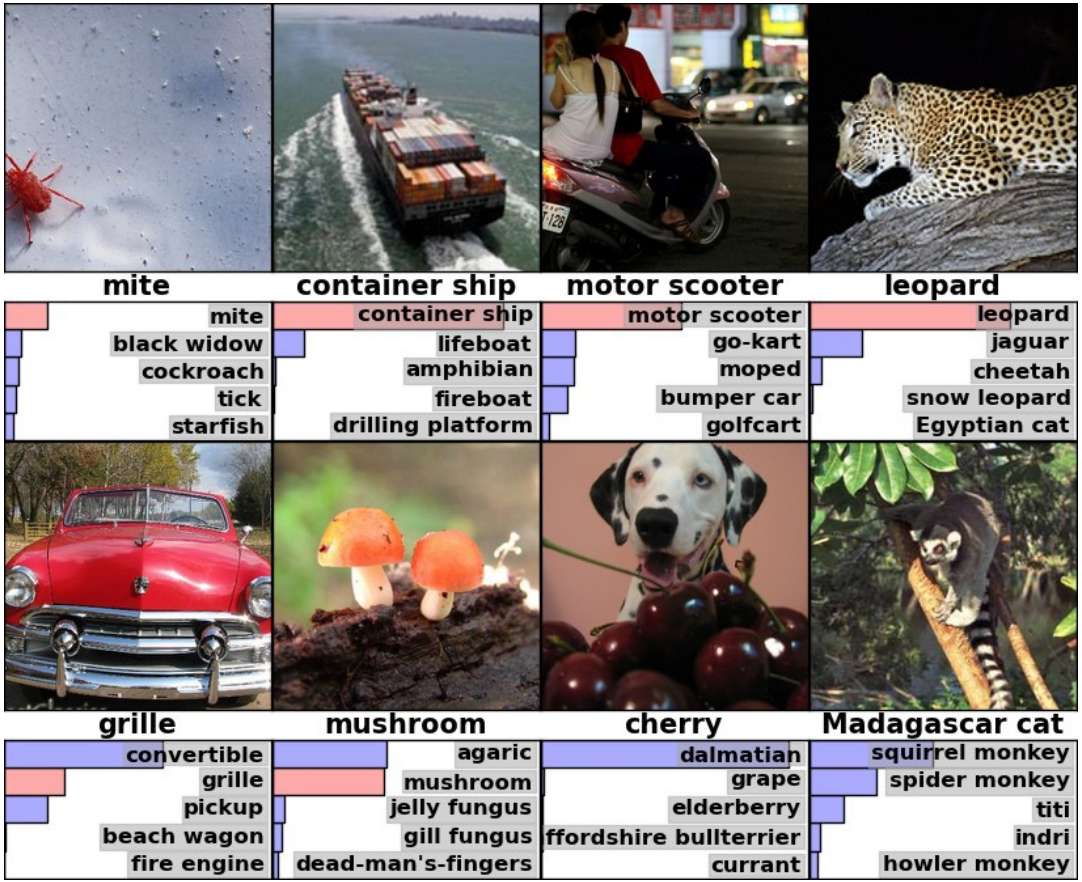
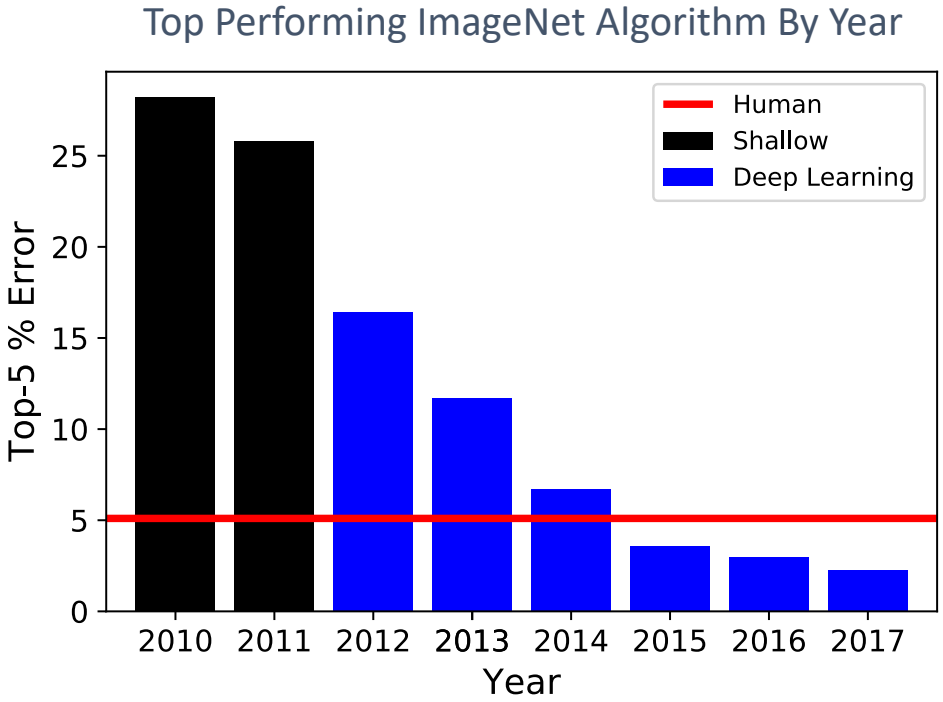


Figure from Krizhevsky *et al* 2012

Deep Learning can surpass human performance

- For ImageNet:
  - Deep Learning was a *huge* jump forward
  - State-of-the-art systems **significantly outperform humans** on the same task
- These use “Convolutional Neural Networks,” which you will learn about tomorrow



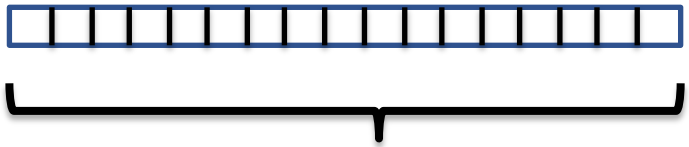
Deep Learning beats human performance in many tasks

- Famously, Google DeepMind trained “AlphaGo” to beat the world champion Go player (a complex game)
- AlphaGo’s largely works by Deep Learning techniques (learned by repeatedly playing the game)
- Many other examples:
  - Voice Recognition
  - Object Detection
  - Text Translation
  - *Etc.*
- So, what is Deep Learning?

We need to understand “shallow” learning before “deep” learning

A “SHALLOW” NETWORK

Learning a Predictive Model Based on Labeled Data

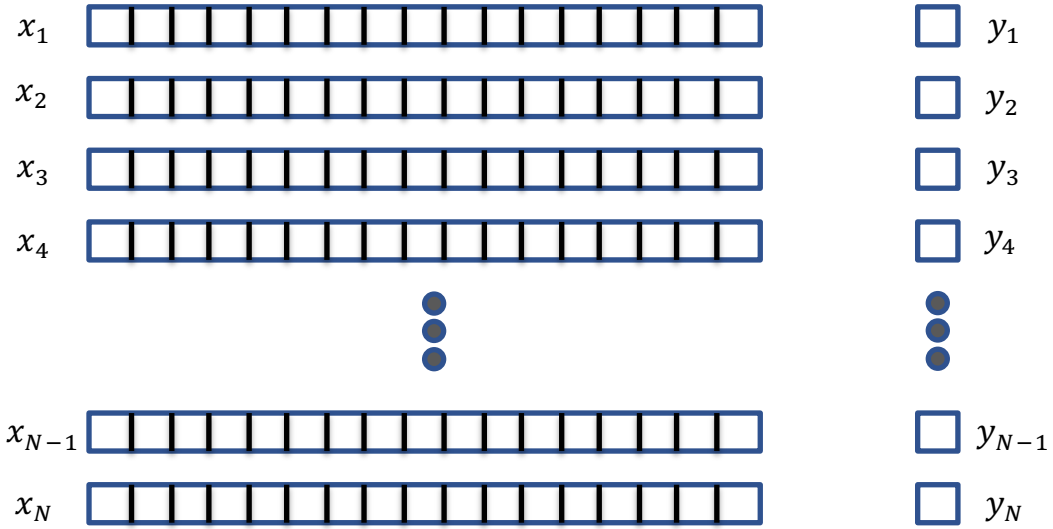


x, data/features for a subject

y, associated label 0/1

End goal: *predict* y from x

Training Set (Historical Data)

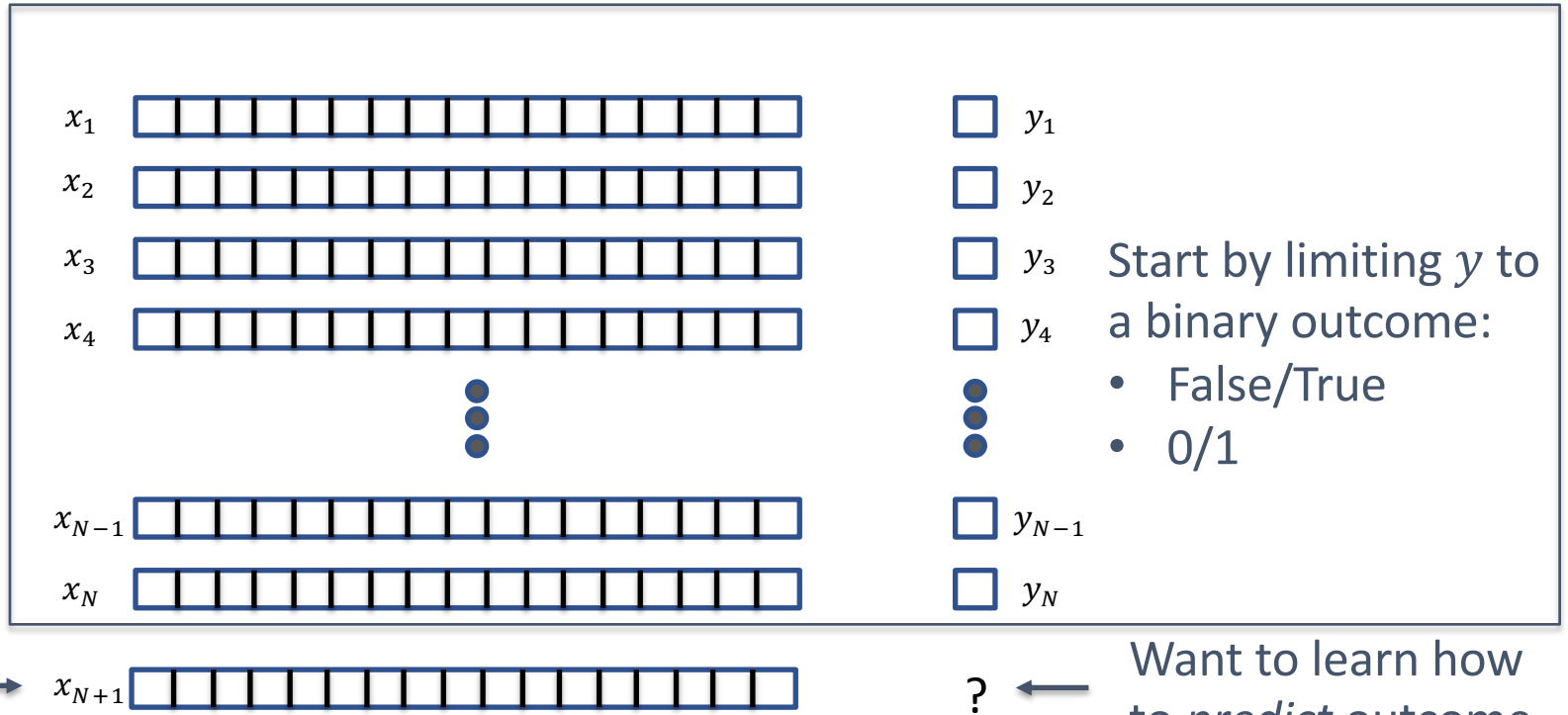


Start by limiting y to a binary outcome:

- False/True
- 0/1

Making Predictions

Learn from Experience

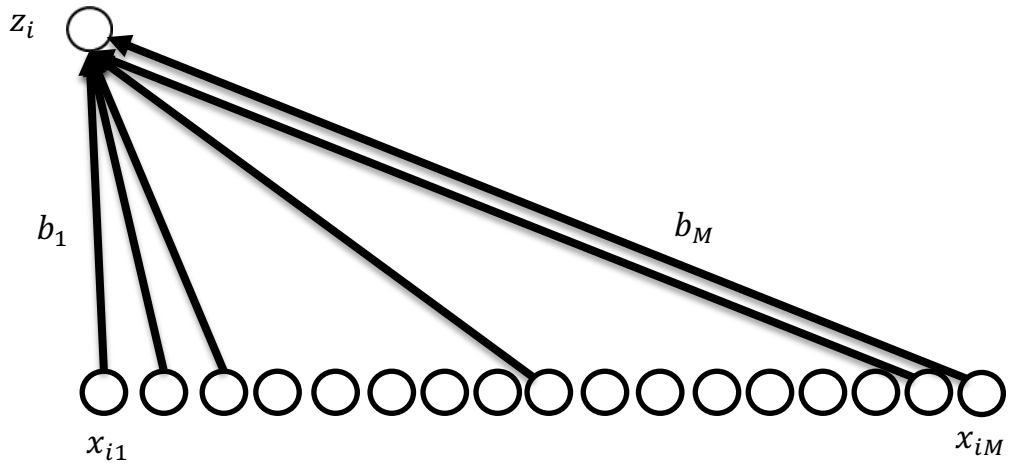


Start by limiting y to a binary outcome:

- False/True
- 0/1

New Data → x_{N+1} → ? ← Want to learn how to *predict* outcome

Linear Predictive Model



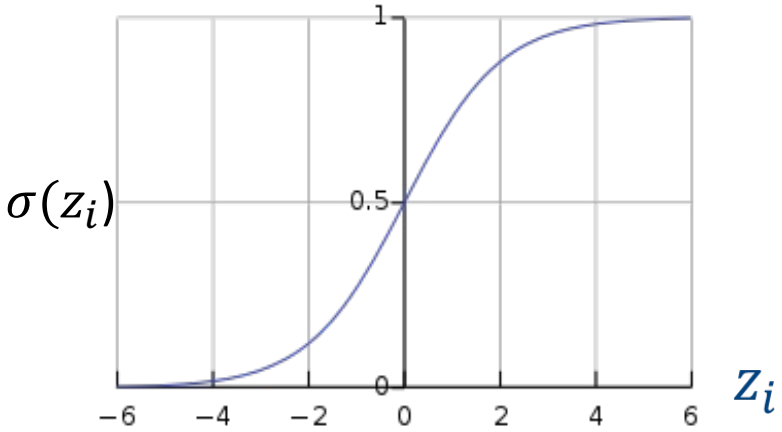
$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

Convert to a Probability

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Extra constant (bias)

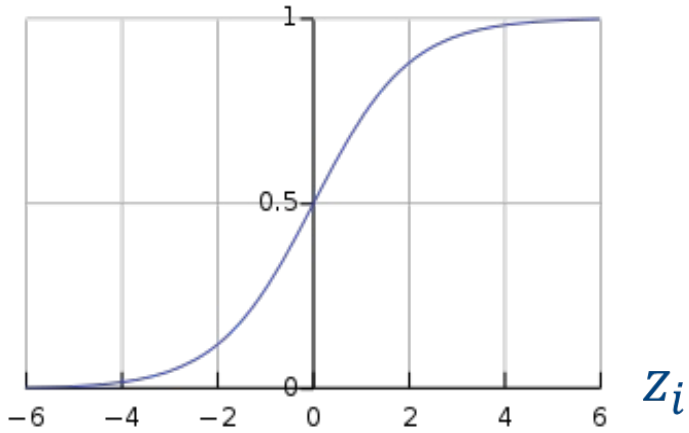
$$p(y_i = 1 | x_i) = \sigma(z_i)$$



Convert to a Probability

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

$$p(y_i = 1|x_i) = \sigma(z_i) = \frac{\exp(z_i)}{1+\exp(z_i)}$$

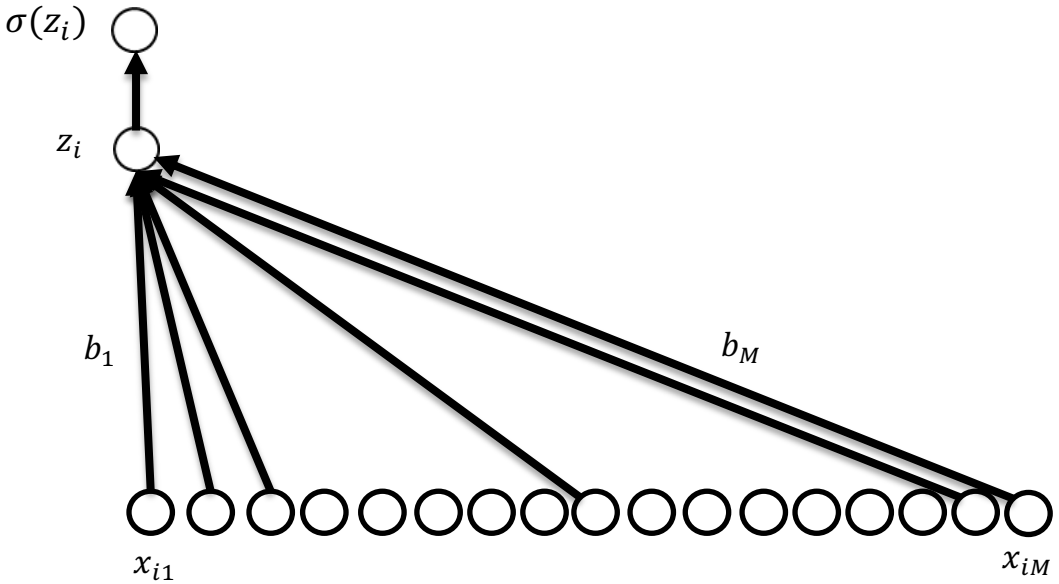


- Large and positive z_i indicates that event $y_i = 1$ is likely
- Large and negative z_i indicates that event $y_i = 0$ is likely

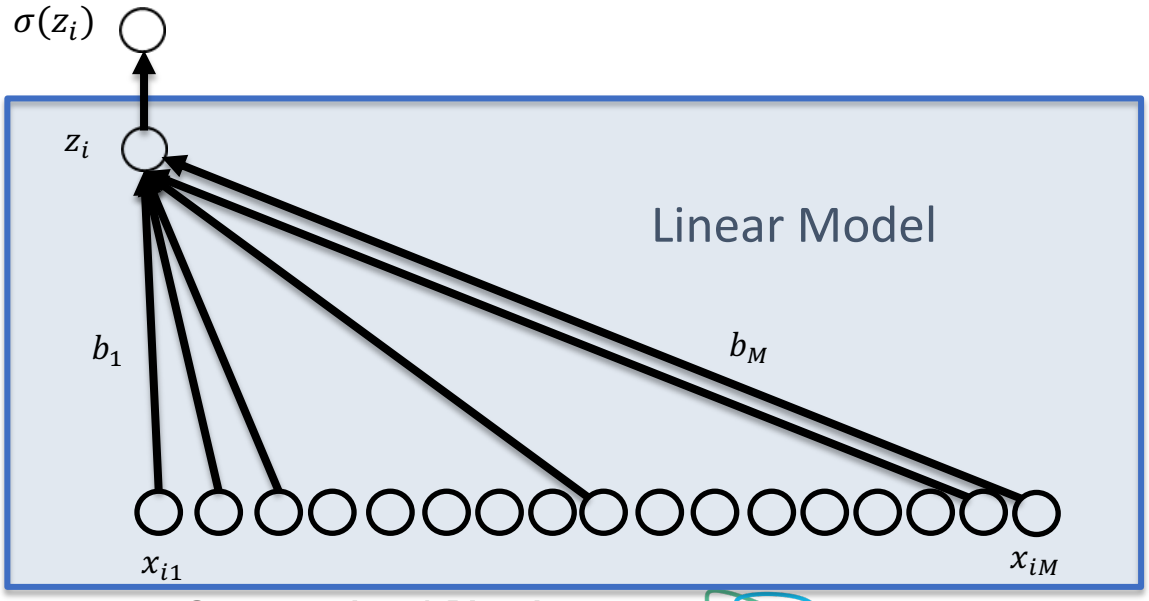
Predictive Machine (Model or Network)

We need to specify a network or model that can make predictions on an outcome given features after being trained by learning from experience

Logistic Regression

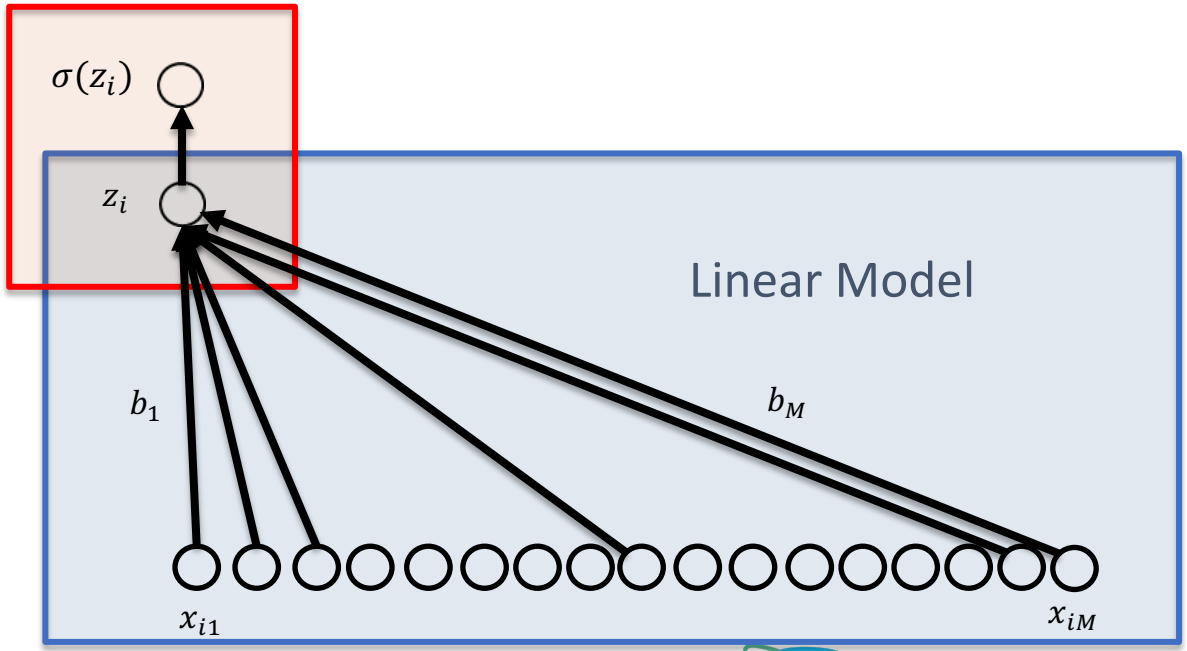


Logistic Regression



Logistic Regression

Convert to Probability

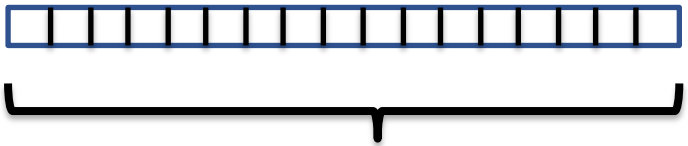


What do the parameters and model mean?

AN EXAMPLE

Example

- Outcome:
  - $y_i = 1$, it rains on day i;
  - $y_i = 0$, it does not rain on day i
- Features:
  - On day i what is the $\{cloud\ cover, humidity, temperature, air\ pressure, \dots\}$



x_i, features for day i

y_i, did it rain on day i

Example

- Outcome: $y_i = 1$, it rains on day i; $y_i = 0$, it does not rain on day i
- Features: On day i what is the

  $\{1: cloud\ cover, 2: humidity, 3: temperature, \dots\}$

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Cloud Cover Humidity

- If cloud cover is positively related to rainfall, b_1 should be positive

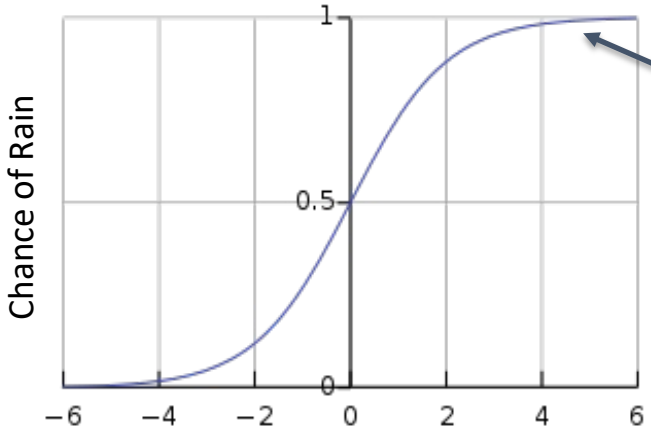
Impact on the Sigmoid Function

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Cloud Cover

Humidity

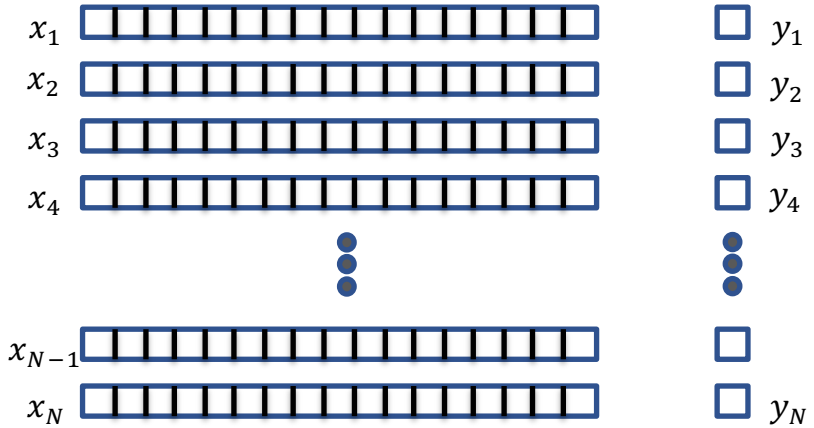
$$p(y_i = 1|x_i) = \sigma(z_i)$$



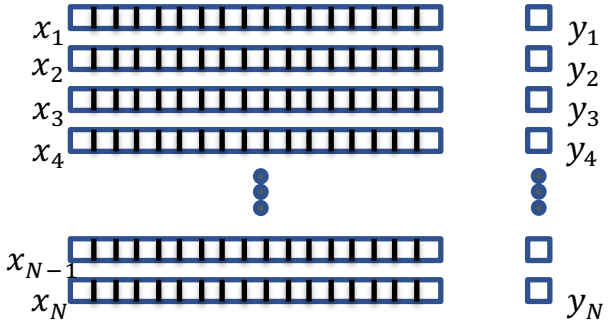
As the value z_i increases, the chance of rain increases

Building the Training Set

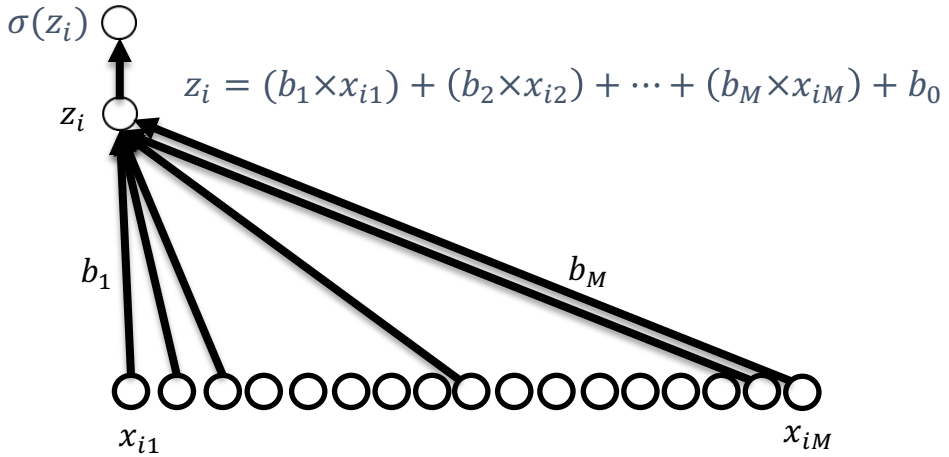
- Need to learn the parameters
- Requires *training data*
- Record data from N days
  - Capture features: $\{cloud\ cover, humidity, temperature, \ldots\}$
  - Did it rain?



Learning Model Parameters



Training Set

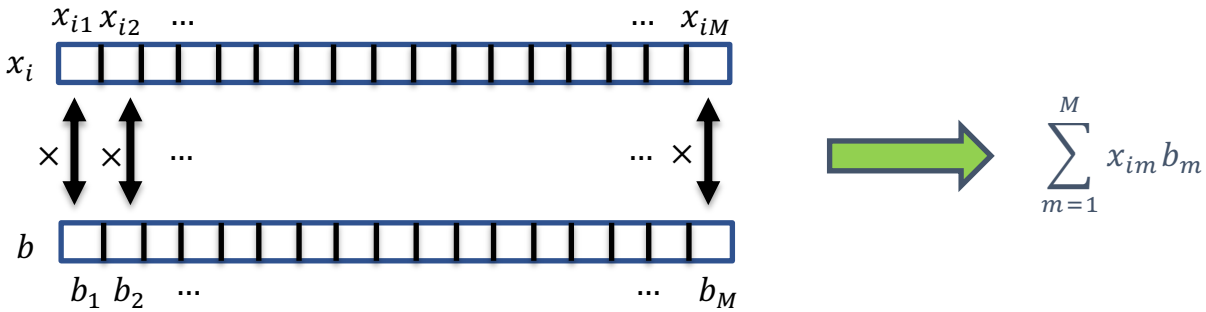


Logistic Regression Model (or "Network")

$(b_0, b_1, \ldots b_N)$ Learned Parameters

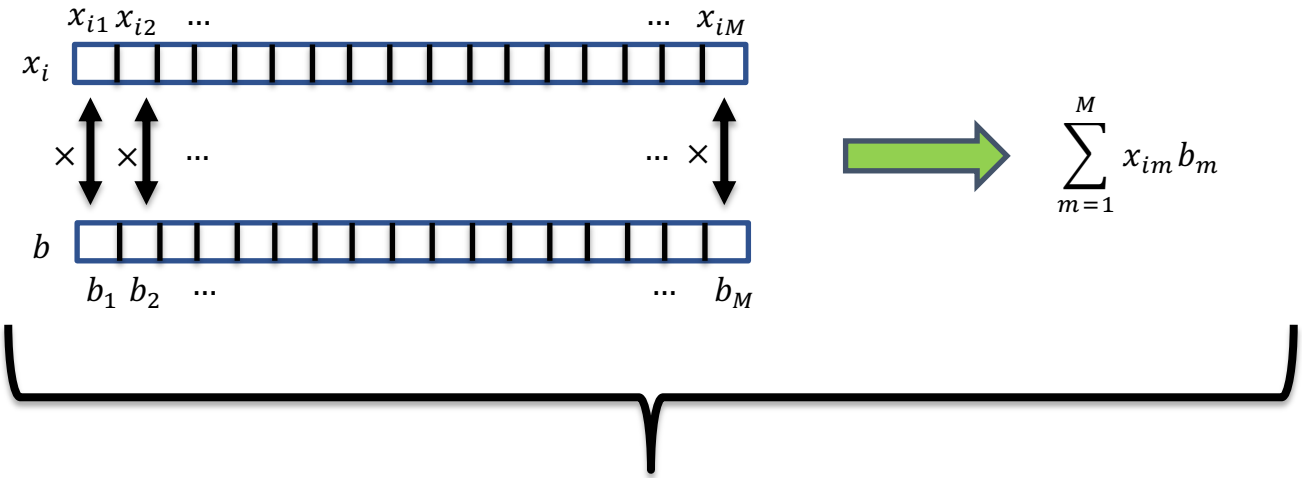
Interpretation of Logistic Regression

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$



Interpretation of Logistic Regression

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$



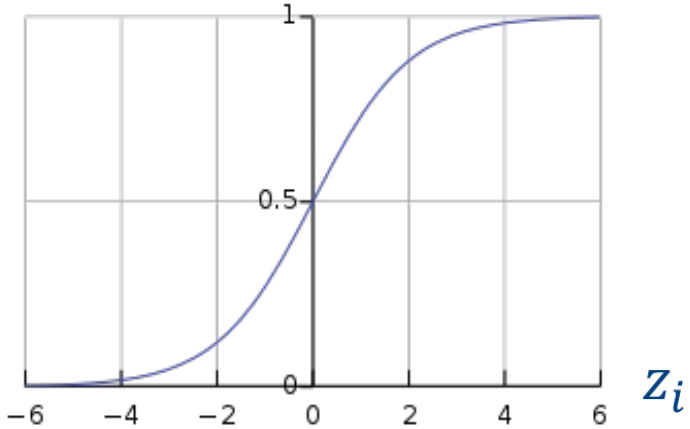
Compact Notation: $x_i \odot b$ (or "inner product")

Interpretation of Logistic Regression

$$z_i = b_0 + (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

$$= b_0 + x_i \odot b$$

$$p(y_i = 1|x_i) = \sigma(z_i)$$



- May think of vector b as a template or filter (will visualize to make clear)
- If x_i is aligned/matched with b, then $x_i \odot b$ will be large
- The parameter b_0 is a bias to correct for class prevalence

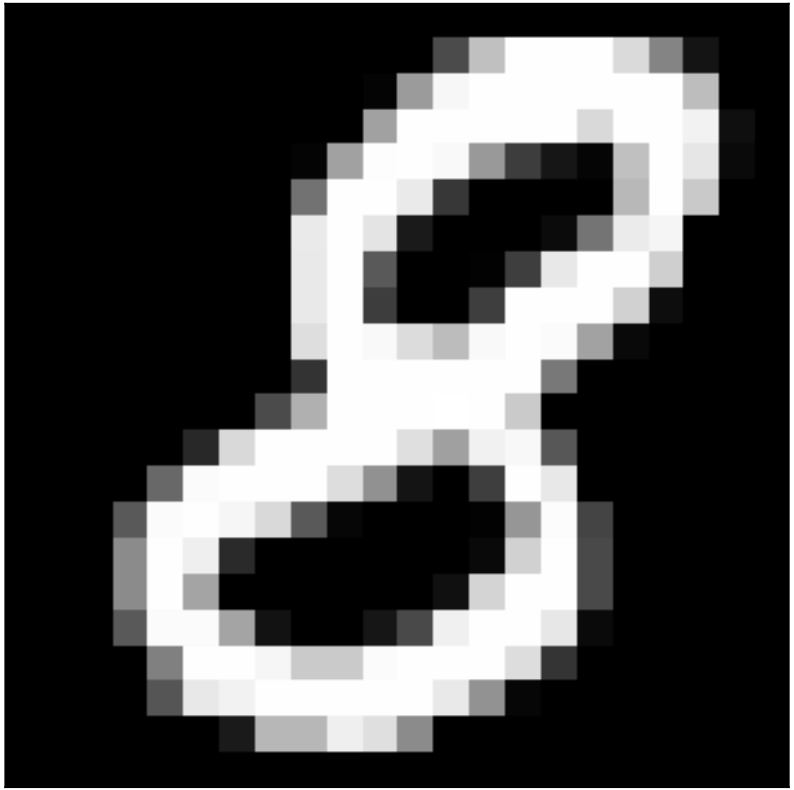
Image classification

RECOGNIZING HANDWRITTEN DIGITS

The MNIST Dataset

- The Modified National Institute of Standards and Technology (MNIST) contains pictures of handwritten digits (0,1,2,…)
- Want to be able to tell what digit each image is (*e.g.*, optical character recognition)

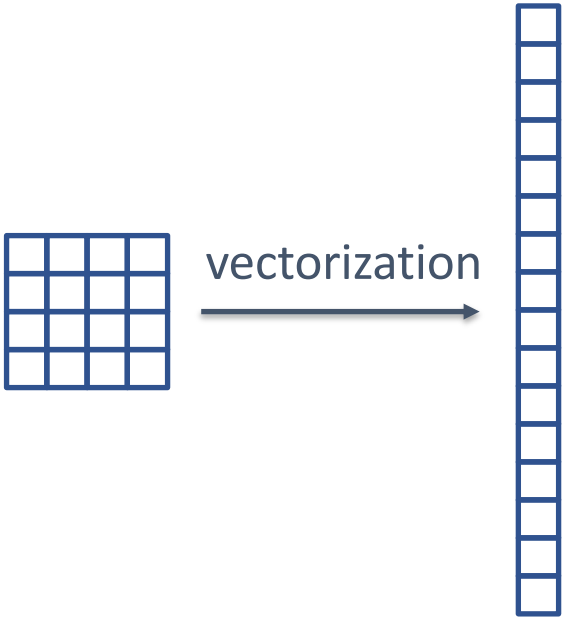
Images are Encoded as Numbers



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	118	200	223	155	155	23	0	0	0	0	0
0	0	0	0	0	0	0	0	6	124	214	253	253	253	253	254	229	213	67	0	0	0
0	0	0	0	0	0	0	43	198	253	254	253	247	175	175	236	253	253	108	0	0	0
0	0	0	0	0	0	9	212	253	253	224	58	19	0	0	16	139	247	247	71	0	0
0	0	0	0	0	0	58	254	254	186	14	0	0	0	0	0	0	139	254	118	0	0
0	0	0	0	0	0	58	253	183	8	0	0	0	0	0	0	36	222	253	118	0	0
0	0	0	0	0	0	54	250	128	2	0	0	0	6	71	192	237	253	247	71	0	0
0	0	0	0	0	0	0	223	253	50	0	18	123	198	253	254	253	247	85	0	0	0
0	0	0	0	0	0	0	69	241	227	136	200	253	253	253	254	192	34	0	0	0	0
0	0	0	0	0	0	0	0	178	254	256	254	254	254	149	59	0	0	0	0	0	0
0	0	0	0	0	0	0	101	253	253	254	253	253	253	42	0	0	0	0	0	0	0
0	0	0	0	0	8	138	247	253	243	159	198	243	253	199	0	0	0	0	0	0	0
0	0	0	0	12	183	253	253	253	50	0	0	49	253	214	0	0	0	0	0	0	0
0	0	0	0	180	254	253	213	50	2	0	0	71	253	214	0	0	0	0	0	0	0
0	0	0	0	234	217	97	10	0	0	0	23	207	254	215	0	0	0	0	0	0	0
0	0	0	174	253	156	0	0	0	0	45	215	253	253	95	0	0	0	0	0	0	0
0	0	9	210	253	163	5	19	49	130	244	253	251	137	4	0	0	0	0	0	0	0
0	0	13	229	253	254	192	253	253	253	254	253	137	0	0	0	0	0	0	0	0	0
0	0	0	160	253	254	253	253	253	253	239	132	4	0	0	0	0	0	0	0	0	0
0	0	0	18	112	194	254	254	254	163	59	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

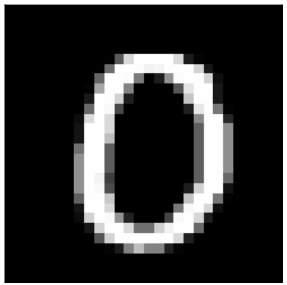
Vectorization

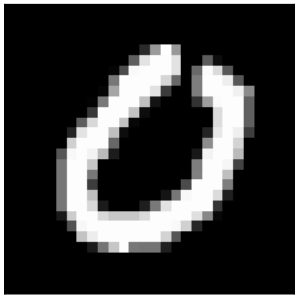
- We will start talking about deep learning *without* using the structure of the image
- We will return to this in a future lecture
- To convert an image into an unstructured set of numbers, we *vectorize it*

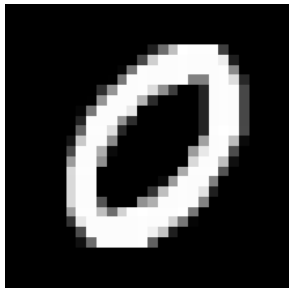


Start With The Binary Case

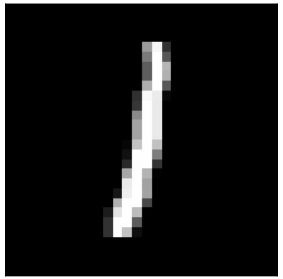
Zeros

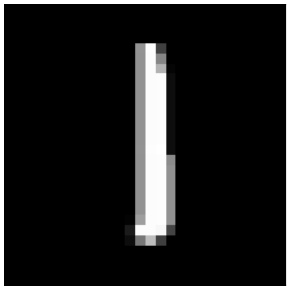


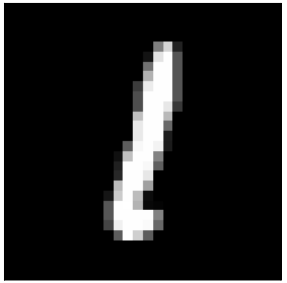




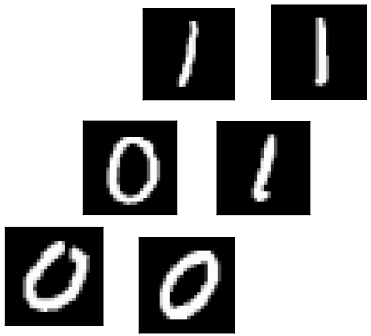
Ones



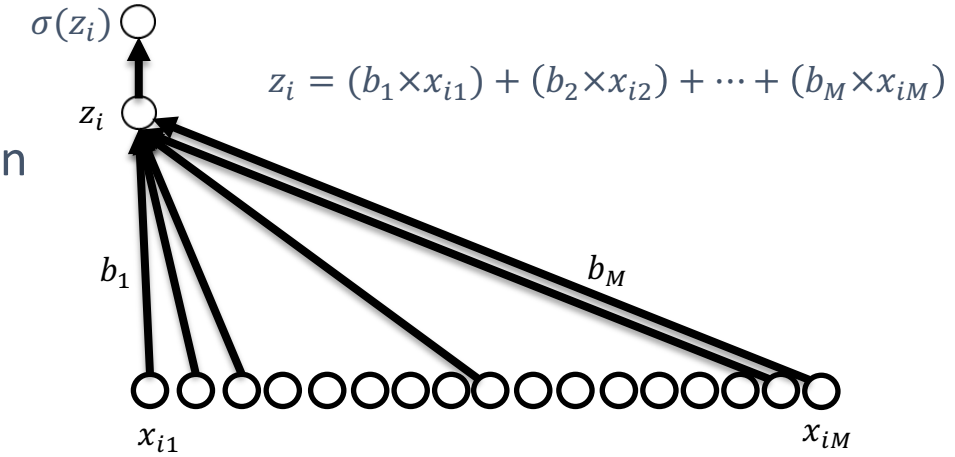




Learning on MNIST



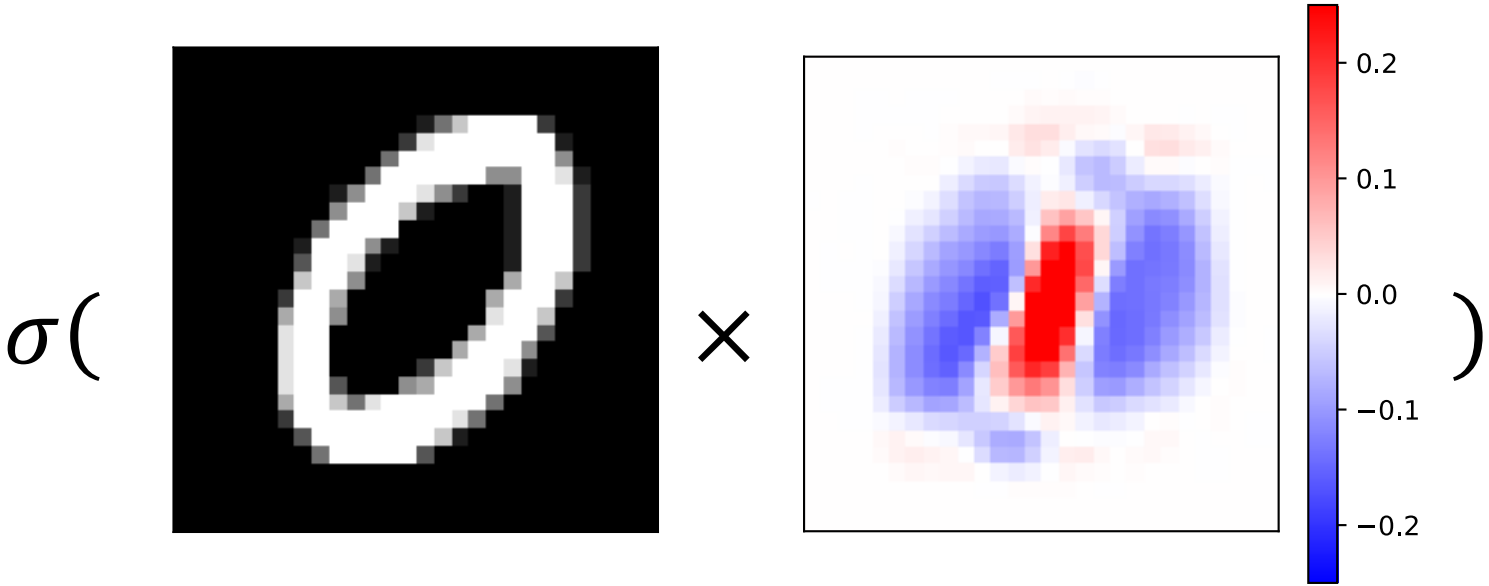
Vectorization



Logistic Regression Model (or "Network")

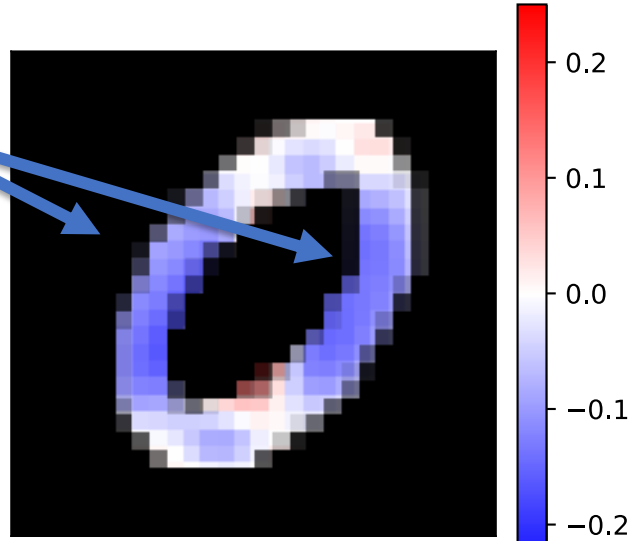
$(b_0, b_1, \ldots b_N)$ Learned Parameters

Zooming in on 0/1



Zooming in on 0/1

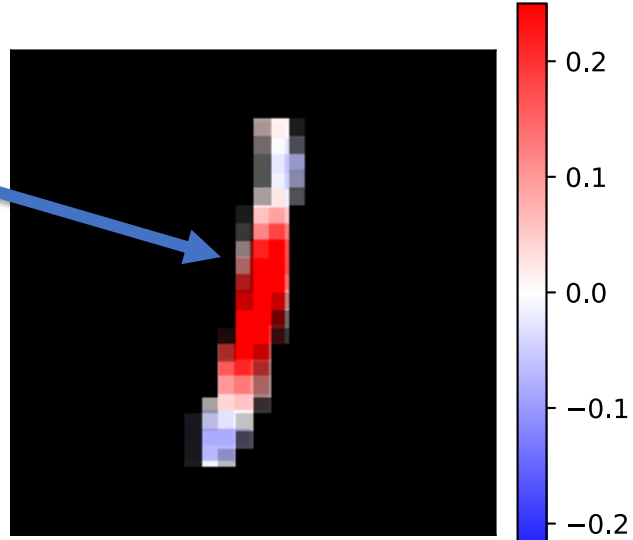
Negative Sections



$$\sigma(\quad) = 0.006$$

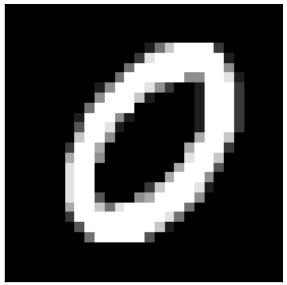
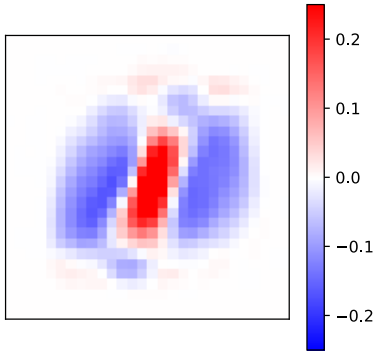
Zooming in on 0/1

Positive Section



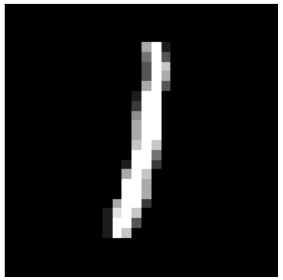
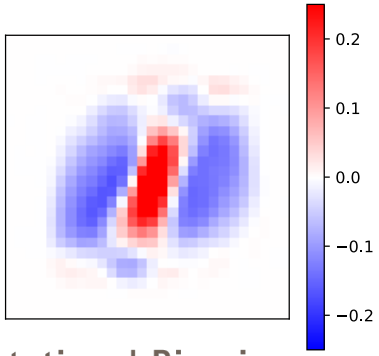
$$\sigma(\quad) = .991$$

Learned Weights for 0/1

$\sigma($  $\times$ 

$) = .006$

We think that this is a “zero” (.6% chance it is a “one”)

$\sigma($  $\times$ 

$) = .991$

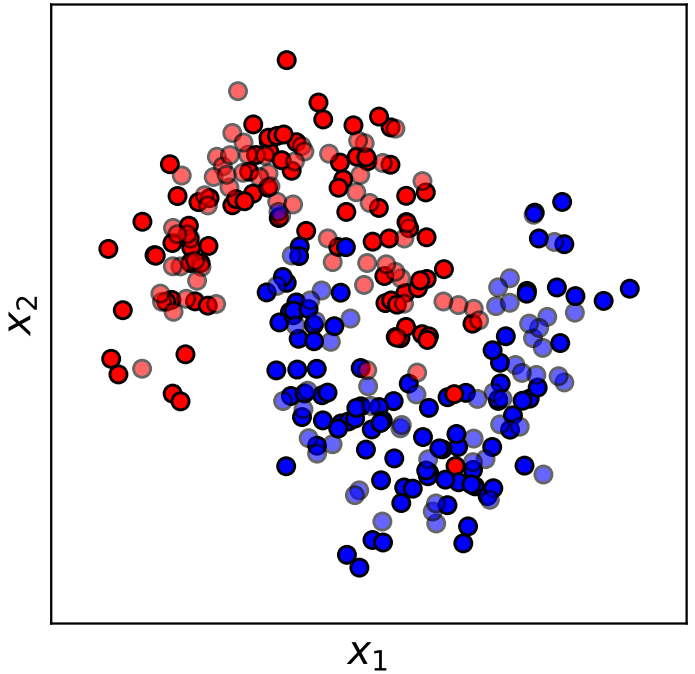
We think that this is a “one” (99.1% chance it is a “one”)

From Shallow to Deep Learning

GENERALIZING LOGISTIC REGRESSION

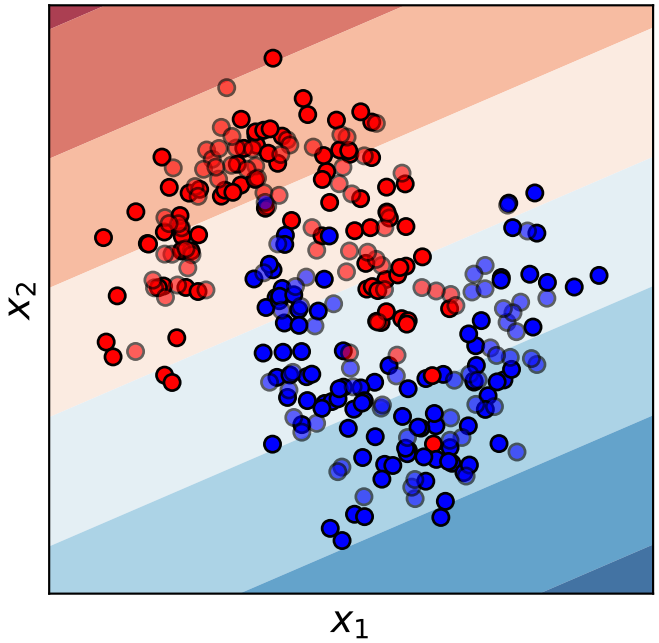
Logistic Regression is a “Linear” Classifier

- Also referred to as a generalized linear model
- Can only split data by linear trends



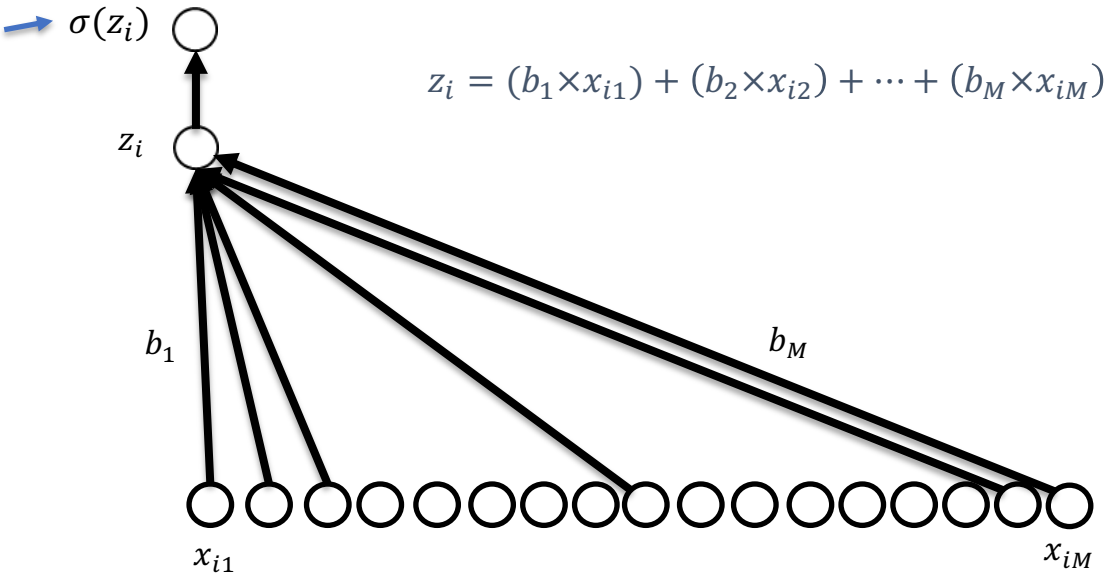
Non-linear classifier

- Linear classifiers can only represent limited relationships
- Often want to use a classifier that can handle non-linearities.
- Many different ways of creating non-linear classifiers



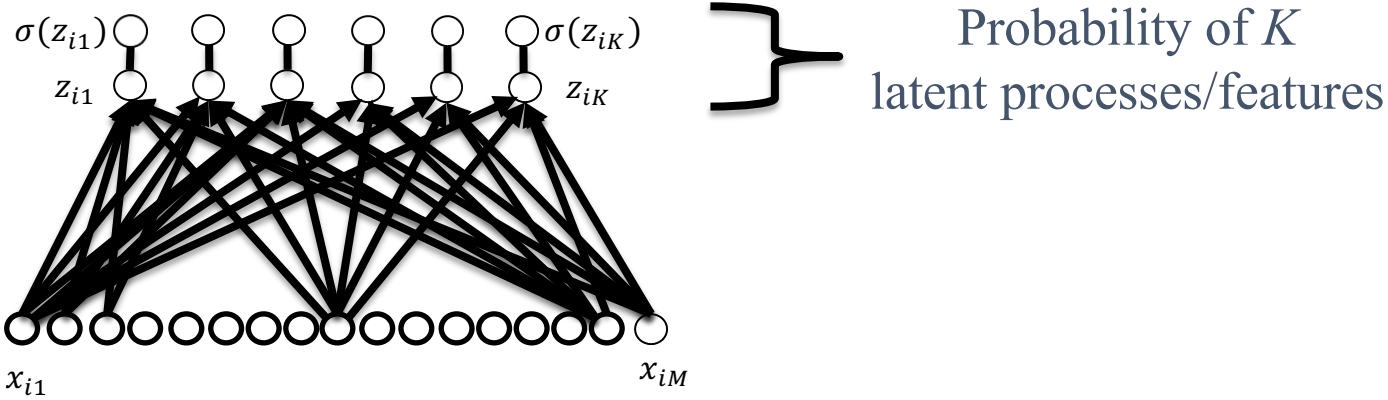
Can we modify Logistic Regression to learn complex structures?

Probability of a single outcome

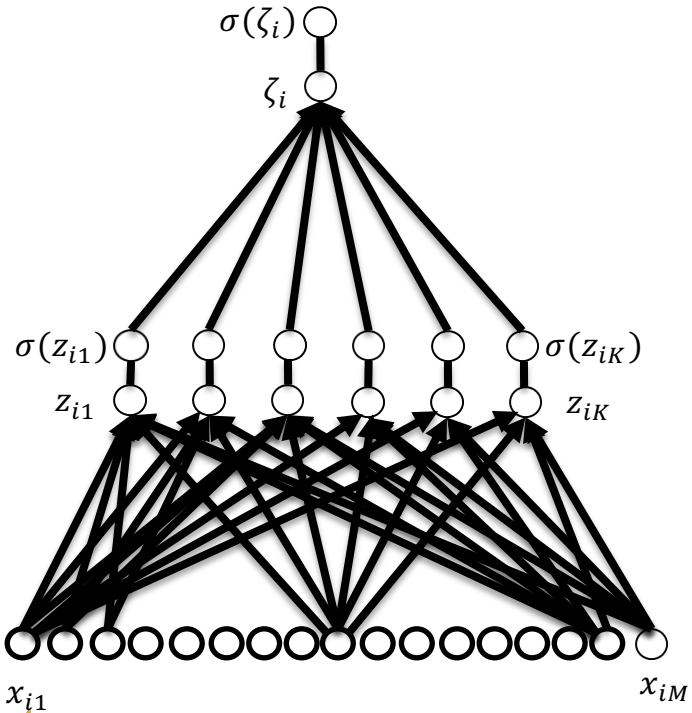


$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

Generalization of Logistic Regression: Learned Features



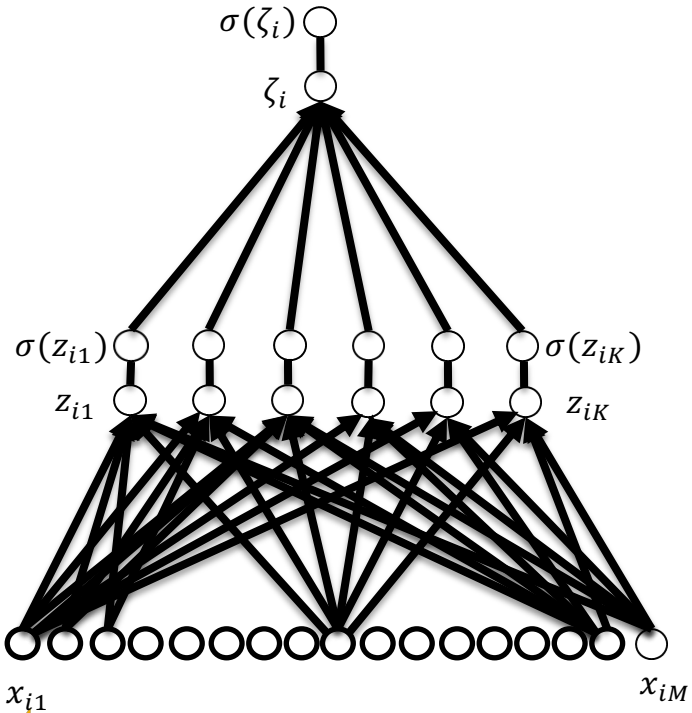
Extended Logistic Regression



Probability of particular outcome

Probability of K latent processes

Extended Logistic Regression



Probability of particular outcome

Probability of K latent processes

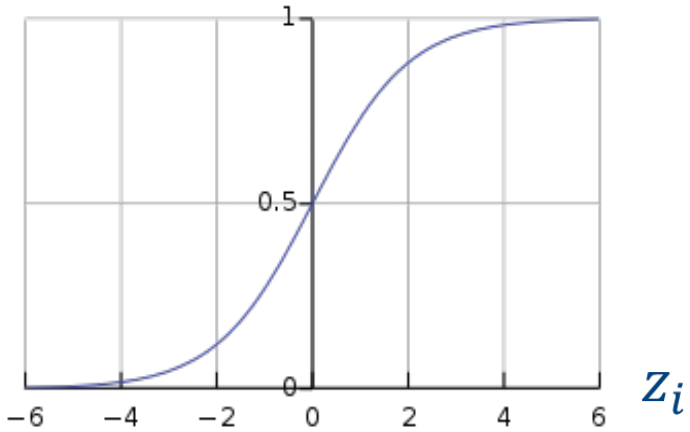
May be viewed as logistic regression on K latent features, rather than directly on the M components of raw data

Recall Logistic Regression

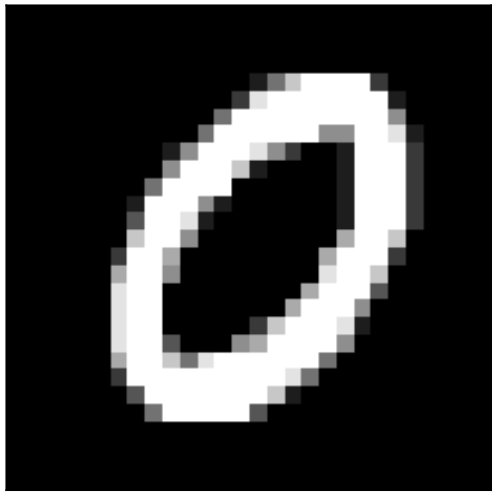
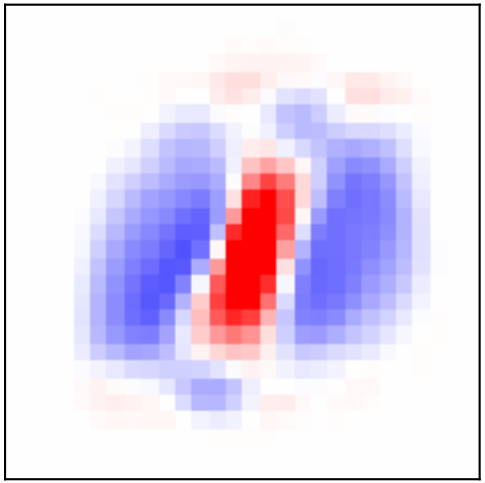
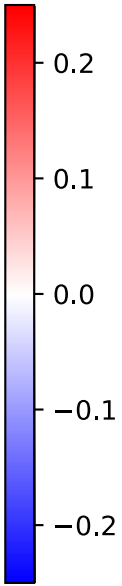
$$z_i = b_0 + (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

$$= b_0 + x_i \odot b$$

$$p(y_i = 1 | x_i) = \sigma(z_i)$$



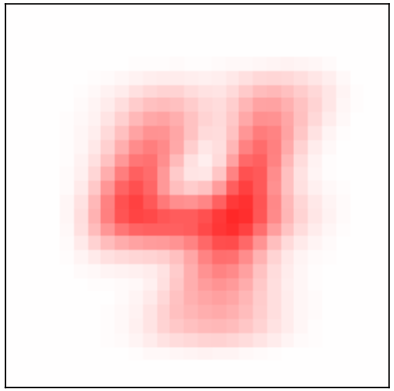
Why Limit Ourselves to Only One Filter?

$\sigma($  $\times$  

$)$

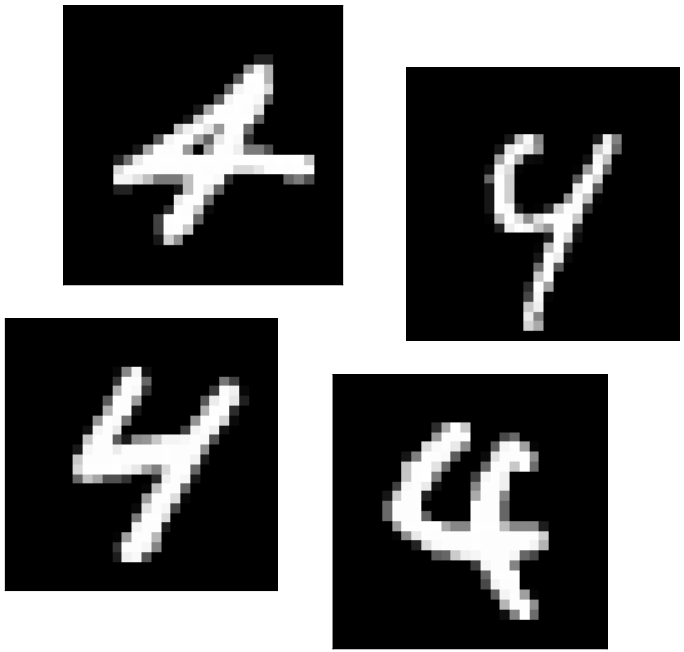
Return to MNIST: Many ways of writing “4”

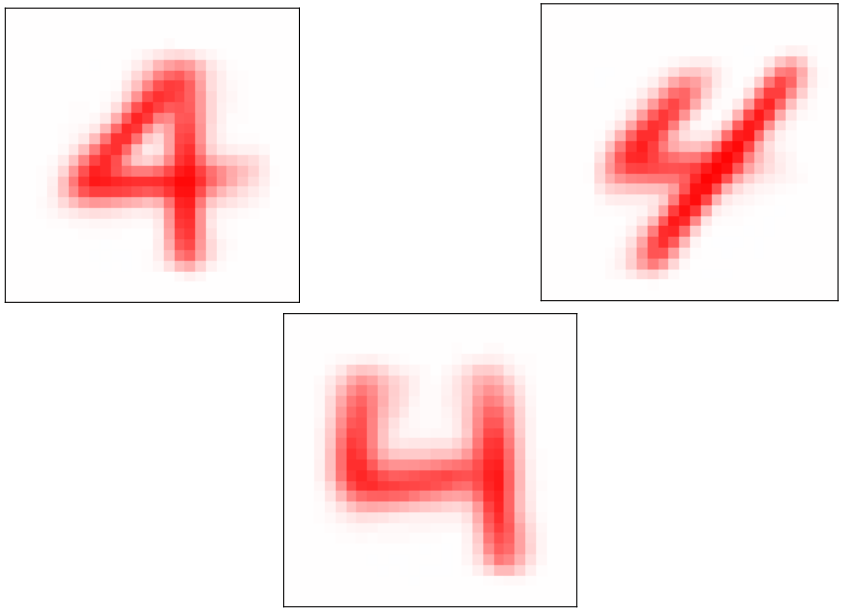
Return to MNIST: Many ways of writing “4”



Single Filter (*e.g.*, Logistic Regression/ “Shallow Learning”) only uses one filter, looks for the average shape

Return to MNIST: Many ways of writing “4”





Multiple filters can look for *subtypes* indicative of different ways of writing “4”

Why Limit Ourselves to Only One Filter?

Introduce *K* Filters

$$z_{i1} = b_{01} + x_i \odot b_1$$
$$z_{i2} = b_{02} + x_i \odot b_2$$
$$\vdots$$
$$z_{iK} = b_{0K} + x_i \odot b_K$$

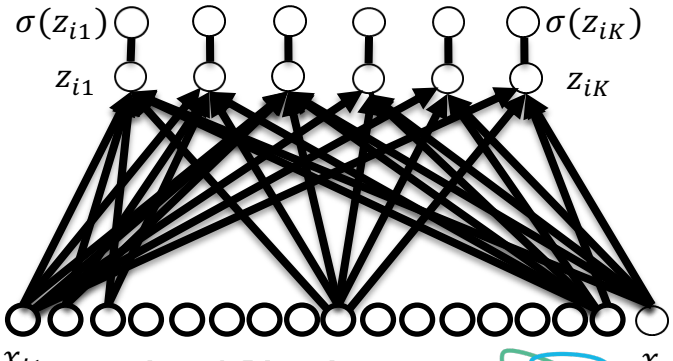
Project data x_i onto *K* filters: $b_1, \ldots, b_K$

Why Limit Ourselves to Only One Filter?

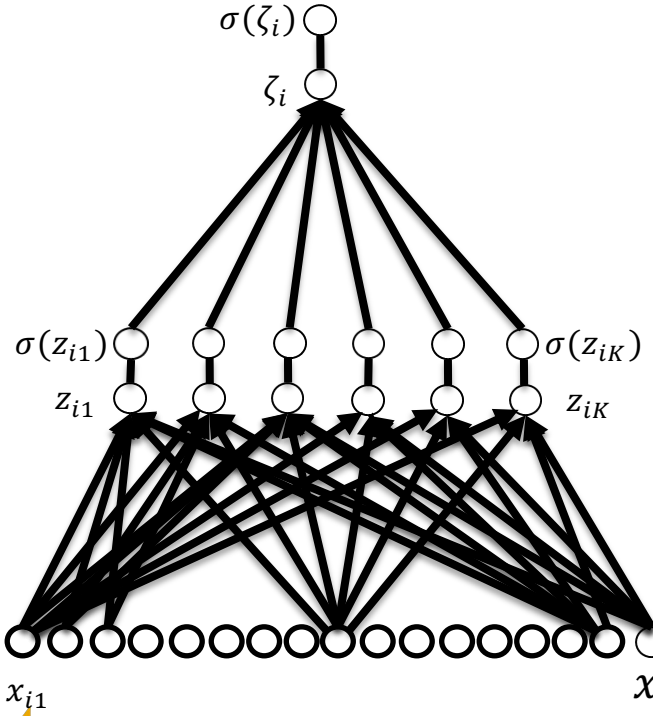
Introduce *K* Filters

$$z_{i1} = b_{01} + x_i \odot b_1$$
$$z_{i2} = b_{02} + x_i \odot b_2$$
$$\vdots$$
$$z_{iK} = b_{0K} + x_i \odot b_K$$

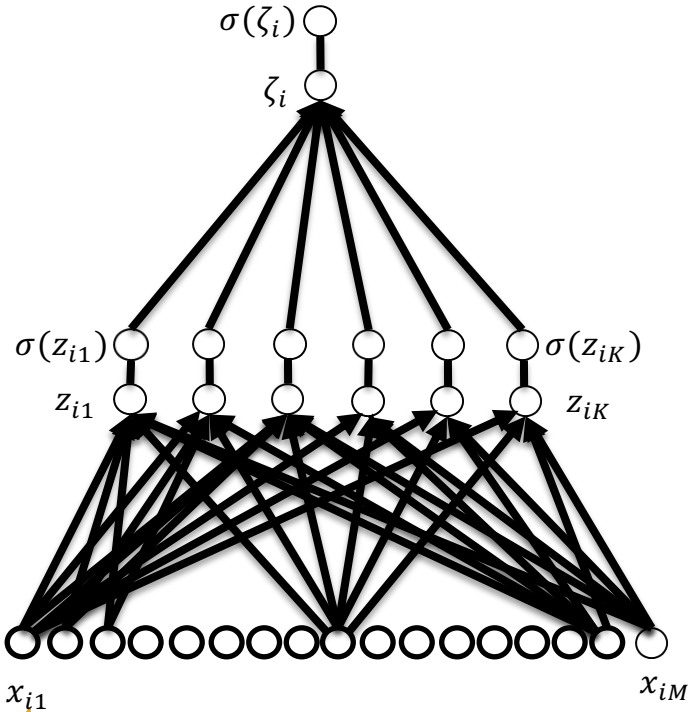
Project data x_i onto *K* filters: $b_1, \ldots, b_K$



Extended Logistic Regression



Extended Logistic Regression

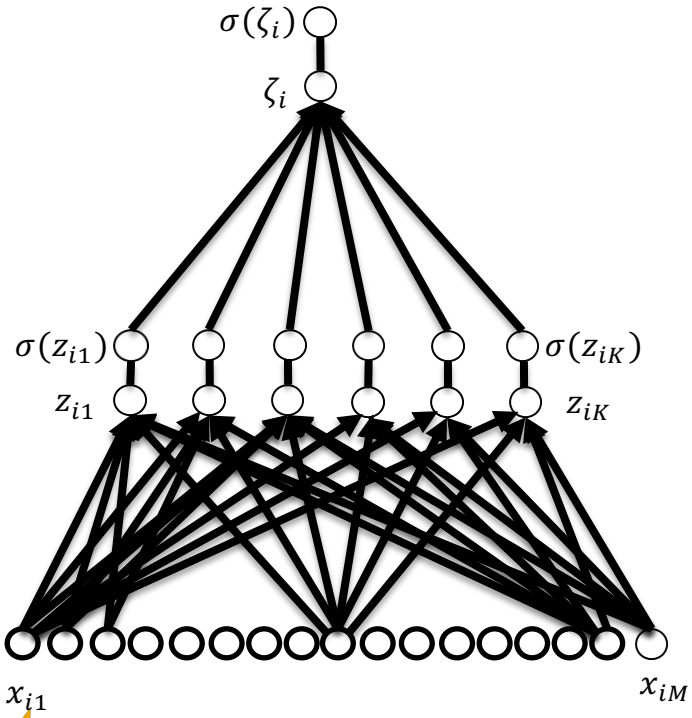


$$z_{i1} = b_{01} + x_i \odot b_1$$
$$z_{i2} = b_{02} + x_i \odot b_2$$
$$\vdots$$
$$z_{iK} = b_{0K} + x_i \odot b_K$$

K Features output from Layer-1

K Layer-1 filters

Extended Logistic Regression



Single Layer-2 Filter

$$\zeta_i = c_0 + \sigma(z_i) \odot c$$

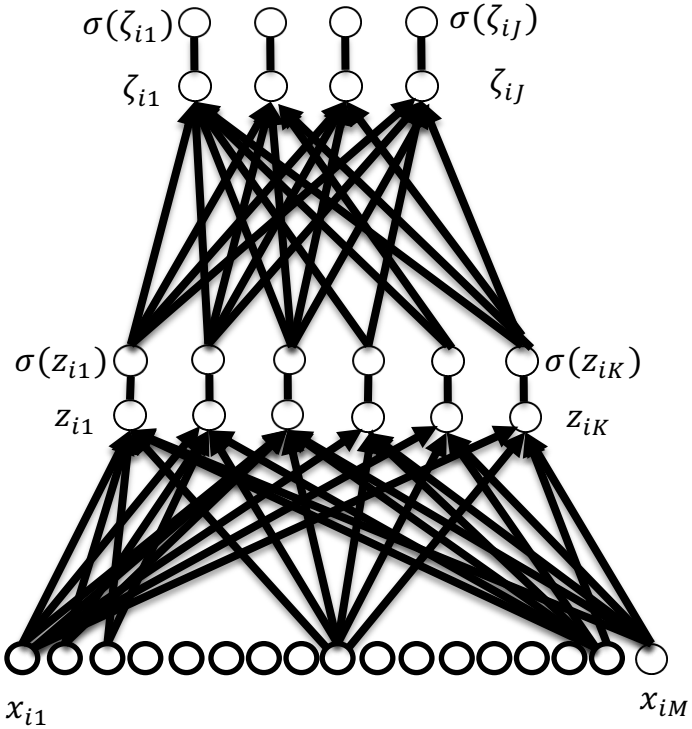
K-Dimensional Feature Vector Output from Layer-1

$$z_{i1} = b_{01} + x_i \odot b_1$$
$$z_{i2} = b_{02} + x_i \odot b_2$$
$$\vdots$$
$$z_{iK} = b_{0K} + x_i \odot b_K$$

K Features output from Layer-1

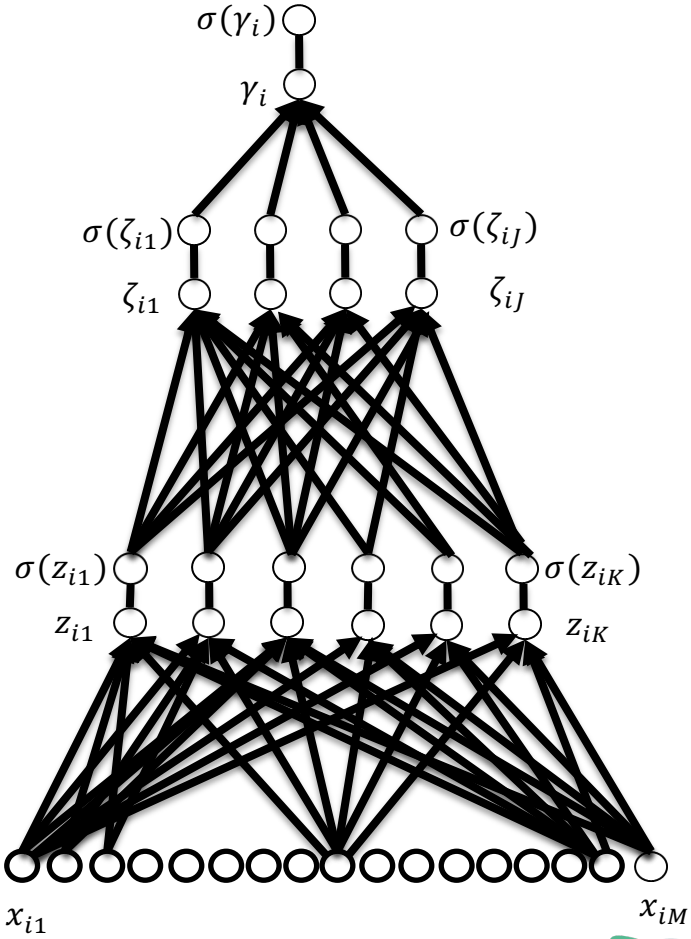
K Layer-1 filters

Going Deeper



Probability of J Layer-2 latent processes

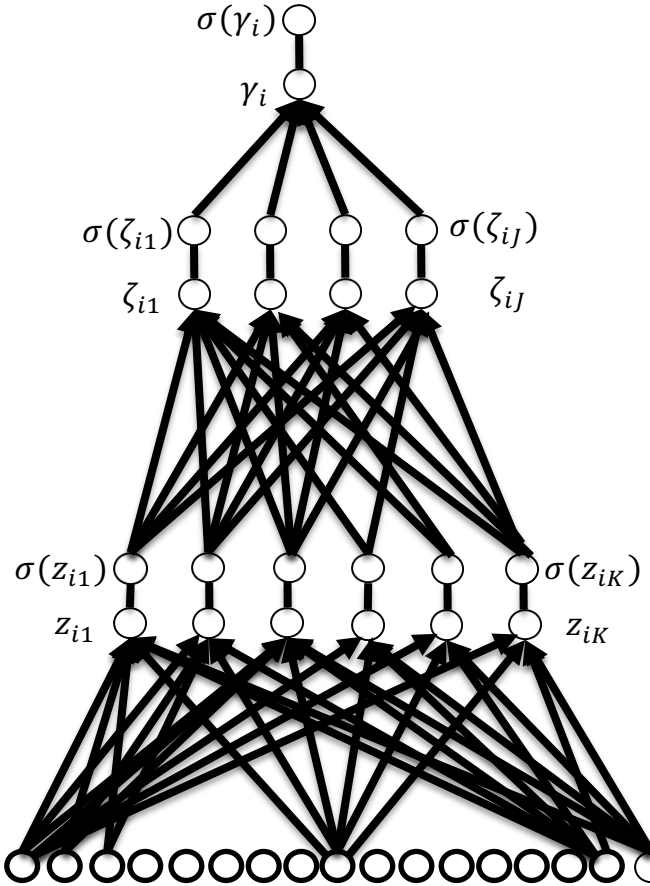
Probability of K Layer-1 latent processes



Probability of particular outcome

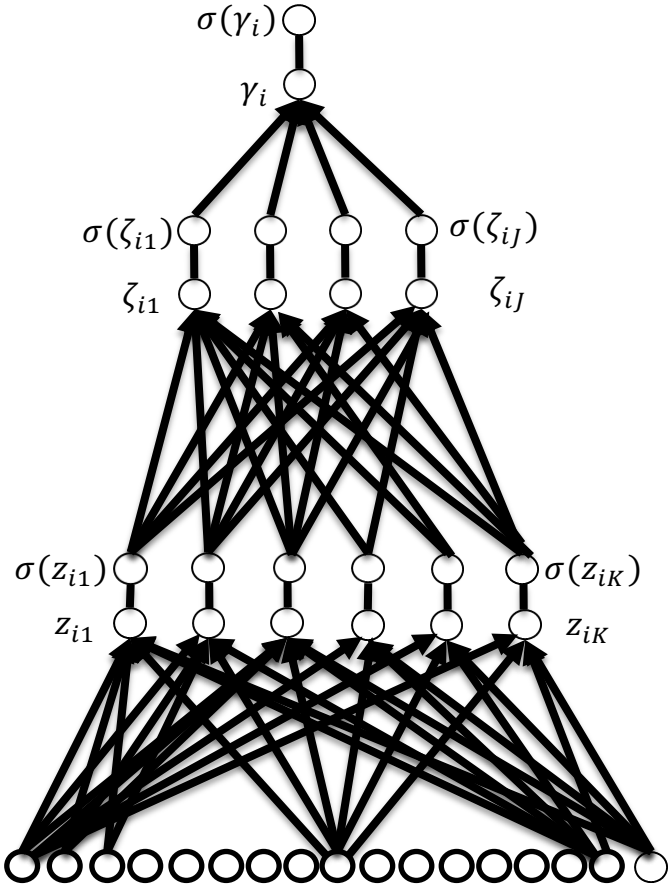
Probability of J Layer-2 latent processes

Probability of K Layer-1 latent processes



$$z_{i1} = b_{01} + x_i \odot b_1$$
$$z_{i2} = b_{02} + x_i \odot b_2$$
$$\vdots$$
$$z_{iK} = b_{0K} + x_i \odot b_K$$

Features of the Data with Layer-1 Filters

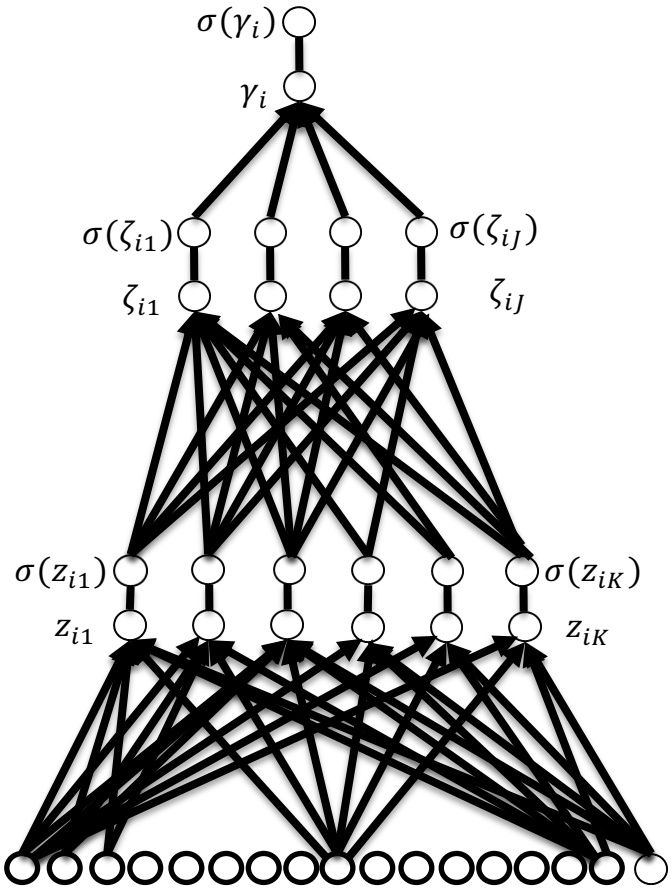


$$\zeta_{i1} = c_{01} + \sigma(z_i) \odot c_1$$
$$\zeta_{i2} = c_{02} + \sigma(z_i) \odot c_2$$
$$\vdots$$
$$\zeta_{iK} = c_{0J} + \sigma(z_i) \odot c_J$$

Features of the Layer-1 Features, with Layer-2 Filters

$$z_{i1} = b_{01} + x_i \odot b_1$$
$$z_{i2} = b_{02} + x_i \odot b_2$$
$$\vdots$$
$$z_{iK} = b_{0K} + x_i \odot b_K$$

Features of the Data with Layer-1 Filters



$$\gamma_i = d_0 + \sigma(\zeta_i) \odot d$$

Logistic Regression Based on the Layer-2 Features

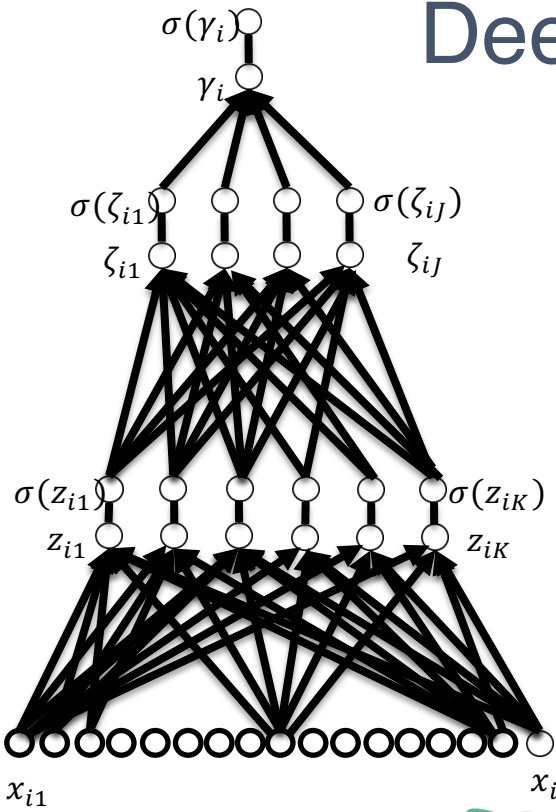
$$\begin{aligned} \zeta_{i1} &= c_{01} + \sigma(z_i) \odot c_1 \\ \zeta_{i2} &= c_{02} + \sigma(z_i) \odot c_2 \\ &\vdots \\ \zeta_{iK} &= c_{0J} + \sigma(z_i) \odot c_J \end{aligned}$$

Features of the Layer-1 Features, with Layer-2 Filters

$$\begin{aligned} z_{i1} &= b_{01} + x_i \odot b_1 \\ z_{i2} &= b_{02} + x_i \odot b_2 \\ &\vdots \\ z_{iK} &= b_{0K} + x_i \odot b_K \end{aligned}$$

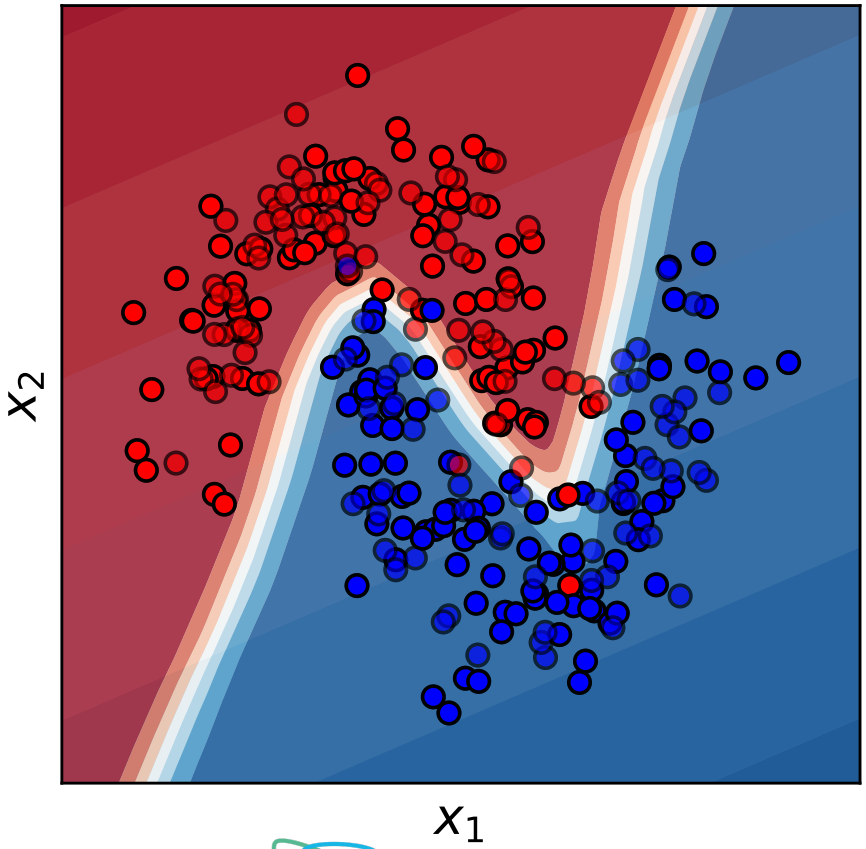
Features of the Data with Layer-1 Filters

Multilayer Perceptron: Neural Network Deep Learning

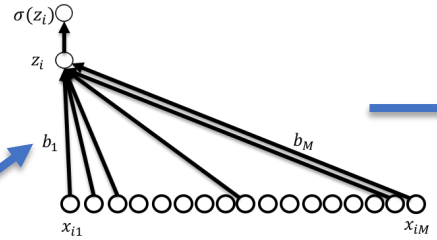


Can learn non-linear classifications

By learning multiple layers and transformation, it is possible to have a non-linear classifier capable of more accurately capturing the data.

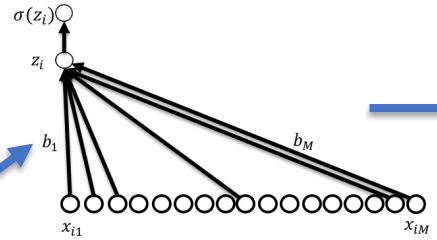


Does this work with MNIST?

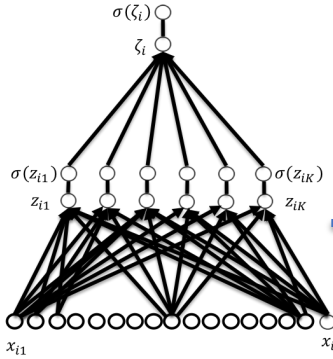


~91% Accurate

Does this work with MNIST?



~91% Accurate



~96% Accurate

An Introduction to Model Validation

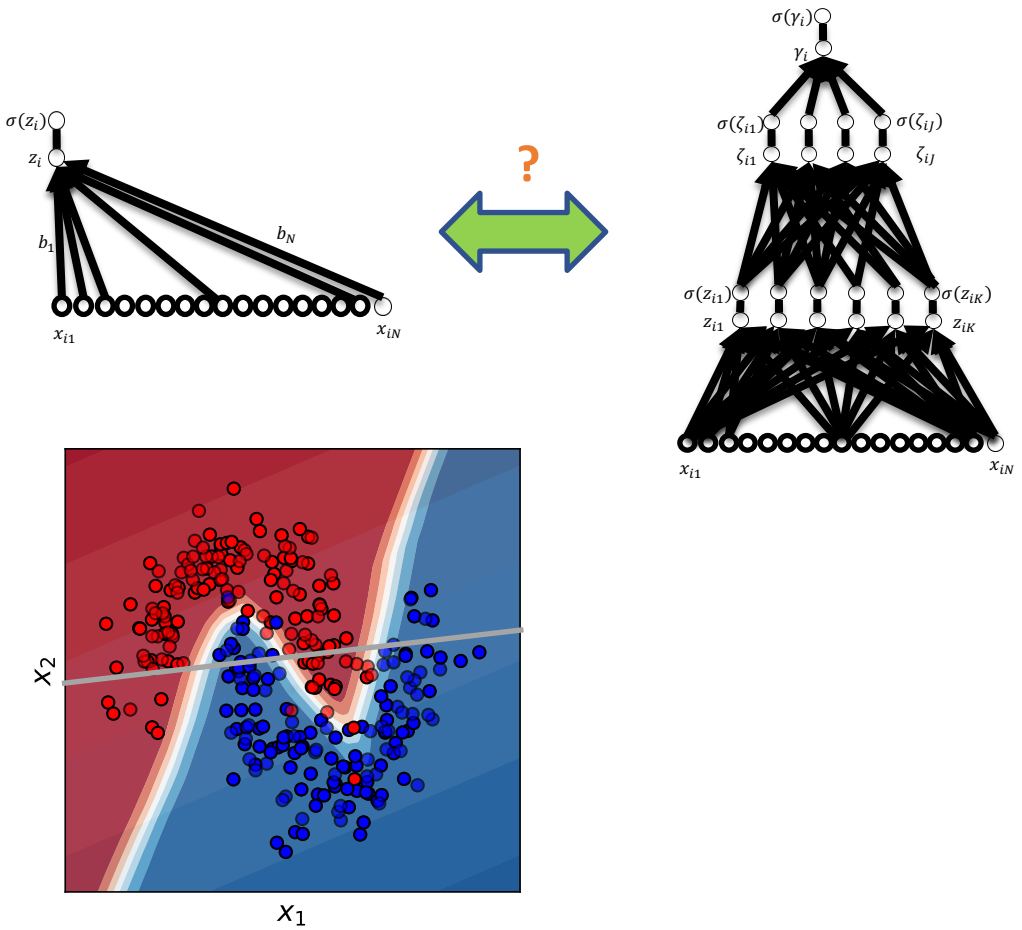
ESTIMATING PERFORMANCE

Creating deep models can help us learn complex relationships

By learning multiple layers and transformation, it is possible to have a non-linear classifier capable of more accurately capturing the data.

However, a deep model can also give “perfect” performance on the training dataset and **fail completely** in the real world

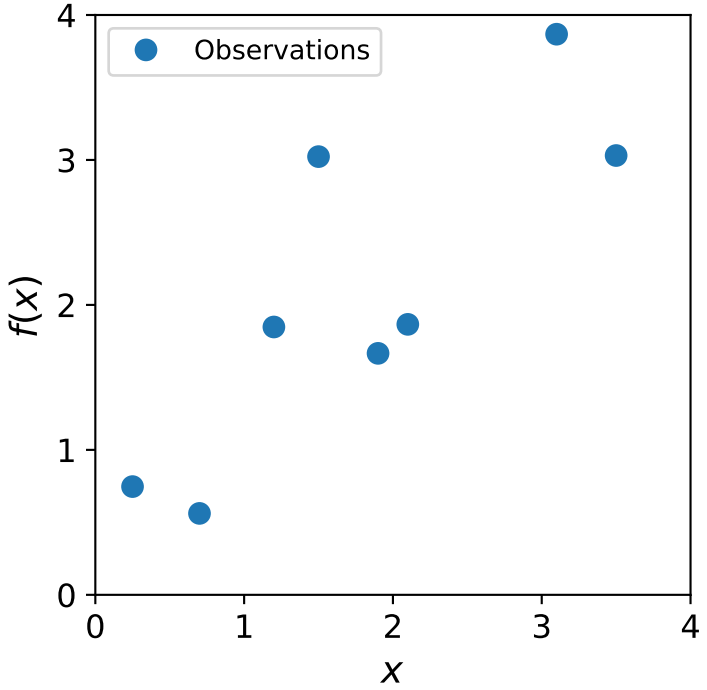
We need to *validate* the performance



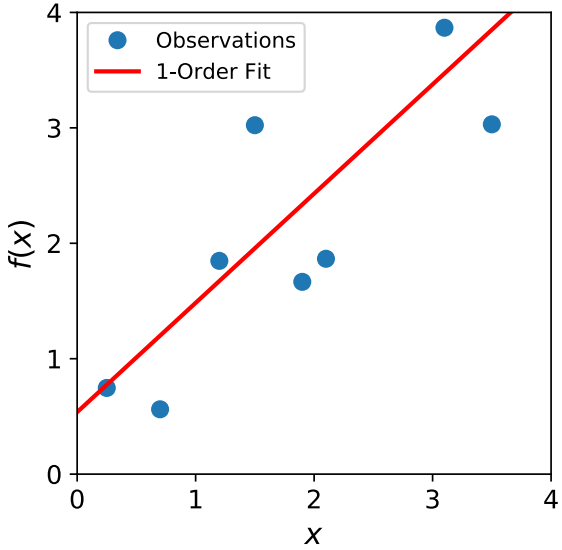
Overfitting

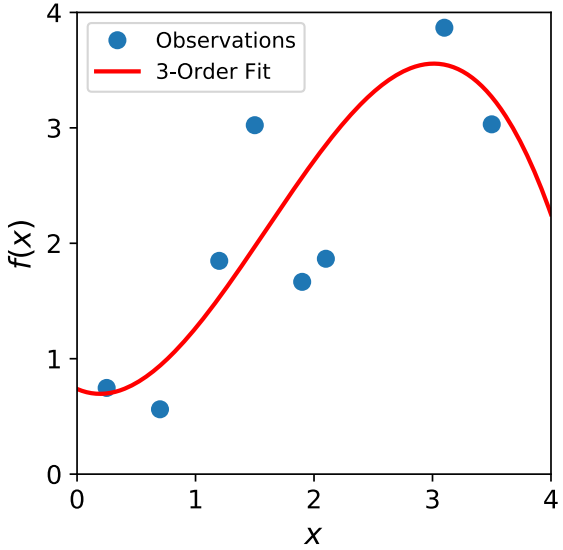
"Overfitting" happens when the learned model increases complexity to fit the observed training data *too well* – will not work to predict future data!

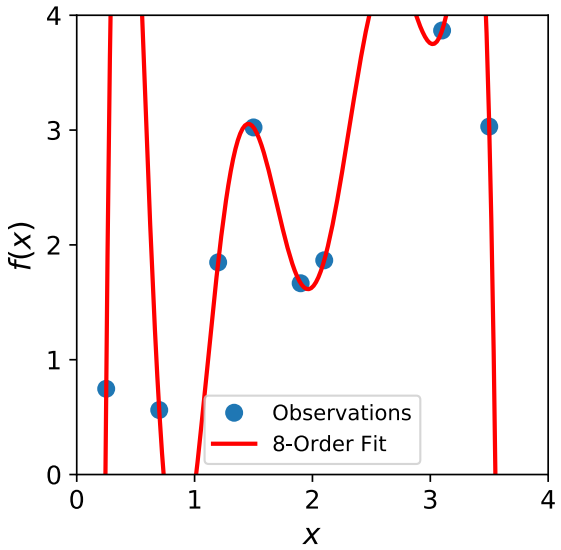
What would we want to use to fit these example data points?



Classic Example: Increasing Polynomial Order





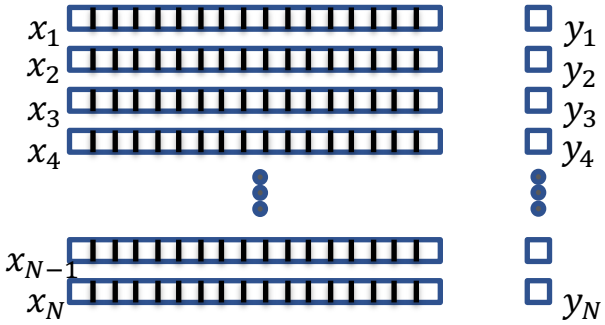


Increasing complexity does not seem appropriate…

What happens in overfitting?

- We are increasing the number of parameters in the model, which means:
  - More parameters to estimate (all their errors add up)
  - Can learn *complex* relationships, maybe too complex for reality

- When we *overfit*, this means that we will not *generalize*
  - We want our models and analysis to generalize, or provide accurate predictions on newly collected data



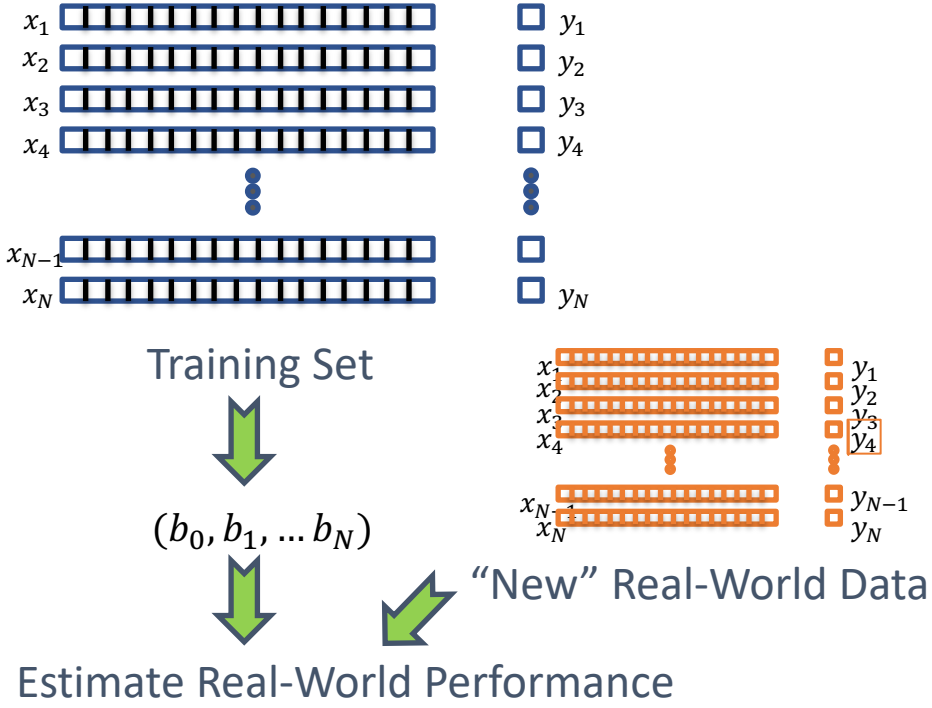
Training Set

$(b_0, b_1, \ldots b_N)$

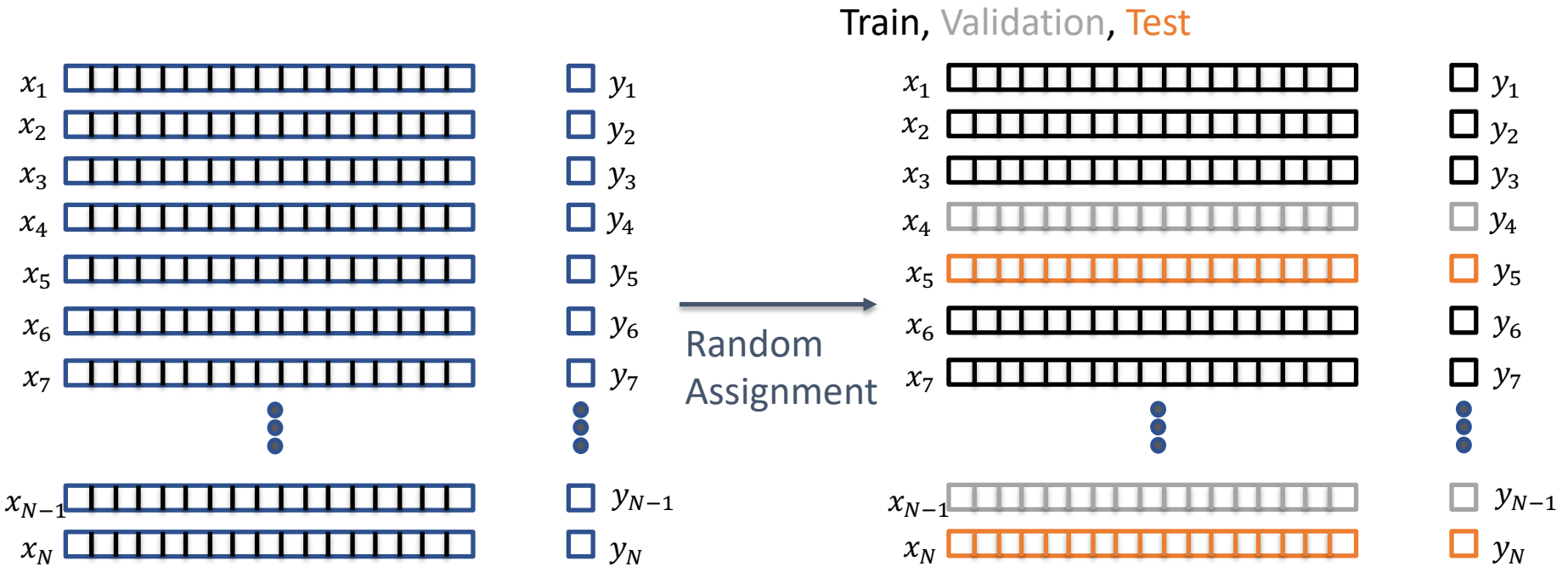
Performance in the Real World?

Standard Validation Strategy

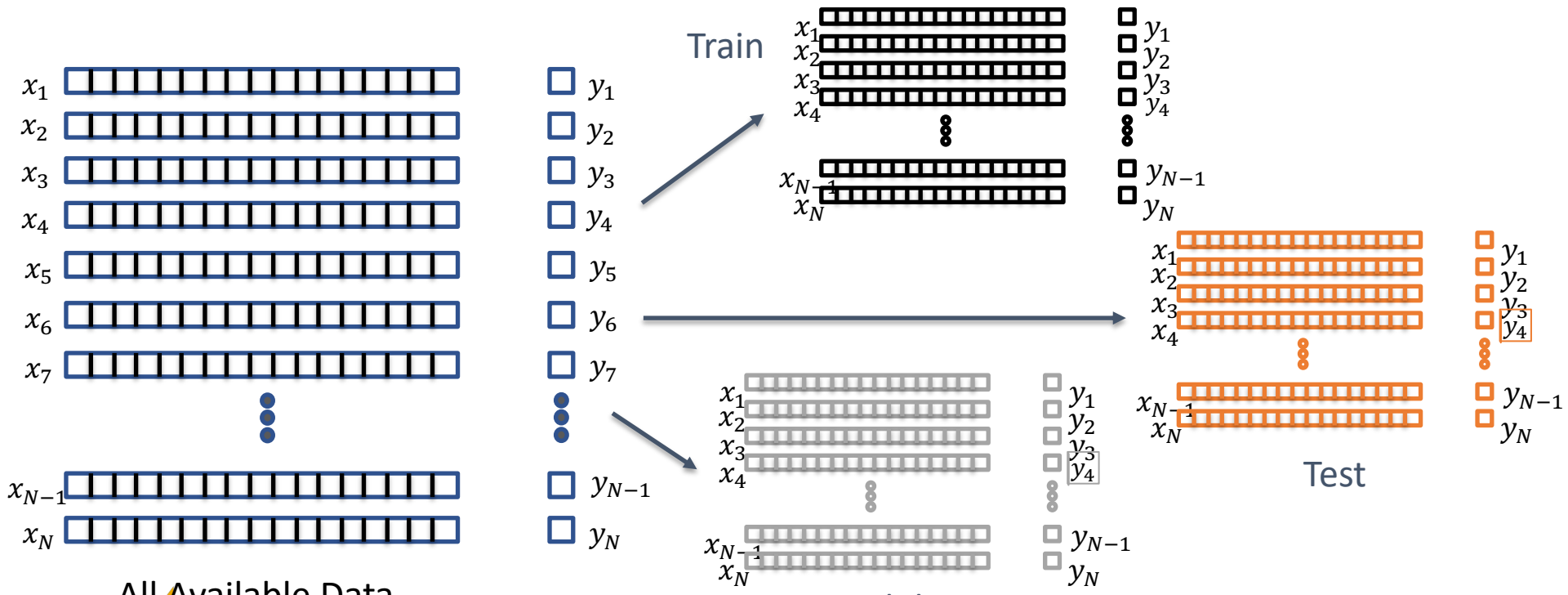
- In the end, we want to know how the network will perform *in the real world*
- Standard approach: try it in the real world
- This is costly; instead, can we use existing data to estimate performance?



Split Data into Separate Groups



Split Data into Separate Groups

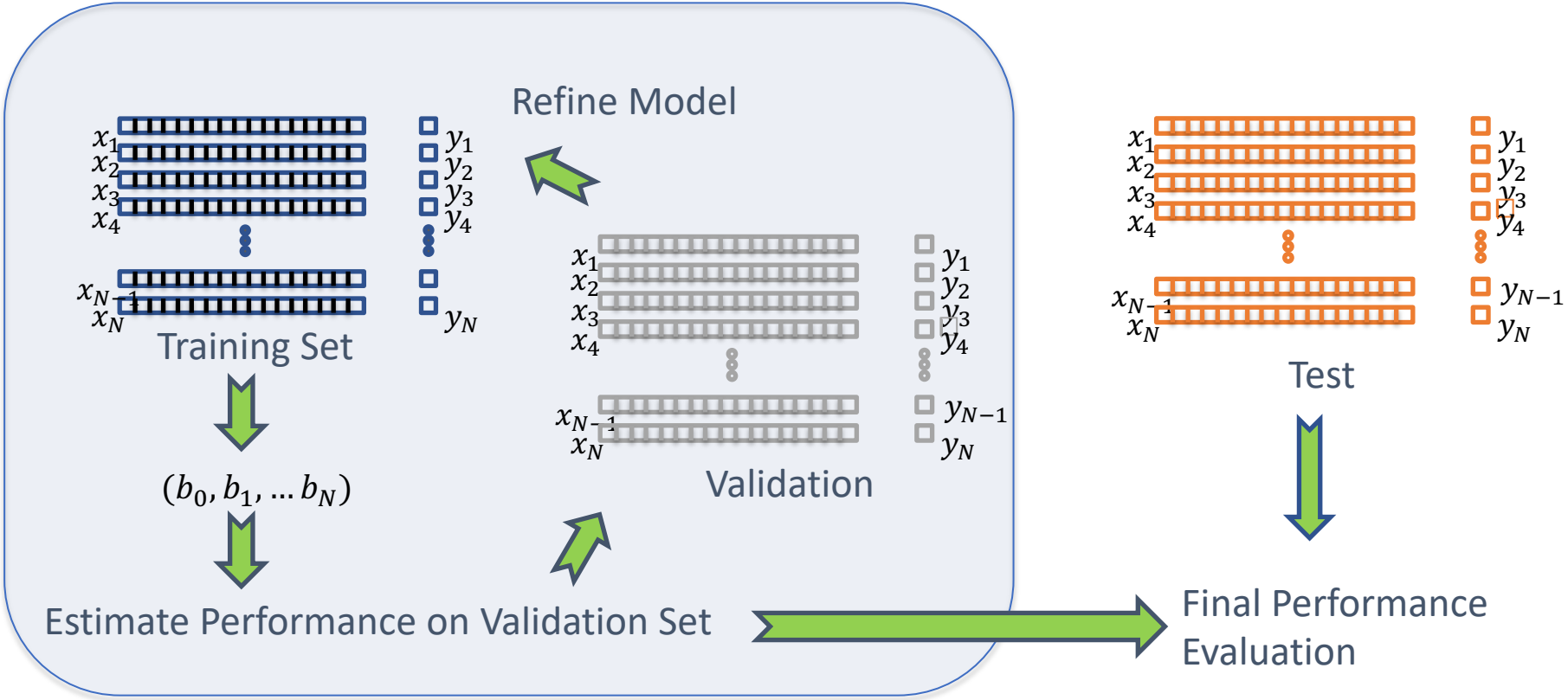


Test set

- Standard practice to create prior to any analysis—**will not be used to learn or fit any parameters**
- After learning the network, can evaluate its performance on the test set
  - This data was not included in the training/fitting, so it is analogous to running a new synthetic experiment
- Ideally, the test set will be used *once*.
  - Reusing the test set leads to bias; performance estimates will be optimistic

Validation Set

- Want to be able to compare which approach is best
  - Problematic if we only want to use a test set once
  - Can create a second held-out dataset
- The validation data is not used for learning parameters, but can be used repeatedly to estimate performance of a model
- We can pick the model with the best performance on the validation set, and run a final evaluation on the test data

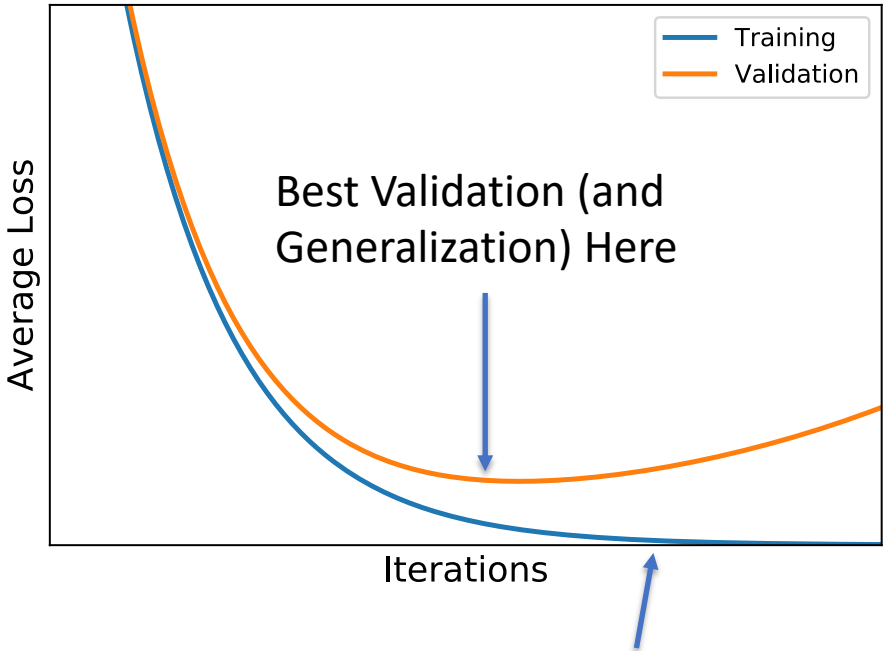

Refine Model
x_1 x_2 x_3 x_4 x_{N-1} x_N
y_1 y_2 y_3 y_4 y_N
Training Set
$(b_0, b_1, \ldots b_N)$
x_1 x_2 x_3 x_4 x_{N-1} x_N
y_1 y_2 y_3 y_4 y_{N-1} y_N
Validation
Estimate Performance on Validation Set
x_1 x_2 x_3 x_4 x_{N-1} x_N
y_1 y_2 y_3 y_4 y_{N-1} y_N
Test
Final Performance Evaluation

Validation During Optimization

- We want to maximize the *generalization* of the network, not our previous optimization goal
- In practice, can we validate the model while running the optimization loop

Early Stopping

- During optimization, we can check the validation loss as we go.
- Instead of optimizing to convergence, we can optimize until the *validation* loss stops improving
  - Saves computational cost
  - Performs better on validation (and test) sets
- Widely used technique in the field



Conclusions

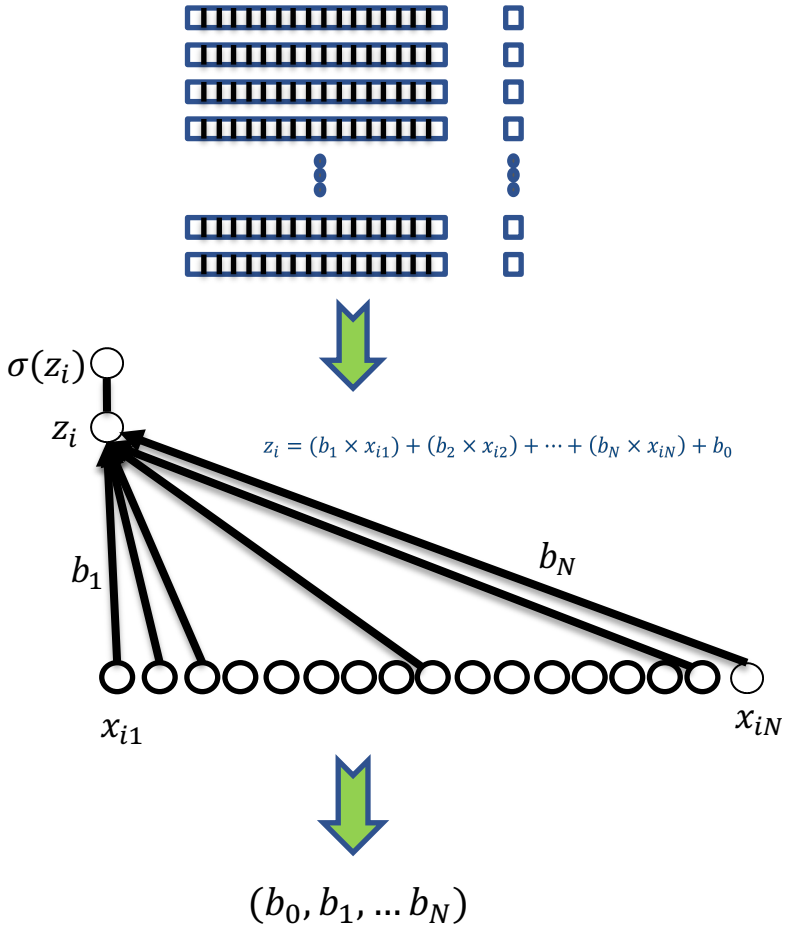
- Learn the parameters of the model (train) from data
- Complex models can be built from simple modules
- Complex models can significantly outperform simple models and even humans
- Model complexity is not always necessary
- Proper model validation is critical to estimate real-world performance

The loss function

LEARNING MODEL PARAMETERS

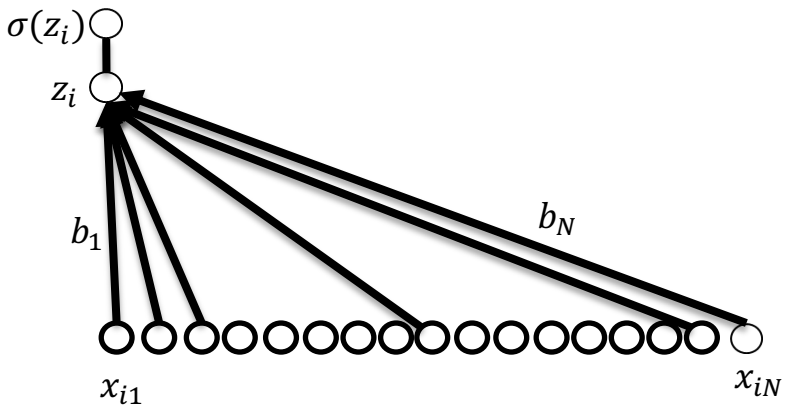
Fundamental Question:

- Given a large amount of data and a model/network to fit, how to *efficiently* and *effectively* learn the model parameters?



What are we trying to do?

- Want to learn parameters that give us the best performance
- Essentially: given data, find the "best" $\boldsymbol{b}$ for that data
- How do we define performance?



Empirical Risk Minimization

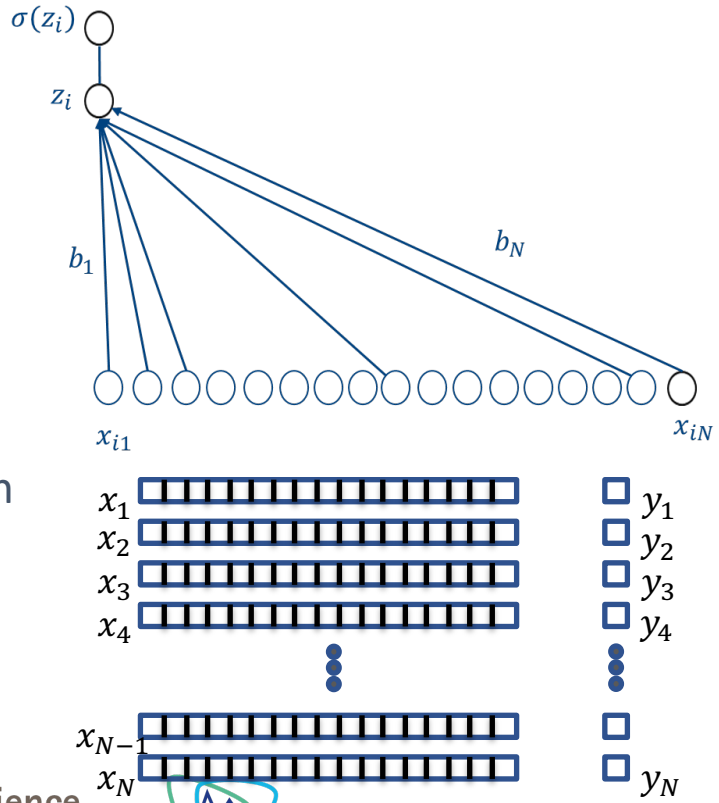
- A *loss function* $\ell(true, prediction)$ defines a penalty for poor predictions
- Want to minimize average loss
- Mathematically, this can be stated as

$$\boldsymbol{b}^* = \arg\min_{\boldsymbol{b}} \frac{1}{N} \sum_{i}^{N} \ell(y_i, \sigma(z_i))$$

Loss function

Guess

True Label



What is the Loss Function?

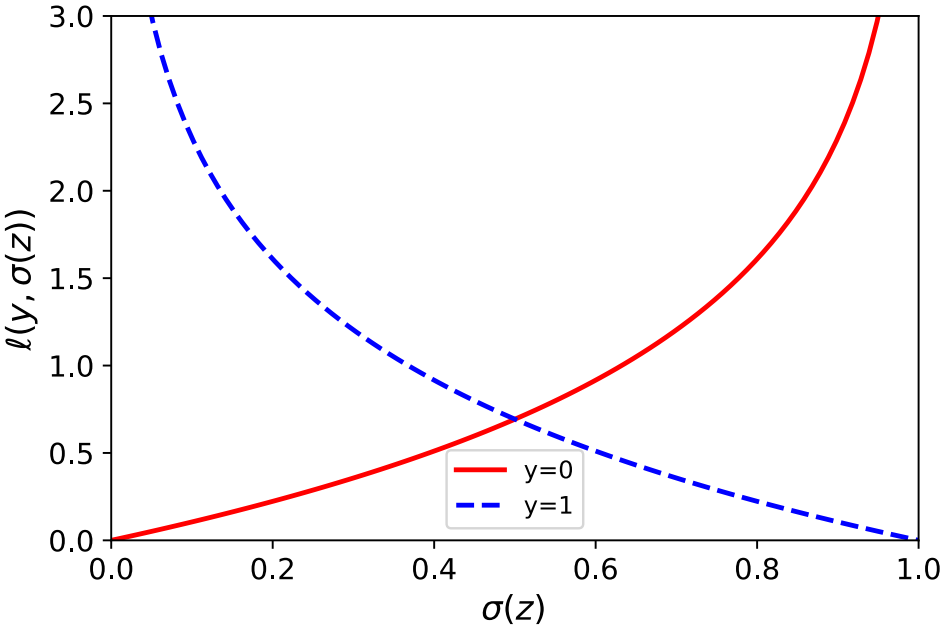
- Define $\sigma(z_i)$ as the predicted probability
- y_i is our true label
- Can be viewed as the negative log-likelihood
  $\ell\big(y_i, \sigma(z_i)\big) = -\log p(y_i|\sigma(z_i))$
- Specific mathematical form is:
  $\ell\big(y, \sigma(z)\big) = -y \log \sigma(z) - (1-y) \log\big(1-\sigma(z)\big)$

Visualization of Logistics Loss Function

- The logistic/cross-entropy loss is:

$$\ell\big(y, \sigma(z)\big) = -y \log \sigma(z) - (1-y) \log\big(1 - \sigma(z)\big)$$

- Dashed blue line shows loss when the true prediction is positive
- When we give 100% of a 1, we pay no penalty
- If we are less confident or wrong, we pay an increasing penalty



Predicted Probability of a "one"

Optimization Goal for Binary Classification

- The optimization goal is to minimize the average loss

$$\boldsymbol{b}^* = \arg\min_{\boldsymbol{b}} \frac{1}{N} \sum_{i}^{N} \ell(y_i, \sigma(z_i))$$

- For binary (0/1) problems, the logistic or cross-entropy loss is

$$\ell\big(y, \sigma(z)\big) = -y \log \sigma(z) - (1-y) \log\big(1 - \sigma(z)\big)$$