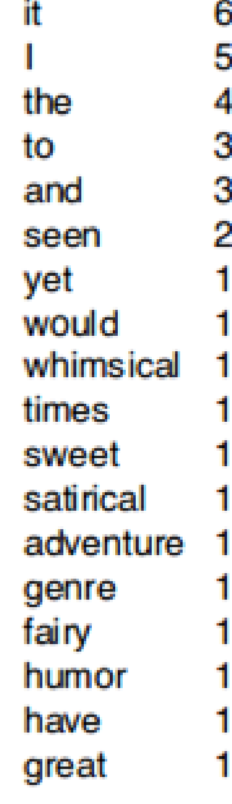
Outline

- HW0 Due
- Lecture on Text classification 2
  - How can we build a sentiment classifier?
  - State of the Art
- Tutorial on building text classifier using PyTorch (Zae)
- Next week:
  - Tutorial on fine-tuning (Karin)
  - Finetuning text classifier using HuggingFace (Karin)
  - HW1 out

Bag of words

Representation of text only as the counts of words that it contains

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

f (

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

) = y

```
[ ] vect_tunned = CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=0.1, max_df=0.7, max_features=100)
```

```
[ ] count_vectorizer = feature_extraction.text.CountVectorizer()

    ## let's get counts for the first 5 tweets in the data
    example_train_vectors = count_vectorizer.fit_transform(x_train[0:5])
```

```
[ ] ## we use .todense() here because these vectors are "sparse" (only non-zero elements are kept to save space)
    print(example_train_vectors[0].todense().shape)
    print(example_train_vectors[0].todense())
```

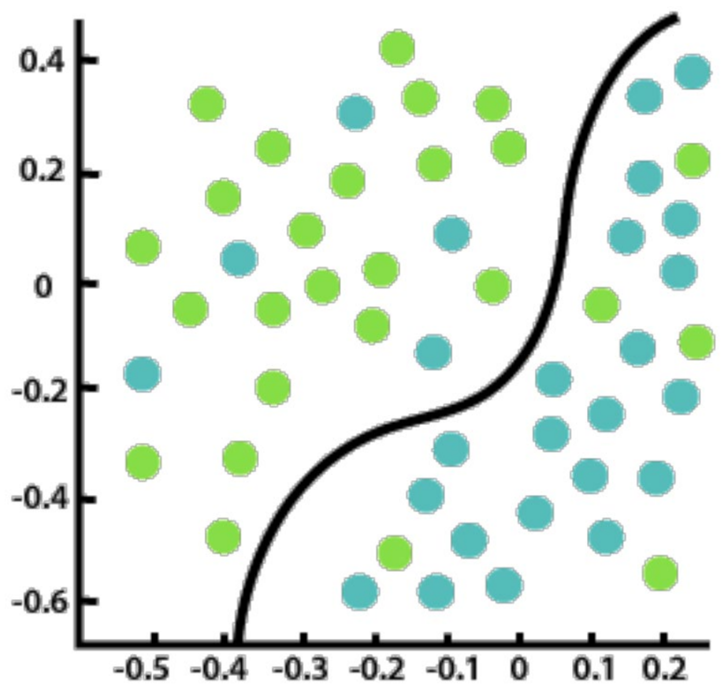
```
(1, 46)
[[1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 2 0 1 2 0 0 1 0
  0 1 0 0 0 0 0 0 2 0]]
```

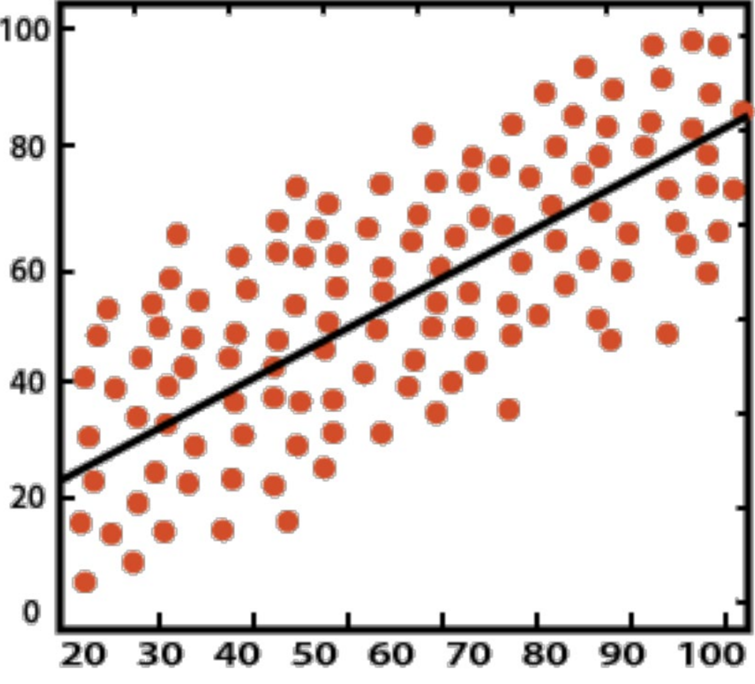
Learning f(x)

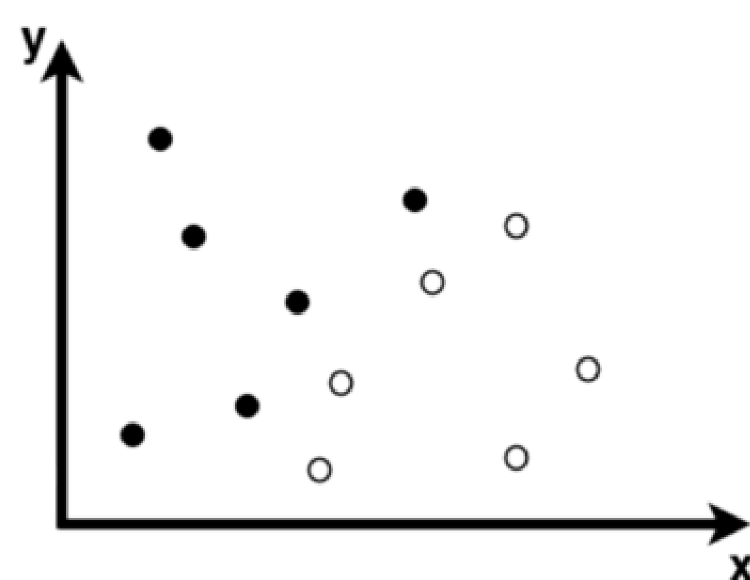
Two components:

- The formal structure of the **learning method**:
  - How x and y are mapped
  - Logistic regression, Naïve Bayes, RNN, CNN, etc
- The **representation** of the data (x)

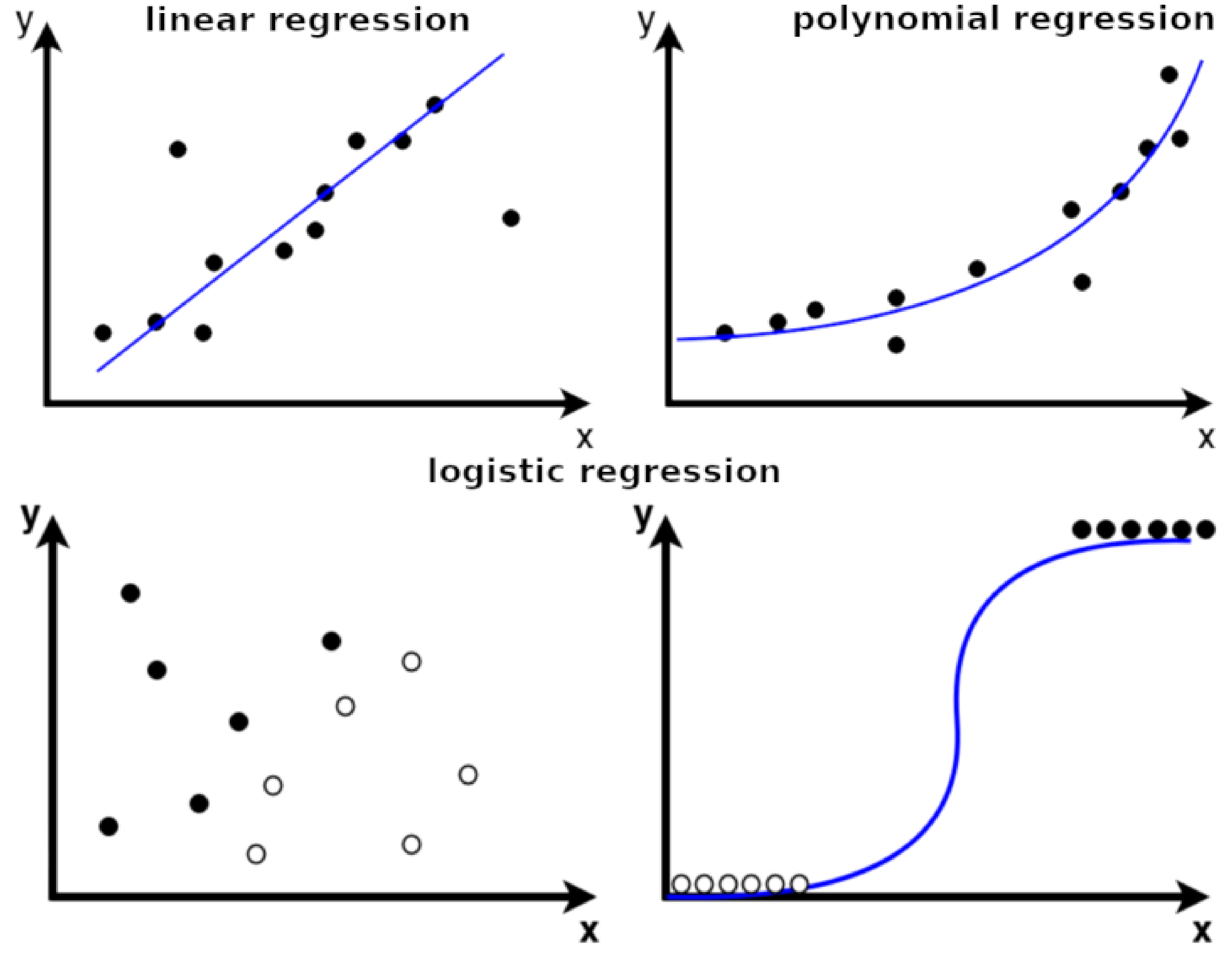
Classification vs Regression





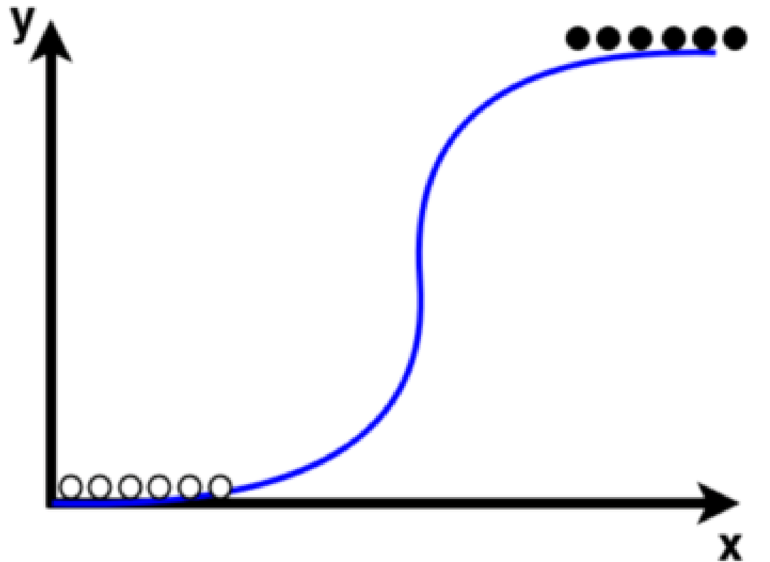


y
x

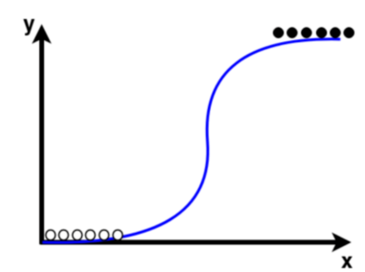

linear regression
y
x
polynomial regression
y
x
logistic regression
y
x
y
x

Logistic regression

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$



Binary logistic regression

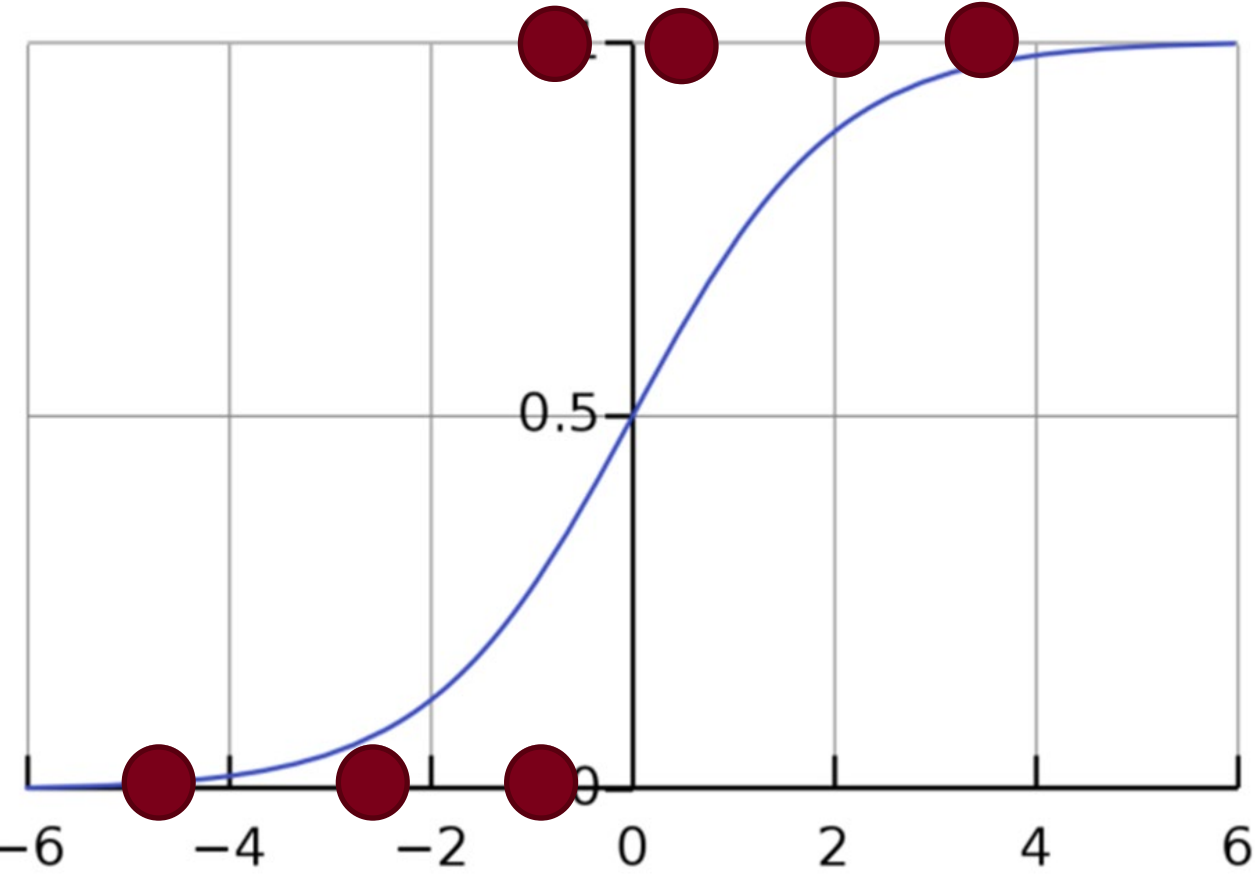


Model parameters to learn

$$P(y = 1 \mid x, \beta) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

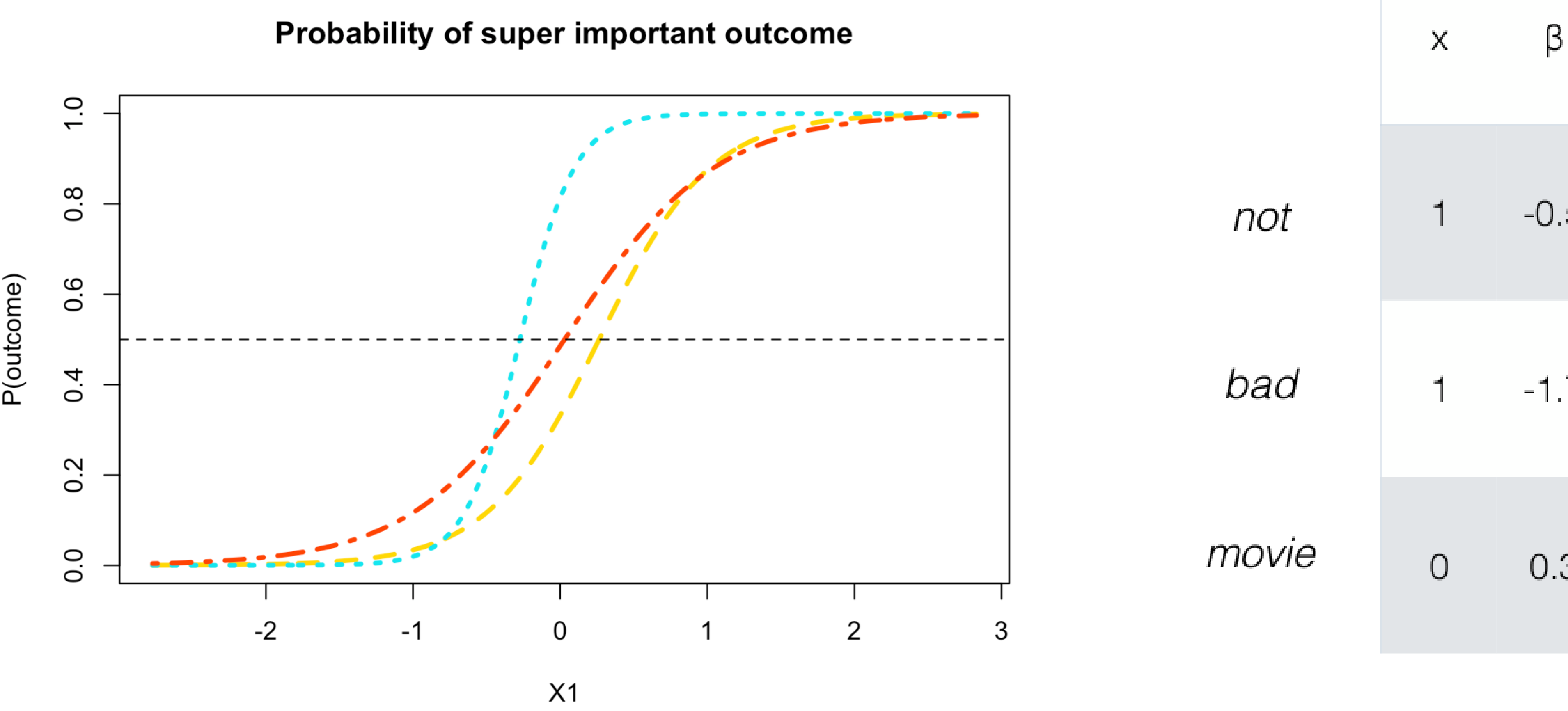
output space $\mathcal{Y} = \{0, 1\}$

Binary logistic regression



f (x) = y

Importance of your features: β



	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

Logistic regression

$$P(y=1 \mid x, \beta) = \frac{1}{1+\exp\left(-\sum_{i=1}^{F} x_i\beta_i\right)}$$

- We want to find the value of β that leads to the highest value of the conditional log likelihood:

$$\ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta)$$

$$\begin{aligned}
\nabla_\beta l(\beta; y, X) &= \nabla_\beta \left(\sum_{i=1}^{N} [-\ln(1+\exp(x_i\beta)) + y_i x_i \beta] \right) \\
&= \sum_{i=1}^{N} (\nabla_\beta [-\ln(1+\exp(x_i\beta)) + y_i x_i \beta]) \\
&= \sum_{i=1}^{N} \left(-\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)} x_i + y_i x_i \right) \\
&= \sum_{i=1}^{N} \left(y_i - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)} \right) x_i \\
&= \sum_{i=1}^{N} \left(y_i - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)} \frac{\exp(-x_i\beta)}{\exp(-x_i\beta)} \right) x_i \\
&= \sum_{i=1}^{N} \left(y_i - \frac{1}{1+\exp(-x_i\beta)} \right) x_i \\
&= \sum_{i=1}^{N} [y_i - S(x_i\beta)] x_i
\end{aligned}$$

Logistic regression

- We want to find the value of β that leads to the highest value of the conditional log likelihood:

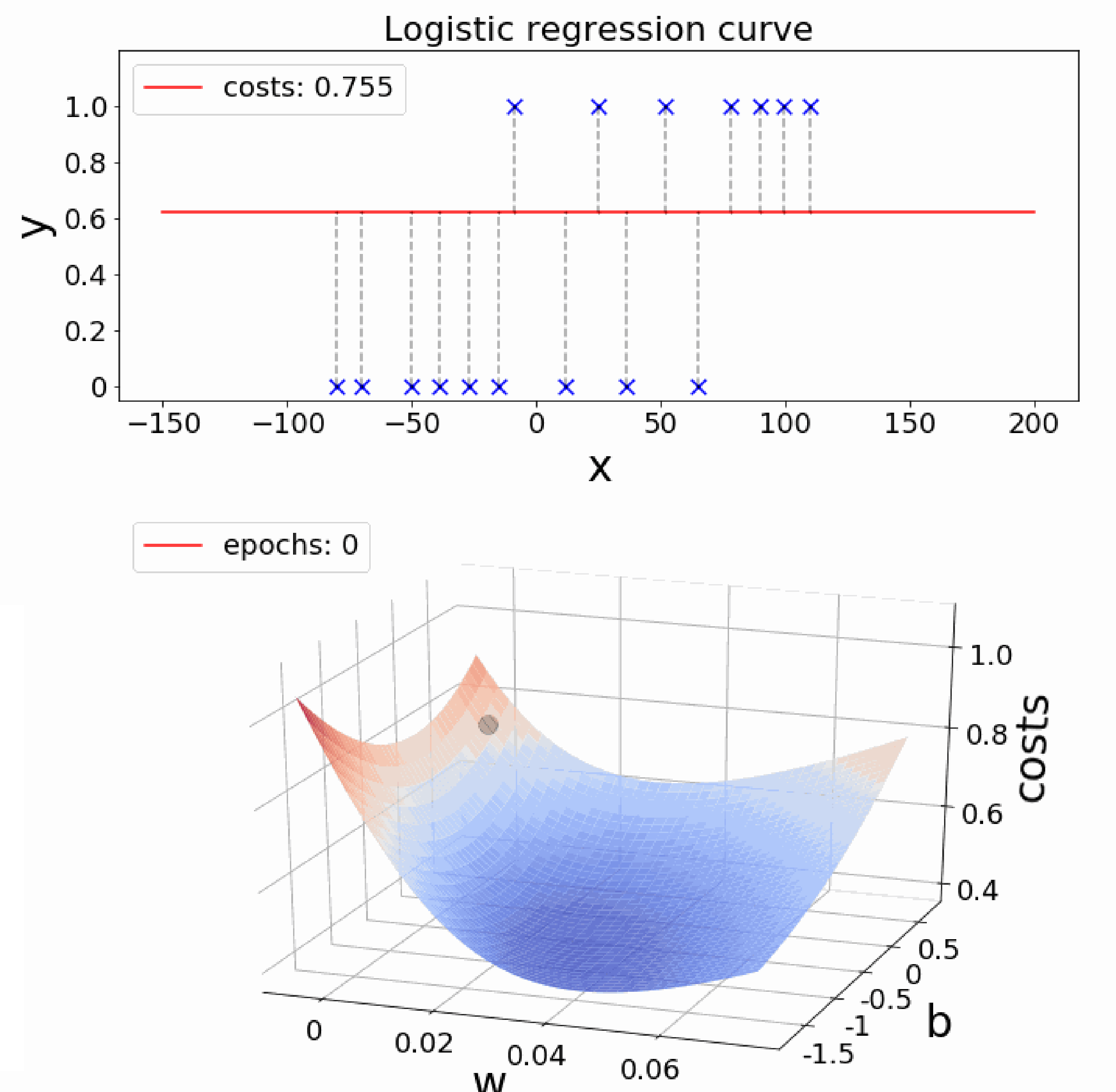
$$\ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta)$$

- Train it with stochastic gradient descent

Algorithm 2 Logistic regression stochastic gradient descent

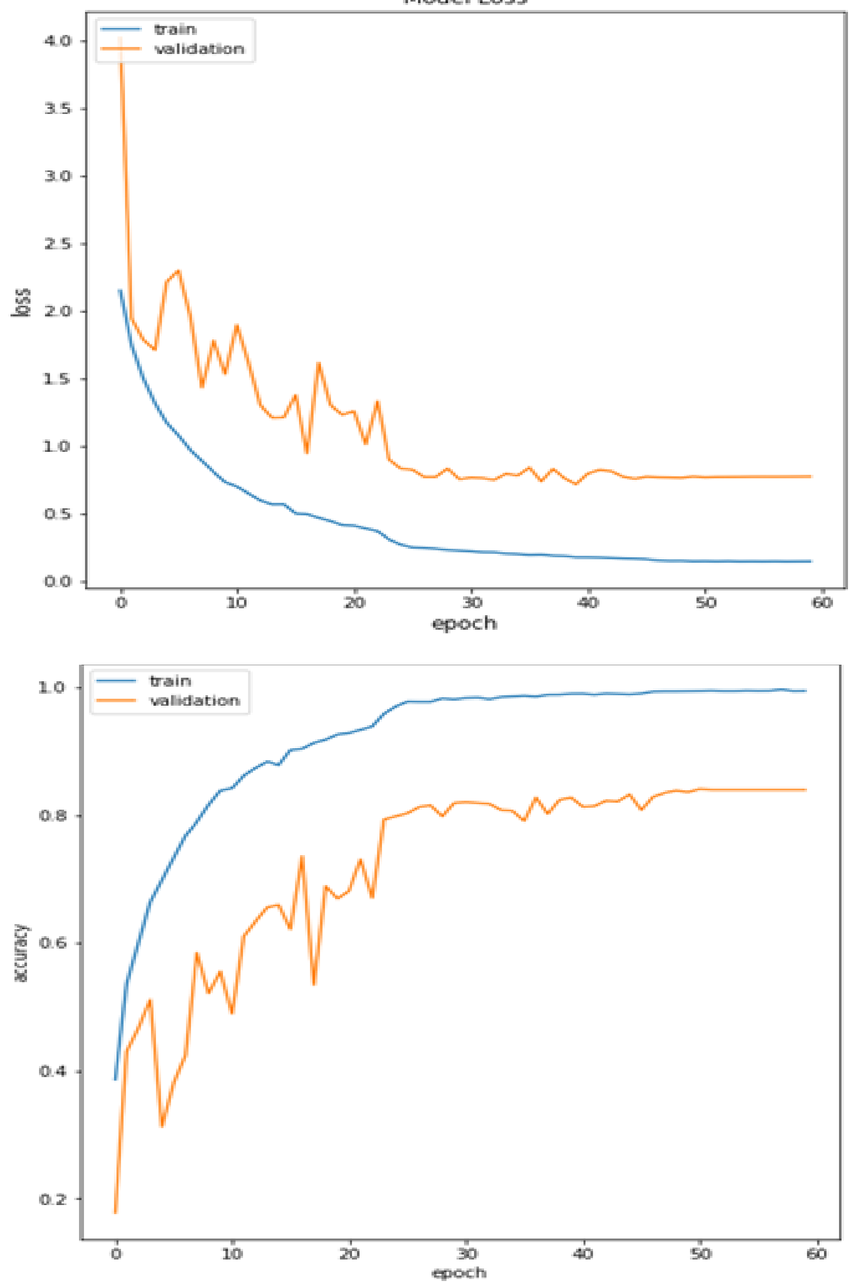
1: Data: training data $x \in \mathbb{R}^F, y \in \{0, 1\}$
2: $\beta = 0^F$
3: **while** not converged **do**
4: **for** $i = 1$ to N **do**
5: $\beta_{t+1} = \beta_t + \alpha\,(y_i - \hat{p}(x_i))\,x_i$
6: **end for**
7: **end while**

$$\begin{aligned}
\nabla_\beta l(\beta; y, X) &= \nabla_\beta\left(\sum_{i=1}^{N}[-\ln(1+\exp(x_i\beta)) + y_i x_i \beta]\right) \\
&= \sum_{i=1}^{N}(\nabla_\beta[-\ln(1+\exp(x_i\beta)) + y_i x_i \beta]) \\
&= \sum_{i=1}^{N}\left(-\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}x_i + y_i x_i\right) \\
&= \sum_{i=1}^{N}\left(y_i - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\right)x_i \\
&= \sum_{i=1}^{N}\left(y_i - \frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}\,\frac{\exp(-x_i\beta)}{\exp(-x_i\beta)}\right)x_i \\
&= \sum_{i=1}^{N}\left(y_i - \frac{1}{1+\exp(-x_i\beta)}\right)x_i \\
&= \sum_{i=1}^{N}[y_i - S(x_i\beta)]x_i
\end{aligned}$$



Algorithm 2 Logistic regression stochastic gradient descent

1: Data: training data $x \in \mathbb{R}^F, y \in \{0, 1\}$
2: $\beta = 0^F$
3: **while** not converged **do**
4: **for** $i = 1$ to N **do**
5: $\beta_{t+1} = \beta_t + \alpha\,(y_i - \hat{p}(x_i))\,x_i$
6: **end for**
7: **end while**



Logistic regression curve

costs: 0.755

y

x

epochs: 0

costs

w

b

Representation of data (x)

- As a discriminative classifier, logistic regression doesn't assume features are independent like Naive Bayes does.
- Its power partly comes in the ability to create richly expressive features without the burden of independence.
- We can represent text through features that are not just the identities of individual words, but any feature that is scoped over the entirety of the input.

Features
Unigrams ("like")
Bigrams ("not like"), trigrams, etc
Prefixes (word that start with "un-"
Words that appear in the positive/negative dictionary
Reviews begin with "I love"
At least 3 mentions of positive verbs (like, love, etc)

Representation of data (x)

f(

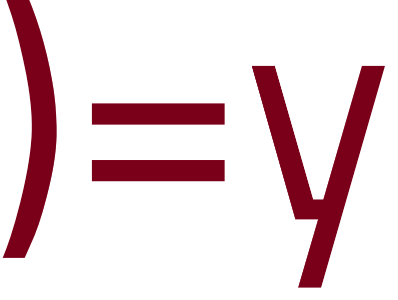
it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

I want	1
have you	2
how many	2
isn't it	1
...	

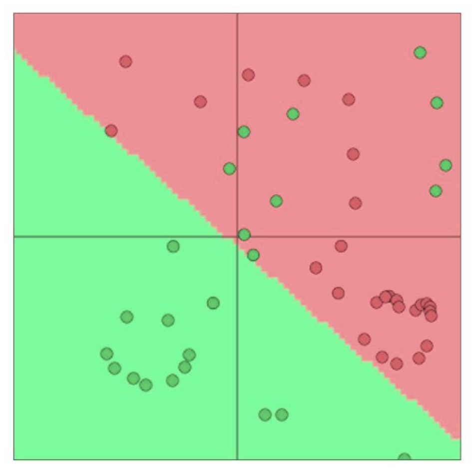
happy	1
love	2
recommend	2

lonely	0
outrage	0
not	2

NP	5
VP	2
Parse depth	5



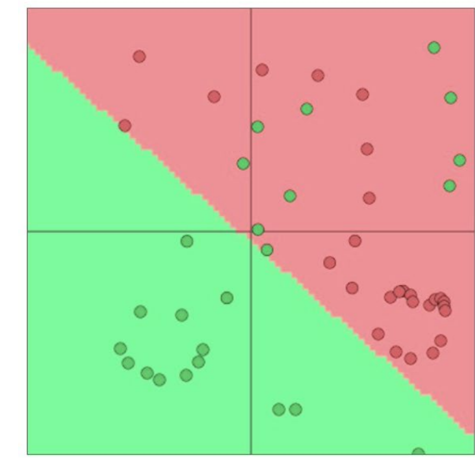
Features
Unigrams ("like")
Bigrams ("not like"), trigrams, etc
Prefixes (word that start with "un-"
Words that appear in the positive/negative dictionary
Reviews begin with "I love"
At least 3 mentions of positive verbs (like, love, etc)

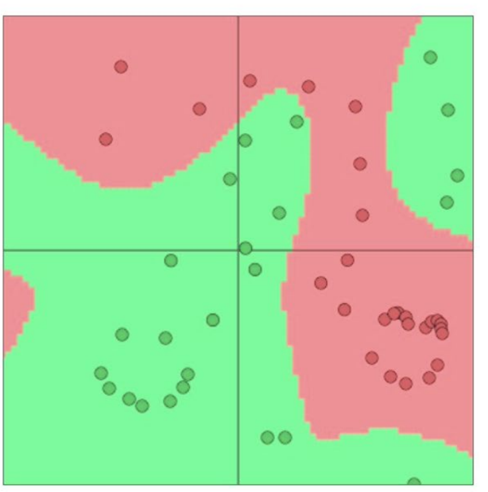


What if your input representation is *complex* and cannot be modeled by simple *linear projection*?

Neural Networks

- Discrete, high-dimensional representation of inputs (one-hot vectors) => low-dimensional "distributed" representations.
  - Distributional semantics and word vectors (To be covered)
- Static representations -> contextual representations, where representations of words are sensitive to local context.
  - Contextualized Word Embeddings (To be covered)
- Multiple layers to capture hierarchical structure

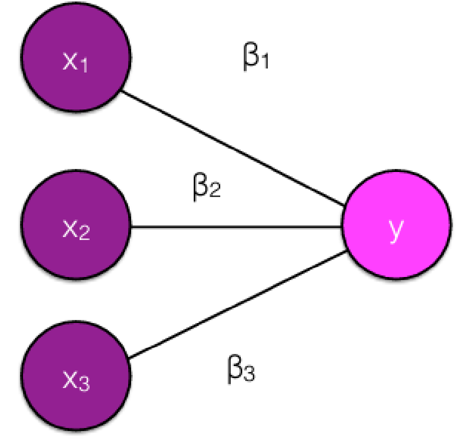




Recap: Logistic regression

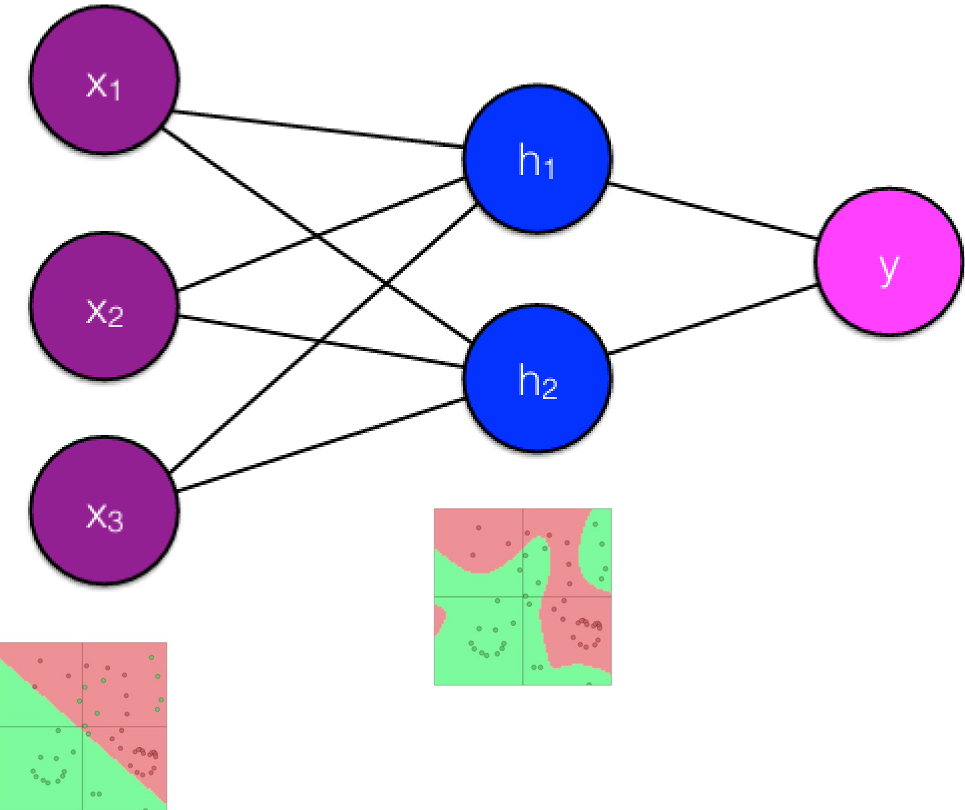
$$P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

	x	β
not	1	-0.5
bad	1	-1.7
movie	0	0.3

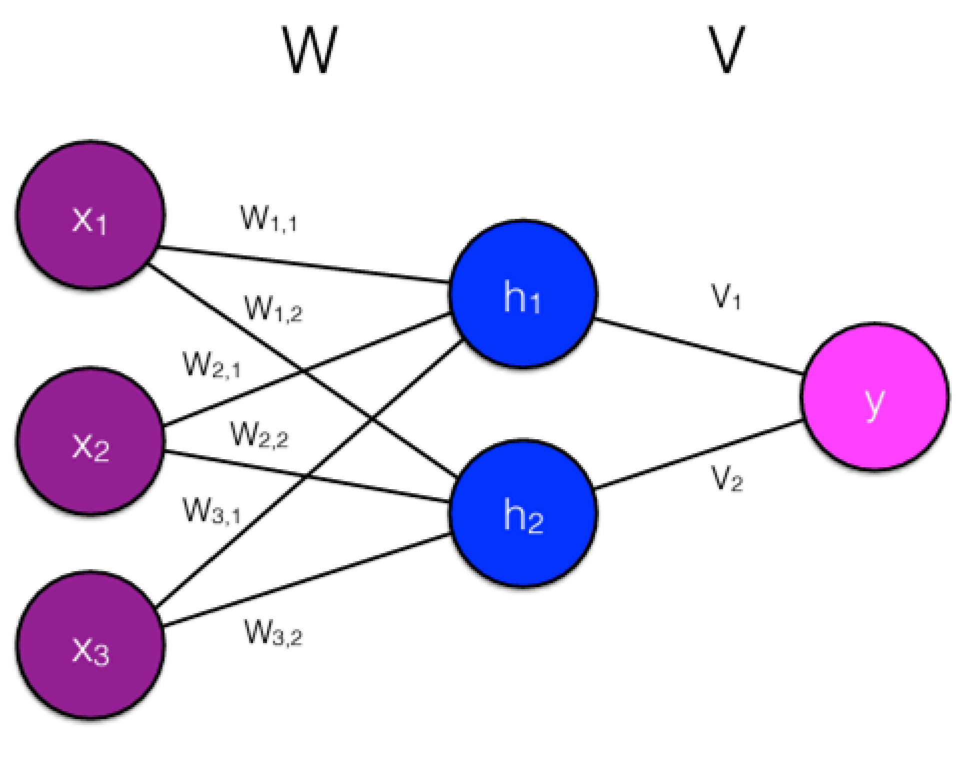


Feedforward neural network

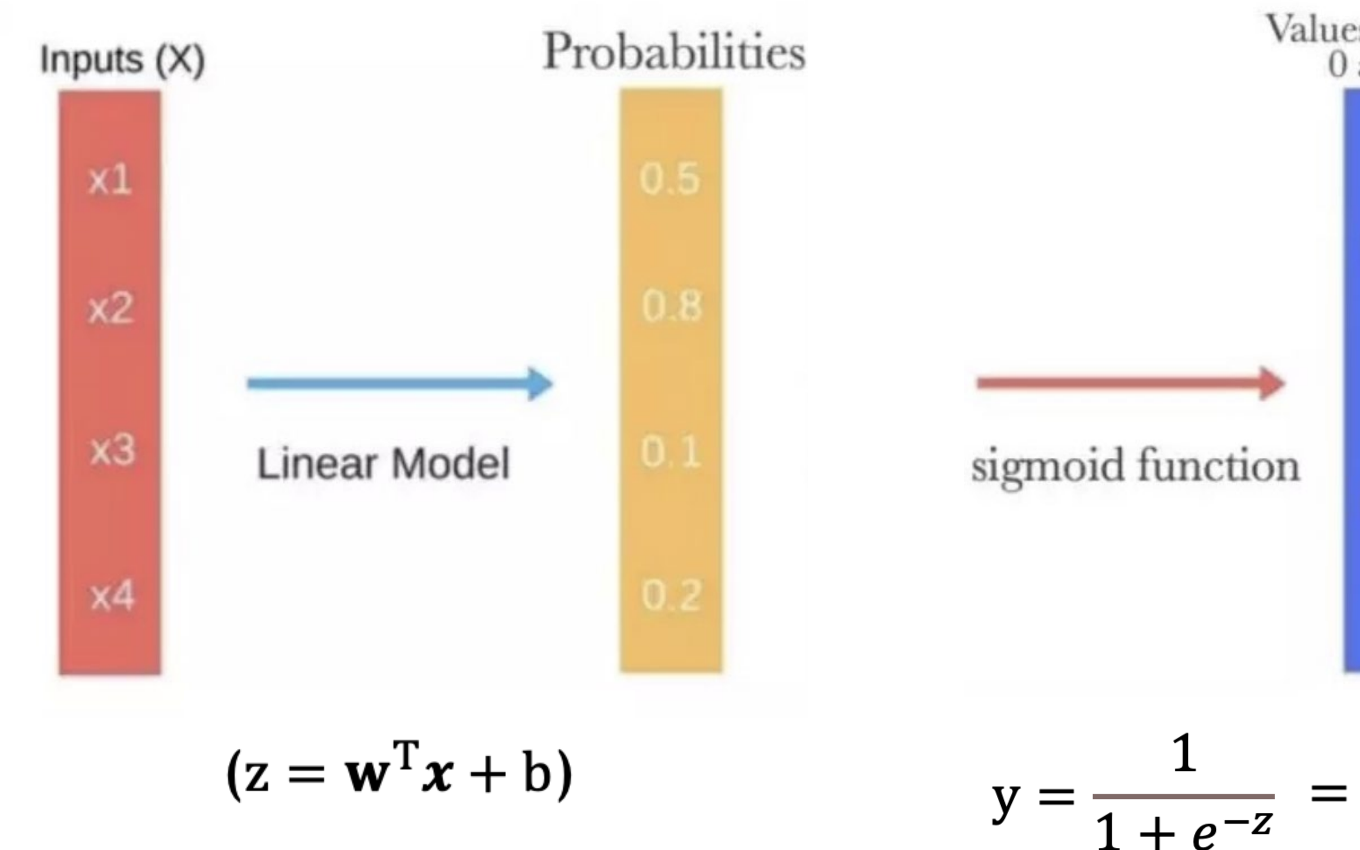
- Input and output are mediated by at least one hidden layer.



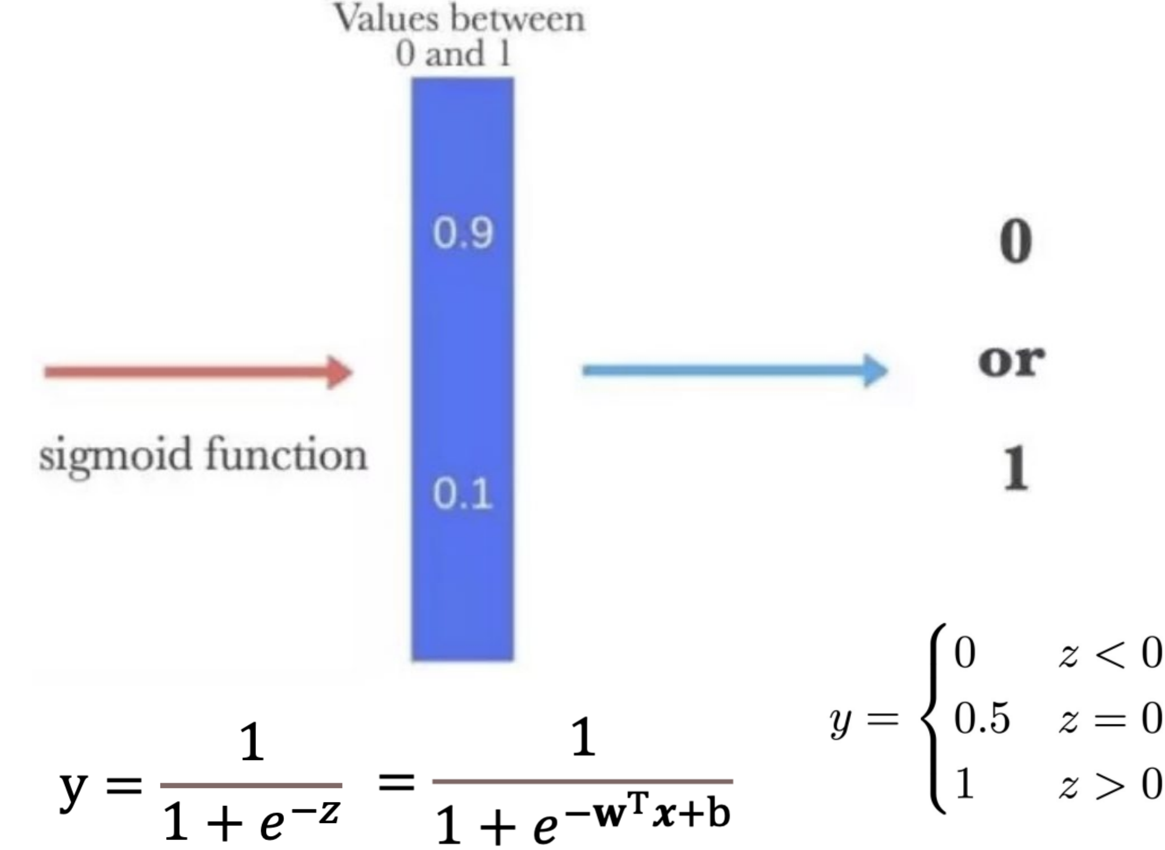
*For simplicity, we're leaving out the bias term, but assume most layers have them as well.



Relations with logistic regression

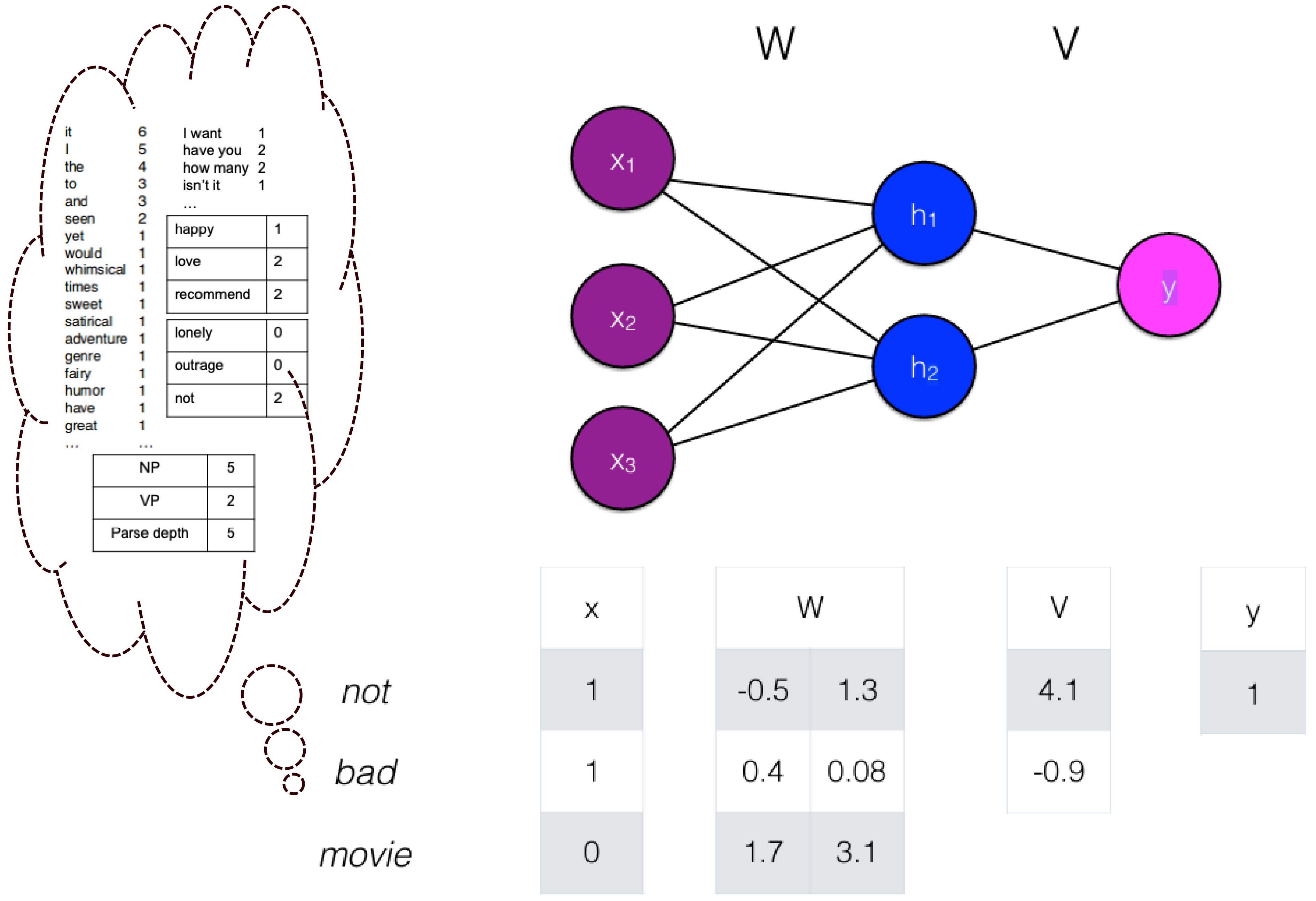


$$(z = \mathbf{w}^T \boldsymbol{x} + b)$$

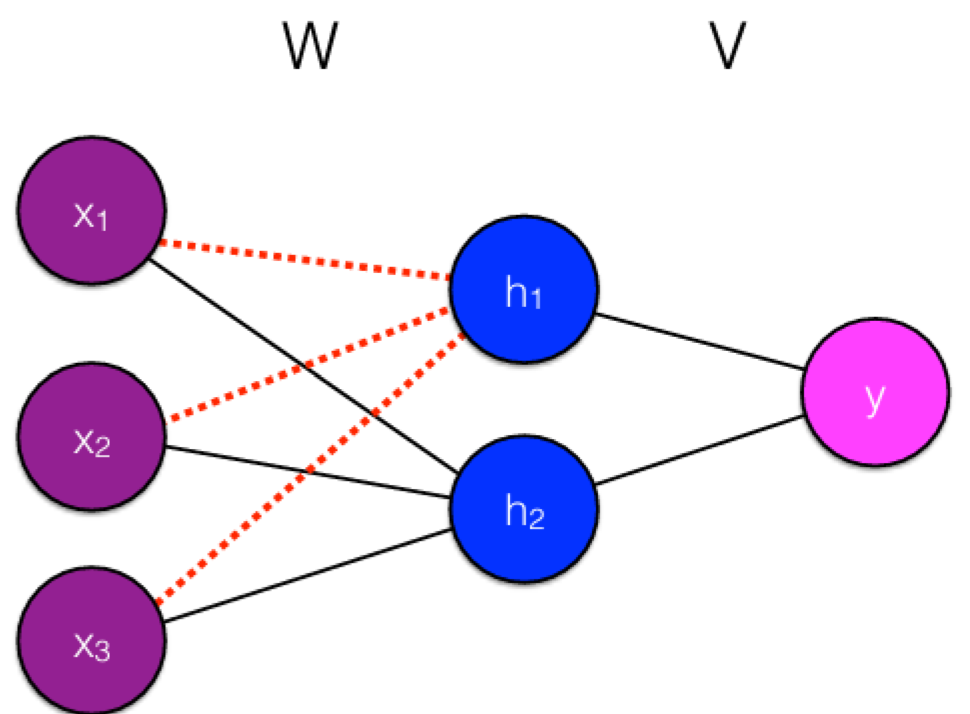


$$y = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\mathbf{w}^T \boldsymbol{x} + b}}$$

$$y = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$$



W
V
x1
x2
x3
h1
h2
y
it 6
I 5
the 4
to 3
and 3
seen 2
yet 1
would 1
whimsical 1
times 1
sweet 1
satirical 1
adventure 1
genre 1
fairy 1
humor 1
have 1
great 1
... ...
I want 1
have you 2
how many 2
isn't it 1
...
happy 1
love 2
recommend 2
lonely 0
outrage 0
not 2
NP 5
VP 2
Parse depth 5
not
bad
movie
x: 1, 1, 0
W: -0.5 1.3; 0.4 0.08; 1.7 3.1
V: 4.1, -0.9
y: 1



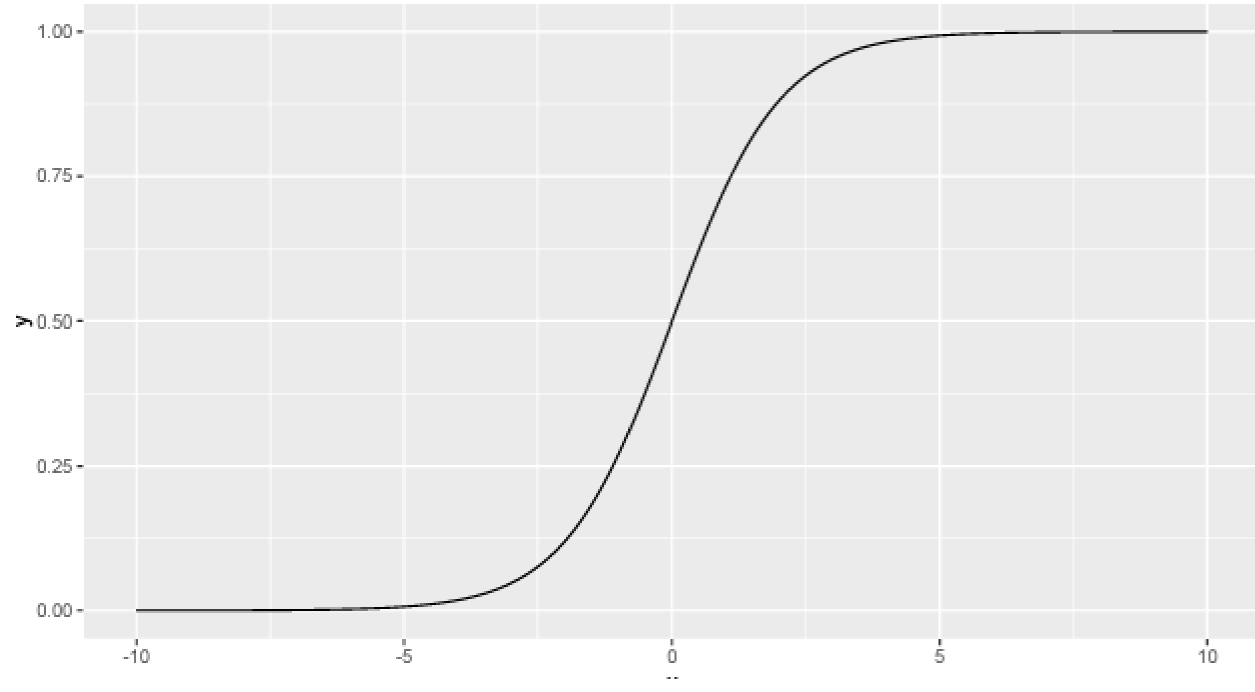
$$h_j = f\left(\sum_{i=1}^{F} x_i W_{i,j}\right)$$

the hidden nodes are completely determined by the input and weights

Activation functions

Squeezing outputs between 0 and 1

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

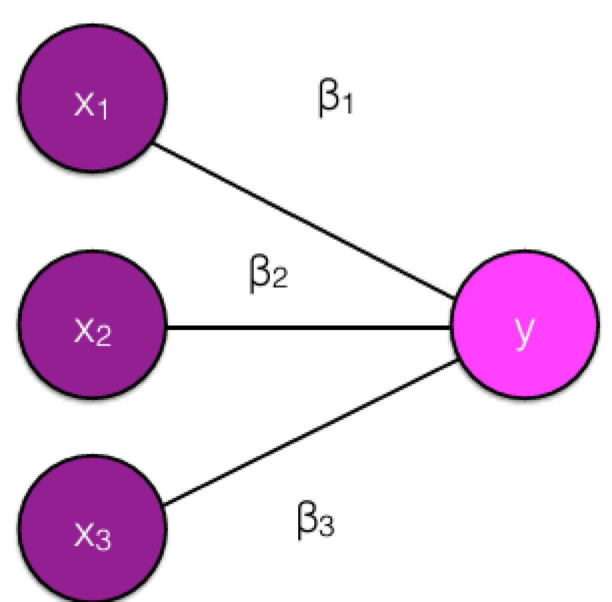


Activation functions

Squeezing outputs between 0 and 1

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

$$P(\hat{y} = 1) = \sigma\left(\sum_{i=1}^{F} x_i \beta_i\right)$$



$$P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

We can think about logistic regression as a neural network with no hidden layers

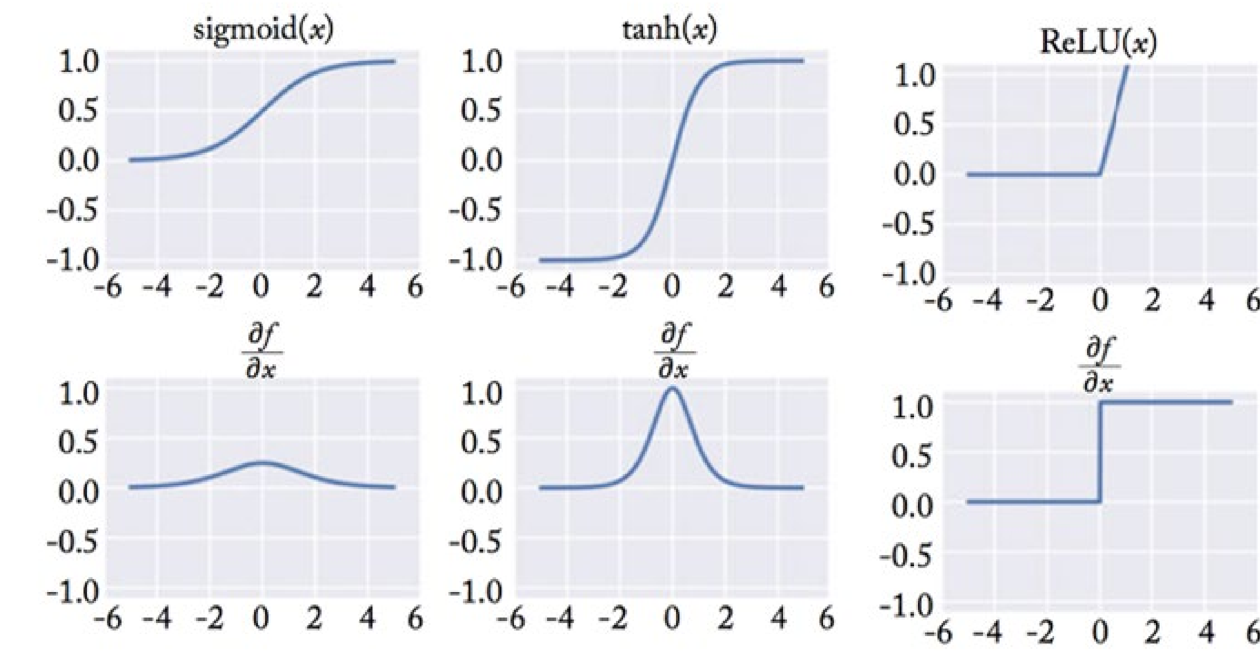
Activation functions

Squeezing outputs between 0 and 1

- ❑ ReLU and tanh are both used extensively in modern systems.
- ❑ Sigmoid is useful for final layer to scale output between 0 and 1, but is not often used in intermediate layers.

function

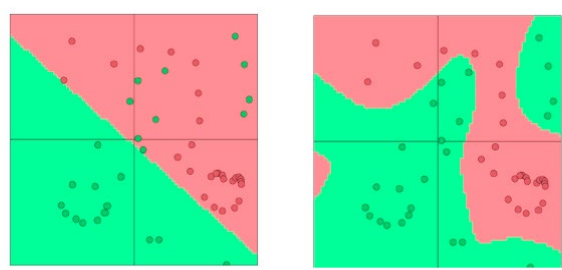
derivative

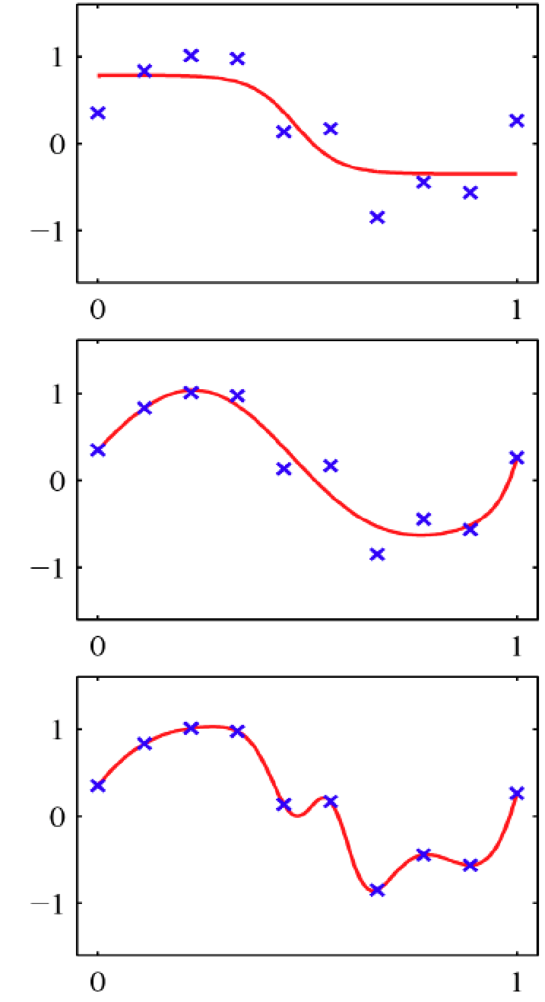


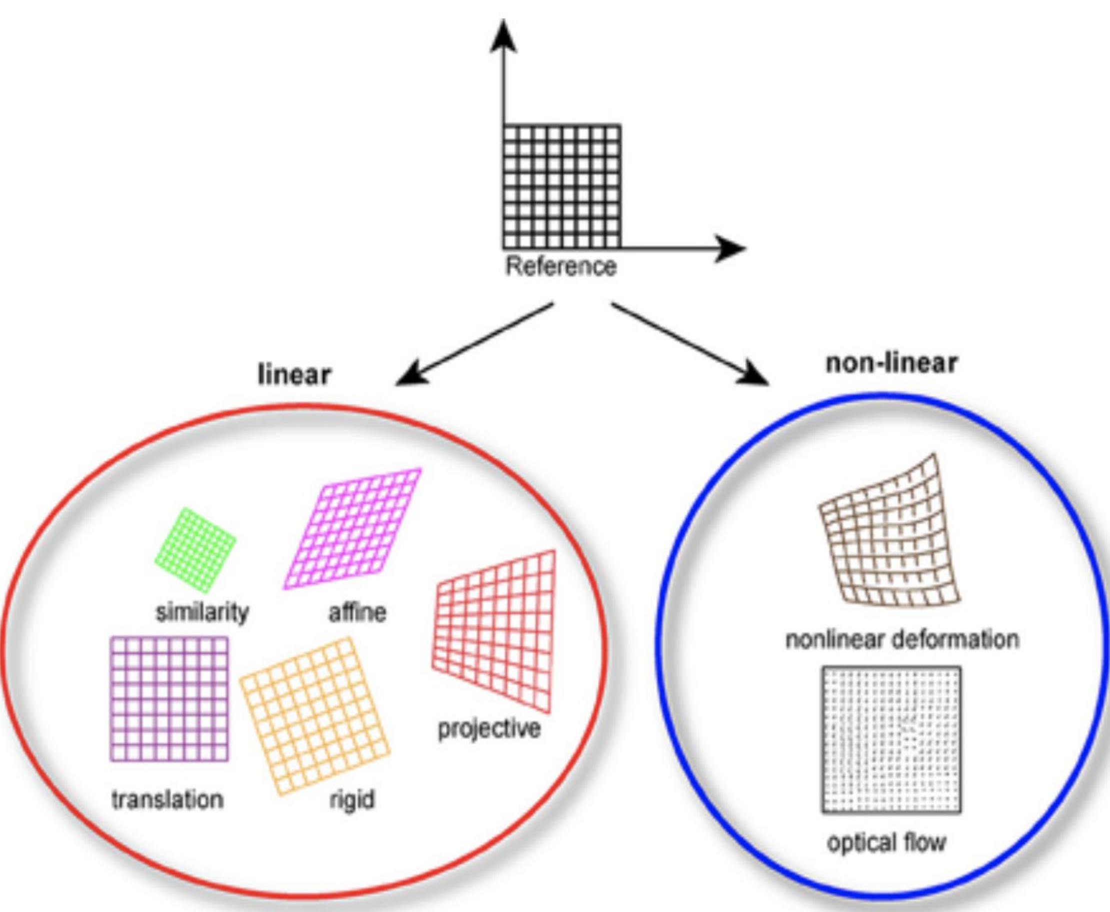
Goldberg 46

Non-linearities (i.e., f): why they're needed?

- Neural nets do function approximation
  - E.g., regression or classification
  - Without non-linearities, deep neural nets can't do anything more than a linear transform.
  - Extra layers could just be complied down into a single linear transform: $W_1 W_2 x = Wx$
  - But, with more layers that include non-linearities, they can approximate more complex functions

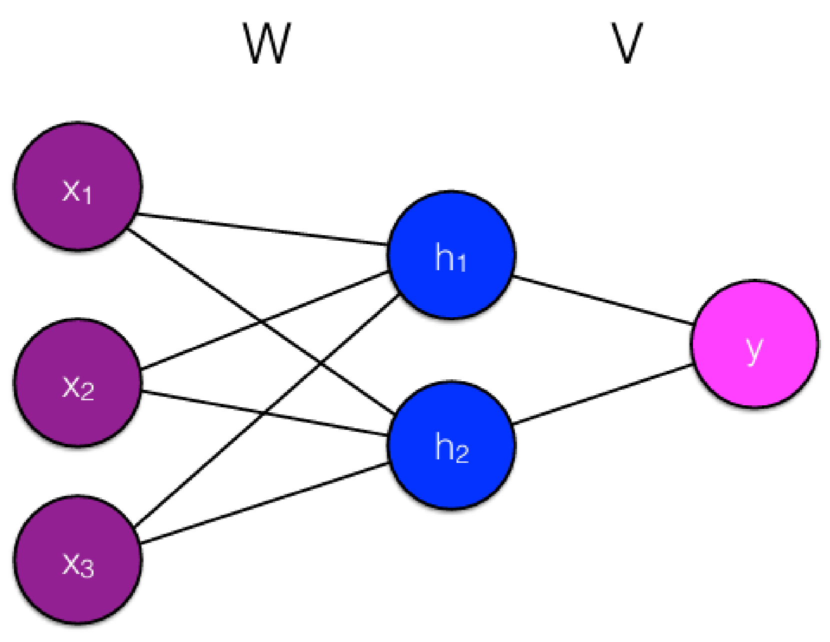






Linear models include translation, rigid (translation + rotation), similarity (translation + rotation + scale), affine and projective transformations. Nonlinear models, which consider non-linear transformations allow for more complex deformations.

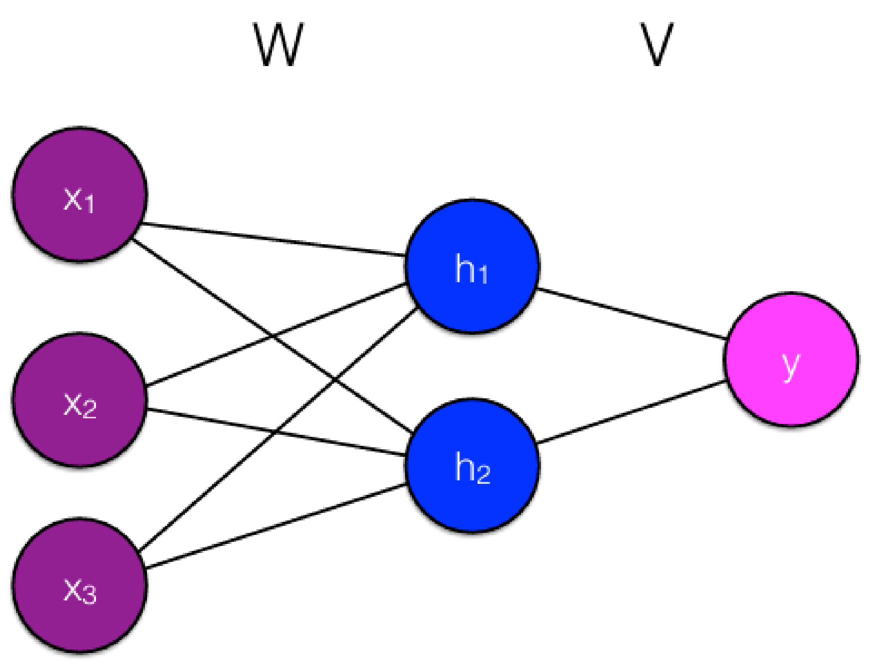
New techniques for motion-artifact-free in vivo cardiac microscopy



$$h_1 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,1}\right)$$

$$h_2 = \sigma\left(\sum_{i=1}^{F} x_i W_{i,2}\right)$$

$$\hat{y} = \sigma\left[V_1 h_1 + V_2 h_2\right]$$

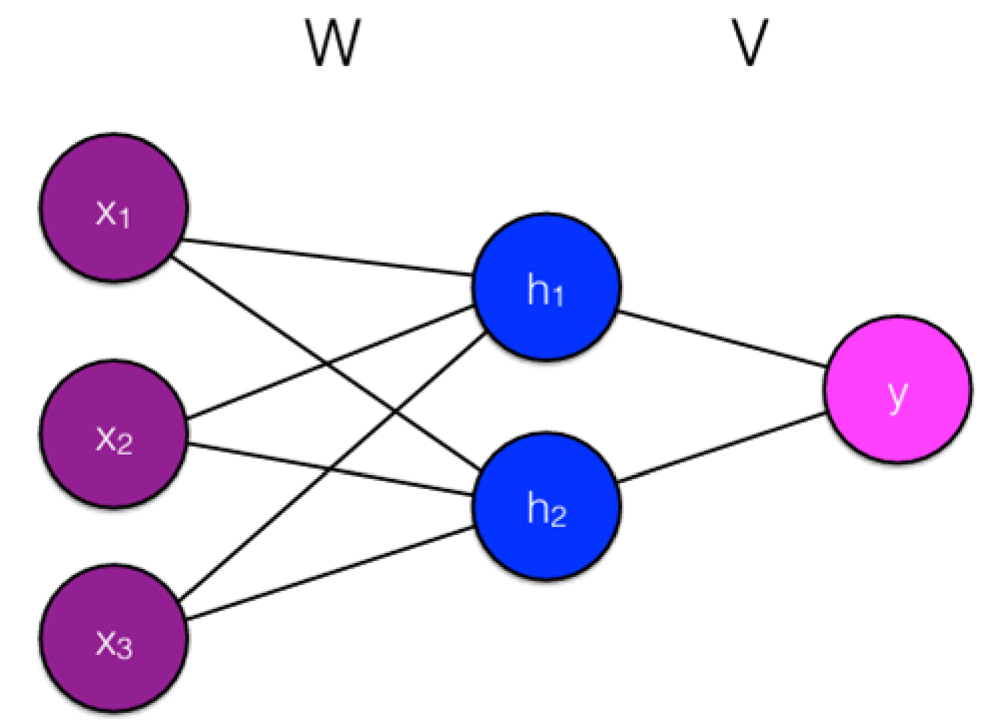


$$\hat{y} = \sigma \left[V_1 \left(\sigma \left(\sum_{i}^{F} x_i W_{i,1} \right) \right) + V_2 \left(\sigma \left(\sum_{i}^{F} x_i W_{i,2} \right) \right) \right]$$

$$\hat{y} = \sigma \left[V_1 \underbrace{\left(\sigma \left(\sum_{i}^{F} x_i W_{i,1} \right) \right)}_{h_1} + V_2 \underbrace{\left(\sigma \left(\sum_{i}^{F} x_i W_{i,2} \right) \right)}_{h_2} \right]$$

This is differentiable via backpropagation

Backpropagation: Given training samples of <x,y> pairs, we can use stochastic gradient descent to find the values of W and V that minimize the loss.



Neural networks are a series of functions chained together

$xW \rightarrow \sigma(xW) \rightarrow \sigma(xW)V \rightarrow \sigma(\sigma(xW)V)$

The loss is another function chained on top

$\log(\sigma(\sigma(xW)V))$

Chain rule

$$\frac{\partial}{\partial V}\log\left(\sigma\left(\sigma\left(xW\right)V\right)\right)$$

$$= \frac{\partial \log\left(\sigma\left(\sigma\left(xW\right)V\right)\right)}{\partial \sigma\left(\sigma\left(xW\right)V\right)} \frac{\partial \sigma\left(\sigma\left(xW\right)V\right)}{\partial \sigma\left(xW\right)V} \frac{\partial \sigma\left(xW\right)V}{\partial V}$$

$$= \overbrace{\frac{\partial \log\left(\sigma\left(hV\right)\right)}{\partial \sigma\left(hV\right)}}^{A} \overbrace{\frac{\partial \sigma\left(hV\right)}{\partial hV}}^{B} \overbrace{\frac{\partial hV}{\partial V}}^{C}$$

$$= \overbrace{\frac{1}{\sigma\left(hV\right)}}^{A} \times \overbrace{\sigma\left(hV\right) \times \left(1 - \sigma\left(hV\right)\right)}^{B} \times \overbrace{h}^{C}$$

$$= \left(1 - \sigma\left(hV\right)\right)h$$

$$= \left(1 - \hat{y}\right)h$$

Backpropagation

- Forward and backward propagation
  - Compute value/gradient of each node with respect to previous nodes
- Good news is that modern automatic differentiation tools do this all for you!
- Deep learning nowadays is like modular programming

```
class Feedforward(torch.nn.Module):
        def __init__(self, input_size, hidden_size):
            super(Feedforward, self).__init__()
            self.input_size = input_size
            self.hidden_size  = hidden_size
            self.fc1 = torch.nn.Linear(self.input_size,
self.hidden_size)
            self.relu = torch.nn.ReLU()
            self.fc2 = torch.nn.Linear(self.hidden_size, 1)
            self.sigmoid = torch.nn.Sigmoid()

        def forward(self, x):
            hidden = self.fc1(x)
            relu = self.relu(hidden)
            output = self.fc2(relu)
            output = self.sigmoid(output)
            return output
```

xW → $\sigma(xW)$ → $\sigma(xW)V$ → $\sigma(\sigma(xW)V)$

```python
class Feedforward(torch.nn.Module):
        def __init__(self, input_size, hidden_size):
            super(Feedforward, self).__init__()
            self.input_size = input_size
            self.hidden_size  = hidden_size
            self.fc1 = torch.nn.Linear(self.input_size,
self.hidden_size)
            self.relu = torch.nn.ReLU()
            self.fc2 = torch.nn.Linear(self.hidden_size, 1)
            self.sigmoid = torch.nn.Sigmoid()


        def forward(self, x):
            hidden = self.fc1(x)
            relu = self.relu(hidden)
            output = self.fc2(relu)
            output = self.sigmoid(output)
            return output
```

```python
model = Feedforward(2, 10)
criterion = torch.nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)
```

```
model.eval()
y_pred = model(x_test)
before_train = criterion(y_pred.squeeze(), y_test)
print('Test loss before training' , before_train.item())

model.train()
epoch = 20

for epoch in range(epoch):

    optimizer.zero_grad()

    # Forward pass
    y_pred = model(x_train)

    # Compute Loss
    loss = criterion(y_pred.squeeze(), y_train)

    print('Epoch {}: train loss: {}'.format(epoch, loss.item()))

    # Backward pass
    loss.backward()
    optimizer.step()
```

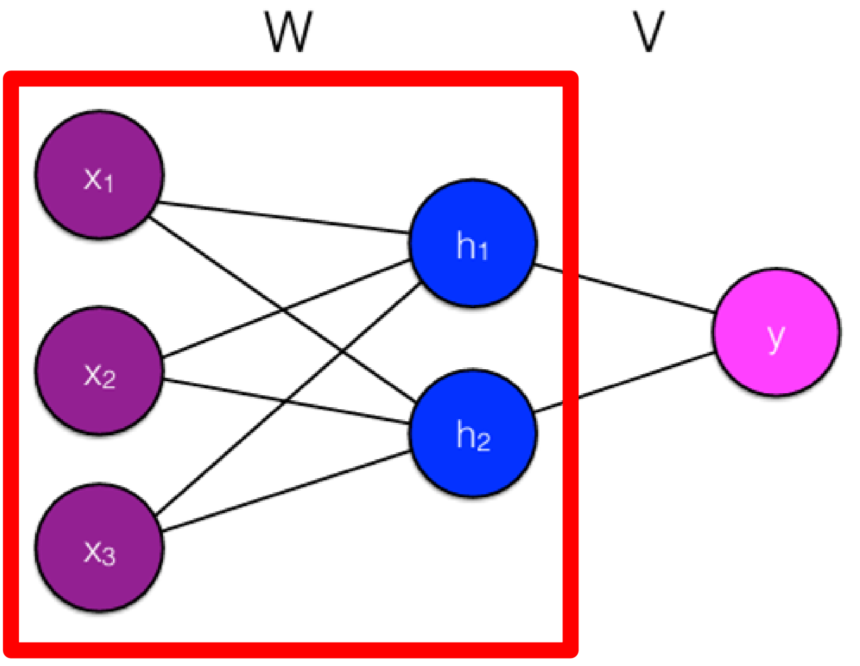
$$\log(\sigma(\sigma(xW)V))$$

```
model.eval()
y_pred = model(x_test)
after_train = criterion(y_pred.squeeze(), y_test)
print('Test loss after Training' , after_train.item())
```

Other tricks in neural network training

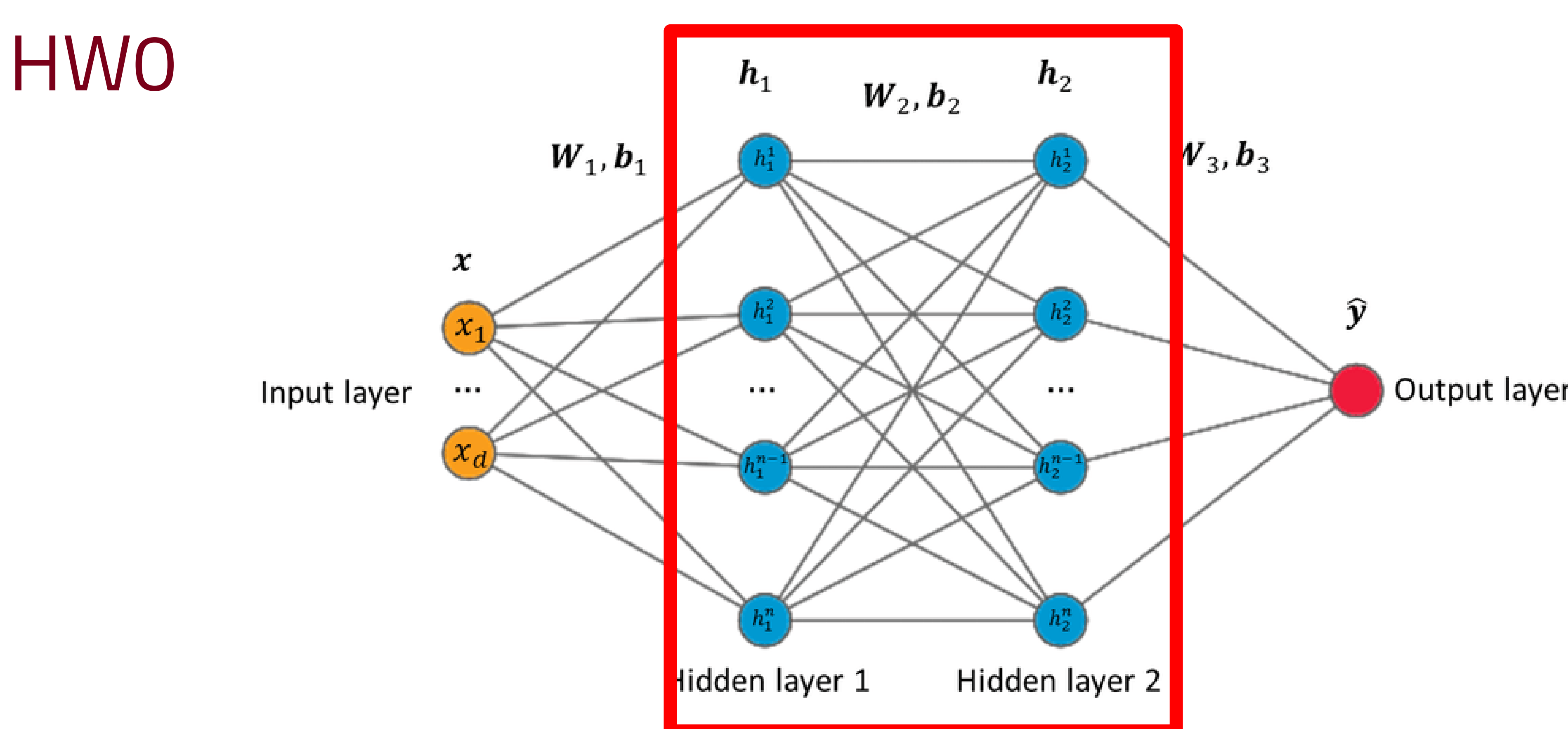
- Avoid overfitting with dropout
- Average/max/min pooling
- Smart initialization
- Adaptive learning rates than SGD
- Gradient clipping
- Early stopping with validation set
- Hyper-parameter tuning

Feedforward Neural Network (i.e., Single-layer Perceptron)

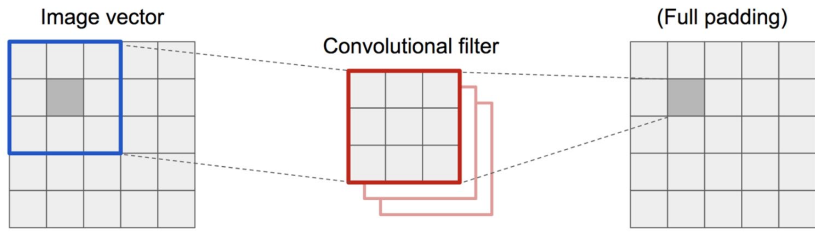


Feedforward Neural Network (i.e., Two-layer Perceptron)

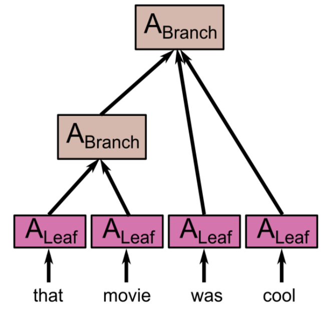
HW0



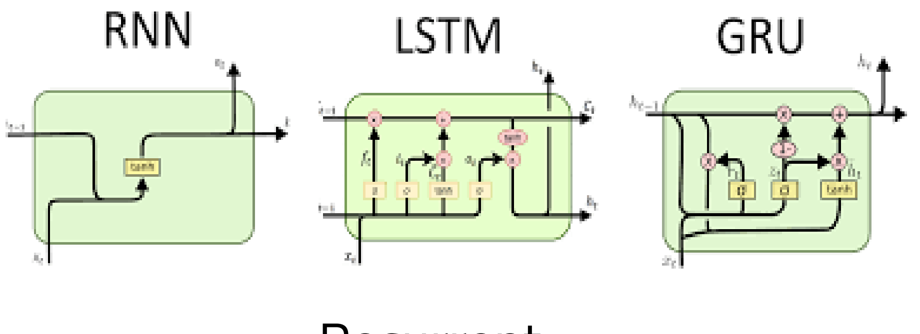
Other neural network models



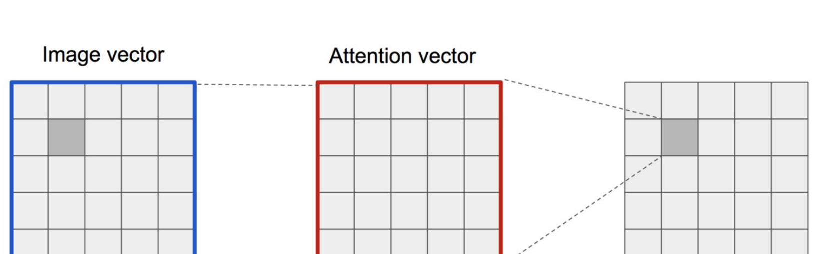
Convolution NN



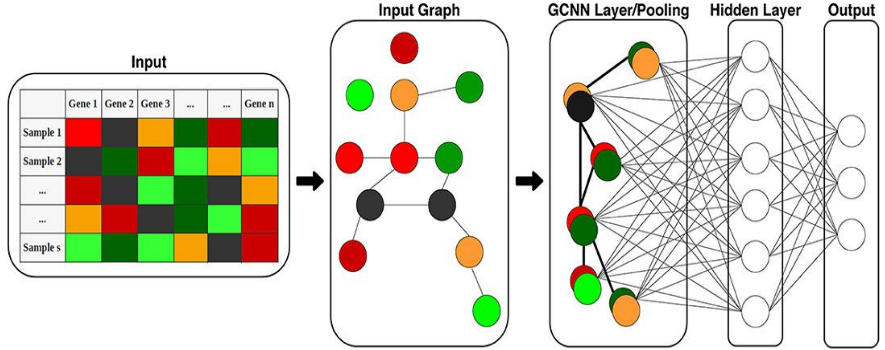
Recursive NN



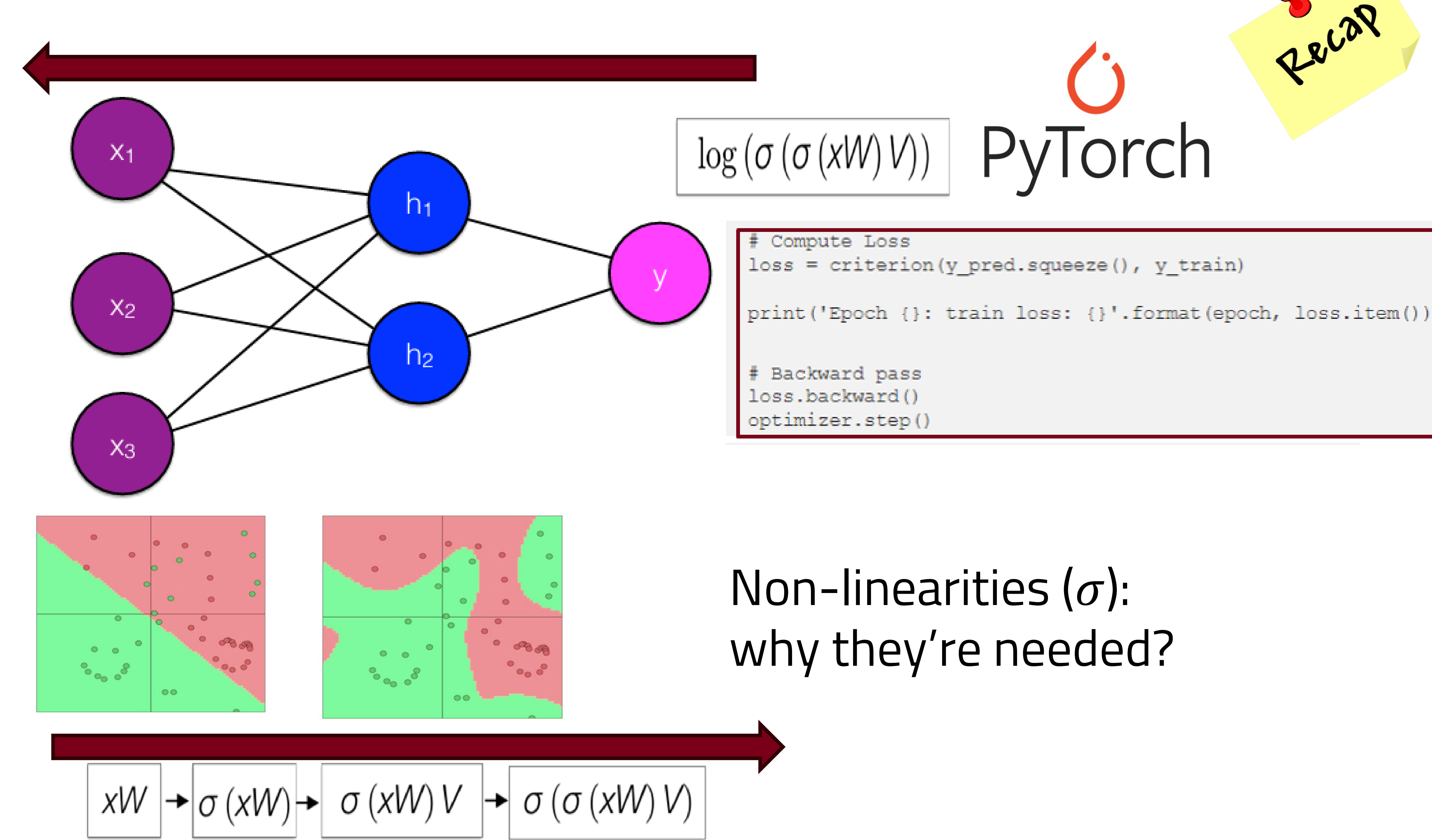
Recurrent-NN/LSTM/GRU



Self-attention / Transformers



Graph Convolutional / Neural Network

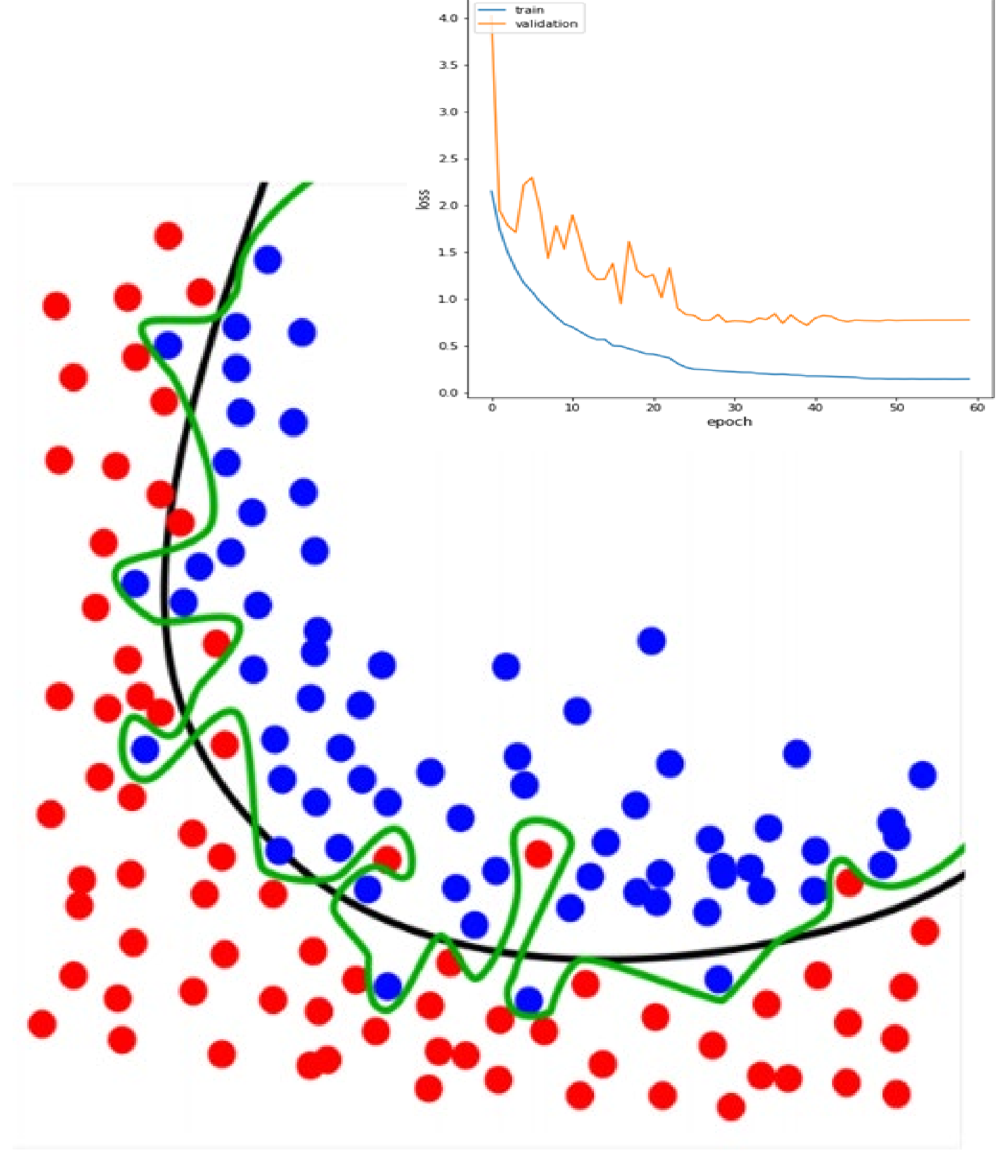



```
# Compute Loss
loss = criterion(y_pred.squeeze(), y_train)

print('Epoch {}: train loss: {}'.format(epoch, loss.item()))

# Backward pass
loss.backward()
optimizer.step()
```

Non-linearities (σ):
why they're needed?

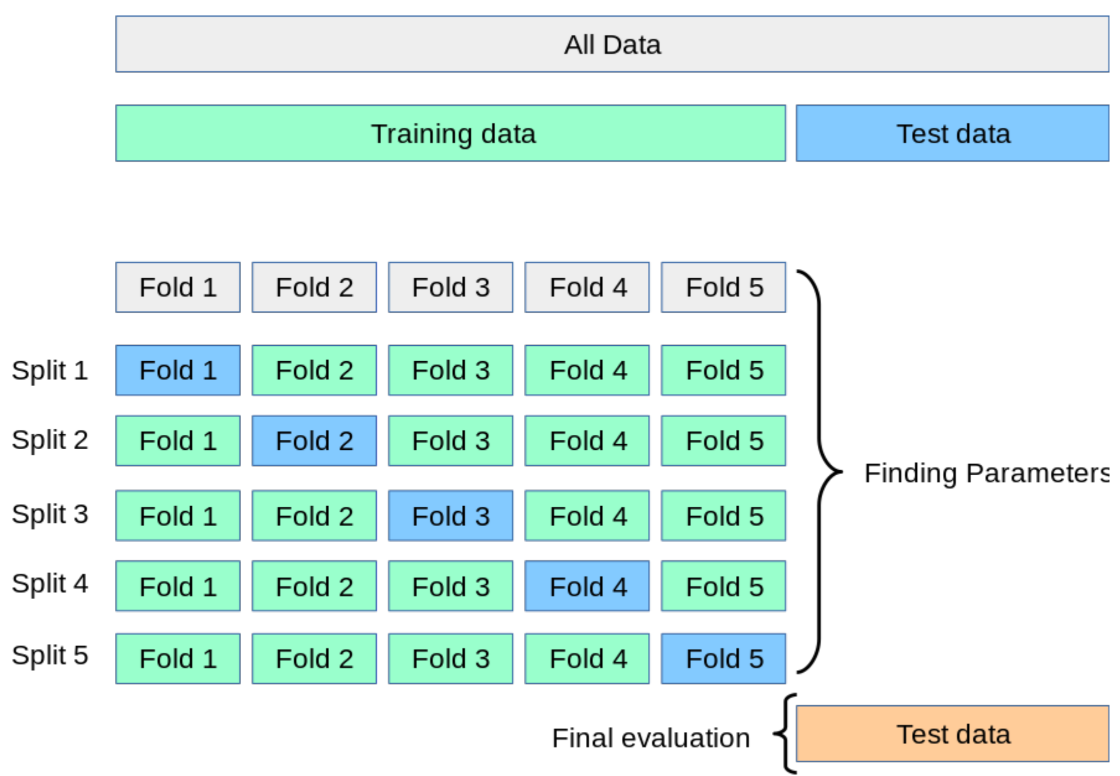
Overfitting

- A model that perfectly match the training data that has a problem
- It will also **overfit** to the data, modeling noise
  - A random word that perfectly predicts y (it happens to only occur in one class) will get a very high weight.
  - Failing to generalize to a test set without this word.
- A good model should be able to generalize



Cross validation

- Break up "training" data into 5 folds
- For each fold
  - Choose the fold as a temporary test set
  - Train on 5 folds, compute performance on test fold
- Report average performance of the 5 runs
- Find the best parameters



https://scikit-learn.org/stable/modules/cross_validation.html

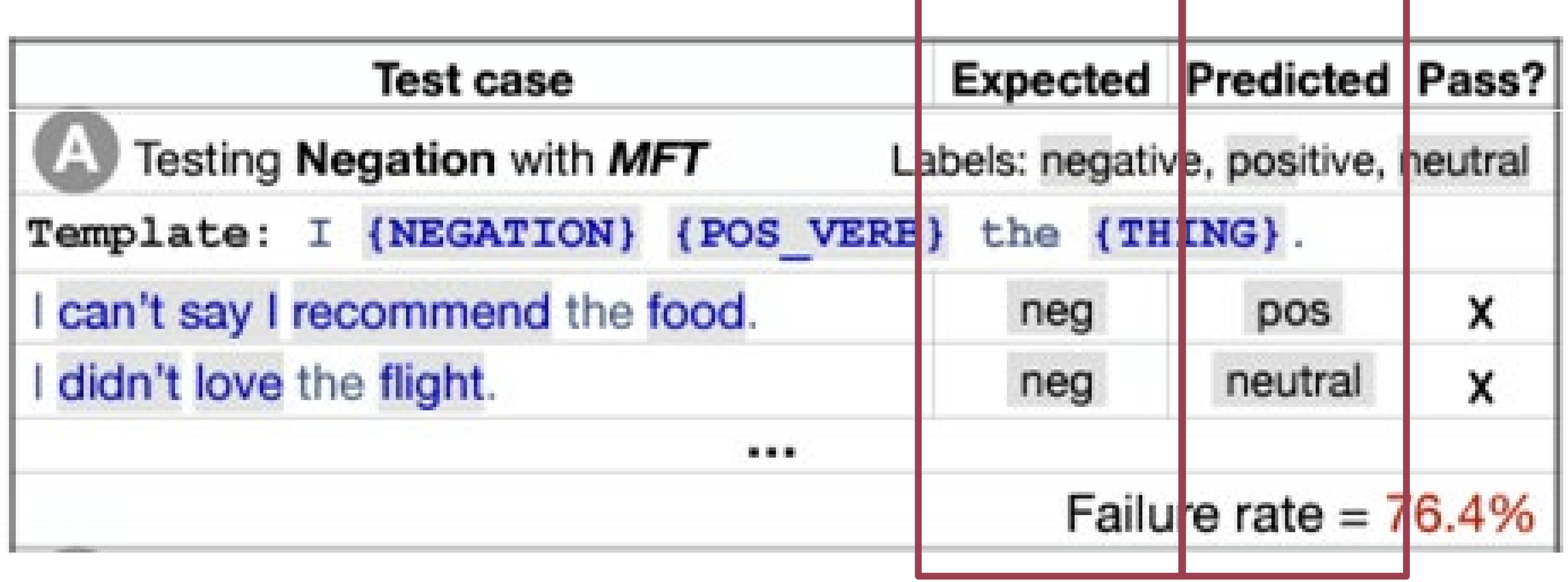
State of the Art

https://paperswithcode.com/

Sentiment Analysis on SST-2 Binary classification

Rank	Model	Accuracy	Paper	Code	Result	Year	Tags
1	**SMART-RoBERTa Large**	97.5	SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization			2019	Transformer
2	**T5-3B**	97.4	Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer			2019	Transformer
3	**MUPPET Roberta Large**	97.4	Muppet: Massive Multi-task Representations with Pre-Finetuning			2021	
4	**ALBERT**	97.1	ALBERT: A Lite BERT for Self-supervised Learning of Language Representations			2019	Transformer
5	**T5-11B**	97.1	Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer			2019	Transformer
6	**StructBERTRoBERTa ensemble**	97.1	StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding			2019	Transformer
7	**XLNet** (single model)	97	XLNet: Generalized Autoregressive Pretraining for Language Understanding			2019	Transformer
8	**ELECTRA**	96.9	ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators			2020	
9	**EFL**	96.9	Entailment as Few-Shot Learner			2021	Transformer
10	**XLNet-Large** (ensemble)	96.8	XLNet: Generalized Autoregressive Pretraining for Language Understanding			2019	Transformer
11	**RoBERTa**	96.7	RoBERTa: A Robustly Optimized BERT Pretraining Approach			2019	Transformer

Robustness of Neural Classifiers



Ribeiro et al., Beyond Accuracy: Behavioral Testing of NLP Models with CheckList, ACL 2020

Robustness of Neural Classifiers

Ribeiro et al., Beyond Accuracy: Behavioral Testing of NLP Models with CheckList, ACL 2020

Robustness of Neural Classifiers

Test case	Expected	Predicted	Pass?
Testing **Vocabulary** with ***DIR***	Sentiment monotonic decreasing (↓)		
@AmericanAir service wasn't great. You are lame.	↓	neg → neutral	X
@JetBlue why won't YOU help them?! Ugh. I dread you.	↓	neg → neutral	X
...			
	Failure rate = 34.6%		

Ribeiro et al., Beyond Accuracy: Behavioral Testing of NLP Models with CheckList, ACL 2020

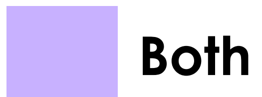
Rank	Model	Accuracy	Paper	Code	Result	Year	Tags
1	**SMART-RoBERTa Large**	97.5	SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization			2019	Transformer
2	**T5-3B**	97.4	Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer			2019	Transformer
3	**MUPPET Roberta Large**	97.4	Muppet: Massive Multi-task Representations with Pre-Finetuning			2021	
4	**ALBERT**	97.1	ALBERT: A Lite BERT for Self-supervised Learning of Language Representations			2019	Transformer
5	**T5-11B**	97.1	Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer			2019	Transformer
6	**StructBERTRoBERTa ensemble**	97.1	StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding			2019	Transformer
7	**XLNet** (single model)	97	XLNet: Generalized Autoregressive Pretraining for Language Understanding			2019	Transformer
8	**ELECTRA**	96.9	ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators			2020	
9	**EFL**	96.9	Entailment as Few-Shot Learner			2021	Transformer
10	**XLNet-Large** (ensemble)	96.8	XLNet: Generalized Autoregressive Pretraining for Language Understanding			2019	Transformer
11	**RoBERTa**	96.7	RoBERTa: A Robustly Optimized BERT Pretraining Approach			2019	Transformer

Interpretability: why? learning dataset, not task

Human: Polite

BERT: Polite

I will understand if you decline, but would very much like you to accept. May I nominate you?



Hayati et al., Does BERT Learn as Humans Perceive? Understanding Linguistic Styles through Lexica

Dataset Characterization

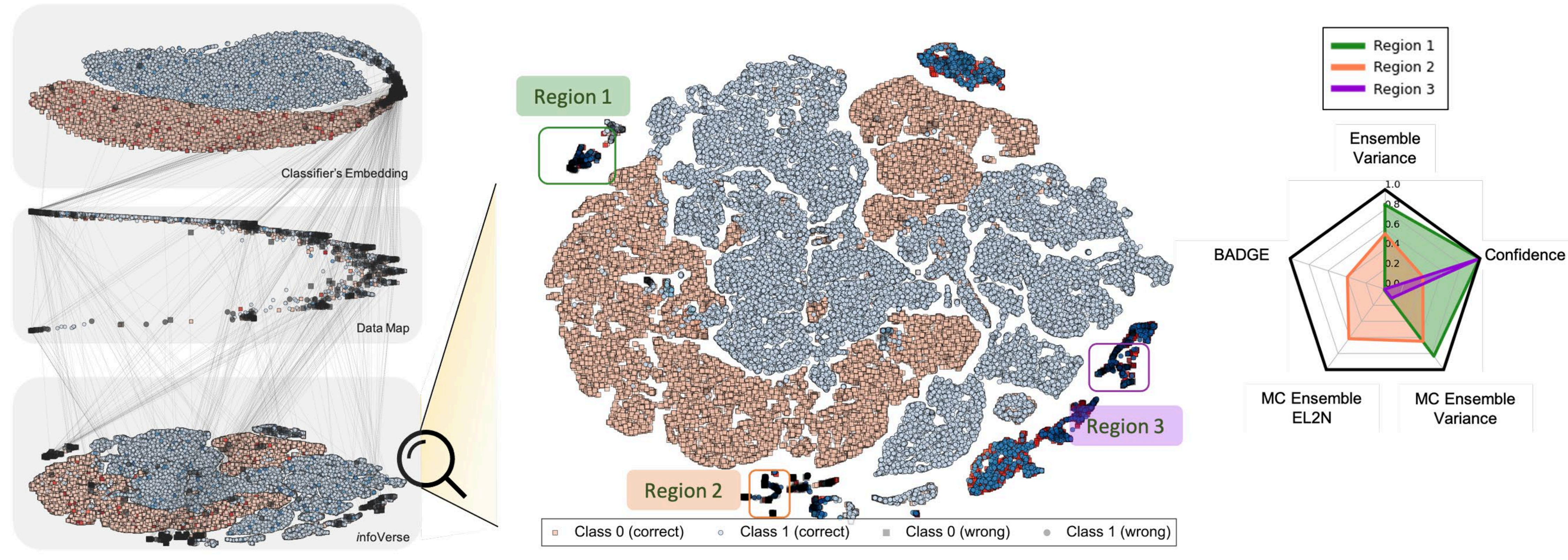


Figure 20: infoVerse (bottom left) on SST-2 along with other feature spaces: classifier embedding (top left) and *data map* (Swayamdipta et al., 2020) (middle left). (middle) Zoomed version of infoVerse is presented. (right) Score distribution of each wrong region characterized by infoVerse.

Kim et al., infoVerse: A Universal Framework for Dataset Characterization with Multidimensional Meta-information, ACL 2023

Run yourself

https://huggingface.co/datasets/sst2

Summary

- Various applications using sentiment analysis in political and social sciences, stock market prediction, advertising, etc.
- Sentiment of text is reflection of the speaker's private state, which is hardly *observable*.
- Lexicon dictionaries have limitations, because sentiment is *contextual*
- Sentiment + X
- Modern deep representations perform better but are hard to *interpret*, and easy to be *biased* to the dataset
- 97.5 accuracy on SST2, but poor *robustness* in practice

Questions

- Is there any way to take advantages from both the classical dictionary based method and modern neural model?
- How can we evaluate and improve robustness of the model? How can we collect even more challenging samples that the current best model can’t predict well?
- How can we make black-box deep learning models to be more interpretable?
- Is benchmarking/leader-boarding a good practice for evaluation? If not, what is the solution?