From Regressions to Transformers

CSE545 - Spring 2023
Stony Brook University

H. Andrew Schwartz

Big Data Analytics, The Class

Goal: Generalizations
A *model* or *summarization* of the data.

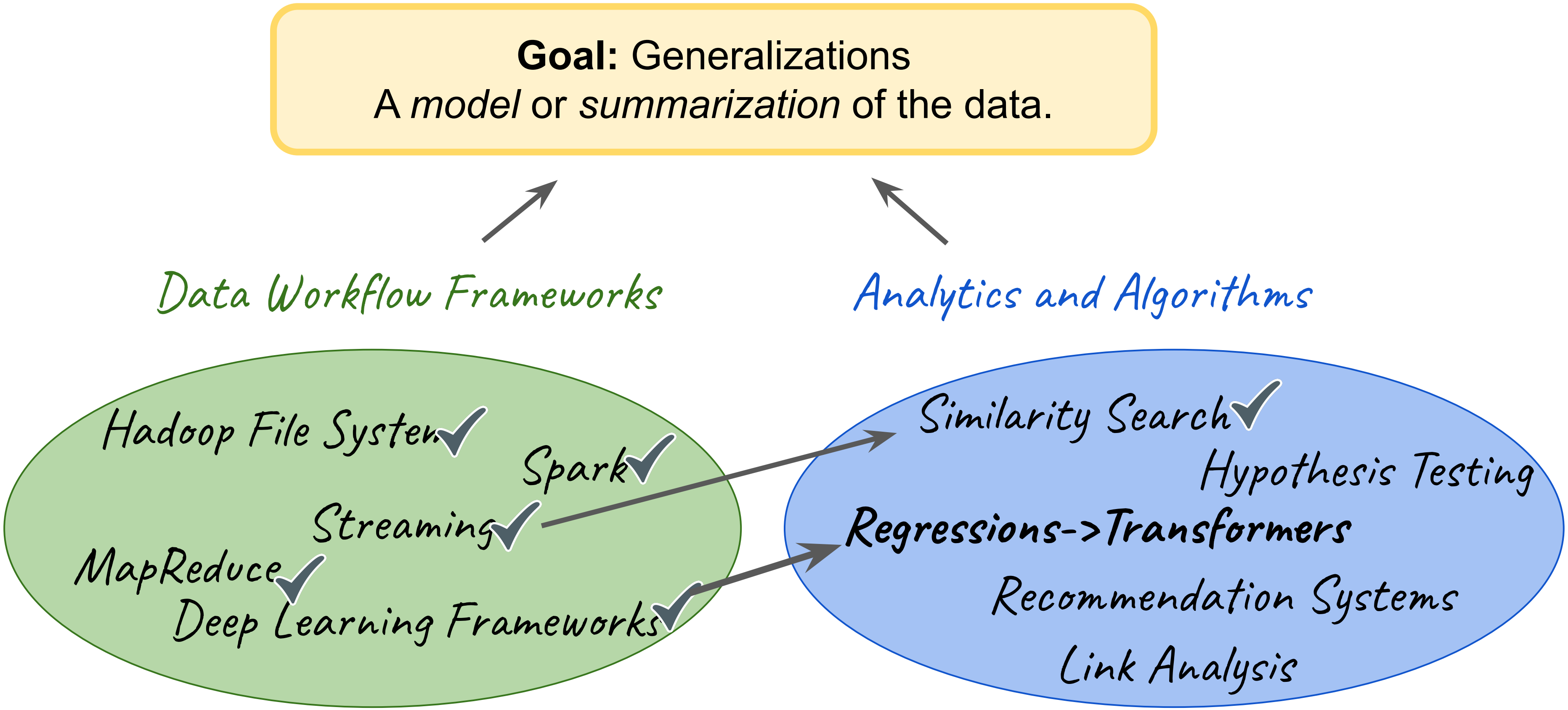
Data Workflow Frameworks

- Hadoop File System ✓
- Spark ✓
- Streaming ✓
- MapReduce ✓
- Deep Learning Frameworks ✓

Analytics and Algorithms

- Similarity Search ✓
- Hypothesis Testing
- **Regressions->Transformers**
- Recommendation Systems
- Link Analysis

Big Data Analytics, The Class



Deep Learning

Already Covered:

- Pytorch as a dataflow system of with a graph of tensors, operations, and building blocks
- Implementation of Linear Regression in PyTorch
- Minimizing error (concept of gradient descent)
- Parallelisms: Data Parallelism and Model Parallelism

(see topic (5) Neural Network Workflow Systems)

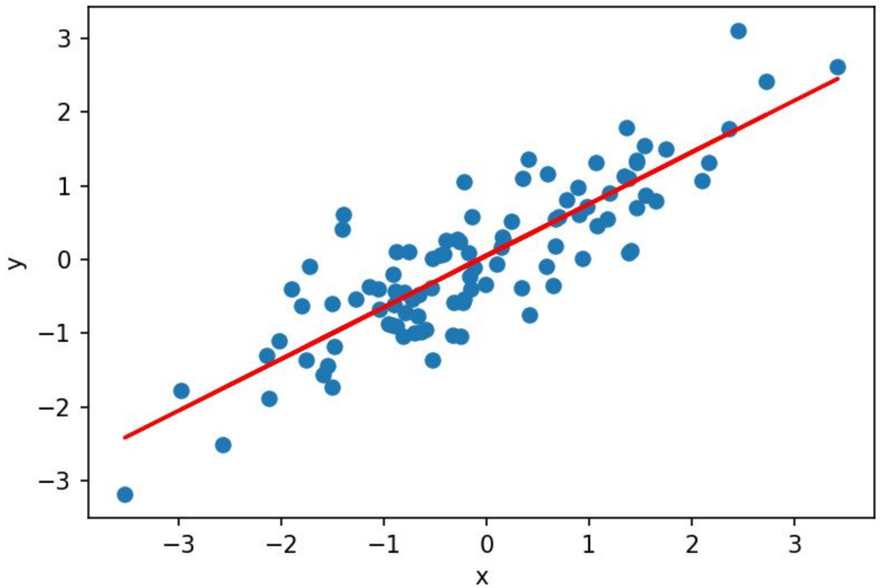
Deep Learning

Already Covered:

- Pytorch as a dataflow system of with a graph of tensors, operations, and building blocks
- **Implementation of Linear Regression in PyTorch**
- **Minimizing error (concept of gradient descent)**
- Parallelisms: Data Parallelism and Model Parallelism

(see topic (5) Neural Network Workflow Systems)

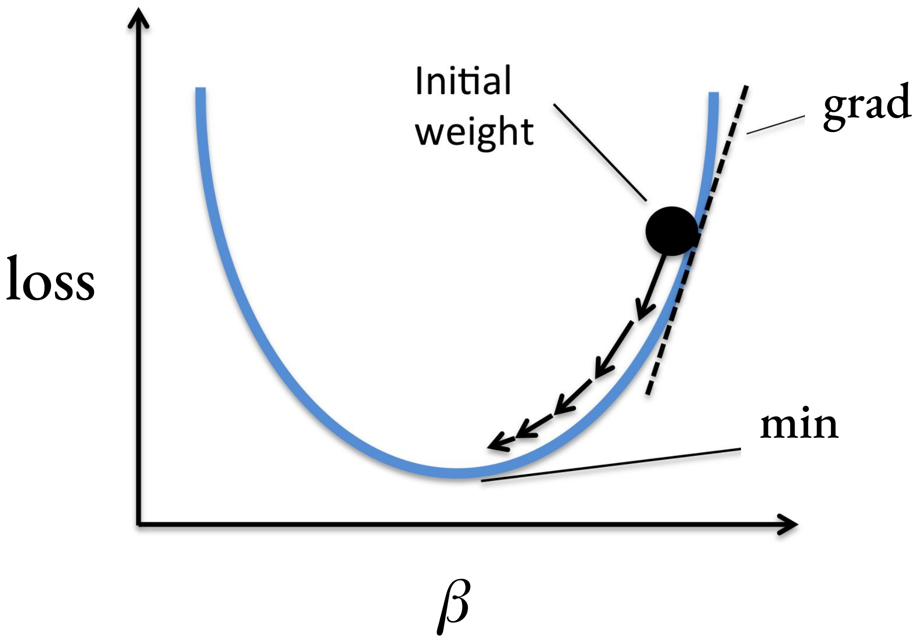
Linear Regression and Gradient Descent



$$y = mx + b$$

$$y = \beta_0 + \beta_1 x$$

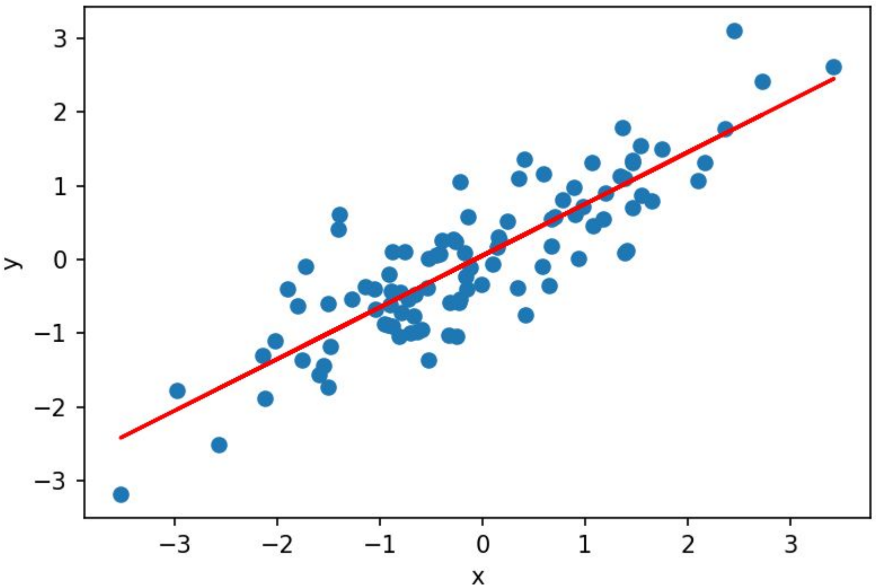
https://www.desmos.com/calculator/y8j7sejtuw
https://www.desmos.com/calculator/2usyk3ykts



$$\beta_{new} = \beta_{prev} - \alpha * \text{grad}$$

(rasbt, http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)

Linear Regression and Gradient Descent



$$y\_pred = mx + b$$

$$y\_pred = \beta_0 + \beta_1 x$$

$$\text{loss} = \Sigma\,(y\_pred - y)^2$$

$$\beta_{new} = \beta_{prev} - \alpha * \text{grad}$$

https://www.desmos.com/calculator/y8j7sejtuw
https://www.desmos.com/calculator/2usyk3ykts

(rasbt, http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)

Regressions

- Linear Regression
- Pearson Product-Moment Correlation
- Multiple Linear Regression
- (Multiple) Logistic Regression
- Ridge Regularized Linear/Logistic Regression

Regressions

- **Linear Regression**
- **Pearson Product-Moment Correlation**
- Multiple Linear Regression
- (Multiple) Logistic Regression
- Ridge Regularized Linear/Logistic Regression

Linear Regression

Finding a linear function based on *X* to best yield *Y*.

X = “covariate” = “feature” = “predictor” = “regressor” = “independent variable”

Y = “response variable” = “outcome” = “dependent variable”

Regression: $$r(x) = \mathrm{E}(Y|X = x)$$

goal: estimate function r

The **expected** value of *Y*, given that the random variable *X* is equal to some specific value, *x*.

Linear Regression

Finding a linear function based on *X* to best yield *Y.*

X = “covariate” = “feature” = “predictor” = “regressor” = “independent variable”

Y = “response variable” = “outcome” = “dependent variable”

Regression:

$$r(x) = \mathrm{E}(Y|X = x)$$

goal: estimate the function r

Linear Regression (univariate version):

$$r(x) = \beta_0 + \beta_1 x$$

goal: find β_0, β_1 such that

$$r(x) \approx \mathrm{E}(Y|X = x)$$

Linear Regression

Simple Linear Regression $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

$$\text{where } \mathbf{E}(\epsilon_i | X_i) = 0 \textit{ and } \mathbf{V}(\epsilon_i | X_i) = \sigma^2$$

more precisely

$$r(x) = \beta_0 + \beta_1 x$$

Linear Regression

intercept slope error

Simple Linear Regression $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

$$\text{where } \mathbf{E}(\epsilon_i | X_i) = 0 \textit{ and } \mathbf{V}(\epsilon_i | X_i) = \sigma^2$$

expected variance

Linear Regression: Estimating Params

Simple Linear Regression $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

$$\text{where } \mathbf{E}(\epsilon_i | X_i) = 0 \; and \; \mathbf{V}(\epsilon_i | X_i) = \sigma^2$$

How to estimate intercept (□$_0$) and slope intercept (□$_1$)?

Least Squares Estimate. Find $\hat{\beta}_0$ and $\hat{\beta}_1$ which minimizes the *residual sum of squares:*

$$J(\square) = RSS = \sum_{i=1}^{n} \hat{\epsilon}_i^2 = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2$$

Pearson Product-Moment Correlation

Covariance

$$
\begin{aligned}
Cov(X, Y) &= \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y) \\
&= \mathbf{E}\left((X - \bar{X})(Y - \bar{Y})\right)
\end{aligned}
$$

Pearson Product-Moment Correlation

Covariance

$$\begin{aligned} Cov(X,Y) &= \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y) \\ &= \mathbf{E}\left((X-\bar{X})(Y-\bar{Y})\right) \end{aligned}$$

Correlation

$$r = r_{X,Y} = \frac{Cov(X,Y)}{s_X s_Y}$$

Pearson Product-Moment Correlation

Covariance

$$
\begin{aligned}
Cov(X,Y) &= \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y) \\
&= \mathbf{E}\left((X-\bar{X})(Y-\bar{Y})\right)
\end{aligned}
$$

Correlation *(standardized covariance)*

$$
r = r_{X,Y} = \frac{Cov(X,Y)}{s_X s_Y}
$$

$$
= \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{X_i-\bar{X}}{s_X}\right)\left(\frac{Y_i-\bar{Y}}{s_Y}\right)
$$

Pearson Product-Moment Correlation

Covariance

$$Cov(X,Y) = \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y)$$
$$= \mathbf{E}\left((X-\bar{X})(Y-\bar{Y})\right)$$

Correlation

$$r = r_{X,Y} = \frac{Cov(X,Y)}{s_X s_Y}$$
$$= \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{X_i - \bar{X}}{s_X}\right)\left(\frac{Y_i - \bar{Y}}{s_Y}\right)$$

Lin Reg Direct Estimates (normal equations)

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1\bar{X}$$

Pearson Product-Moment Correlation

Covariance

$$
\begin{aligned}
Cov(X,Y) &= \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y) \\
&= \mathbf{E}\left((X-\bar{X})(Y-\bar{Y})\right)
\end{aligned}
$$

Correlation

$$
\begin{aligned}
r = r_{X,Y} &= \frac{Cov(X,Y)}{s_X s_Y} \\
&= \frac{1}{n-1}\sum_{i=1}^{n} \boxed{\left(\frac{X_i-\bar{X}}{s_X}\right)\left(\frac{Y_i-\bar{Y}}{s_Y}\right)}
\end{aligned}
$$

Lin Reg Direct Estimates (normal equations)

$$
\hat{\beta}_1 = \boxed{\frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sum_{i=1}^{n}(X_i-\bar{X})^2}}
$$

$$
\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1\bar{X}
$$

Pearson Product-Moment Correlation

Covariance

$$Cov(X,Y) = \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y) \\ = \mathbf{E}\left((X-\bar{X})(Y-\bar{Y})\right)$$

Correlation

$$r = r_{X,Y} = \frac{Cov(X,Y)}{s_X s_Y}$$

$$= \frac{1}{n-1}\sum_{i=1}^{n} \boxed{\left(\frac{X_i - \bar{X}}{s_X}\right)\left(\frac{Y_i - \bar{Y}}{s_Y}\right)}$$

Lin Reg Direct Estimates (normal equations)

$$\hat{\beta}_1 = \boxed{\frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

If one standardizes *X* and *Y* (i.e. subtract the mean and divide by the standard deviation) before running linear regression, then:

??

Pearson Product-Moment Correlation

Covariance

$$\begin{aligned} Cov(X,Y) &= \mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y) \\ &= \mathbf{E}\left((X-\bar{X})(Y-\bar{Y})\right) \end{aligned}$$

Correlation

$$\begin{aligned} r = r_{X,Y} &= \frac{Cov(X,Y)}{s_X s_Y} \\ &= \frac{1}{n-1}\sum_{i=1}^{n} \boxed{\left(\frac{X_i-\bar{X}}{s_X}\right)\left(\frac{Y_i-\bar{Y}}{s_Y}\right)} \end{aligned}$$

Lin Reg Direct Estimates (normal equations)

$$\hat{\beta}_1 = \boxed{\frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sum_{i=1}^{n}(X_i-\bar{X})^2}}$$

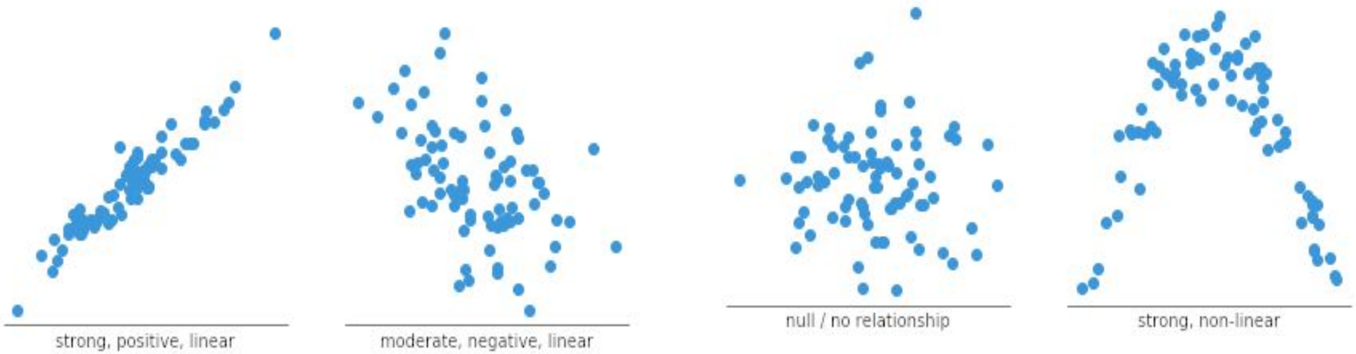
$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1\bar{X}$$

If one standardizes *X* and *Y* (i.e. subtract the mean and divide by the standard deviation) before running linear regression, then:

$\hat{\beta}_0 = 0$ and $\hat{\beta}_1 = r$ --- *i.e.* $\hat{\beta}_1$ *is the Pearson correlation!*

Useful Plots: Correlation

Scatter Plot: for two variables expected to be associated (with optional regression line)



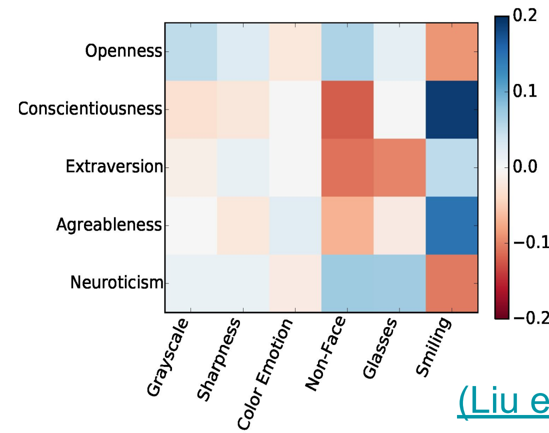
(Chartio)

Correlation Matrix: for comparing associations between many variables (use Bonferroni correction if hyp testing)

	FriendSize	Intelligence Quotient	Income	Sat W/ Life	Depression
F1	0.03	0.04	0.12	0.02	-0.1
F2	0.04	-0.26	-0.19	-0.09	0.11
F3	-0.07	-0.13	0.02	-0.02	-0.02
F4	-0.03	0.27	-0.08	-0.12	0.11
F5	-0.01	0.23	0.29	0.07	-0.21

Fig 3. Individual factor correlations with outcomes. Note how F4 which captures the use of swear words negatively correlates with Satisfaction with Life (SWL).

https://doi.org/10.1371/journal.pone.0201703.g003



(Liu et al., 2016)

Regressions

- **Linear Regression**
- **Pearson Product-Moment Correlation**
- Multiple Linear Regression
- (Multiple) Logistic Regression
- Ridge Regularized Linear/Logistic Regression

Regressions

- Linear Regression
- Pearson Product-Moment Correlation
- **Multiple Linear Regression**
- (Multiple) Logistic Regression
- Ridge Regularized Linear/Logistic Regression

Multiple Linear Regression

Simple Linear Regression $$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

$$\text{where } \mathbf{E}(\epsilon_i | X_i) = 0 \ and \ \mathbf{V}(\epsilon_i | X_i) = \sigma^2$$

Estimated intercept and slope

$$\hat{r}(x) = \hat{\beta}_0 + \hat{\beta}_1 x \qquad \hat{Y}_i = \hat{r}(X_i)$$

Residual: $$\hat{\epsilon}_i = Y_i - \hat{Y}_i$$

Multiple Linear Regression

Suppose we have multiple X that we'd like to fit to Y at once:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_m X_{m1} + \epsilon_i$$

If we include and $X_{oi} = 1$ for all i (i.e. adding the intercept to X), then we can say:

$$Y_i = \sum_{j=0}^{m} \beta_j X_{ij} + \epsilon_i$$

Multiple Linear Regression

Suppose we have multiple X that we'd like to fit to Y at once:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_m X_{m1} + \epsilon_i$$

If we include and $X_{oi} = 1$ for all i, then we can say:

$$Y_i = \sum_{j=0}^{m} \beta_j X_{ij} + \epsilon_i$$

Or in vector notation across all i:

$$Y = X\beta + \epsilon$$

where β and ϵ are vectors and X is a matrix.

Multiple Linear Regression

Suppose we have multiple *X* that we'd like to fit to *Y* at once:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_m X_{m1} + \epsilon_i$$

If we include and $X_{oi} = 1$ for all *i*, then we can say:

$$Y_i = \sum_{j=0}^{m} \beta_j X_{ij} + \epsilon_i$$

Or in vector notation across all i:

$$Y = X\beta + \epsilon$$

where β and ϵ are vectors and *X* is a matrix.

Estimating β :

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

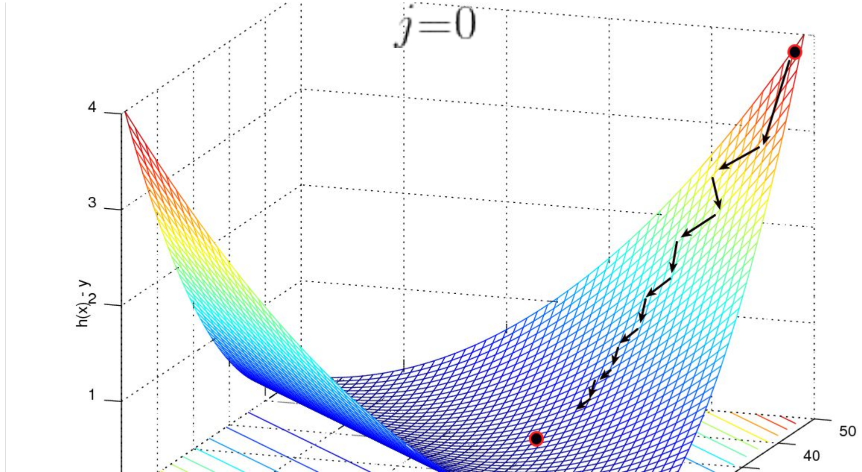
Multiple Linear Regression

Suppose we have multiple X that we'd like to fit to Y at once:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_m X_{m1} + \epsilon_i$$

If we include and $X_{oi} = 1$ for all i, then we can say:

$$Y_i = \sum_{j=0}^{m} \beta_j X_{ij} + \epsilon_i$$



Or in vector notation across all i:

$$Y = X\beta + \epsilon$$

where β and ϵ are vectors and X is a matrix.

Estimating β :

← Use Gradient Descent

Regressions

- Linear Regression
- Pearson Product-Moment Correlation
- **Multiple Linear Regression**
- (Multiple) Logistic Regression
- Ridge Regularized Linear/Logistic Regression

Regressions

- Linear Regression
- Pearson Product-Moment Correlation
- Multiple Linear Regression
- **(Multiple) Logistic Regression**
- Ridge Regularized Linear/Logistic Regression

Logistic Regression on a single feature (*x*)

$Y_i \in \{0, 1\}$; X is a **single value** and can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}$$

Logistic Regression on a single feature (*x*)

$Y_i \in \{0, 1\}$; X is a **single value** and can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}$$

$$= \frac{1}{1 + e^{-(\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij})}}$$

Logistic Regression on a single feature (*x*)

$Y_i \in \{0, 1\}$; X is a **single value** and can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}$$

$$= \frac{1}{1 + e^{-(\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij})}}$$

$$= \frac{1}{1+e^{-\beta x}}$$ (vector multiply)

Logistic Regression on a single feature (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}$$

The goal of this function is to: take in the variable *x* and
return a probability that *Y* is 1.

Logistic Regression on a single feature (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$p_i \equiv P(Y_i = 1 | X_i = x) = \frac{e^{\boxed{\beta_0} + \boxed{\beta_1}\boxed{x_i}}}{1 + e^{\boxed{\beta_0} + \boxed{\beta_1}\boxed{x_i}}}$$

The goal of this function is to: <u>take in the variable *x*</u> and <u>return a probability that *Y* is 1</u>.

Note that there are only three variables on the right: X_i, B_0, B_1

Logistic Regression on a single feature (x)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$p_i \equiv P(Y_i = 1 | X_i = x) = \frac{e^{\boxed{\beta_0} + \boxed{\beta_1}\boxed{x_i}}}{1 + e^{\boxed{\beta_0} + \boxed{\beta_1}\boxed{x_i}}}$$

The goal of this function is to: take in the variable *x* and return a probability that *Y* is 1.

Note that there are only three variables on the right: X_i, B_0, B_1

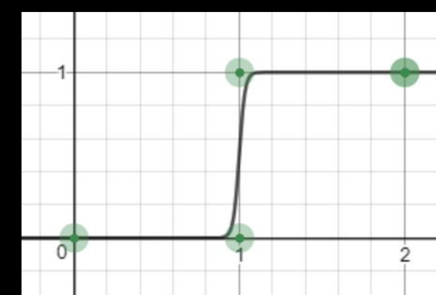
X is given. B_0 and B_1 must be learned.

Logistic Regression on a single feature (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$p_i \equiv P(Y_i = 1 | X_i = x) = \frac{e^{\boxed{\beta_0}+\boxed{\beta_1}x_i}}{1 + e^{\boxed{\beta_0}+\boxed{\beta_1}x_i}}$$

HOW? Essentially, try different B_0 and B_1 values until "best fit" to the training data (example X and Y).



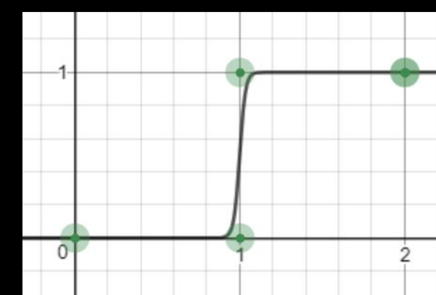
X is given. B_0 and B_1 must be **learned**.

"best fit" : whatever maximizes the likelihood function:

$$\boxed{L(\beta_0, \beta_1 | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}}$$

$$p_i \equiv P(Y_i = 1 | X_i = x) = \frac{e^{\boxed{\beta_0} + \boxed{\beta_1} x_i}}{1 + e^{\boxed{\beta_0} + \boxed{\beta_1} x_i}}$$

HOW? Essentially, try different B_0 and B_1 values until "best fit" to the training data (example X and Y).



X is given. B_0 and B_1 must be **learned**.

“best fit” : whatever maximizes the likelihood function:

$$\boxed{L(\beta_0, \beta_1 | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}}$$

“best fit” : whatever maximizes the *likelihood* function:

$$\boxed{L(\beta_0, \beta_1 | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}}$$

“best fit” : more efficient to maximize *log likelihood* :

“best fit” : whatever maximizes the *likelihood* function:

$$\boxed{L(\beta_0, \beta_1 | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}}$$

“best fit” : more efficient to maximize *log likelihood* :

$$\ell(\beta) = \sum_{i=1}^{N} y_i log\, p(x_i) + (1 - y_i) log\, (1 - p(x_i))$$

“best fit” : whatever maximizes the *likelihood* function:

$$\boxed{L(\beta_0, \beta_1 | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}}$$

“best fit” : more efficient to maximize *log likelihood* :

$$\ell(\beta) = \sum_{i=1}^{N} y_i log\, p(x_i) + (1 - y_i) log\, (1 - p(x_i))$$

“best fit” for neural networks: software designed to **minimize** rather than maximize (typically, normalized by N, the number of examples.)

From Likelihood to Cross Entropy Loss

Logistic Regression Likelihood: $L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

?

Final Cost Function: $J^{(t)} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j}^{(t)} log\ \hat{y}_{i,j}^{(t)}$ -- ”cross entropy error”

From Likelihood to Cross Entropy Loss

Logistic Regression Likelihood: $L(\beta_0, \beta_1, \ldots, \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

Log Likelihood: $\ell(\beta) = \sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p(x_i))$

Log Loss: $J(\beta) = -\frac{1}{N} \sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p)(x_i))$

Final Cost Function: $J^{(t)} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_{i,j}^{(t)} log\ \hat{y}_{i,j}^{(t)}$ -- "cross entropy error"

From Likelihood to Cross Entropy Loss

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
          #where did this come from?
```

Logistic Regression Likelihood: $L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

Log Likelihood: $\ell(\beta) = \sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p(x_i))$

Log Loss: $J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p)(x_i))$

Cross-Entropy Cost: $J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ p(x_{i,j})$ (a “multiclass” log loss)

V is classes

Final Cost Function: $J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j} log\ \hat{y}_{i,j}$ -- ”cross entropy error”

From Likelihood to Cross Entropy Loss

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
            #where did this come from?
```

Logistic Regression Likelihood: $L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

Log Likelihood: $\ell(\beta) = \sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p(x_i))$

Log Loss: $J(\beta) = -\frac{1}{N} \sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p)(x_i))$

Cross-Entropy Cost: $J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_i log\ p(x_{i,j})$ (a “multiclass” log loss)

V is classes

Final Cost Function: $J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_{i,j} log\ \hat{y}_{i,j}$ -- ”cross entropy error”

How to train in torch

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
```

$$\hat{y}_{i,j} = P(Y_i = 1 | X_i = x) = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}$$

$$= \frac{1}{1+e^{-\beta x}}$$

Final Cost Function: $J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_{i,j} log\ \hat{y}_{i,j}$ -- ”cross entropy error”

How to train in torch

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
```

$$\hat{y}_{i,j} = P(Y_i = 1 | X_i = x) = \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}$$

$$= \frac{1}{1+e^{-\beta x}}$$

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i \, log\left(\frac{1}{1+e^{-\beta x}}\right)$$

Final Cost Function: $J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j} \, log \, \hat{y}_{i,j}$ -- "cross entropy error"

How to train in torch

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
```

As a Graph?

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\left(\frac{1}{1+e^{-\beta x}}\right)$$

How to train in torch

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Final Cost Function: $J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j} log\ \hat{y}_{i,j}$ -- ”cross entropy error”

How to train in torch

```
loss  = torch.mean(-torch.sum(y*torch.log(y_pred))
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

<u>To Optimize Betas</u> (all weights/parameters within the neural net):

Stochastic Gradient Descent (SGD)

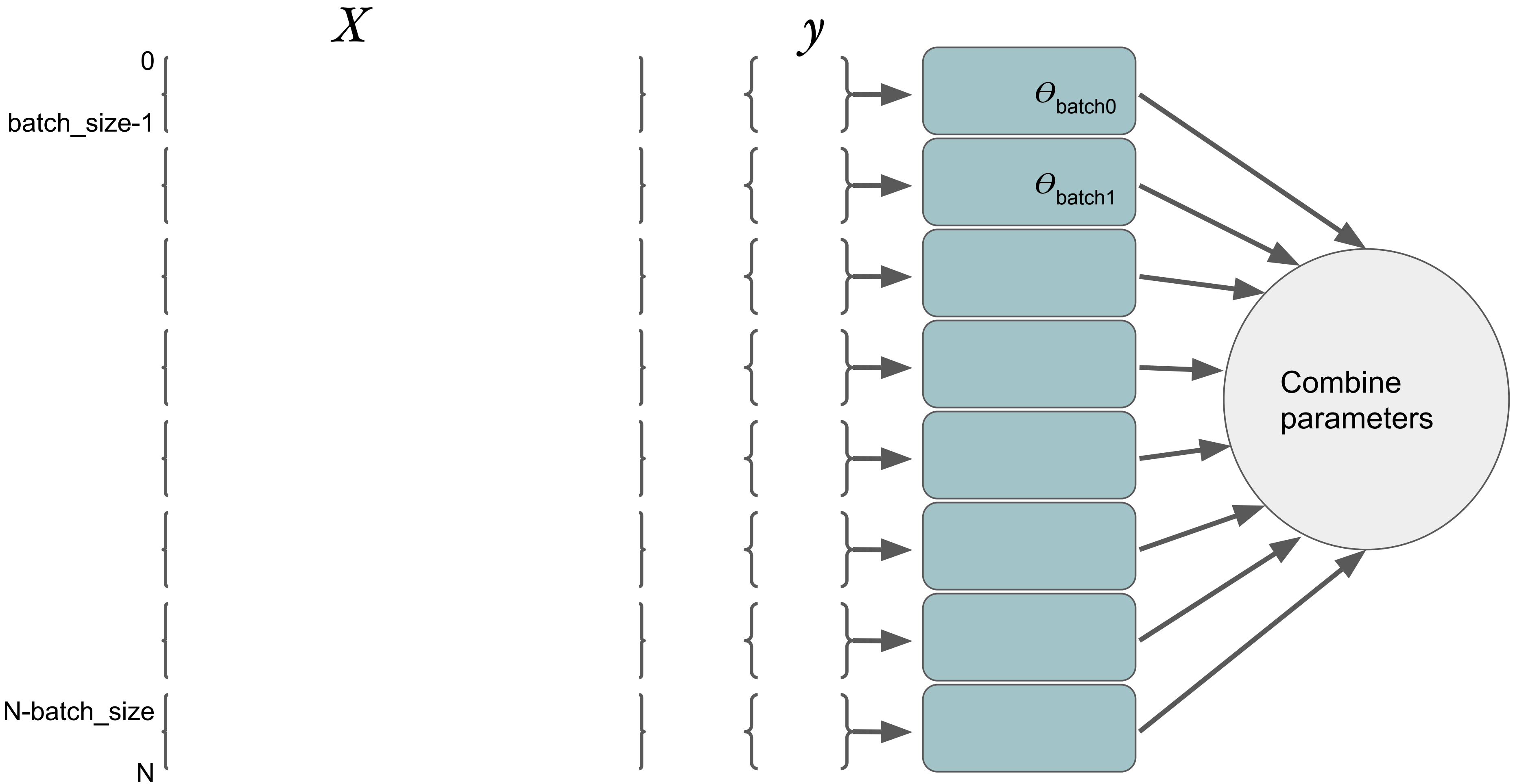
-- optimize over one sample each iteration

Mini-Batch SDG:

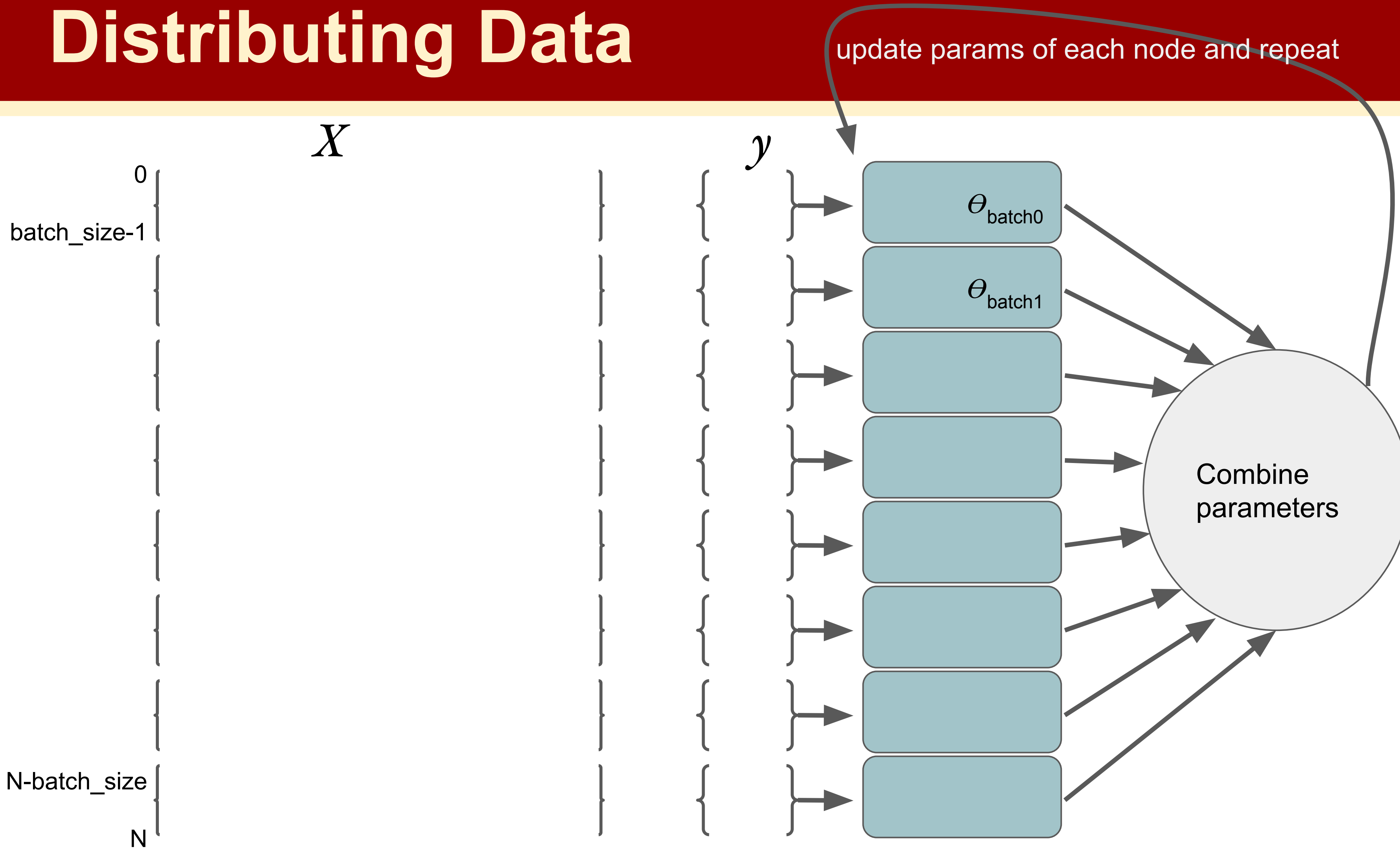
--optimize over *b* samples each iteration

Final Cost Function: $J \quad = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j} log\ \hat{y}_{i,j}$ -- ”cross entropy error”

Distributing Data



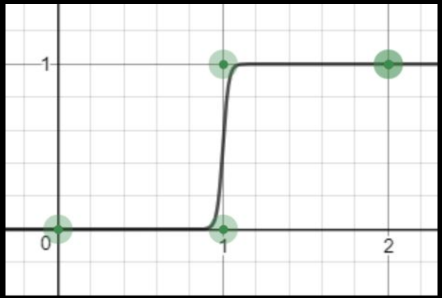
Distributing Data



X can be multiple features

Often we want to make a classification based on multiple features:

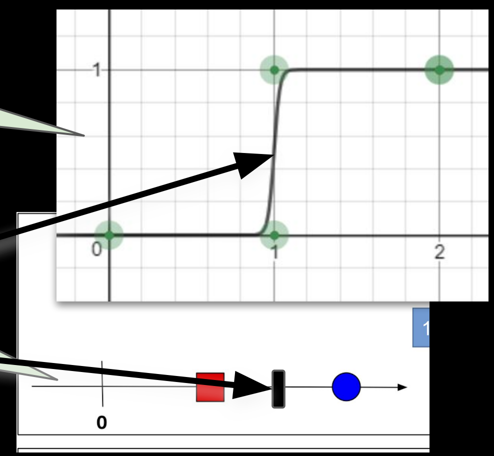
- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}



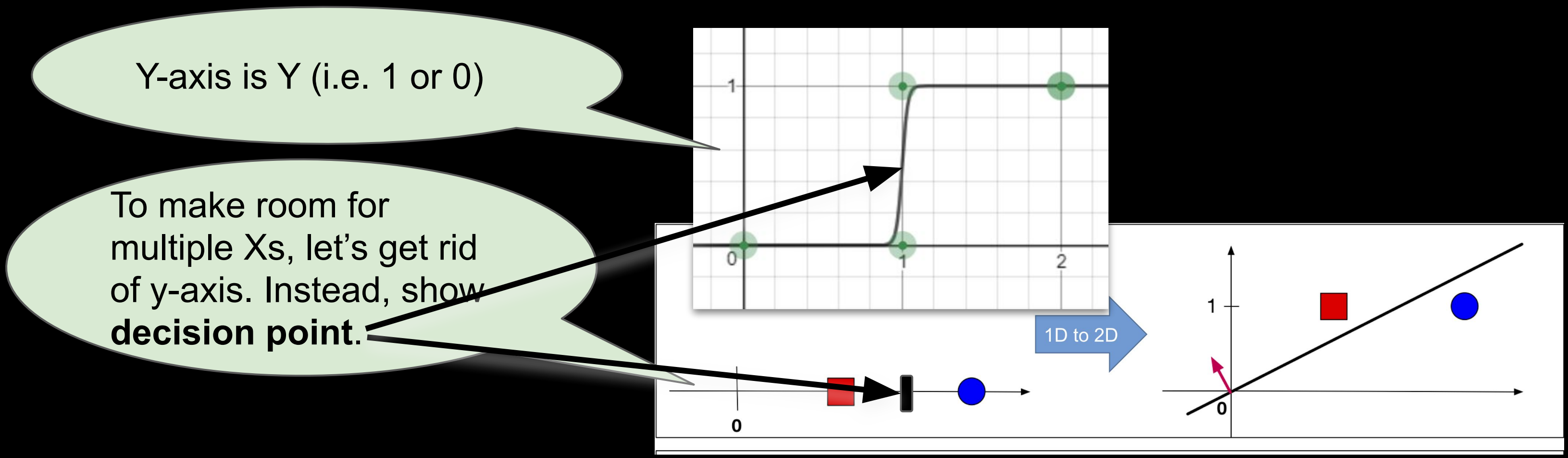
Logistic Regression is still "linear modeling"

Y-axis is Y (i.e. 1 or 0)

To make room for multiple Xs, let's get rid of y-axis. Instead, show **decision point**.



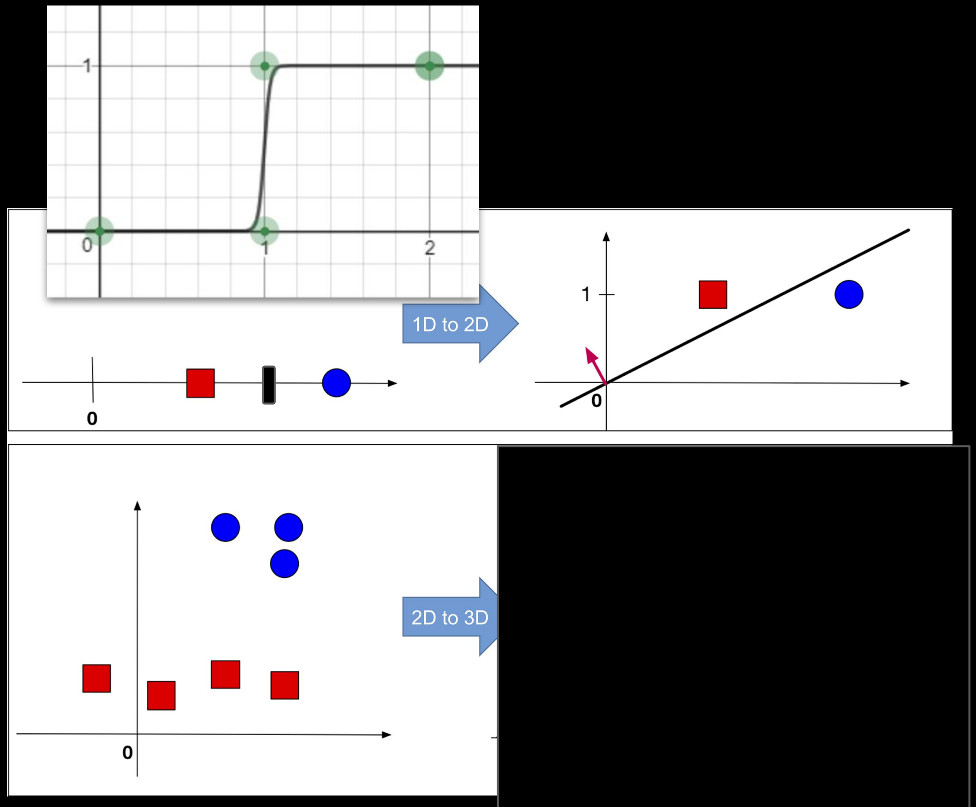
Logistic Regression is still "linear modeling"



1 feature

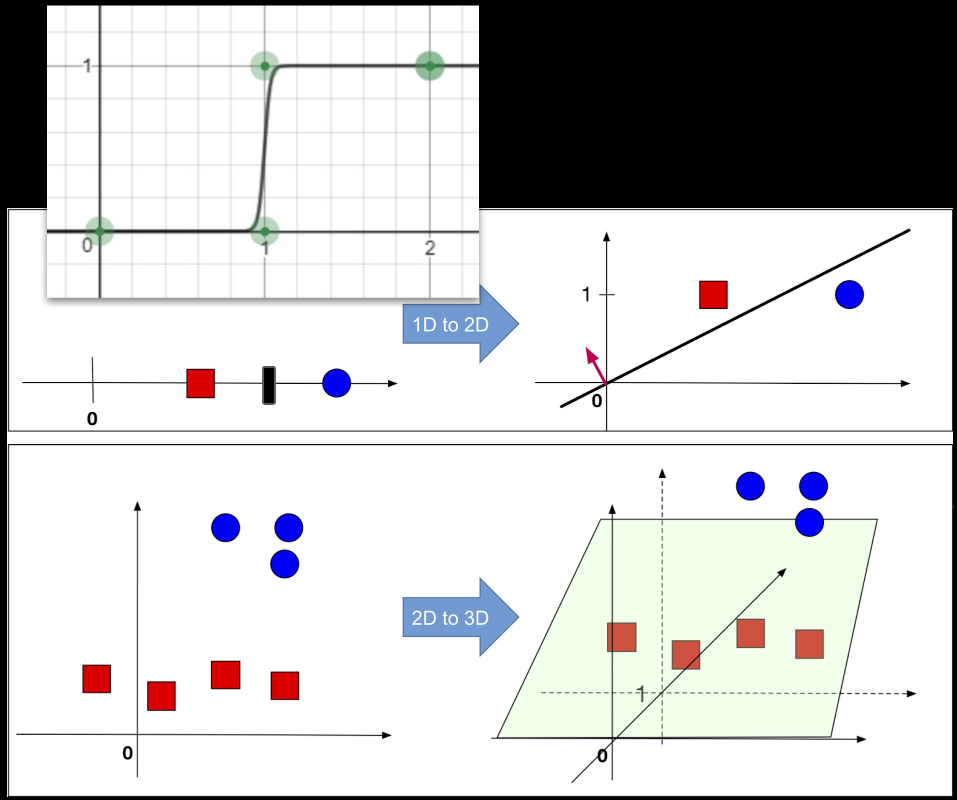
2 features

Logistic Regression is still "linear modeling"

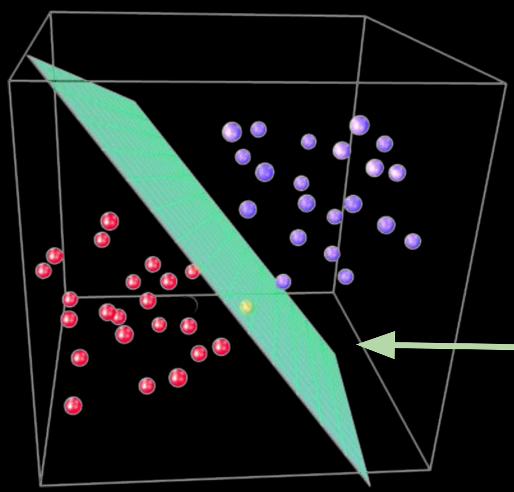


Logistic Regression is still "linear modeling"

- Because we're still learning a linear "hyperplane"



X can be multiple features



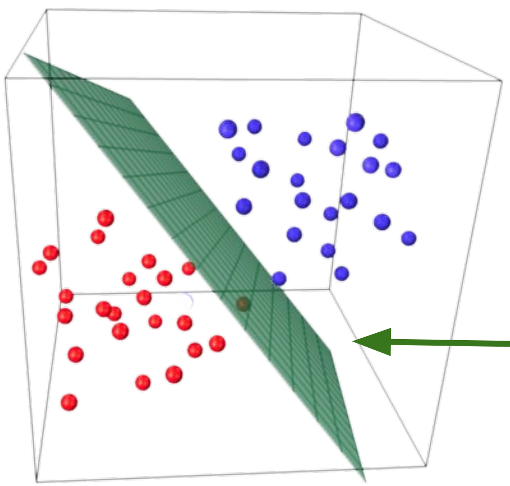
We're learning a linear (i.e. flat) *separating hyperplane*, but fitting it to a *logit* outcome.

(https://www.linkedin.com/pulse/predicting-outcomes-probabilities-logistic-regression-konstantinidis/)

Logistic Regression

What if $Y_i \in \{0, 1\}$? (i.e. we want "classification")

$$p_i \equiv p_i(\beta) \equiv \mathbf{P}(Y_i = 1 | X = x) = \frac{e^{\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij}}}{1 + e^{\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij}}}$$



$$logit(p_i) = log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \sum_{j=1}^{m} \boxed{\beta_j x_{ij}}$$

We're still learning a linear *separating hyperplane*, but fitting it to a *logit* outcome.

(https://www.linkedin.com/pulse/predicting-outcomes-probabilities-logistic-regression-konstantinidis/)

Uses of Regressions

1. Testing the relationship between variables given other variables. β is an "effect size" -- a score for the magnitude of the relationship; can be tested for significance.

2. Building a predictive model that generalizes to new data. $\hat{Y}$ is an estimate value of Y given X.

Uses of Regressions

1. Testing the relationship between variables given other variables. β is an "effect size" -- a score for the magnitude of the relationship; can be tested for significance.

2. Building a predictive model that generalizes to new data. $\hat{Y}$ is an estimate value of Y given X.

X_1 X_2 X_3 X_4 X_5 X_6
X_7 X_8 X_9 X_{10} X_{11} X_{12} $\rightarrow$ $\hat{Y}$
X_{13} X_{14} X_{15} ... X_m

Task: Determine a function, f (or parameters to a function) such that $f(X) = Y$

Uses of Regressions

1. Testing the relationship between variables given other variables. β is an "effect size" -- a score for the magnitude of the relationship; can be tested for significance.

2. Building a predictive model that generalizes to new data. $\hat{Y}$ is an estimate value of Y given X.
   However, when $|features|$ $close\ to\ number\ of\ observatations$ then the model might "overfit".

   -> Regularized linear regression (a ML technique)

Regressions

- Linear Regression
- Pearson Product-Moment Correlation
- Multiple Linear Regression
- **(Multiple) Logistic Regression**
- Ridge Regularized Linear/Logistic Regression

Regressions

- Linear Regression
- Pearson Product-Moment Correlation
- Multiple Linear Regression
- (Multiple) Logistic Regression
- **Ridge Regularized Linear/Logistic Regression**

Supervised Statistical Learning

X_1 X_2 X_3 X_4 X_5 X_6

X_7 X_8 X_9 X_{10} X_{11} X_{12} $\rightarrow$ Y

X_{13} X_{14} X_{15} … X_m

Task: Determine a function, f (or parameters to a function) such that $f(X) = Y$

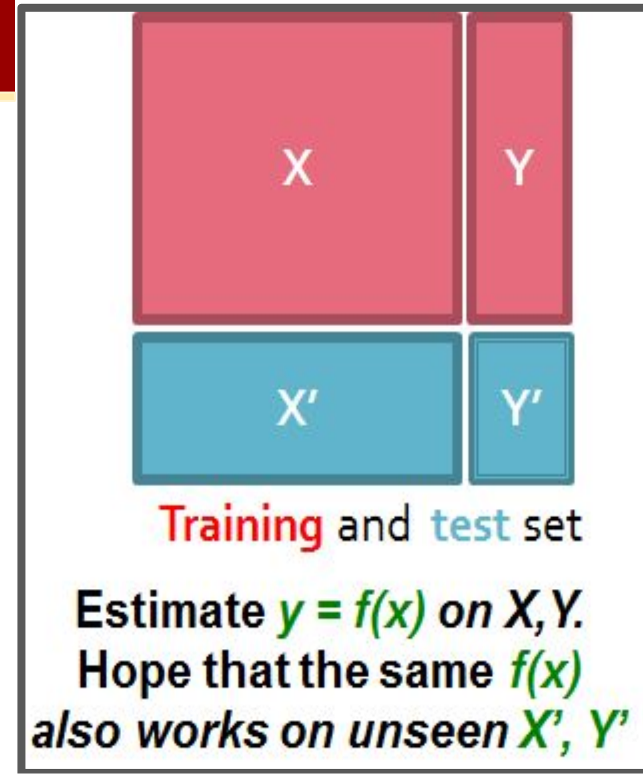
Supervised Learning

$X_1 \quad X_2 \quad X_3 \quad Y$

$X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6$

$X_7 \quad X_8 \quad X_9 \quad X_{10} \quad X_{11} \quad X_{12}$

$X_{13} \quad X_{14} \quad X_{15} \quad \ldots \quad X_m$



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Task: Determine a function, f (or parameters to a function) such that $f(X) = Y$

Logistic Regression - Regularization

$$X \quad = \quad Y$$

0.5	0	0.6	1	0	0.25	1
0	0.5	0.3	0	0	0	1
0	0	1	1	1	0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

Logistic Regression - Regularization

$$X \quad = \quad Y$$

0.5	0	0.6	1	0	0.25	1
0	0.5	0.3	0	0	0	1
0	0	1	1	1	0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

Logistic Regression - Regularization

X $=$ Y

0.5	0	0.6	1	0	0.25	1
0	0.5	0.3	0	0	0	1
0	0	1	1	1	0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

$$1.2 + -63 * x_1 + 179 * x_2 + 71 * x_3 + 18 * x_4 + -59 * x_5 + 19 * x_6 = logit(Y)$$

Logistic Regression - Regularization

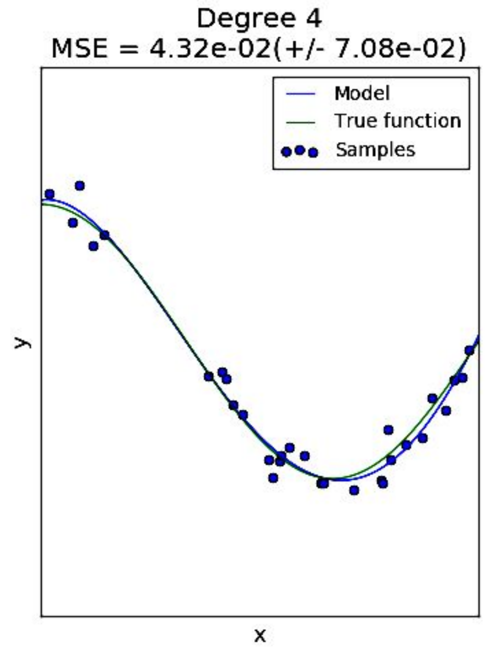
$$X \qquad = \qquad Y$$

							Y
colab	0.5	0	0.6	1	0	0.25	1
	0	0.5	0.3	0	[illegible]	0	1
	0	0	[illegible]	[illegible]	[illegible]	0.5	0
	0	0	0	0	1	1	0
	0.25	1	1.25	1	0.1	2	1

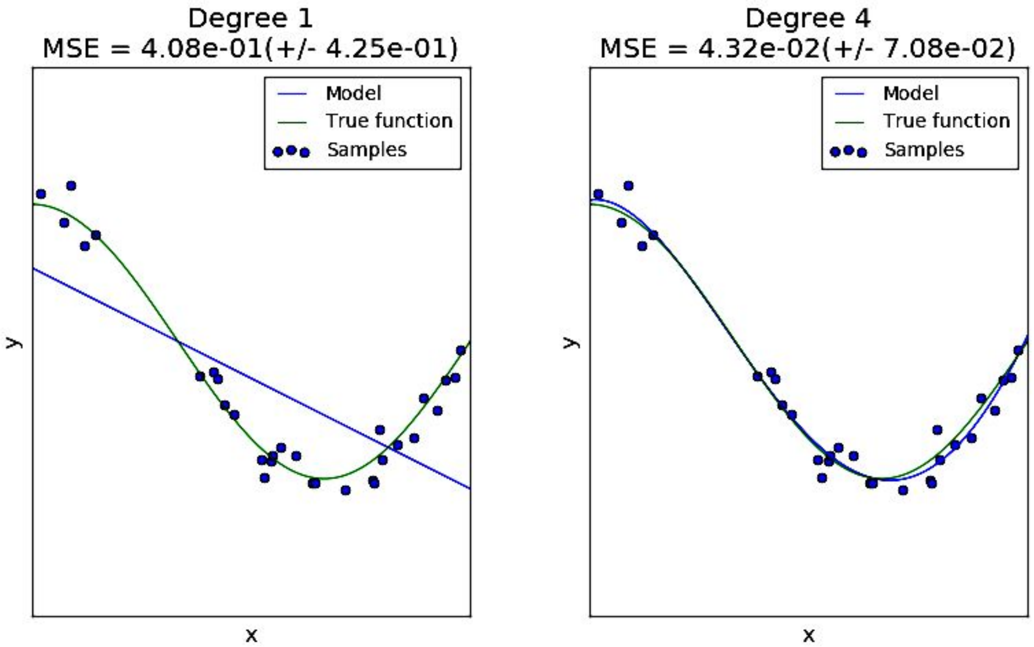
“overfitting”

$$1.2 + -63*x_1 + 179*x_2 + 71*x_3 + 18*x_4 + -59*x_5 + 19*x_6 = logit(Y)$$

Overfitting (1-d example)



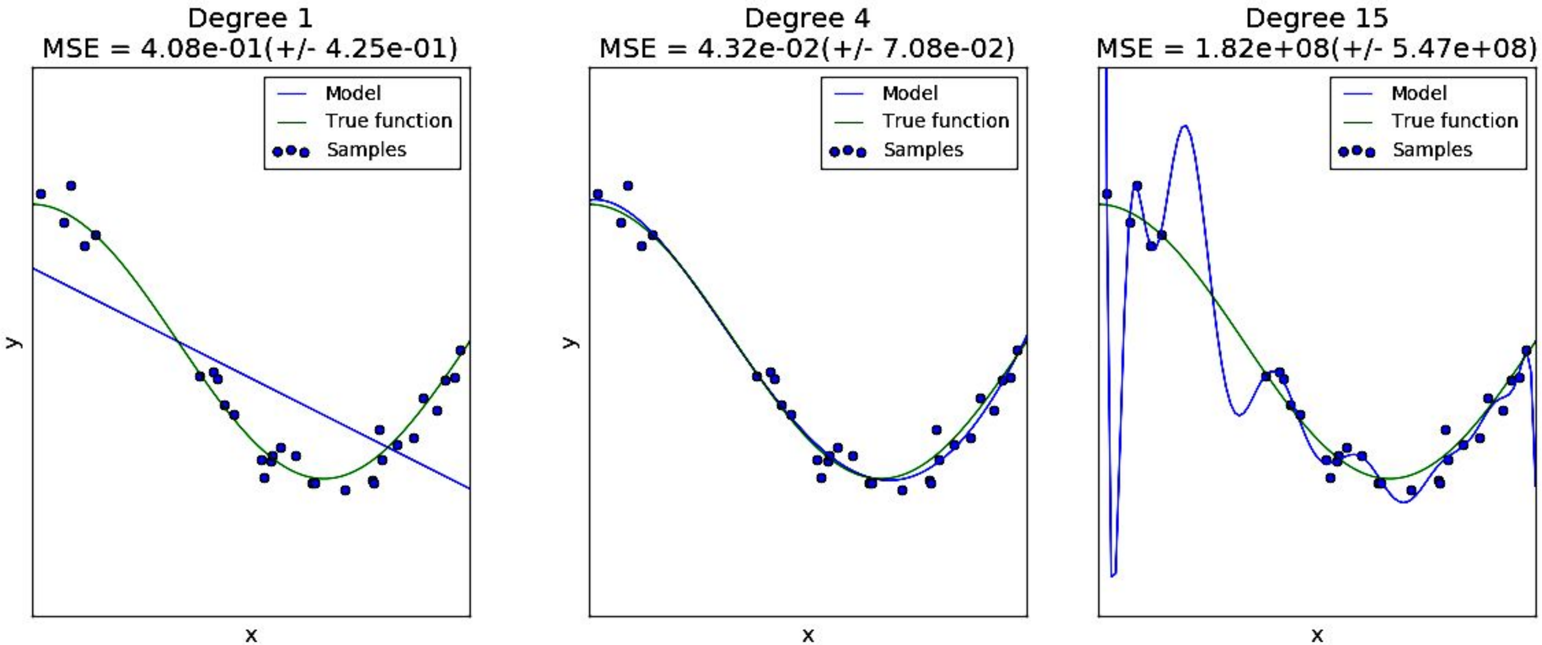
Overfitting (1-d example)



Underfit

(image credit: Scikit-learn; in practice data are rarely this clear)

Overfitting (1-d example)



Underfit

Overfit

(image credit: Scikit-learn; in practice data are rarely this clear)

Overfitting (multidimenstional example)

$$X \quad = \quad Y$$

0.5	0	0.6	1	0	0.25	1
0	0.5	0.3	0		0	1
0	0				0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

“overfitting”

$$1.2 + -63 \cdot x_1 + 179 \cdot x_2 + 71 \cdot x_3 + 18 \cdot x_4 + -59 \cdot x_5 + 19 \cdot x_6 = logit(Y)$$

Overfitting (multidimenstional example)

$$X \qquad\qquad\qquad\qquad = \qquad Y$$

0.5	0	0.6	[illegible]	[illegible]	[illegible]	1
0	0.5	[illegible]	[illegible]	[illegible]	[illegible]	1
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0
0.25	[illegible]	[illegible]	[illegible]	0.1	2	1

1.2 +	$-63 * x_1$ +	17[illegible]x_2 +	$71 * x_3$ +	$18 * x_4$ +	$-59 * x_5$ +	$19 * x_6$ $= logit(Y)$

“overfitting”: generally due to trying to fit too many features given the number of observations.

Overfitting (multidimenstional example)

X

x_1	x_2
0.5	0
0	0.5
0	0
0	0
0.25	1

What if only 2 predictors?

Y

1
1
0
0
1

Overfitting (multidimenstional example)

X $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ Y

x_1	x_2
0.5	0
0	0.5
0	0
0	0
0.25	1

Y
1
1
0
0
1

What if only 2 predictors?

A: better fit

$0 + 2*x_1 + 2*x_2 = logit(Y)$

Regularization: stepwise feature selection

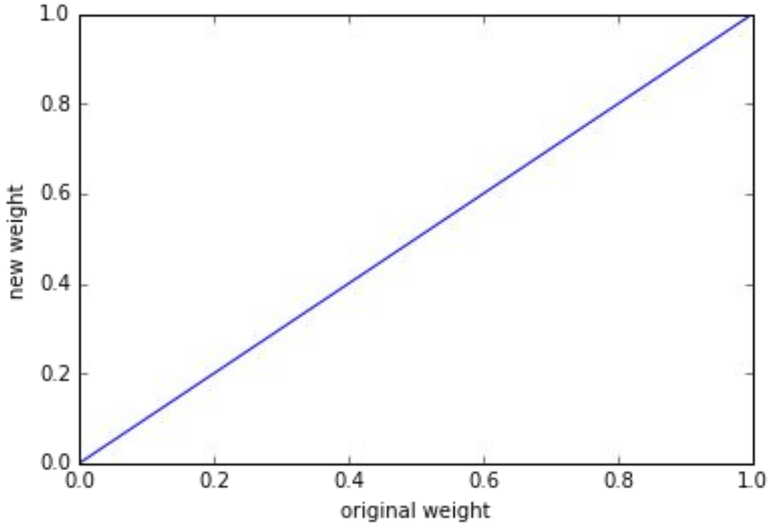
(bad) solution to overfit problem

Use less features based on Forward Stepwise Selection:

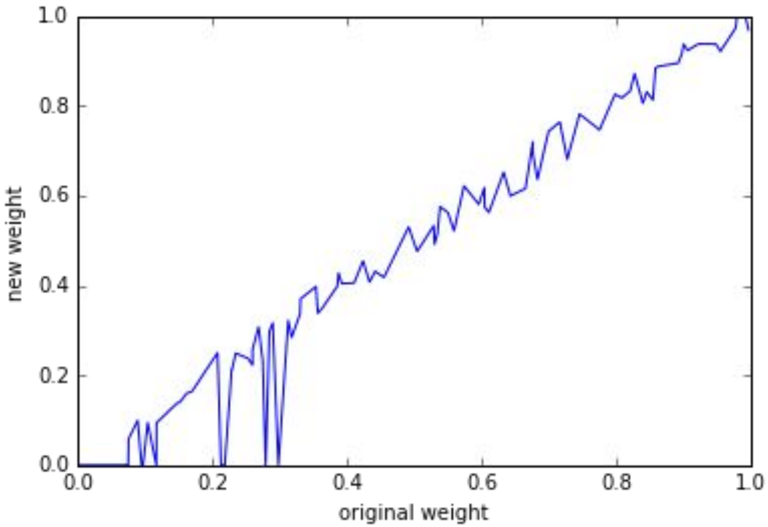
- start with current_model just has the intercept (mean)
  remaining_predictors = all_predictors

```
 for i in range(k):
   #find best p to add to current_model:
   for p in remaining_prepdictors
      refit current_model with p
       #add best p, based on RSS_p to current_model
   #remove p from remaining predictors
```

Regularization: shrinkage



No selection (weight=beta)



forward stepwise

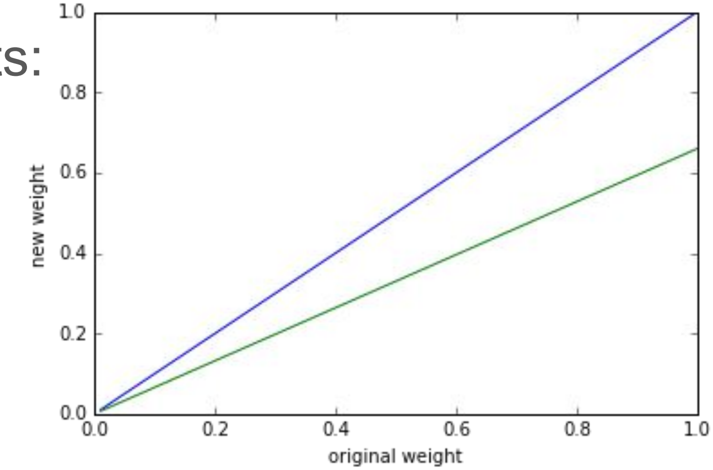
Why just keep or discard features?

Regularization: L2 (Ridge) Penalized Loss

Idea: Impose a penalty on size of weights:

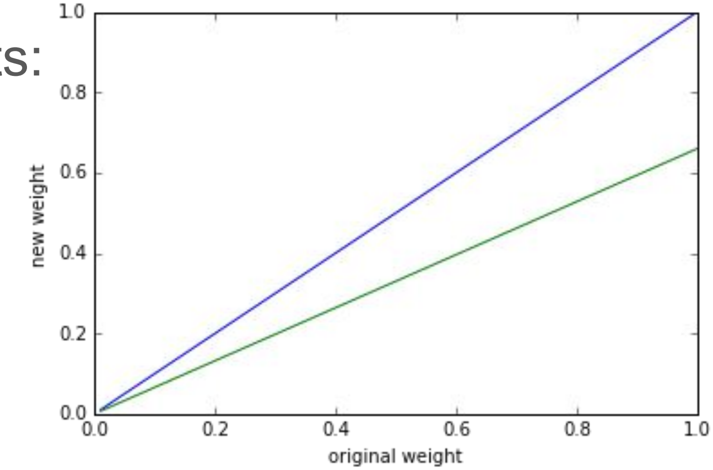
Ordinary least squares objective:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2\}$$



Regularization: L2 (Ridge) Penalized Loss

Idea: Impose a penalty on size of weights:



Ordinary least squares objective:

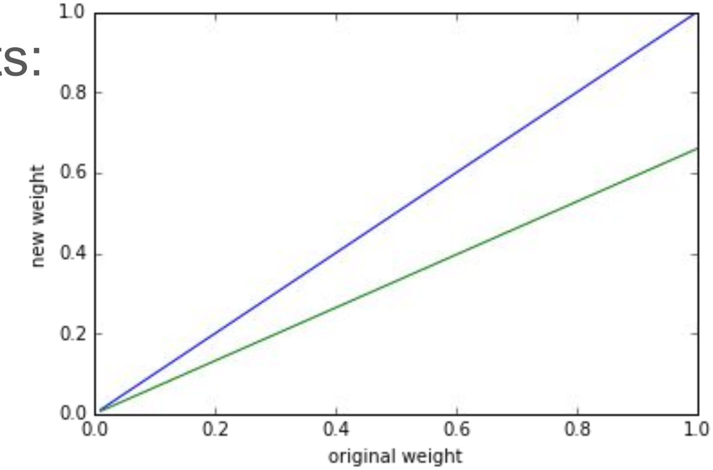
$$\hat{\beta} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2\}$$

Ridge regression:

$$\hat{\beta}^{ridge} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2\}$$

Regularization: L2 (Ridge) Penalized Loss

Idea: Impose a penalty on size of weights:



Ordinary least squares objective:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2\}$$

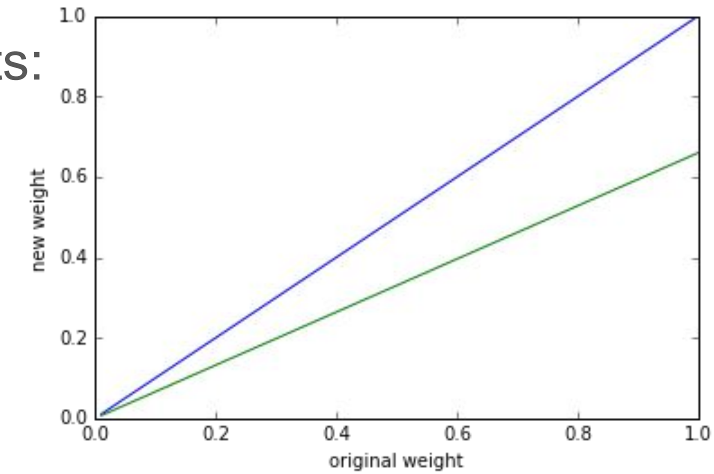
Ridge regression:

$$\hat{\beta}^{ridge} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{m}\beta_j^2\}$$

$$\lambda\sum_{j=1}^{m}\beta_j^2 \longrightarrow \lambda||\beta||_2^2$$

Regularization: L2 (Ridge) Penalized Loss

Idea: Impose a penalty on size of weights:



Ordinary least squares objective:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2\}$$

Ridge regression:

$$\hat{\beta}^{ridge} = argmin_{\beta}\{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2\}$$

$$\lambda||\beta||_2^2$$

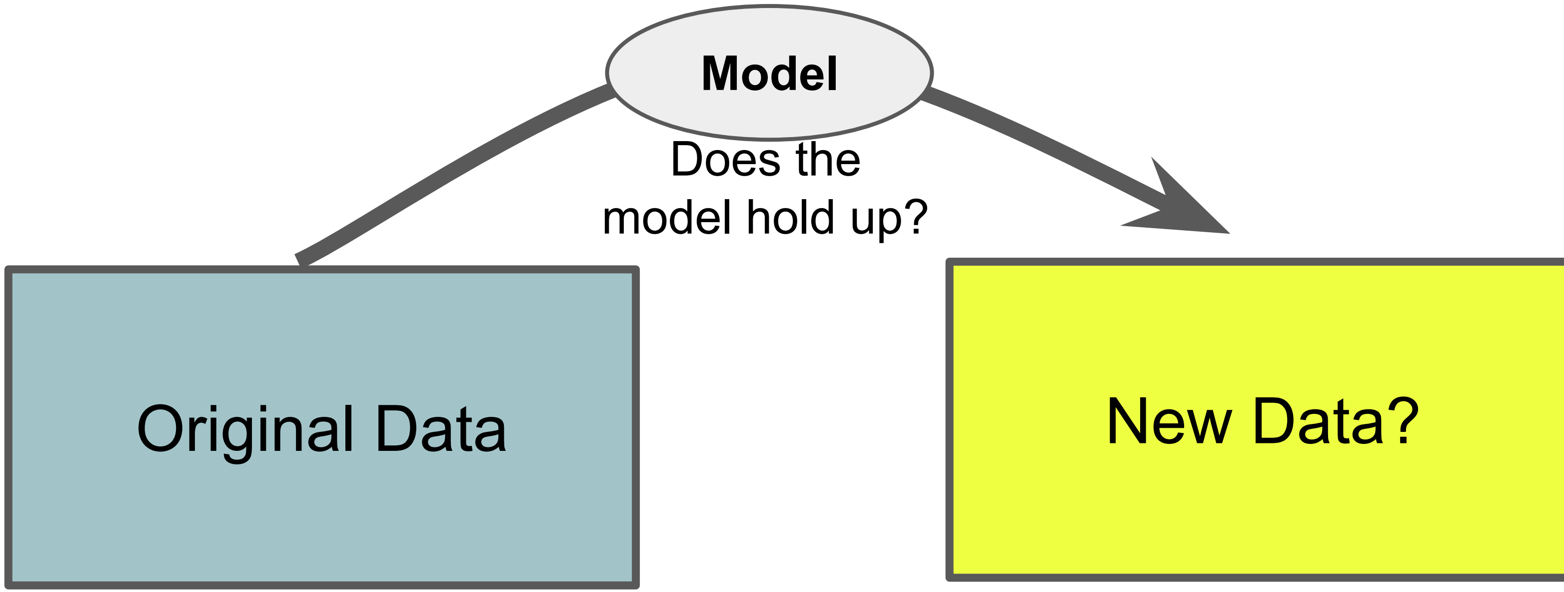
In Matrix Form:

$$\mathrm{RSS}(\lambda) = (y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta$$

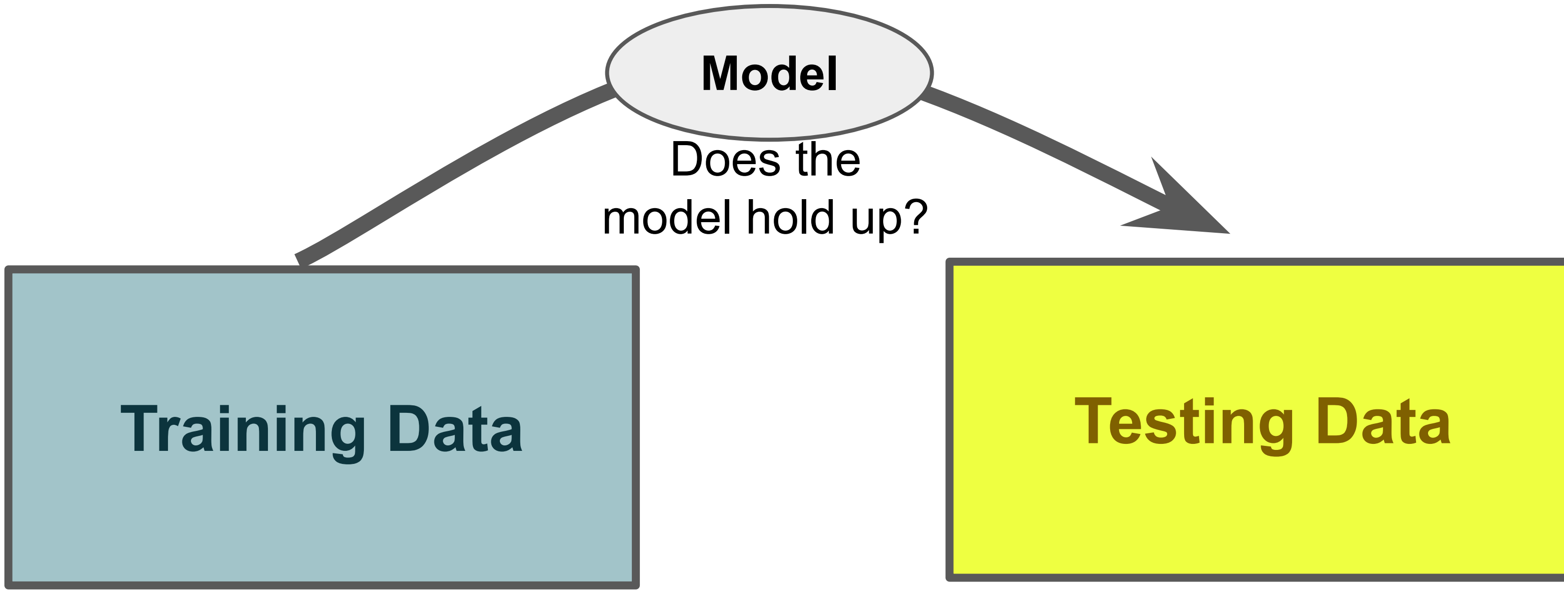
$$\hat{\beta}^{ridge} = (X^TX + \lambda I)^{-1}X^Ty$$

I: m x m identity matrix

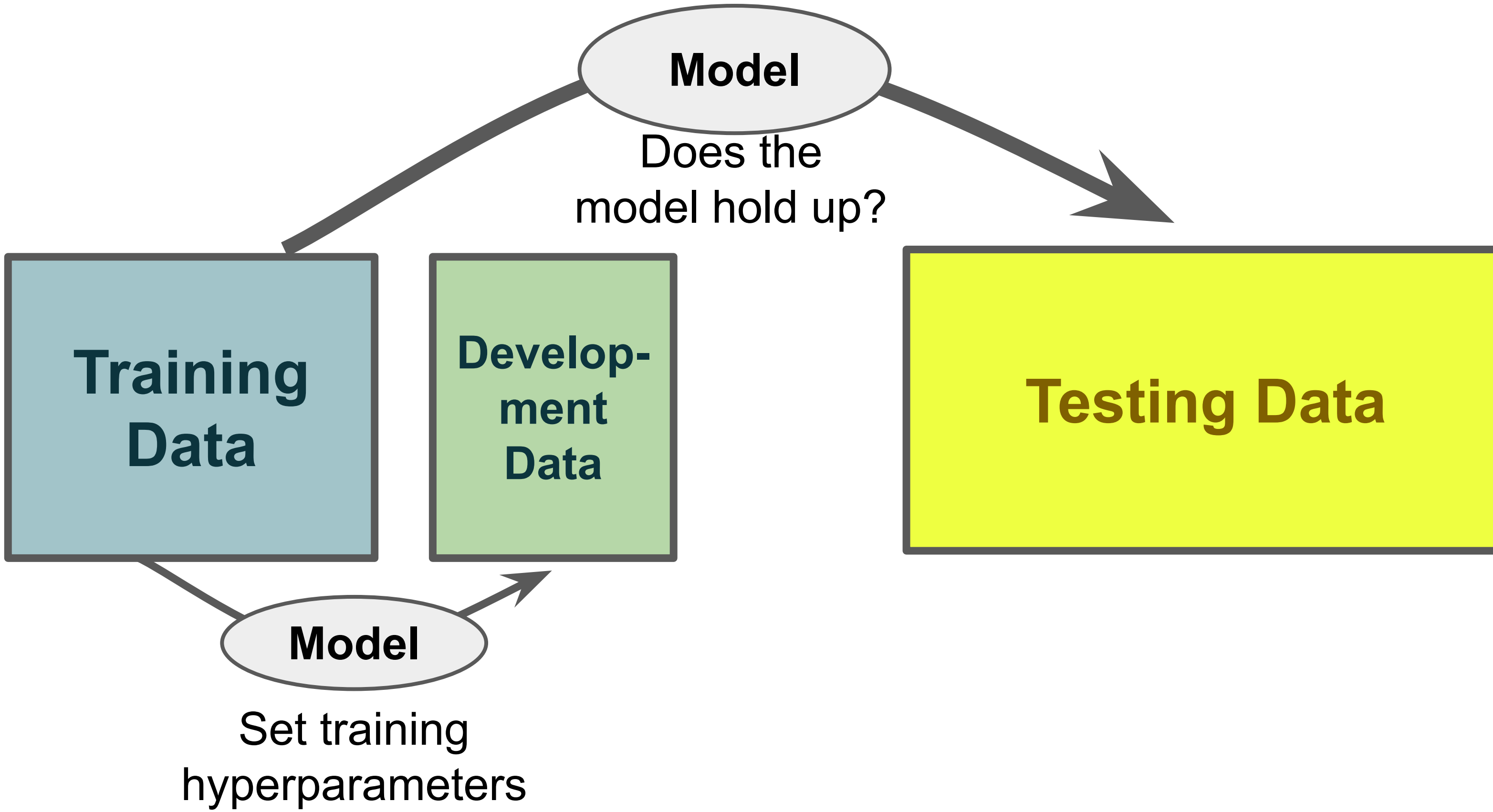
Common Goal: Generalize to new data



Common Goal: Generalize to new data

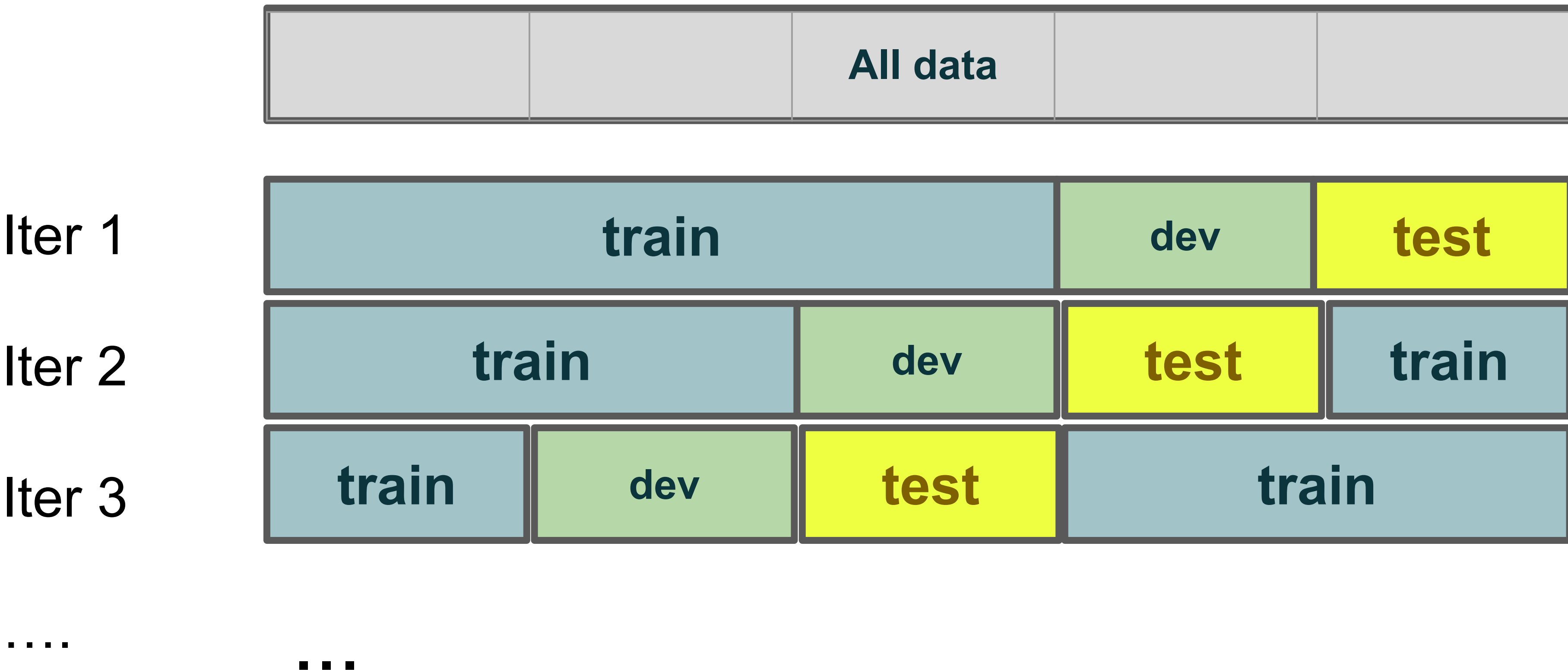


ML: GOAL


Model
Does the model hold up?
Training Data
Develop-ment Data
Testing Data
Model
Set training hyperparameters

N-Fold Cross Validation

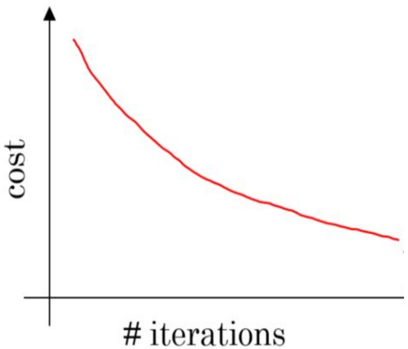
Goal: Decent estimate of model accuracy



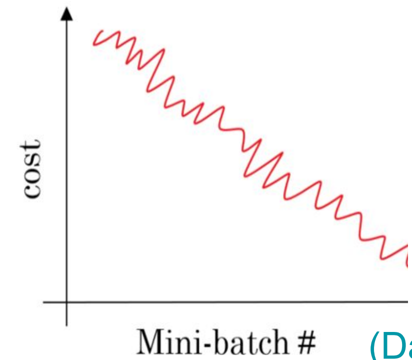
Useful Plots: Prediction

Learning Curve: for plotting error from gradient descent.

for a model with convex optimization (i.e. linear regression)

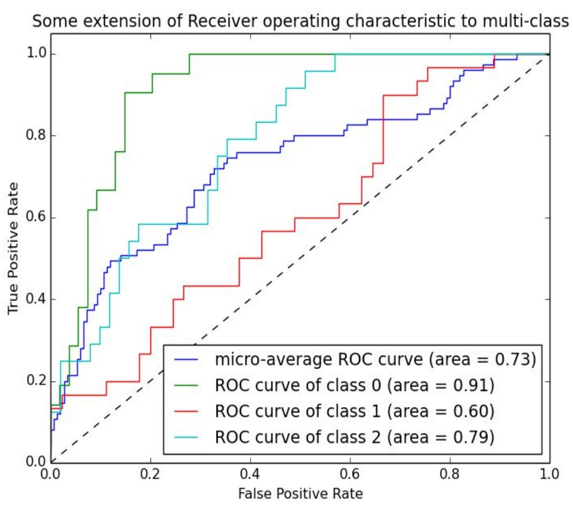


for a model with non-convex optimization (i.e. most deep learning)

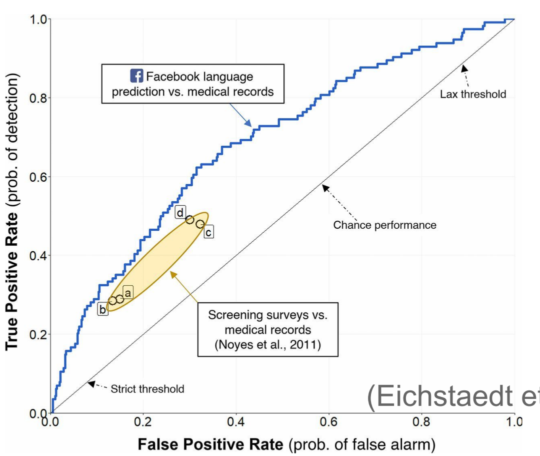


(Dabura, 2017)

ROC Plot: for visualizing true-positive to false-positive rates (used for AUC metric)



(PLOT_ROC)

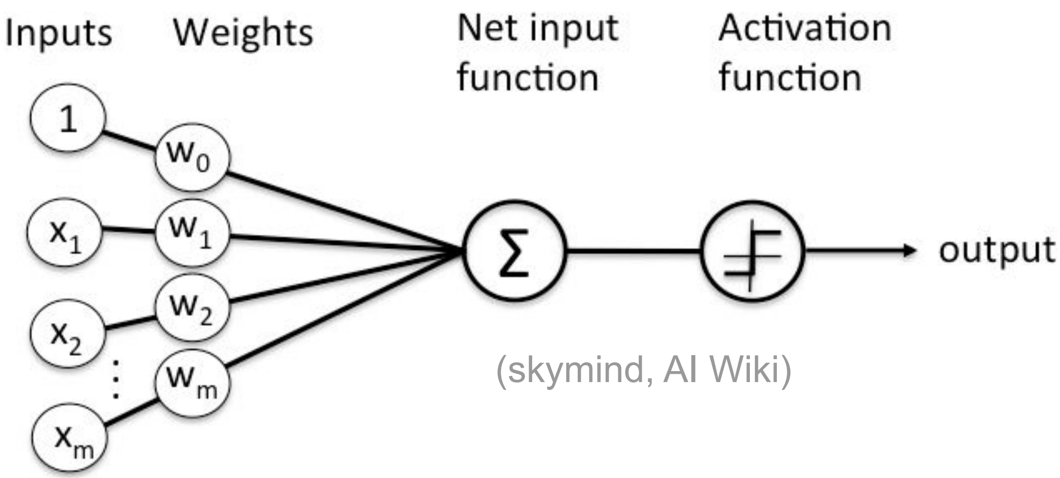


(Eichstaedt et al., 2018)

From Linear Models to Neural Nets

Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$

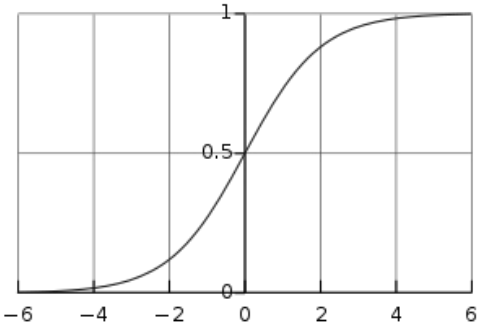


(skymind, AI Wiki)

Common Activation Functions

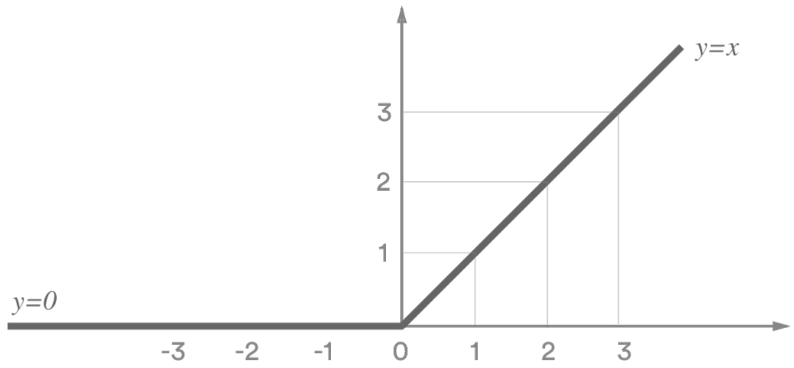
$z = wX$

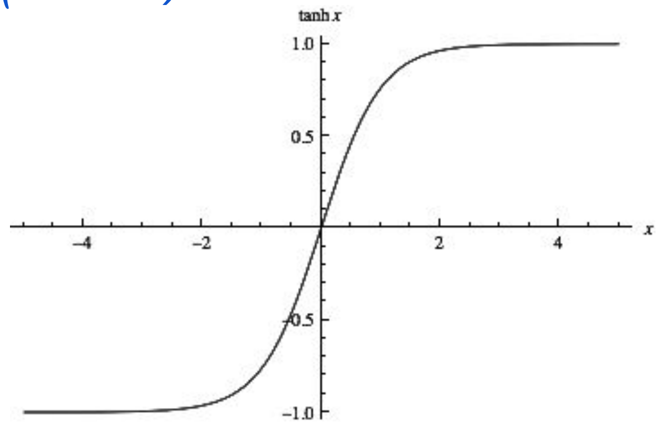
Logistic: $\sigma(z) = 1 / (1 + e^{-z})$



Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$

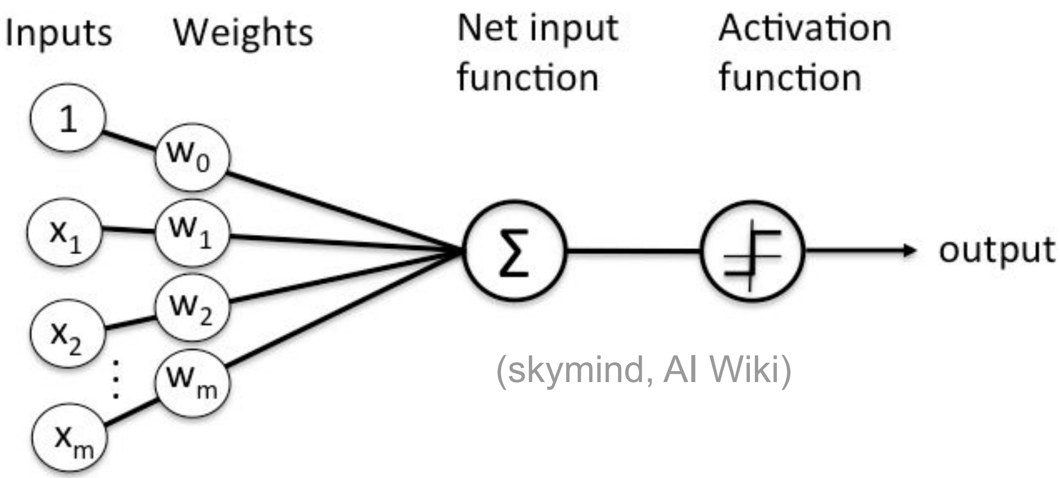




From Linear Models to Neural Nets

Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$

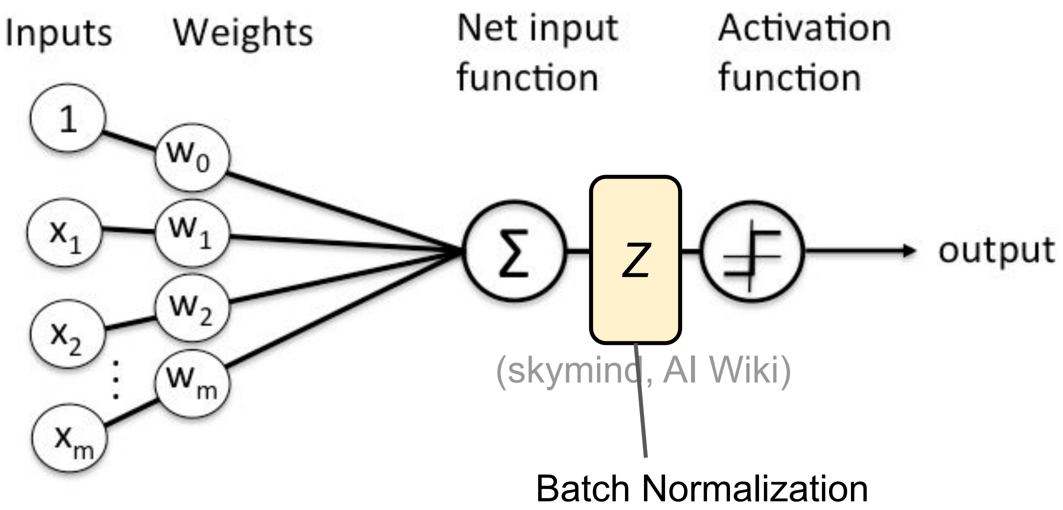


(skymind, AI Wiki)

From Linear Models to Neural Nets

Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$



(skymind, AI Wiki)

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

Batch Normalization

This is just standardizing!
(but within the current batch of observations)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

Batch Normalization

This is just standardizing!
(but within the current batch of observations)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

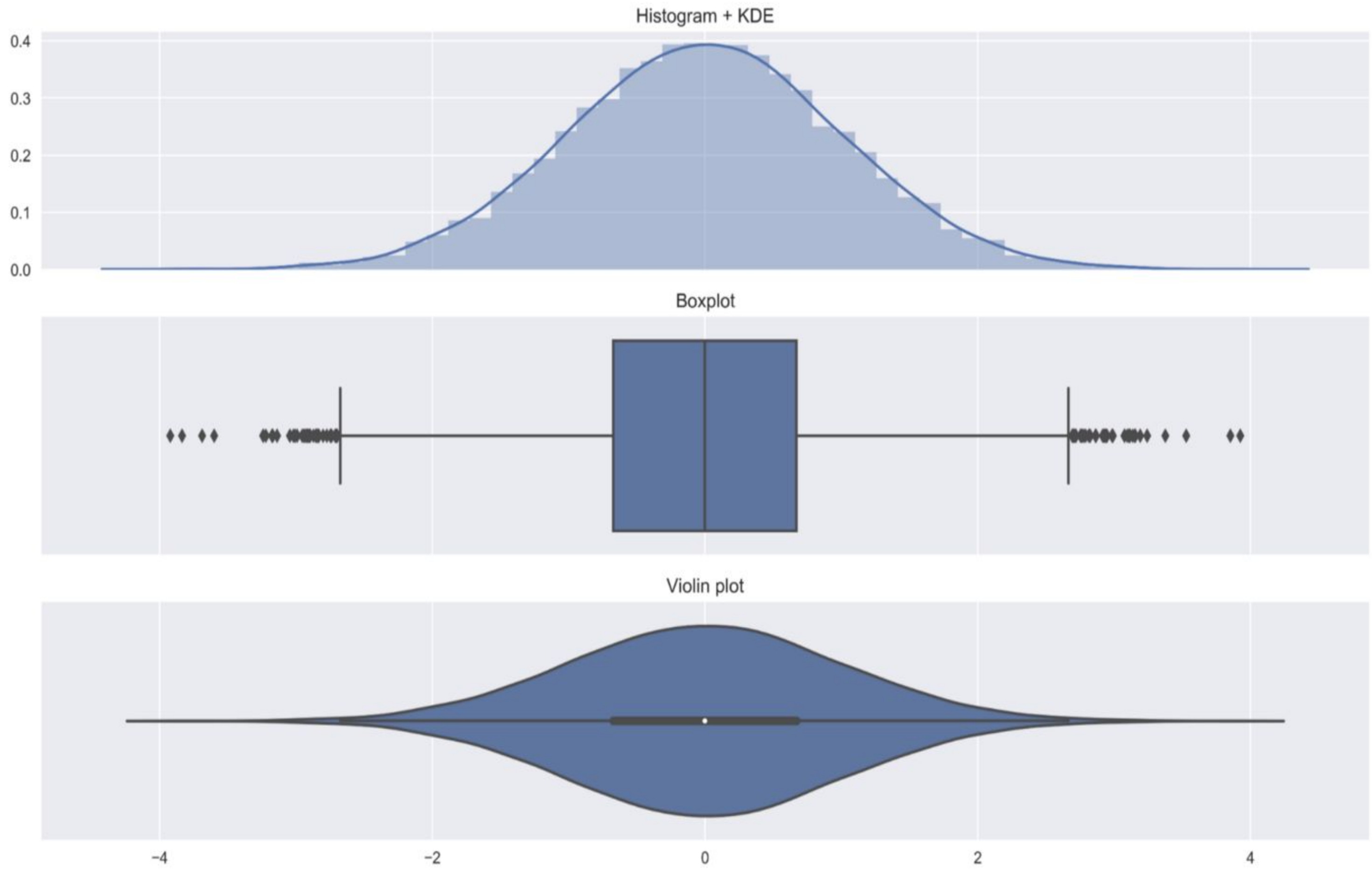
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

Why?

- Empirically, it works!
- Conceptually, generally good for weight optimization to keep data within a reasonable range (dividing by sigma) and such that positive weights move it up and negative down (centering).
- Small effect: When done over mini-batches, adds regularization due to differences between batches.

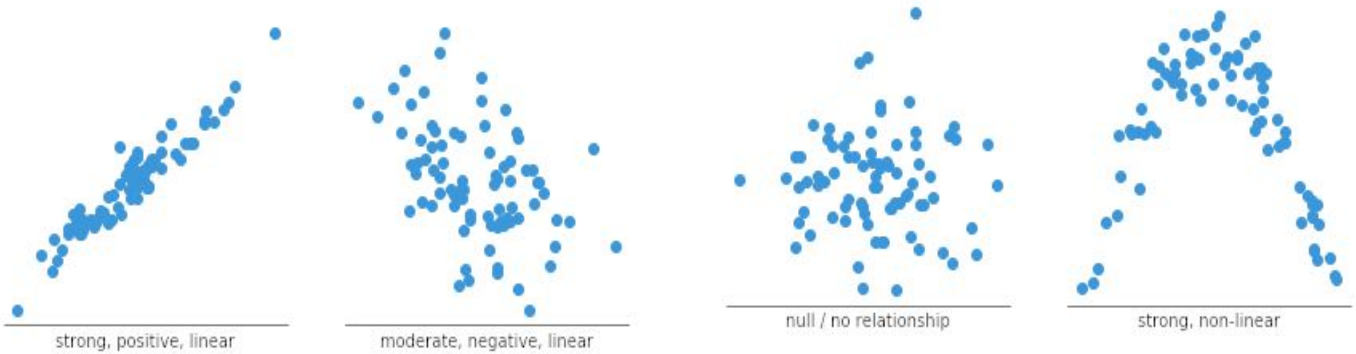
Useful Plots: For distributions



(Lewinson, 2019)

Useful Plots: Correlation

Scatter Plot: for two variables expected to be associated (with optional regression line)



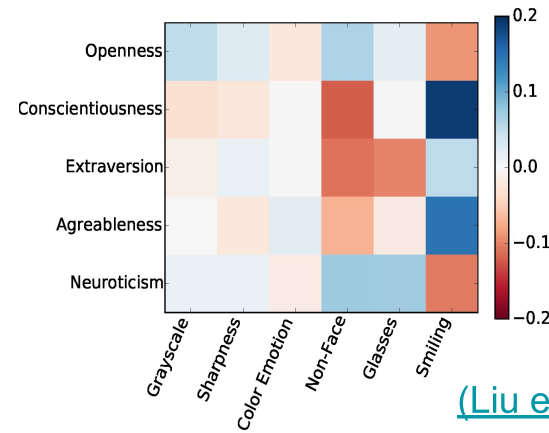
(Chartio)

Correlation Matrix: for comparing associations between many variables (use Bonferroni correction if hyp testing)

	FriendSize	Intelligence Quotient	Income	Sat W/ Life	Depression
F1	0.03	0.04	0.12	0.02	-0.1
F2	0.04	-0.26	-0.19	-0.09	0.11
F3	-0.07	-0.13	0.02	-0.02	-0.02
F4	-0.03	0.27	-0.08	-0.12	0.11
F5	-0.01	0.23	0.29	0.07	-0.21

Fig 3. Individual factor correlations with outcomes. Note how F4 which captures the use of swear words negatively correlates with Satisfaction with Life (SWL).

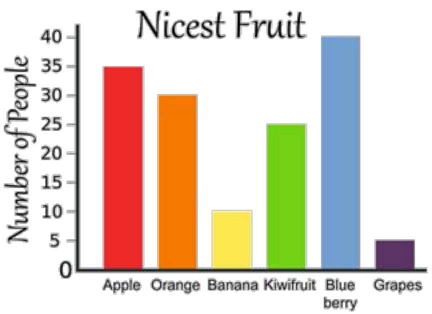
https://doi.org/10.1371/journal.pone.0201703.g003



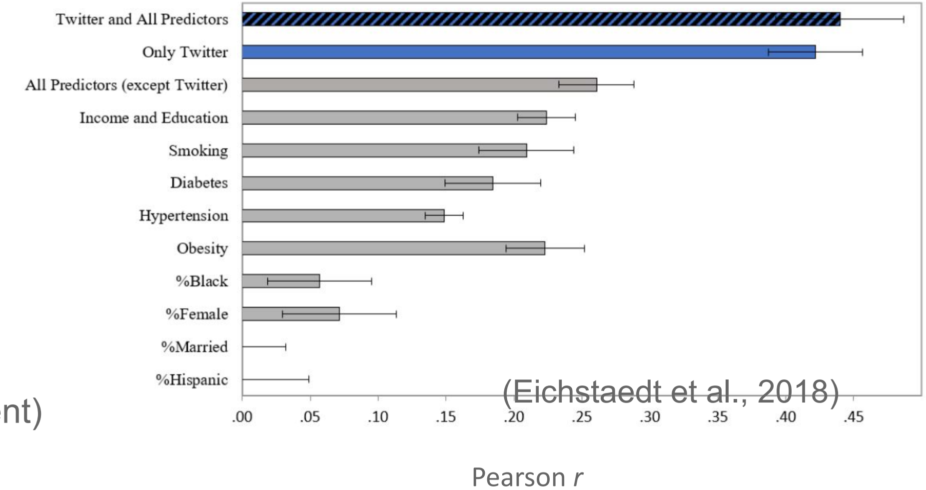
[(Liu et al., 2016)]

Useful Plots: Any Values

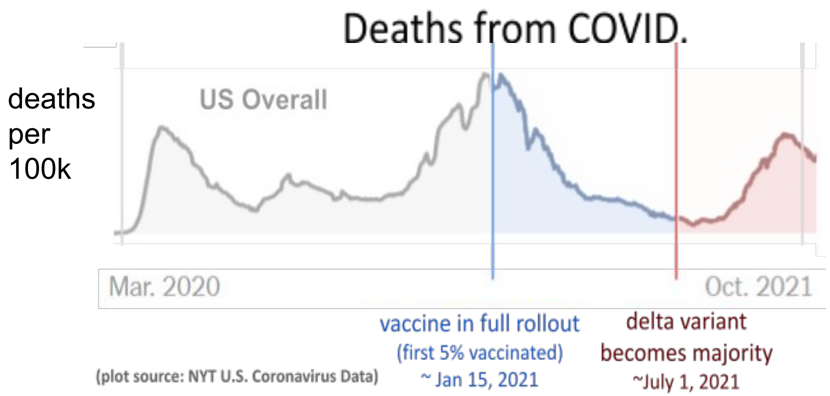
Bar Plot: To visually compare values under different selections/conditions.



(Science sEDiment)



Line Plot: When one variable has a natural ordering (e.g. time)



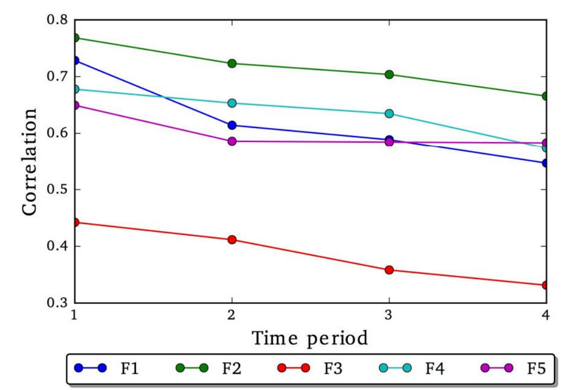


Fig 6. Test re-test validity of our learned factors.

https://doi.org/10.1371/journal.pone.0201703.g006

The Transformer: NN Sequence Modeling

-- assigning a probability to a sequences of words.

The Transformer: NN Sequence Modeling

-- assigning a probability to a sequences of elements.

Common Formulation: Model $P(e_n \mid e_1, e_2, \ldots, e_{n-1})$

:the probability of a next element given history

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n | w_1, w_2, \ldots, w_{n-1})$

:the probability of a next word given history

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n | w_1, w_2, \ldots, w_{n-1})$

:the probability of a next word given history

Task Formulation:

Input: the previous words, $w_1, w_2, \ldots, w_{n-1}$

Output: a probability for the next word, w_n

$$P(w_4 \quad | w_1='Im', w_2='feeling', w_3='very') = ??$$

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n | w_1, w_2, \ldots, w_{n-1})$

:the probability of a next word given history

Task Formulation:

Input: the previous words, $w_1, w_2, \ldots, w_{n-1}$

Output: a probability for the next word, w_n

$$P(w_4 = \text{'rhapsodic'} \mid w_1 = \text{'Im'}, w_2 = \text{'feeling'}, w_3 = \text{'very'}) = ??$$

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n | w_1, w_2, ..., w_{n-1})$

:the probability of a next word given history

Task Formulation:

Input: the previous words, $w_1, w_2, ..., w_{n-1}$

Output: a probability for the next word, w_n

$P(w_4='rhapsodic' \mid w_1='Im', w_2='feeling', w_3='very') = 0.0012$

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n | w_1, w_2, \ldots, w_{n-1})$

:the probability of a next word given history

Task Formulation:

Input: the previous words, $w_1, w_2, \ldots, w_{n-1}$

Output: a probability for the next word, w_n

"maximum likelihood estimate"
Simple way to estimate, but mostly only works ok for short phrases.

$$P('good' \mid 'Im', 'feeling', 'very') = \frac{count('Im\ feeling\ very\ good')}{count('Im\ feeling\ very\ *')}$$

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n | w_1, w_2, \ldots, w_{n-1})$

:the probability of a next word given history

Task Formulation:

Input: the previous words, $w_1, w_2, \ldots, w_{n-1}$

Output: a probability for the next word, w_n

(i.e. a "probability distribution")

$P(w_n \mid \text{'Im'}, \text{'feeling'}, \text{'very'}) =$

'good' 'clever' 'stressed' 'a' 'rhapsodic' 'blue'

Language Modeling

-- assigning a probability to a sequences of words.

Common Formulation: Model $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:the probability of a next word given history

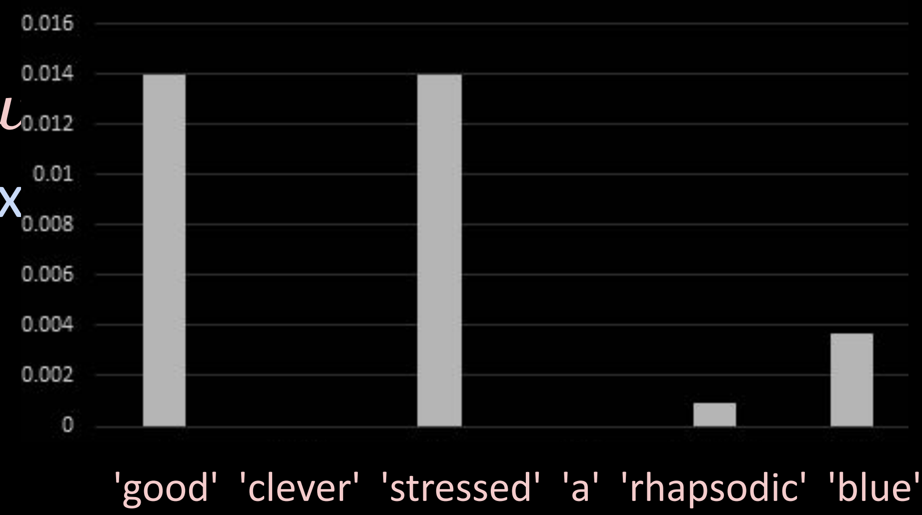
Task Formulation:

Input: the previous words, w_1, w

Output: a probability for the nex

(i.e. a "probability distribution")

$P(w_n \mid \text{'Im'}, \text{'feeling'}, \text{'very'}) =$

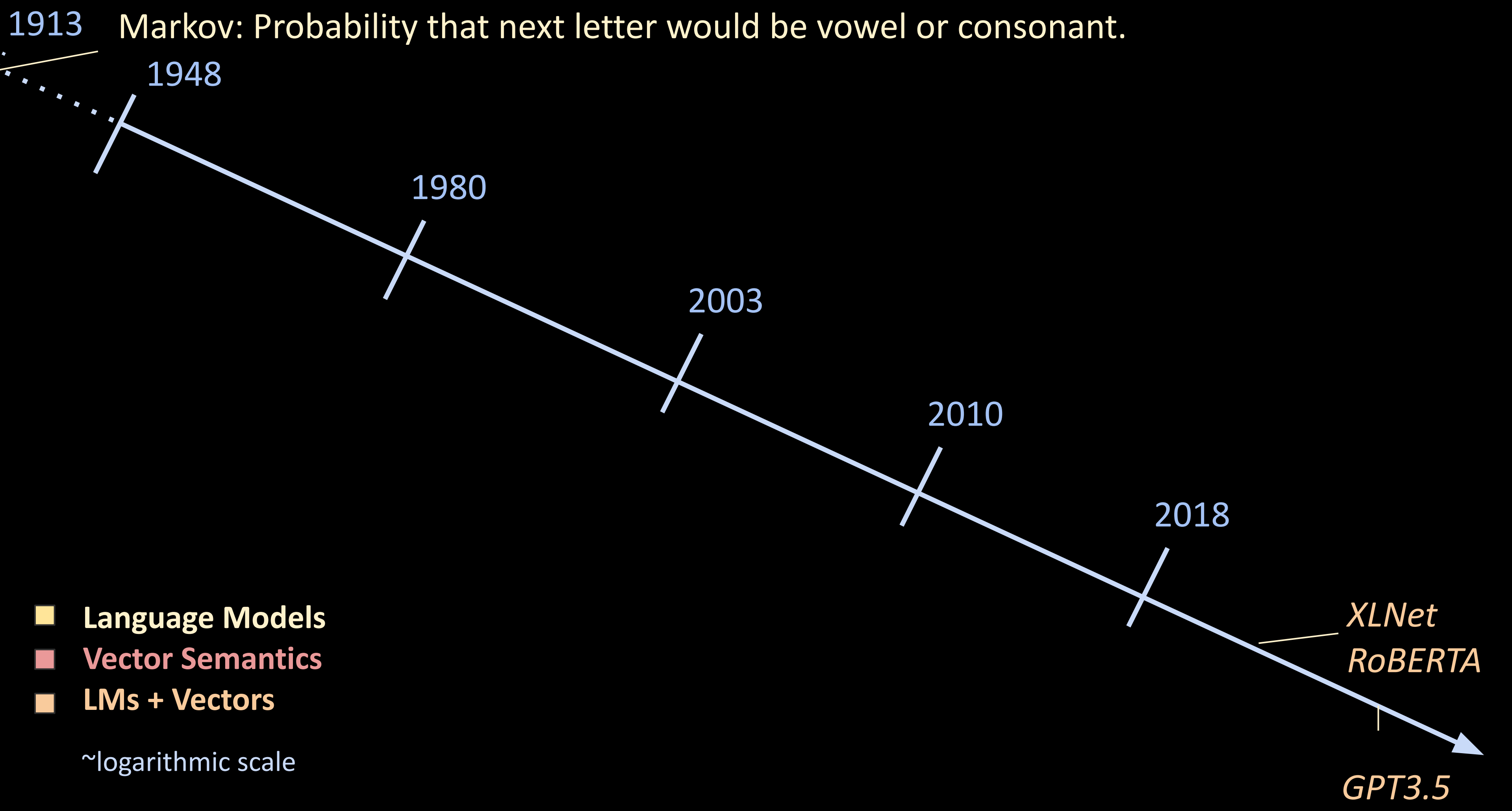


Language Modeling

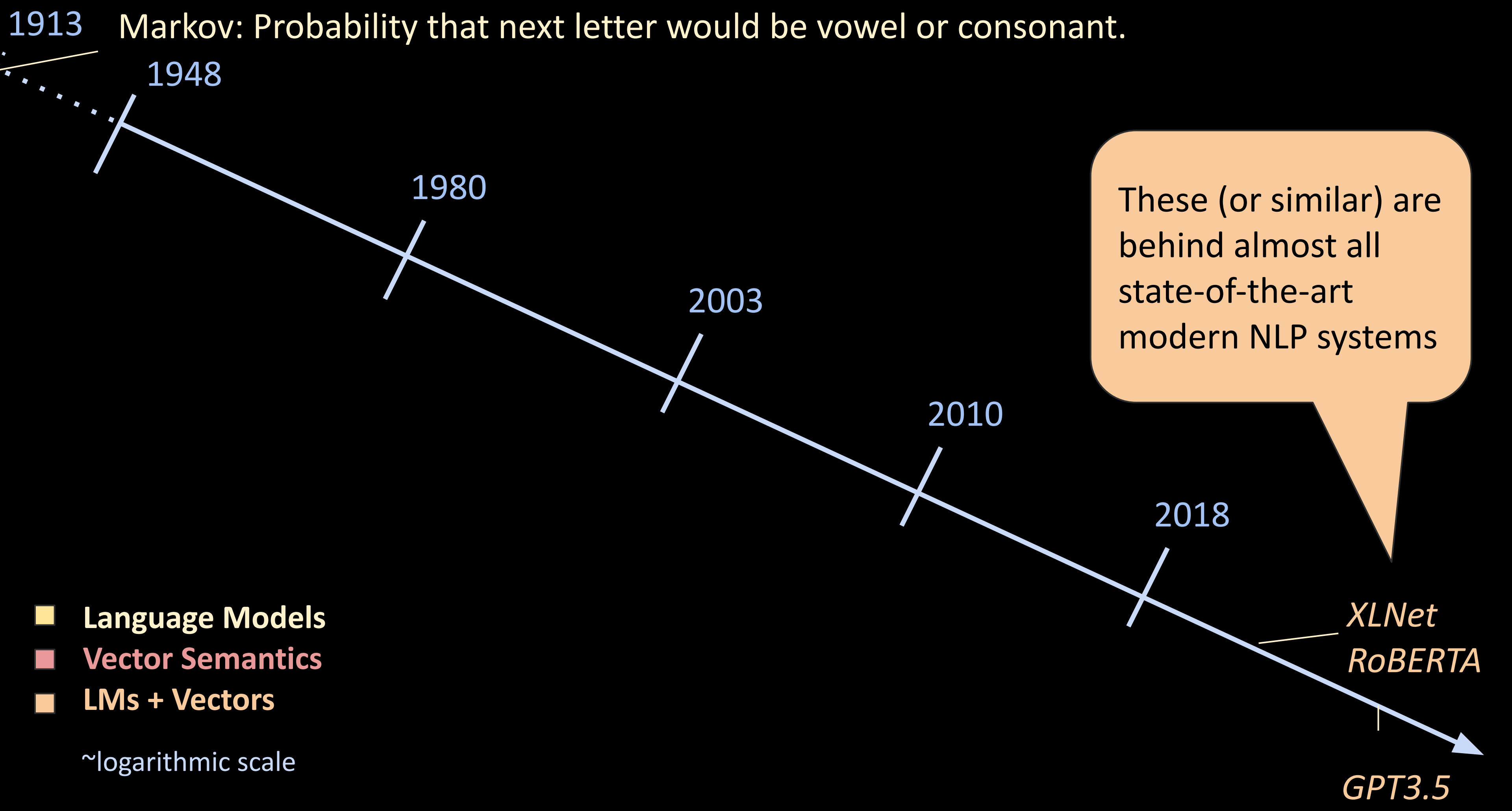
Applications:

- Auto-complete: What word is next?
- Machine Translation: Which translation is most likely?
- Spell Correction: Which word is most likely given error?
- Speech Recognition: What did they just say?
  "eyes aw of an"
  (example from Jurafsky, 2017)

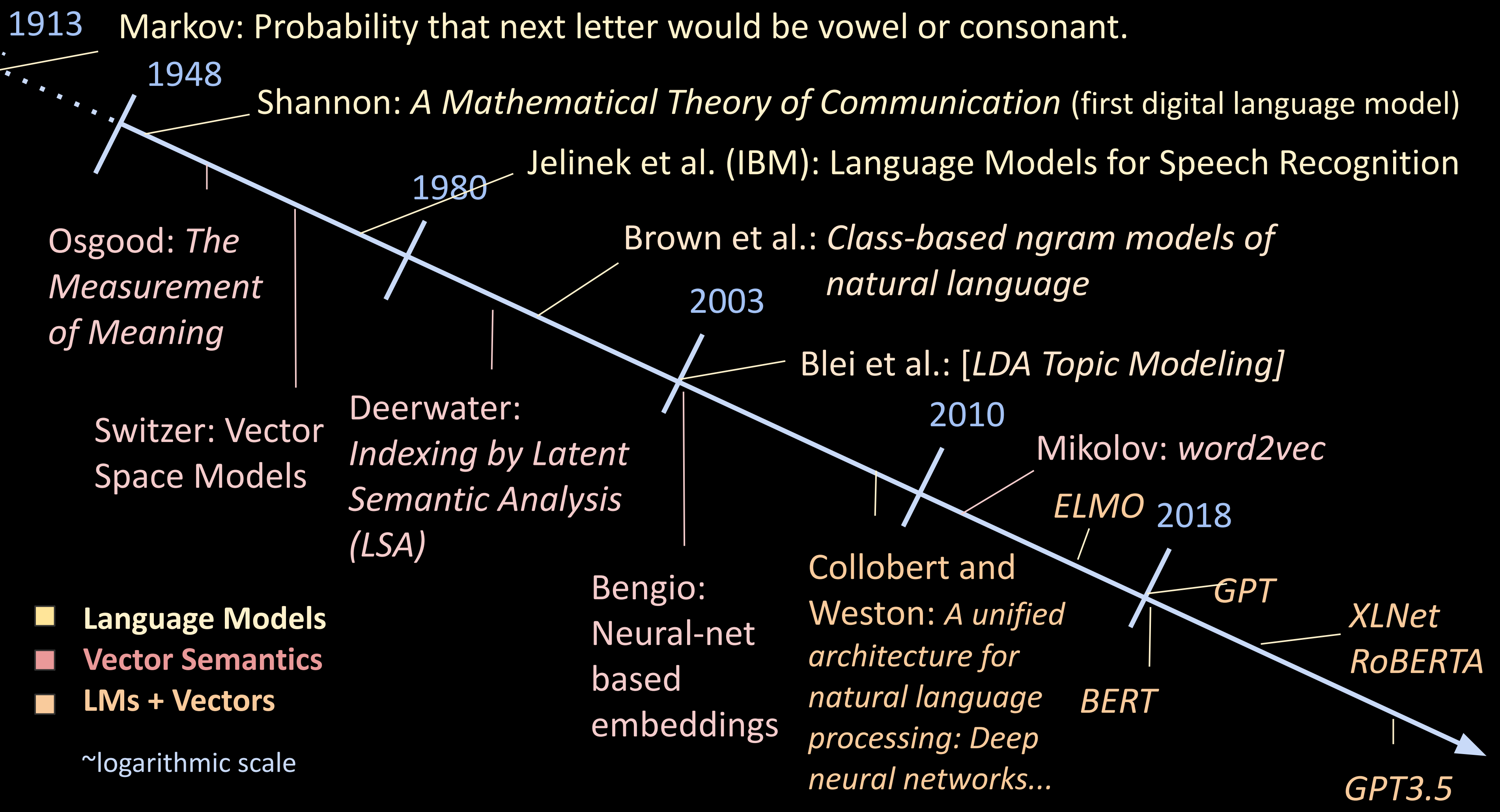
Timeline: *Language Modeling* and *Vector Semantics*



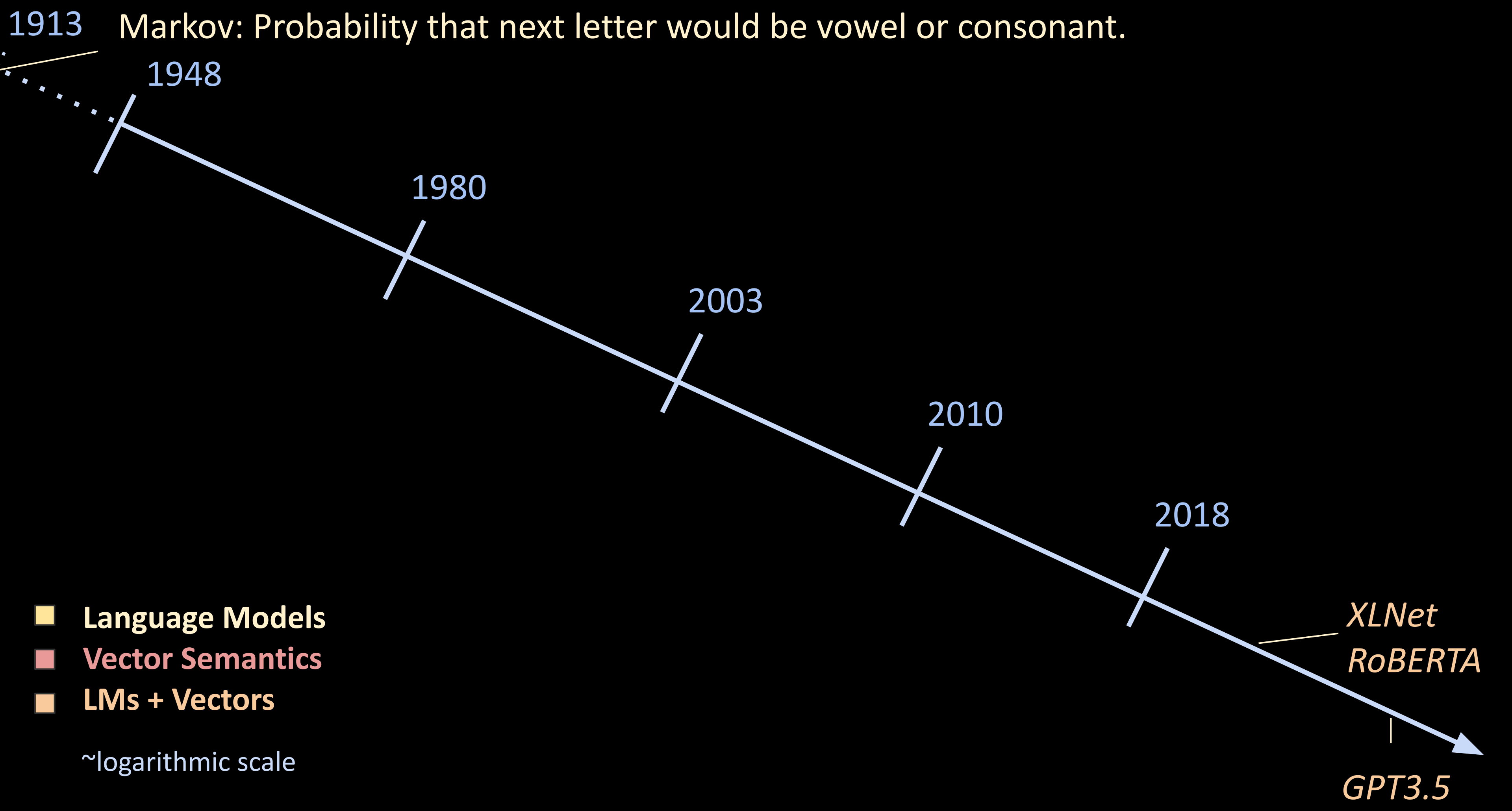
Timeline: *Language Modeling* and *Vector Semantics*



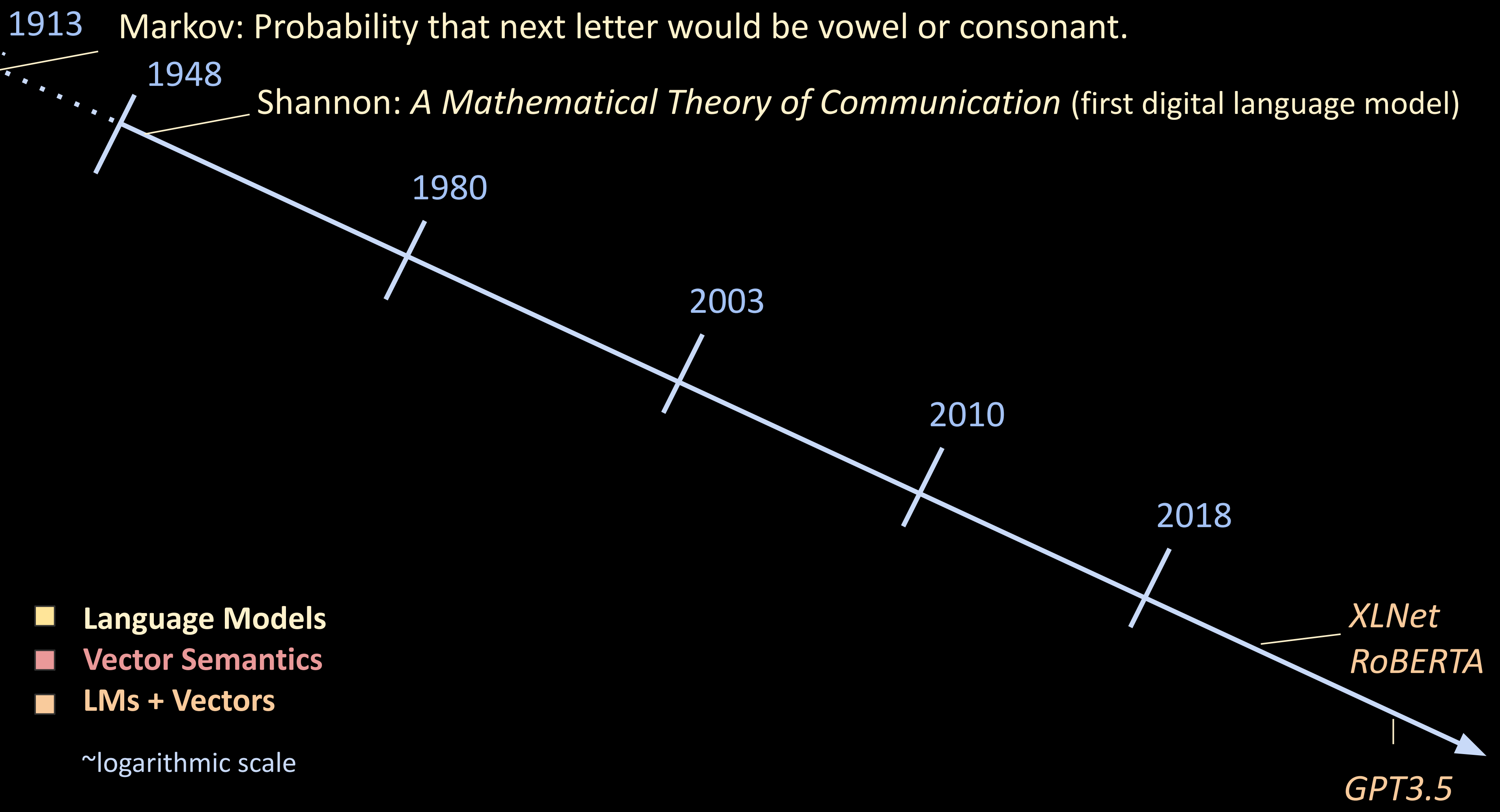
Timeline: *Language Modeling* and *Vector Semantics*



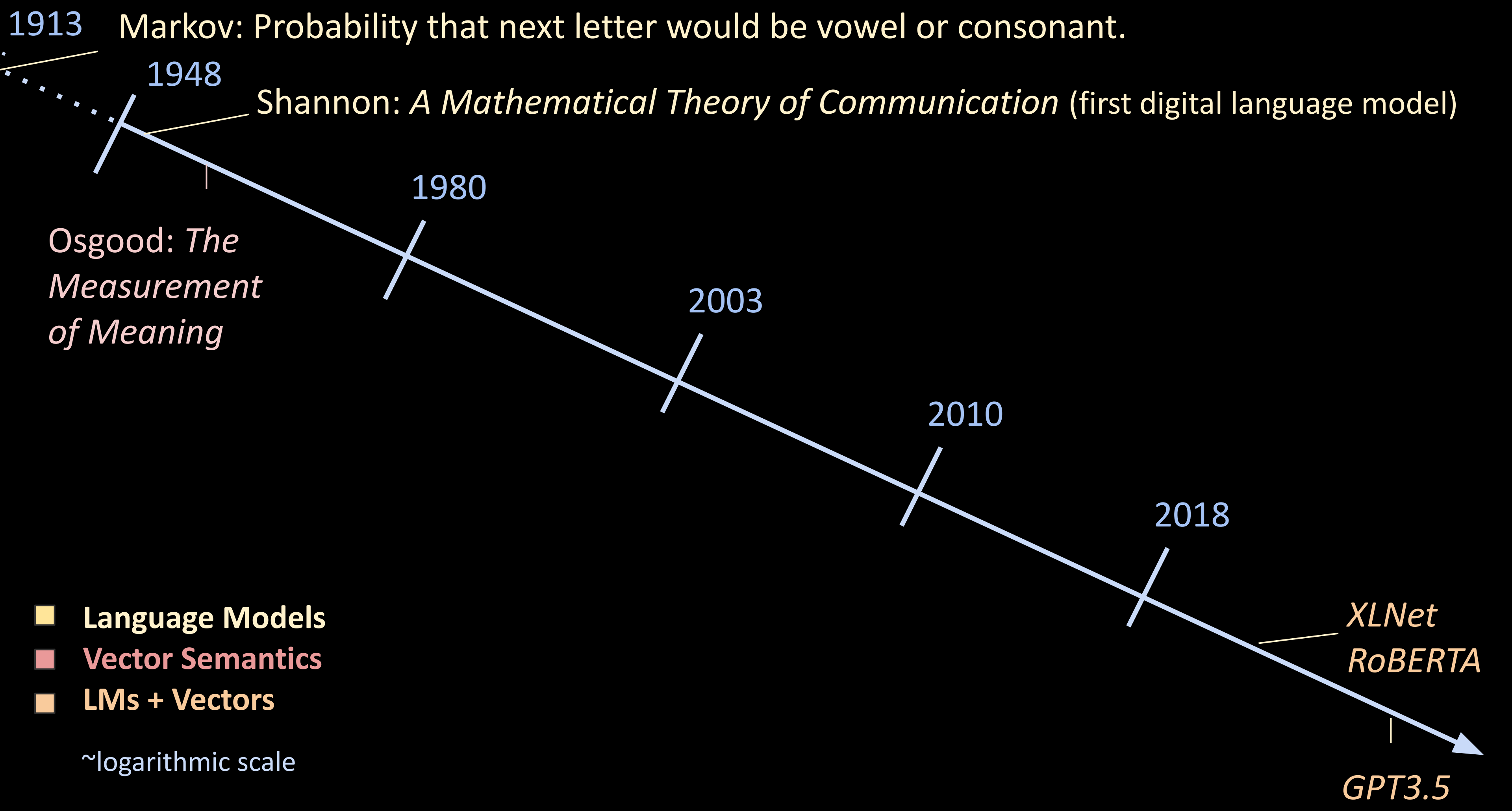
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*

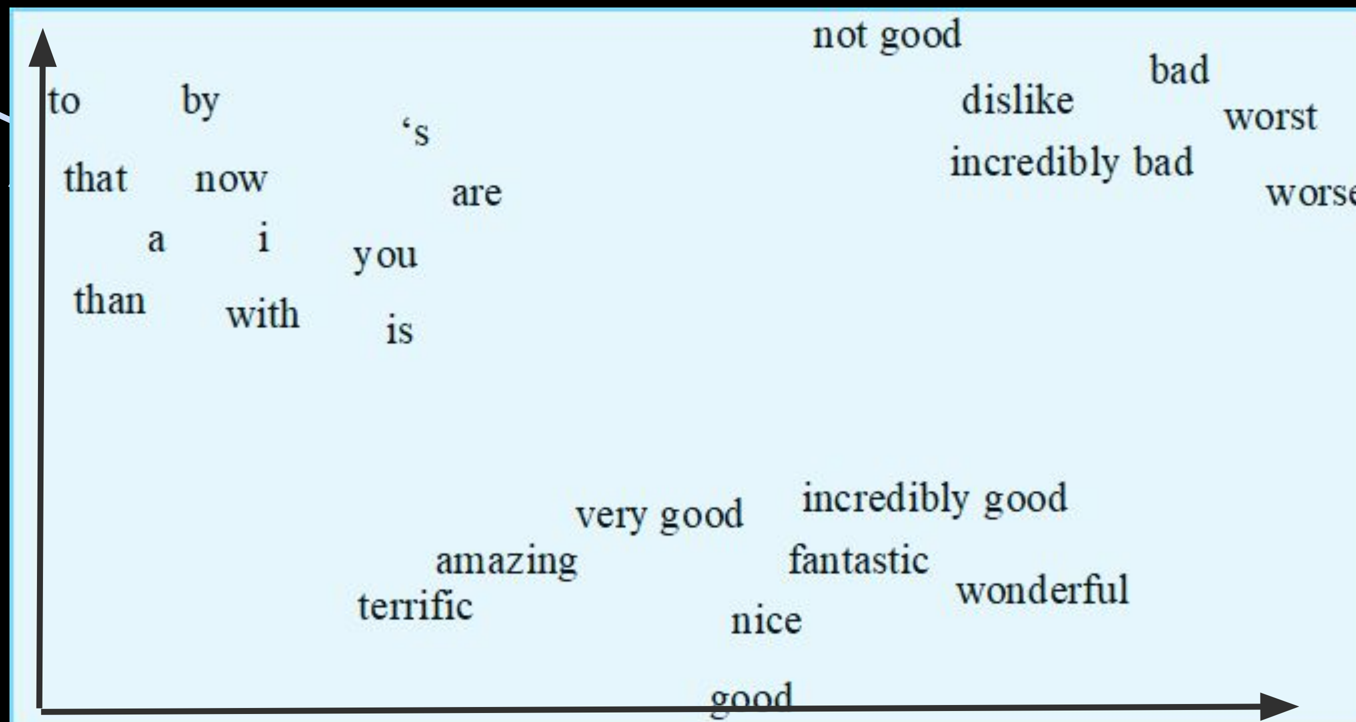


Timeline: *Language Modeling* and *Vector Semantics*

1913 Markov: Probability that next letter would be vowel or consonant.

1948 Shannon: *A Mathematical Theory of Communication* (first digital language model)

Osgood: *The Measurement of Meaning*



(Li et al. ,2015; Jurafsky et al., 2019)

- Language Models
- Vector Semantics
- LMs + Vectors

~logarithmic scale

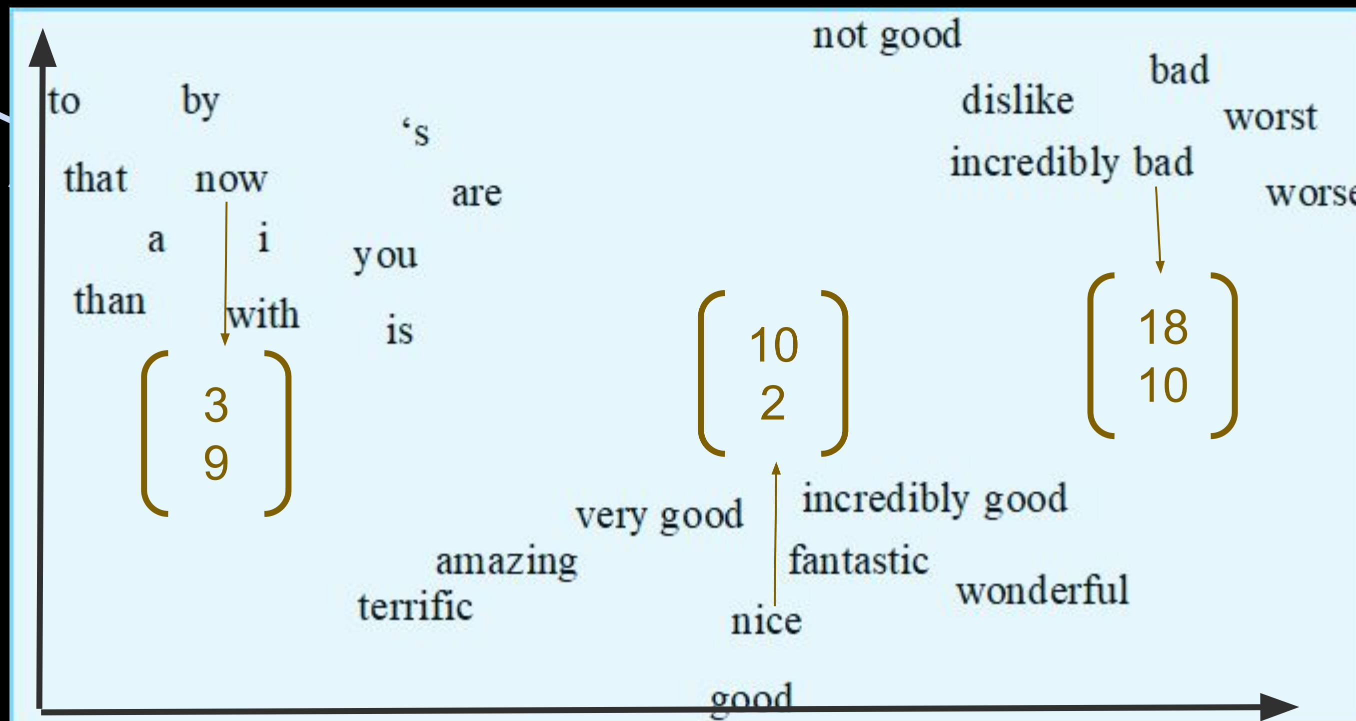
GPT3.5

Timeline: *Language Modeling* and *Vector Semantics*

1913 Markov: Probability that next letter would be vowel or consonant.

1948 Shannon: *A Mathematical Theory of Communication* (first digital language model)

Osgood: *The Measurement of Meaning*



(Li et al. ,2015; Jurafsky et al., 2019)

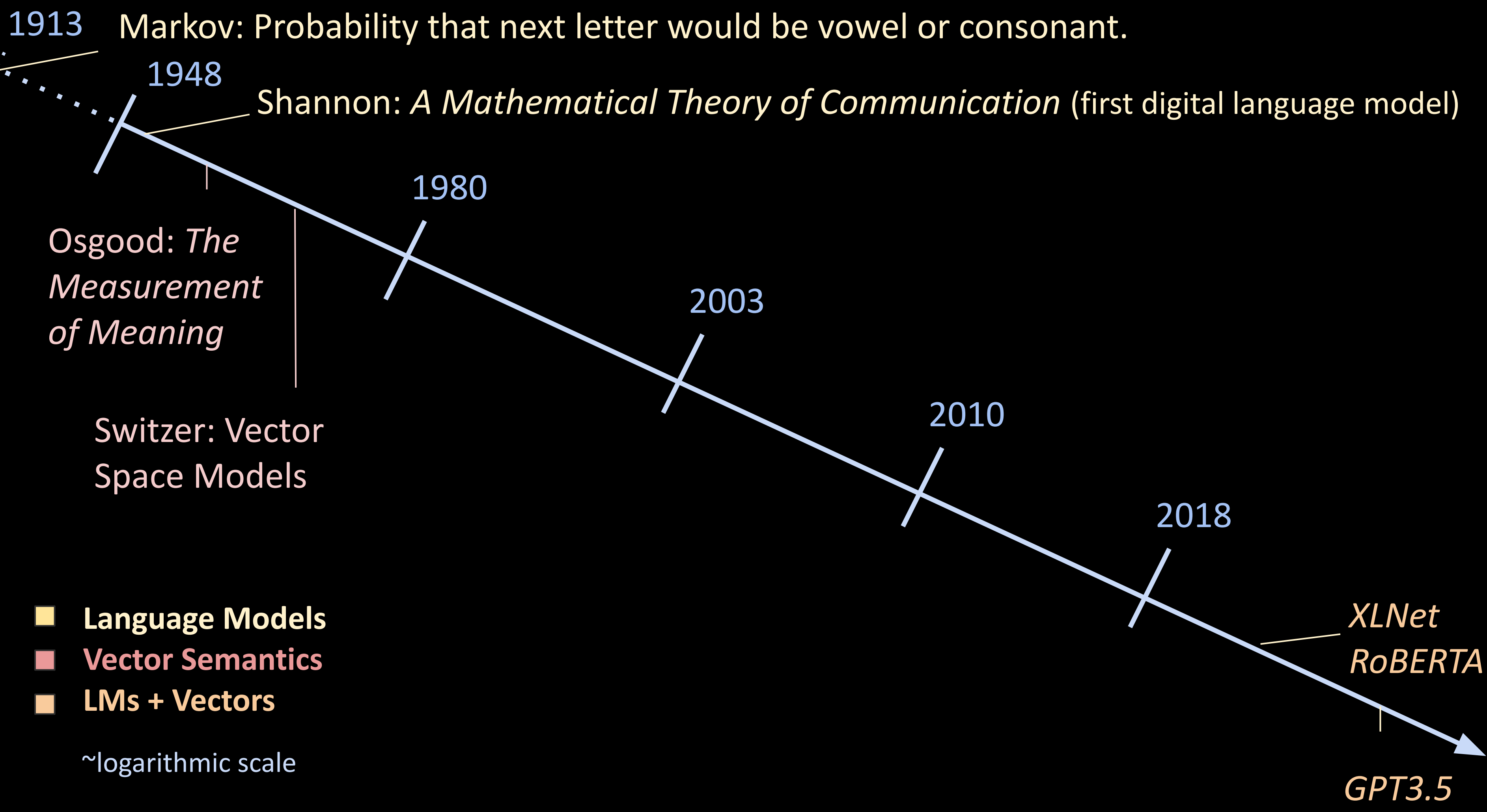
A

GPT3.5

- Language Models
- Vector Semantics
- LMs + Vectors

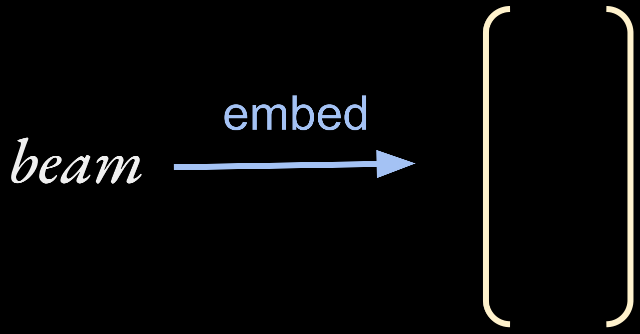
~logarithmic scale

Timeline: *Language Modeling* and *Vector Semantics*



Word Vectors

To embed: convert a token (or sequence) to a vector that **represents meaning**.



Word Vectors

To embed: convert a token (or sequence) to a vector that **represents meaning**.

"one-hot encoding"

$$\textit{beam} \xrightarrow{\text{embed}} \begin{bmatrix} 0 \\ \dots \\ 0 \\ 1 \\ \dots \\ 0 \end{bmatrix}$$

Word Vectors

To embed: convert a token (or sequence) to a vector that **represents meaning**.

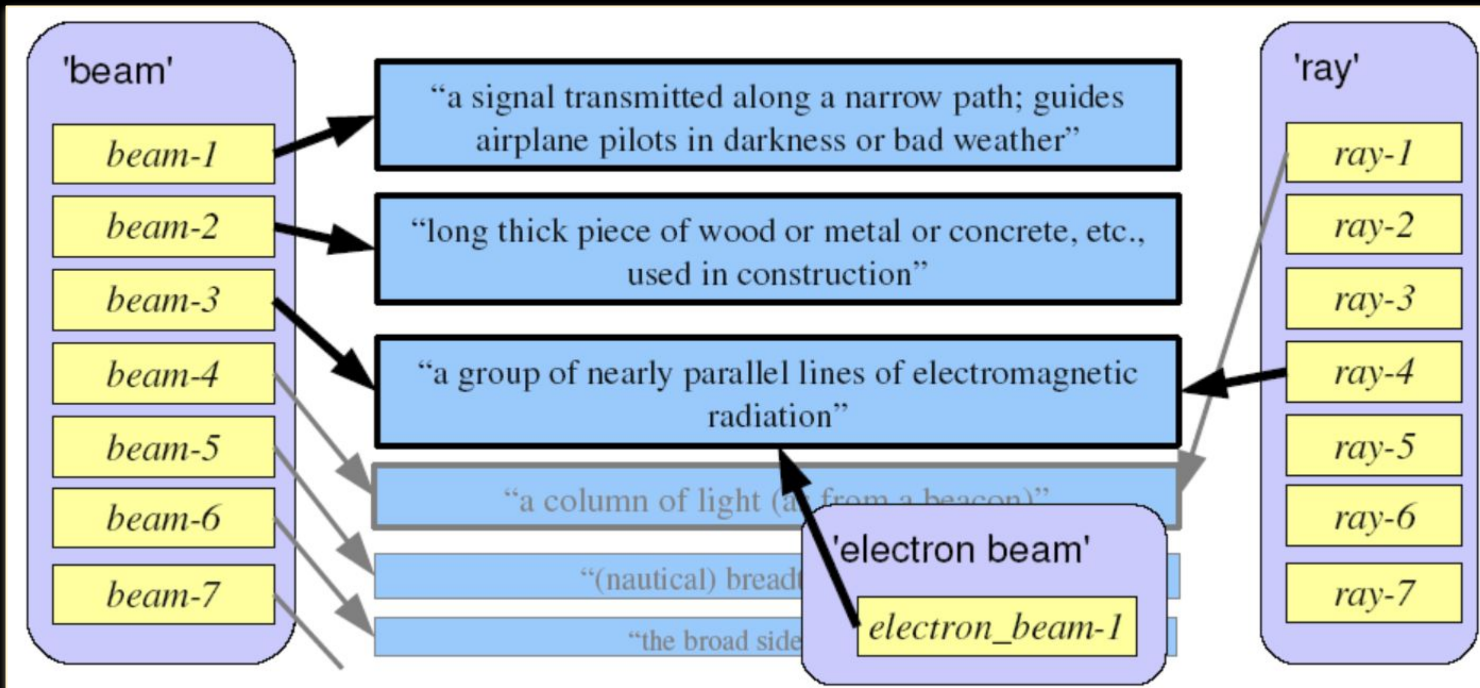
"one-hot encoding"

$$\textit{beam} \xrightarrow{\text{embed}} \begin{bmatrix} 0 \\ \dots \\ 0 \\ 1 \\ \dots \\ 0 \end{bmatrix}$$

Prefer dense vectors; why?

- Less parameters (weights) for machine learning model.
- May generalize better implicitly.
- May capture synonyms

Word Vectors



The nail hit the beam behind the wall.

Word Vectors

To embed: convert a token (or sequence) to a vector that represents **meaning.**

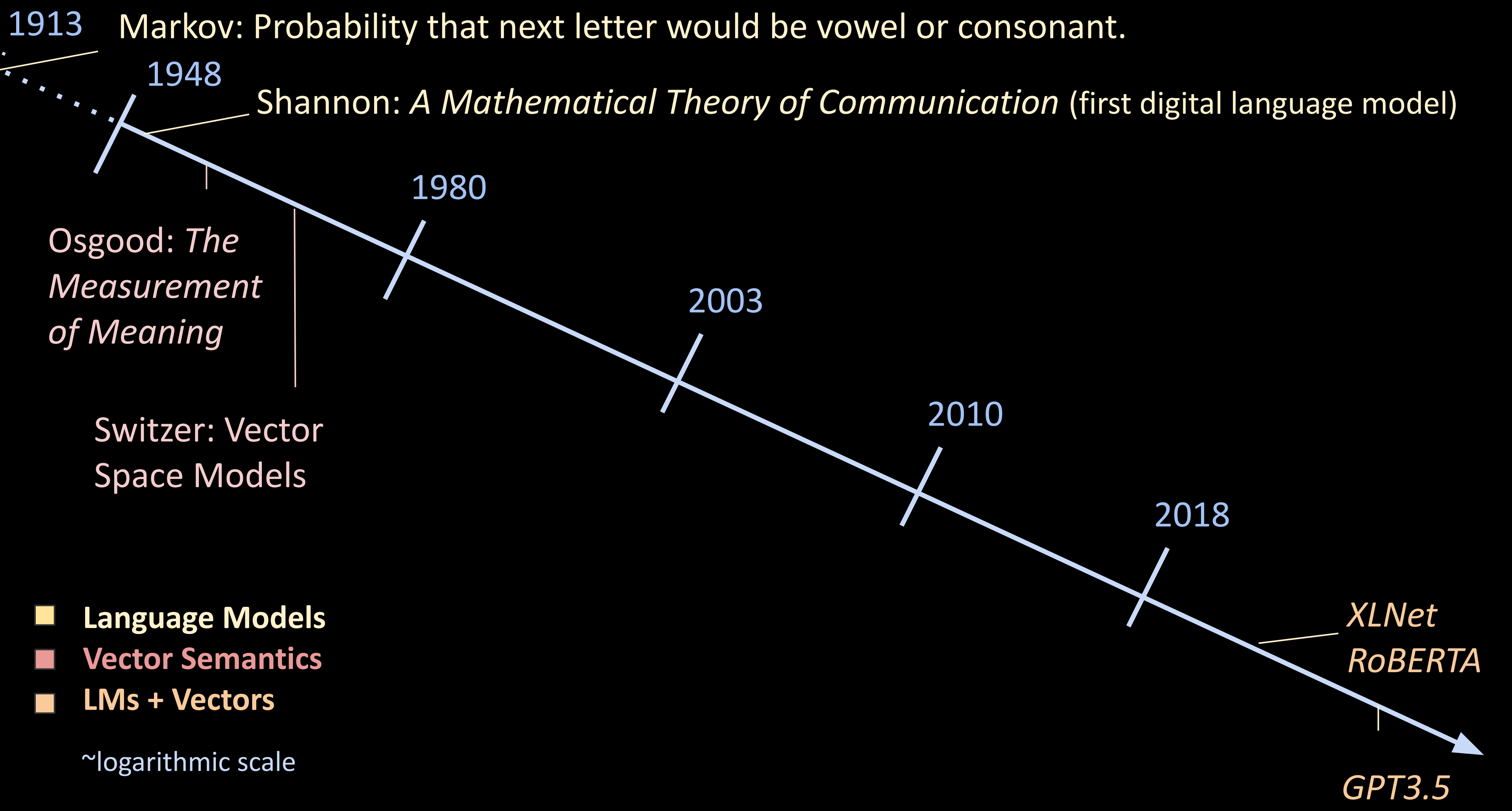
Wittgenstein, 1945: "*The meaning of a word is its use in the language*"

Distributional hypothesis -- A word's meaning is defined by all the different contexts it appears in (i.e. how it is "distributed" in natural language).

Firth, 1957: "*You shall know a word by the company it keeps*"

The nail hit the beam behind the wall.

Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*

1913 Markov: Probability that next letter would be vowel or consonant.

1948 Shannon: *A Mathematical Theory of Communication* (first digital language model)

Osgood: *The Measurement of Meaning*

Switzer: Vector Space Models

1980

Deerwater: *Indexing by Latent Semantic Analysis (LSA)*

2003

2010

2018

XLNet
RoBERTA

GPT3.5

- Language Models
- Vector Semantics
- LMs + Vectors

~logarithmic scale

Word Vectors

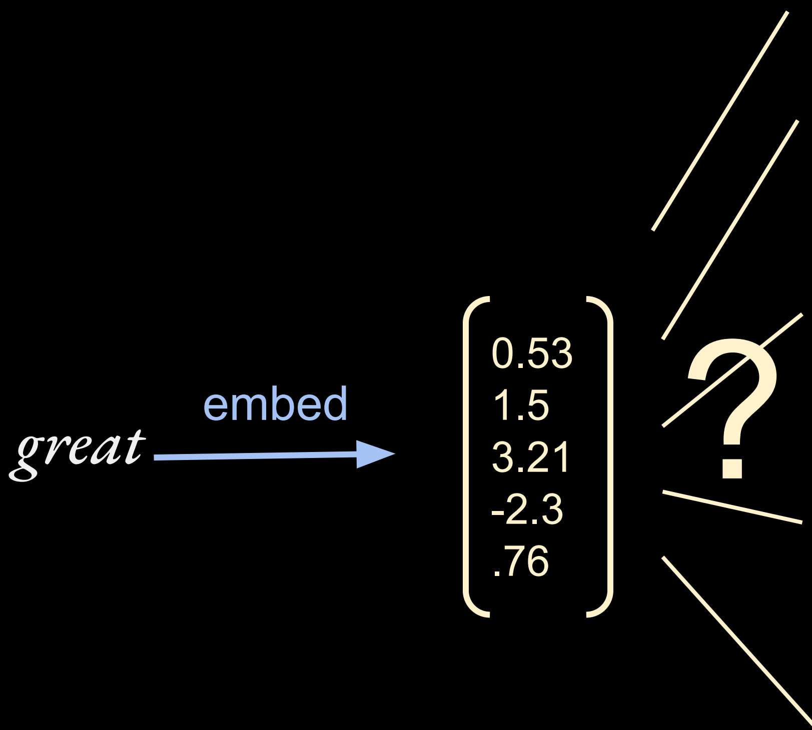
	Person A	**Person B**
How are you?	I feel ***fine*** –even ***great***!	My life is a ***great*** mess! I'm having a very hard time being happy.
What is going on?	Earlier, I ***played*** the ***game*** Yahtzee with my ***partner*.** I could not get that ***die*** to roll a 1! Now I'm ***lying*** on my bed for a ***rest***.	My business ***partner*** was ***lying*** to me. He was trying to ***game*** the system and ***played*** me. I think I am going to ***die*** –he left and now I have to pay the ***rest*** of his ***fine*.**

(Kjell, Kjell, and Schwartz, 2023)

Objective

$$\textit{great} \xrightarrow{\text{embed}} \begin{pmatrix} 0.53 \\ 1.5 \\ 3.21 \\ -2.3 \\ .76 \end{pmatrix}$$

Objective



great.a.1 (relatively large in size or number or extent; larger than others of its kind)

great.a.2, outstanding (of major significance or importance)

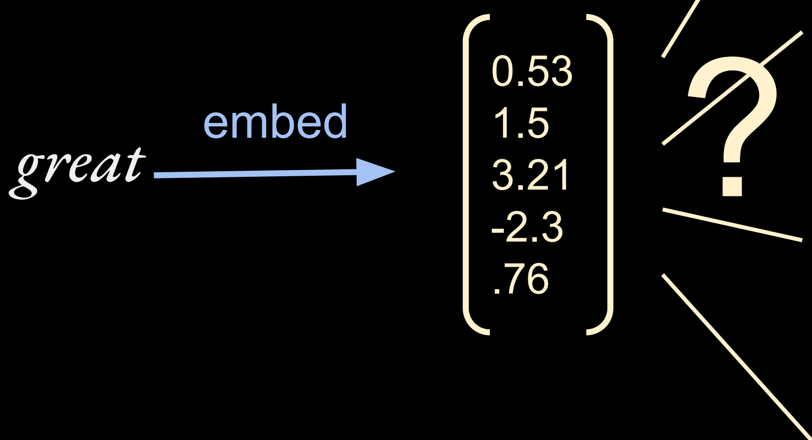
great.a.3 (remarkable or out of the ordinary in degree or magnitude or effect)

bang-up, bully, corking, cracking, dandy, **great.a.4**, groovy, keen, neat, nifty, not bad, peachy, slap-up, swell, smashing, old (very good)

capital, **great.a.5**, majuscule (uppercase)

big, enceinte, expectant, gravid, **great.a.6**, large, heavy, with child (in an advanced stage of pregnancy)

Objective



great.a.1 (relatively large in size or number or extent; larger than others of its kind)

great.a.2, outstanding (of major significance or importance)

great.a.3 (remarkable or out of the ordinary in degree or magnitude or effect)

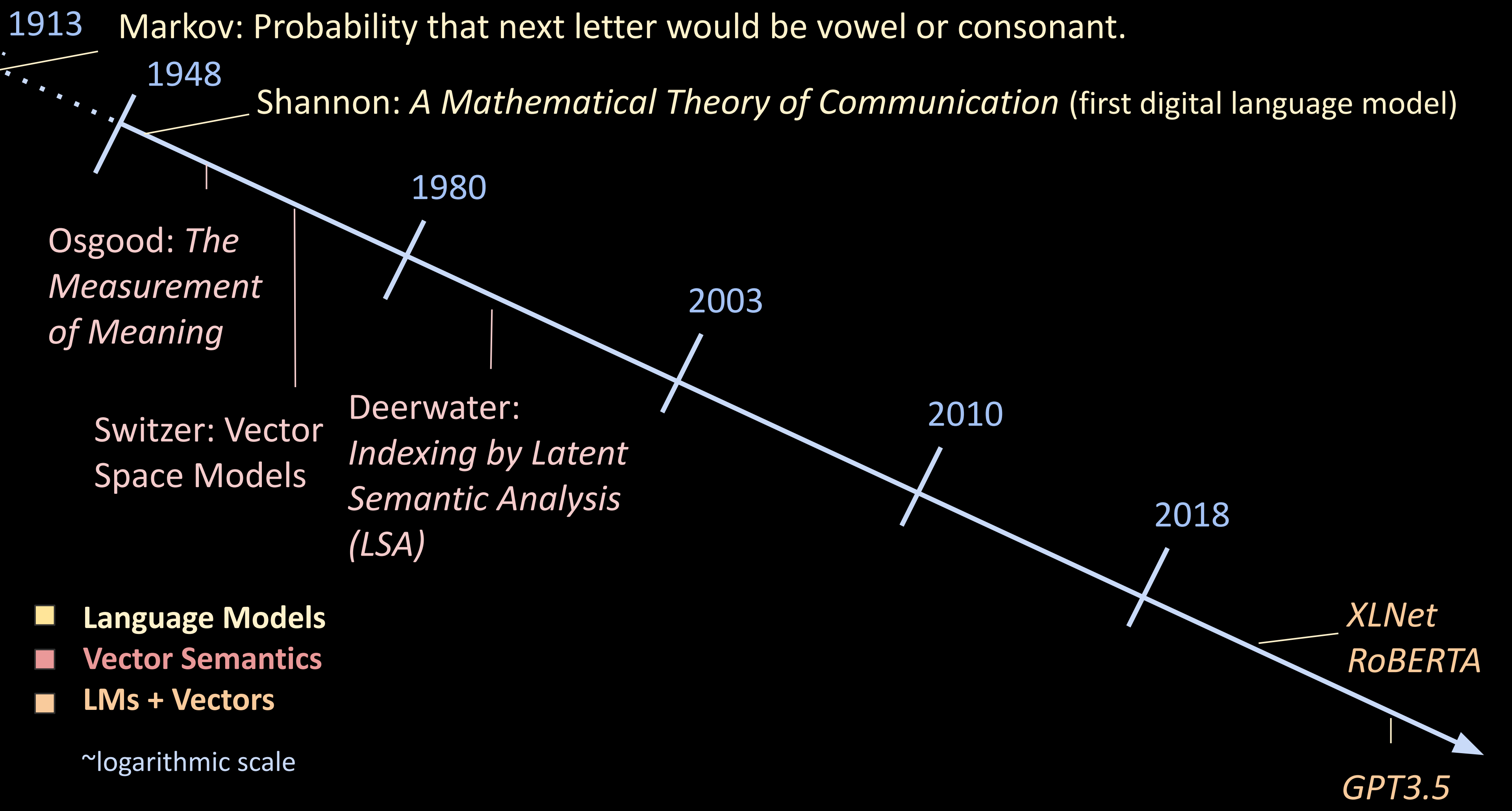
bang-up, bully, corking, cracking, dandy, **great.a.4**, groovy, keen, neat, nifty, not bad, peachy, slap-up, swell, smashing, old (very good)

capital, **great.a.5**, majuscule (uppercase)

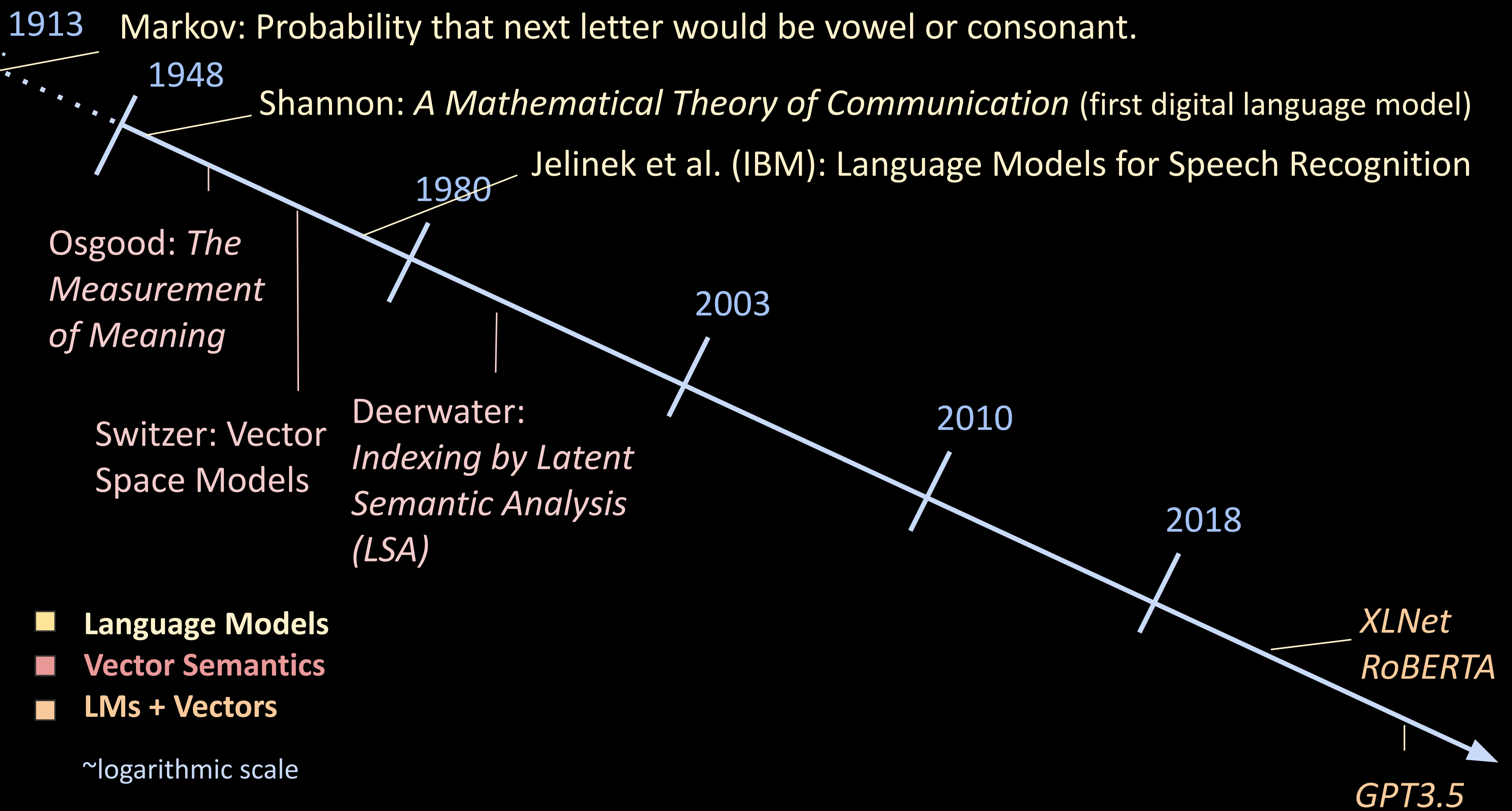
big, enceinte, expectant, gravid, **great.a.6**, large, heavy, with child (in an advanced stage of pregnancy)

great.n.1 (a person who has achieved distinction and honor in some field)

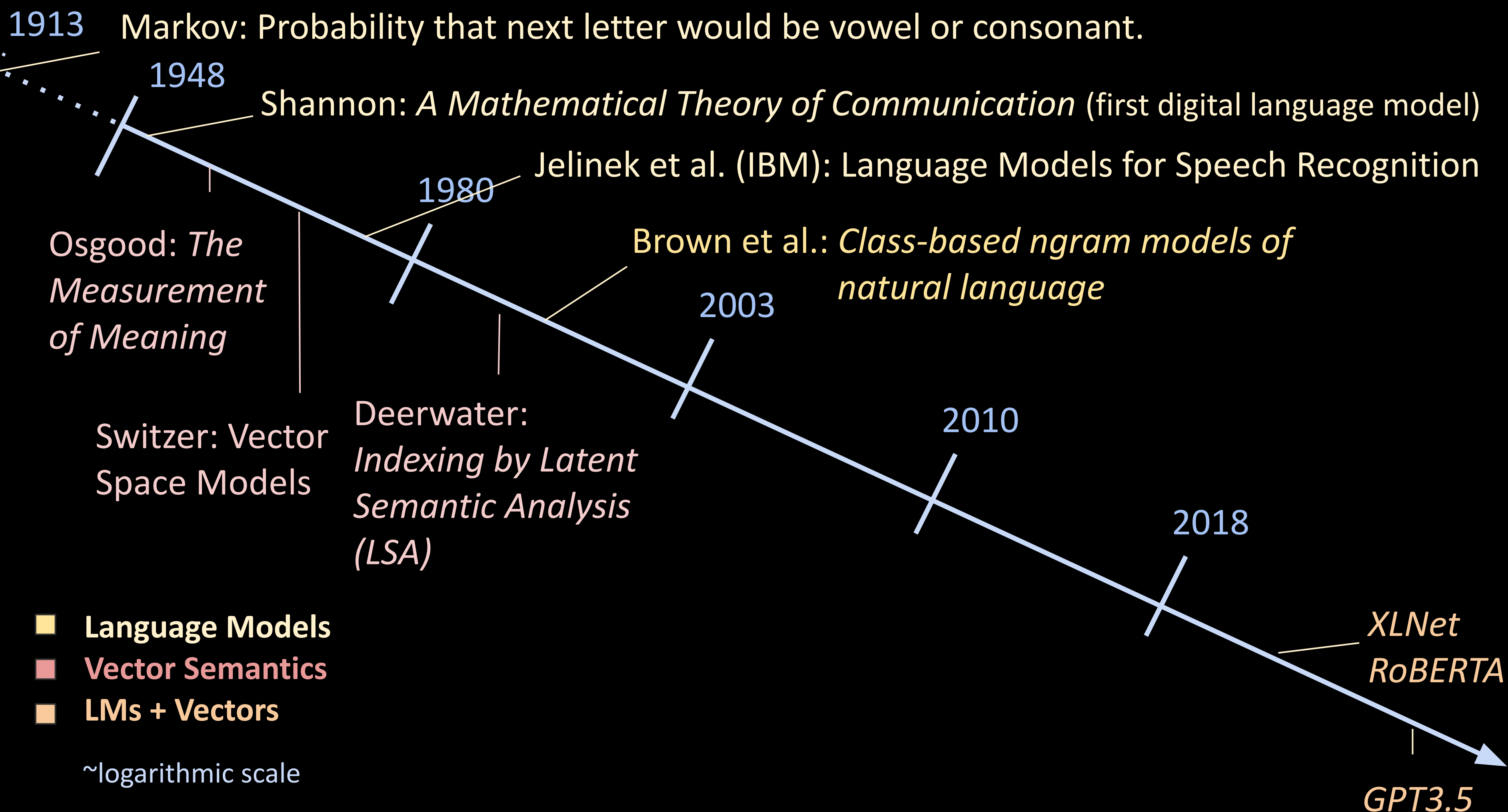
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*

1913 Markov: Probability that next letter would be vowel or consonant.

1948 Shannon: *A Mathematical Theory of Communication* (first digital language model)

Osgood: *The Measurement of Meaning*

Switzer: Vector Space Models

1980 Jelinek et al. (IBM): Language Models for Speech Recognition

Deerwater: *Indexing by Latent Semantic Analysis (LSA)*

Brown et al.: *Class-based ngram models of natural language*

2003

http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html

2010

2018

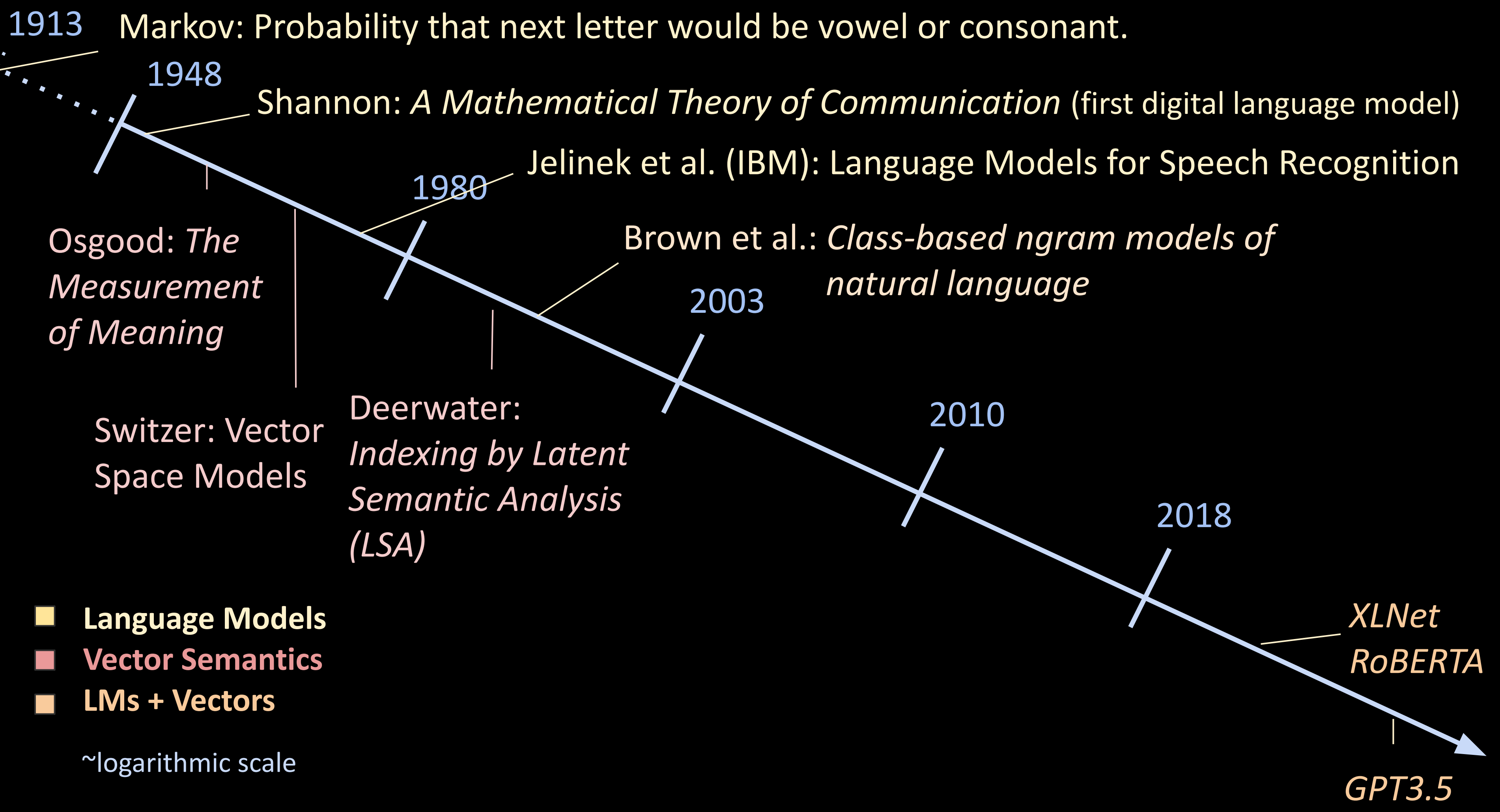
XLNet
RoBERTA

GPT3.5

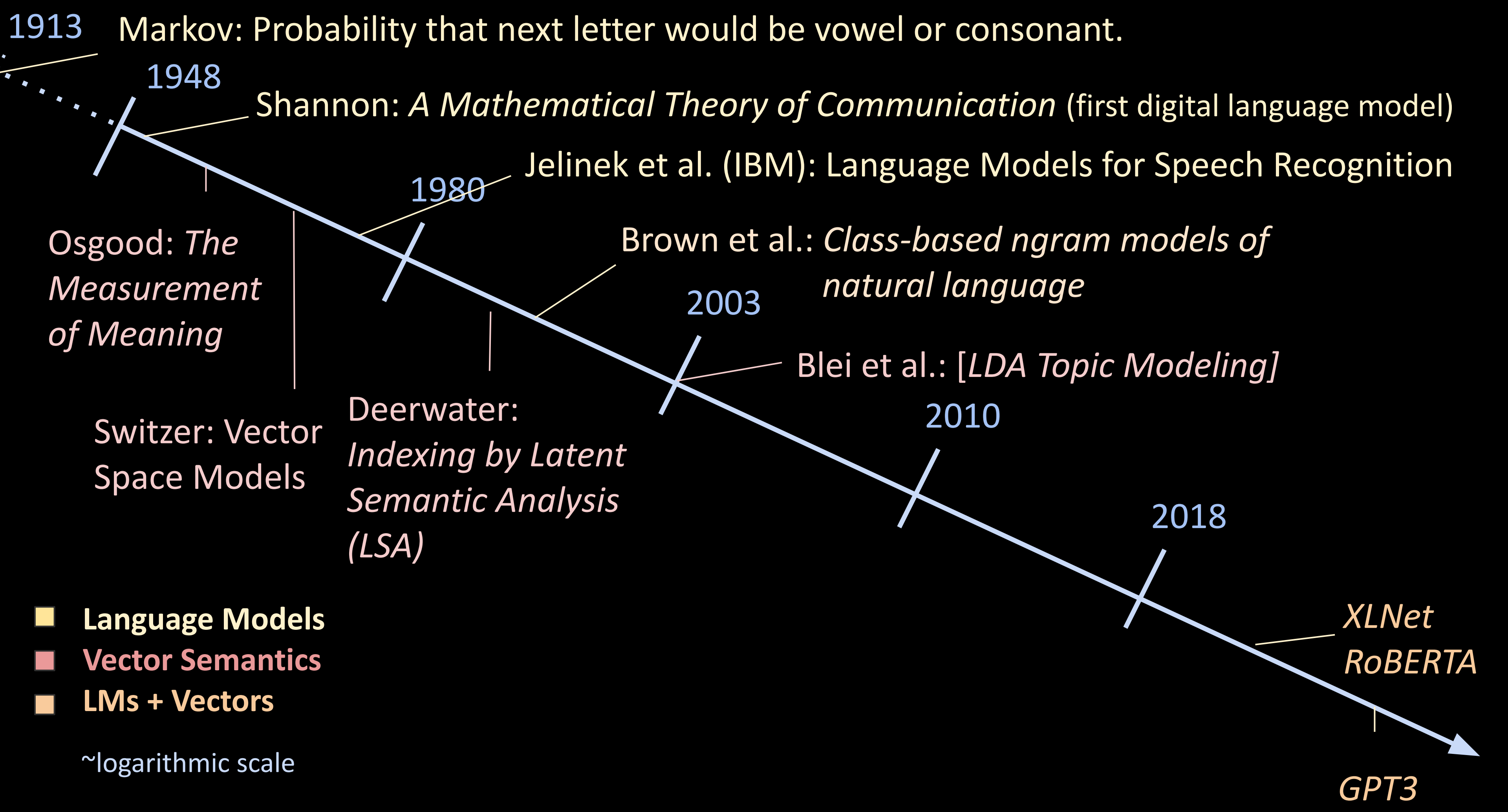
- **Language Models**
- **Vector Semantics**
- **LMs + Vectors**

~logarithmic scale

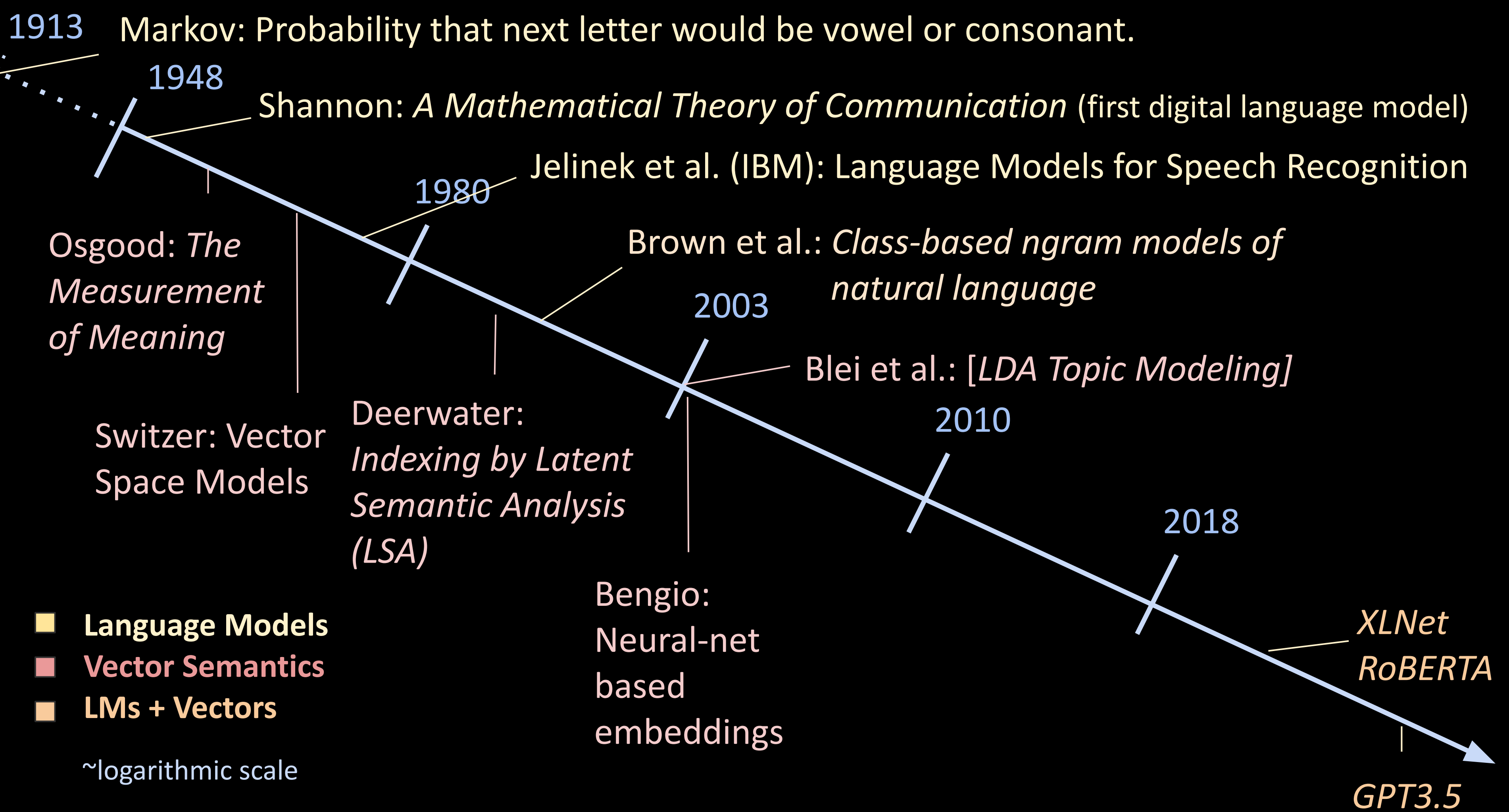
Timeline: *Language Modeling* and *Vector Semantics*



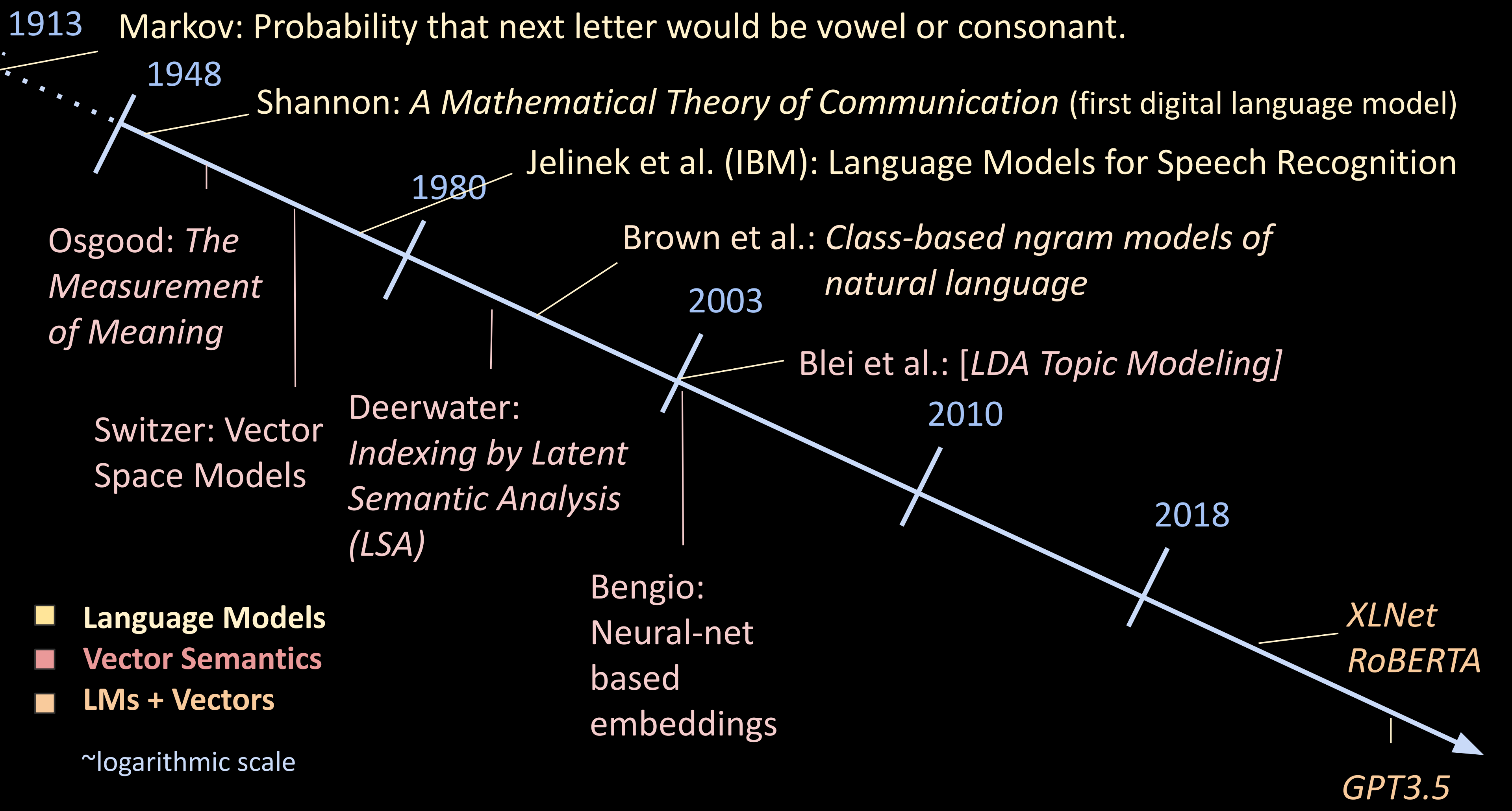
Timeline: *Language Modeling* and *Vector Semantics*



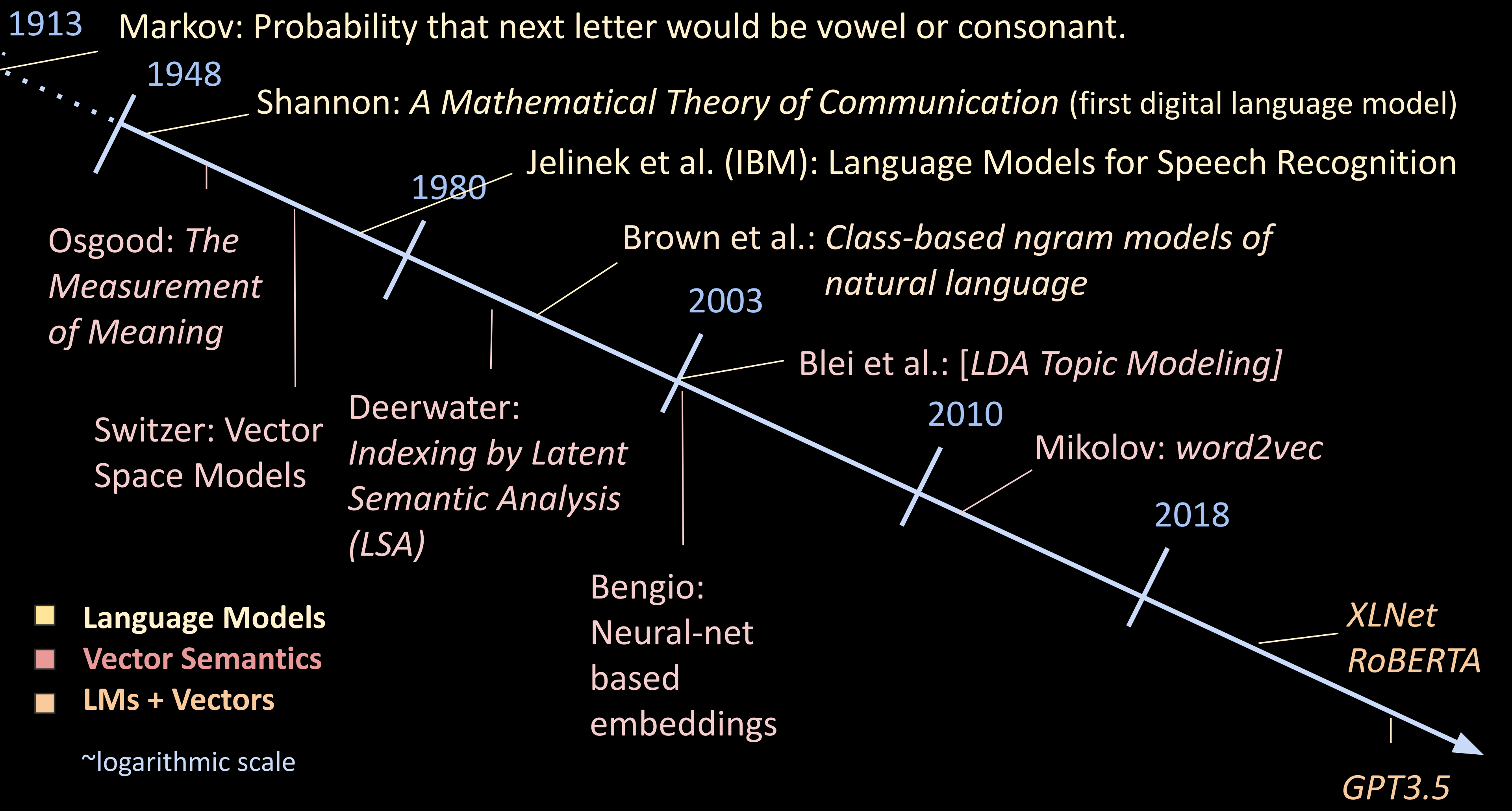
Timeline: *Language Modeling* and *Vector Semantics*



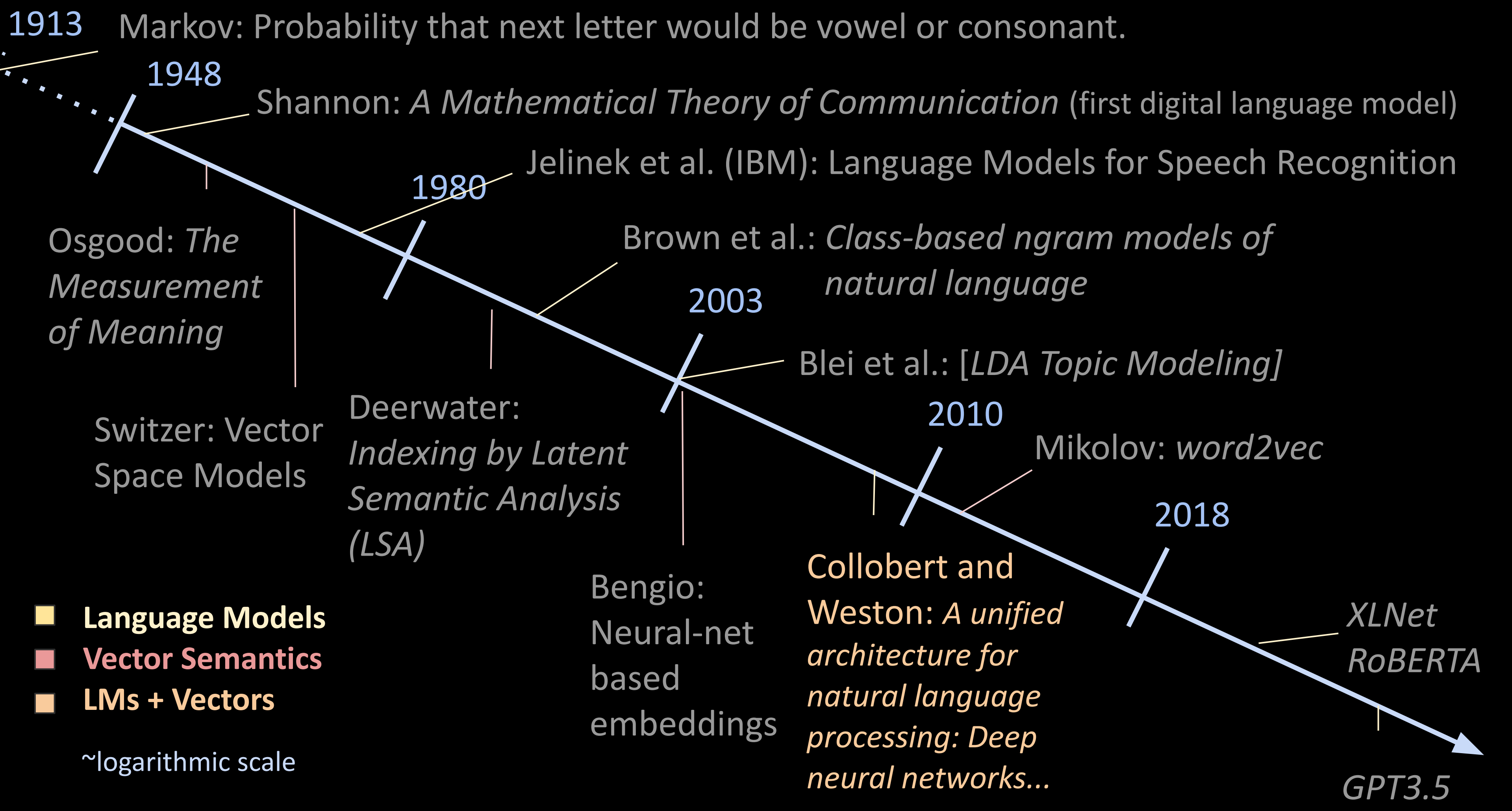
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



T... ...odeling and *Vector Semantics*

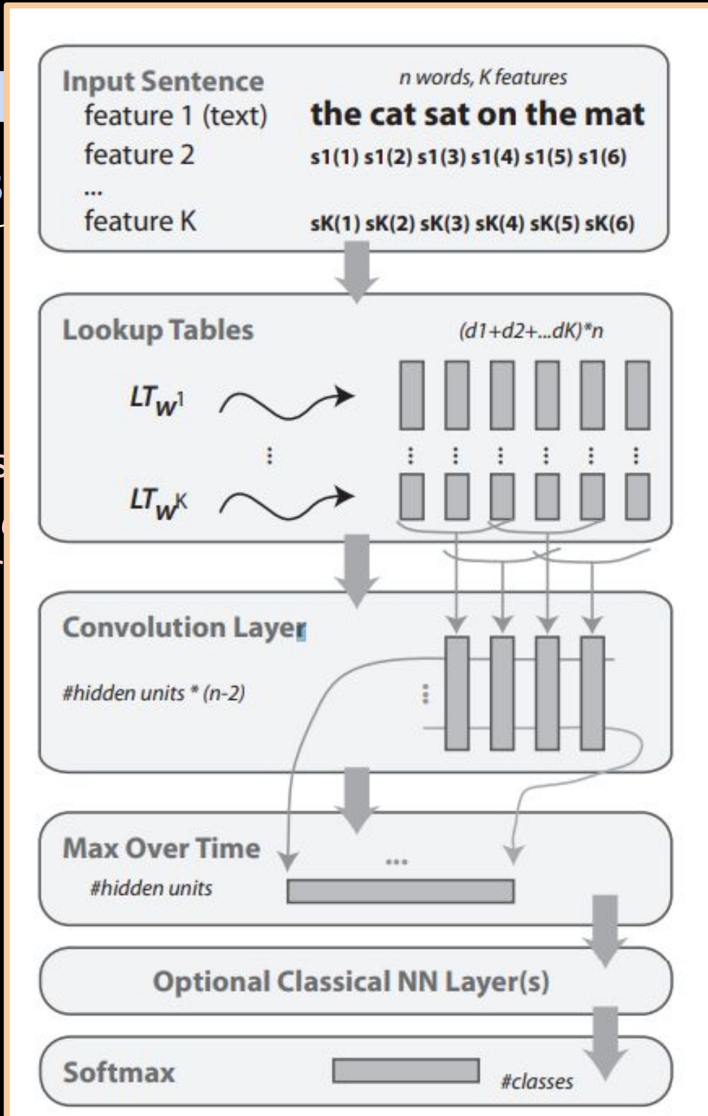
1913 ...r would be vowel or consonant.

...l *Theory of Communication* (first digital language model)

...et al. (IBM): Language Models for Speech Recognition

...rown et al.: *Class-based ngram models of natural language*

Os... *M...* *of...*



2003

Blei et al.: [*LDA Topic Modeling*]

...gio: ...ural-net ...ed ...beddings

2010

Mikolov: *word2vec*

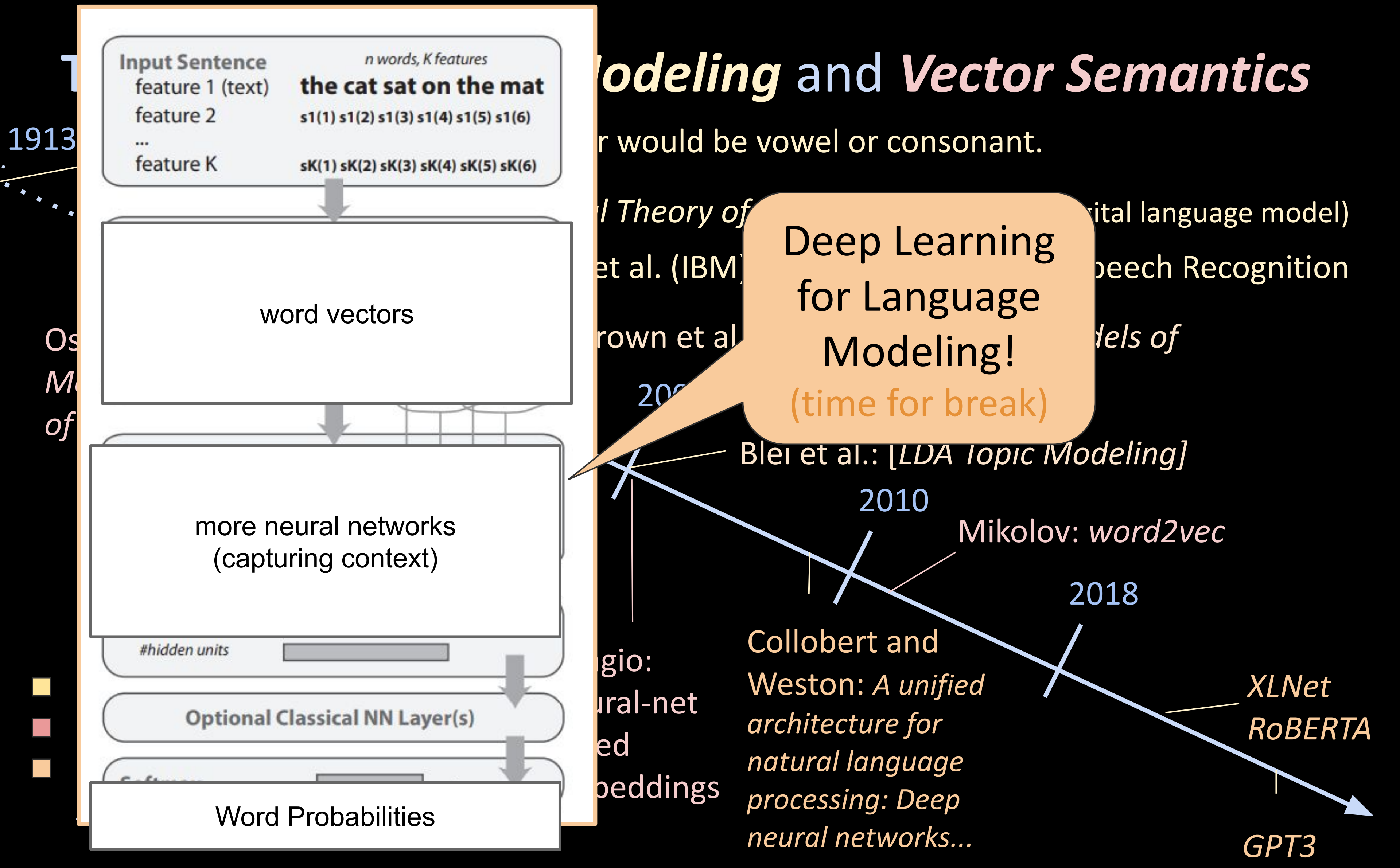
Collobert and Weston: *A unified architecture for natural language processing: Deep neural networks...*

2018

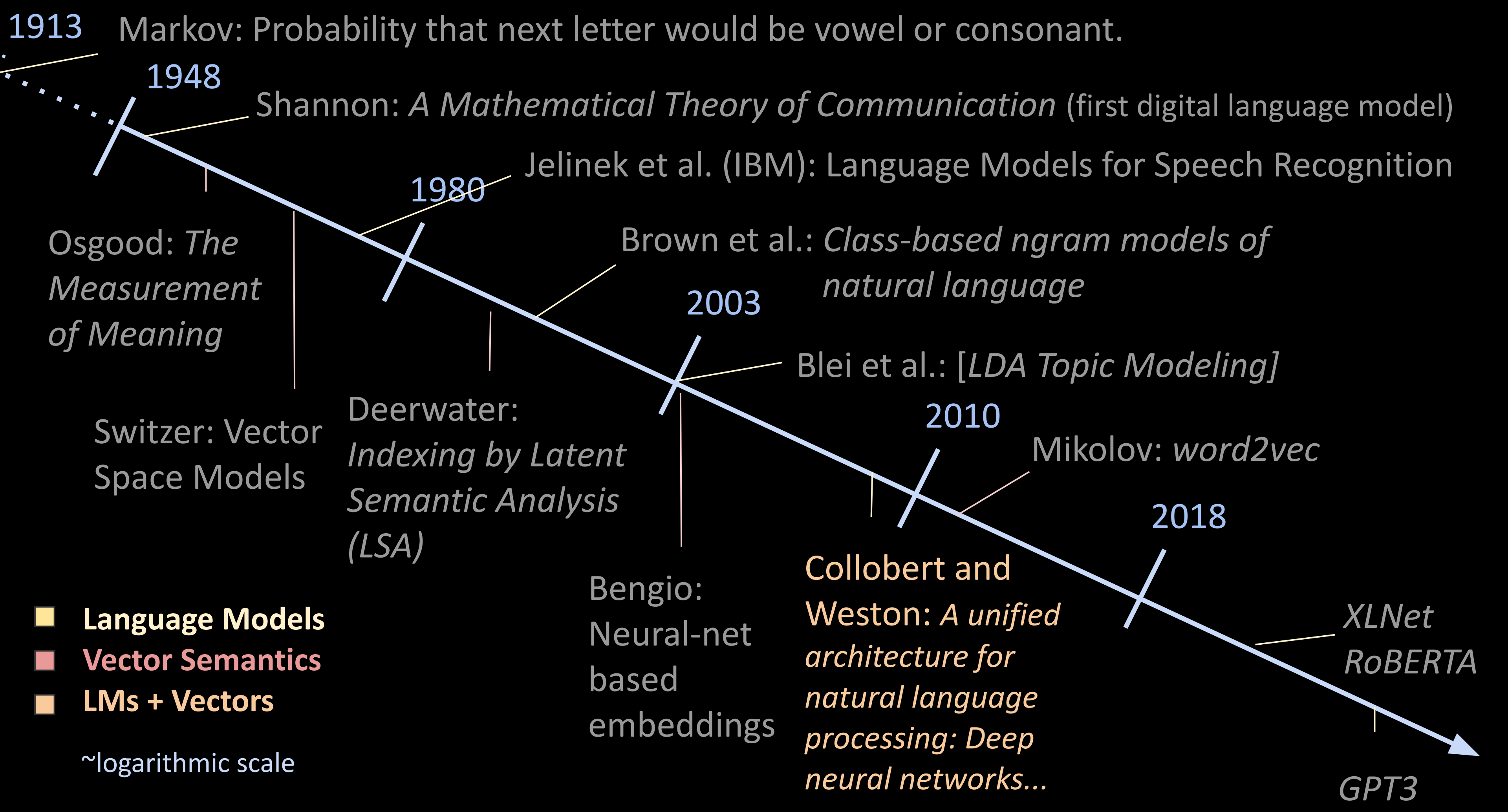
XLNet
RoBERTA

GPT3

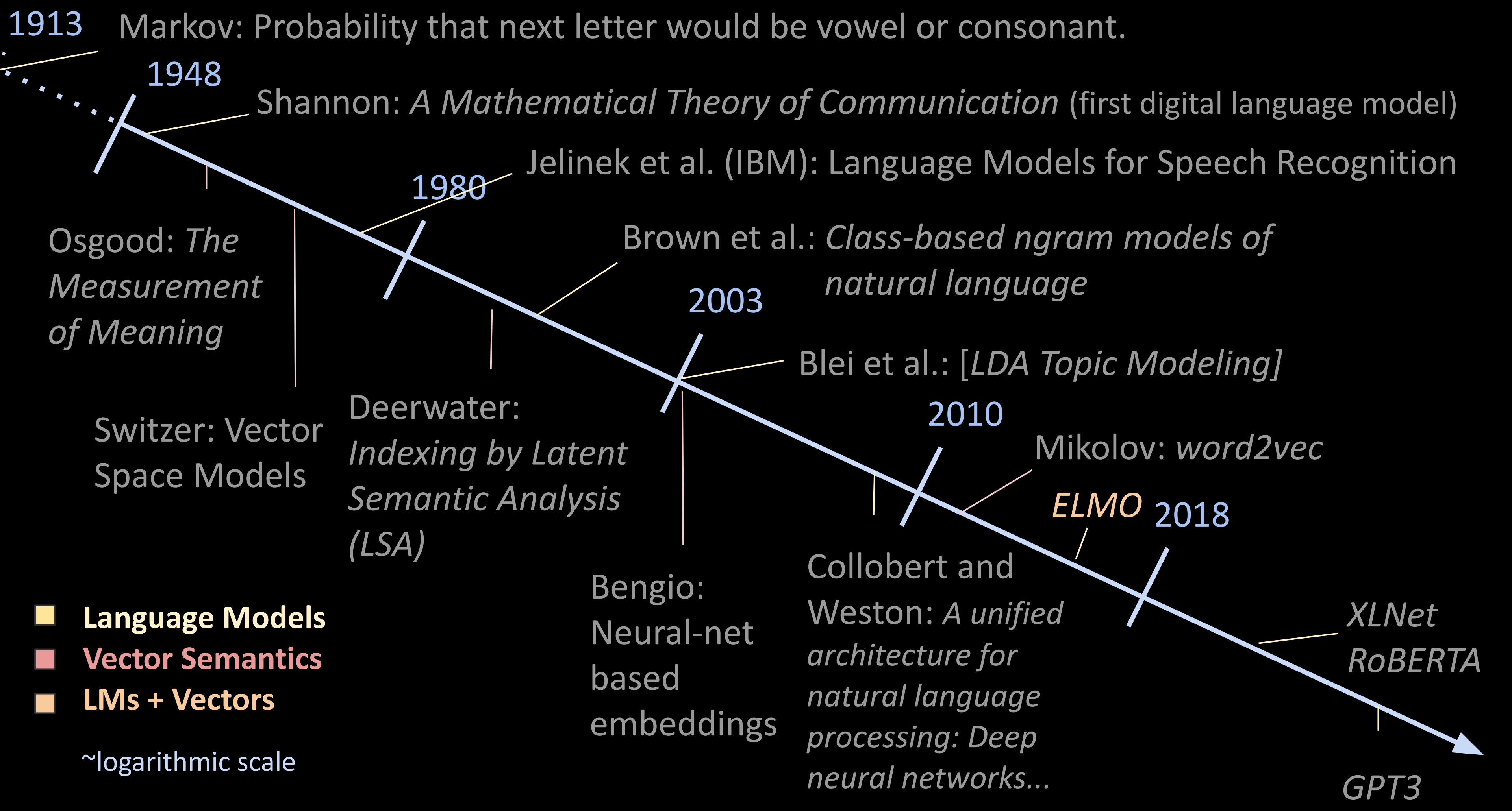
...logarithmic scale


odeling and Vector Semantics
1913
r would be vowel or consonant.
l Theory of
ital language model)
et al. (IBM)
peech Recognition
rown et al
dels of
Os
M
of
20
Blei et al.: [LDA Topic Modeling]
2010
Mikolov: word2vec
2018
gio:
ural-net
ed
beddings
Collobert and
Weston: A unified
architecture for
natural language
processing: Deep
neural networks...
XLNet
RoBERTA
GPT3
Input Sentence
n words, K features
feature 1 (text)
the cat sat on the mat
feature 2
s1(1) s1(2) s1(3) s1(4) s1(5) s1(6)
...
feature K
sK(1) sK(2) sK(3) sK(4) sK(5) sK(6)
word vectors
more neural networks
(capturing context)
#hidden units
Optional Classical NN Layer(s)
Word Probabilities
Deep Learning
for Language
Modeling!
(time for break)

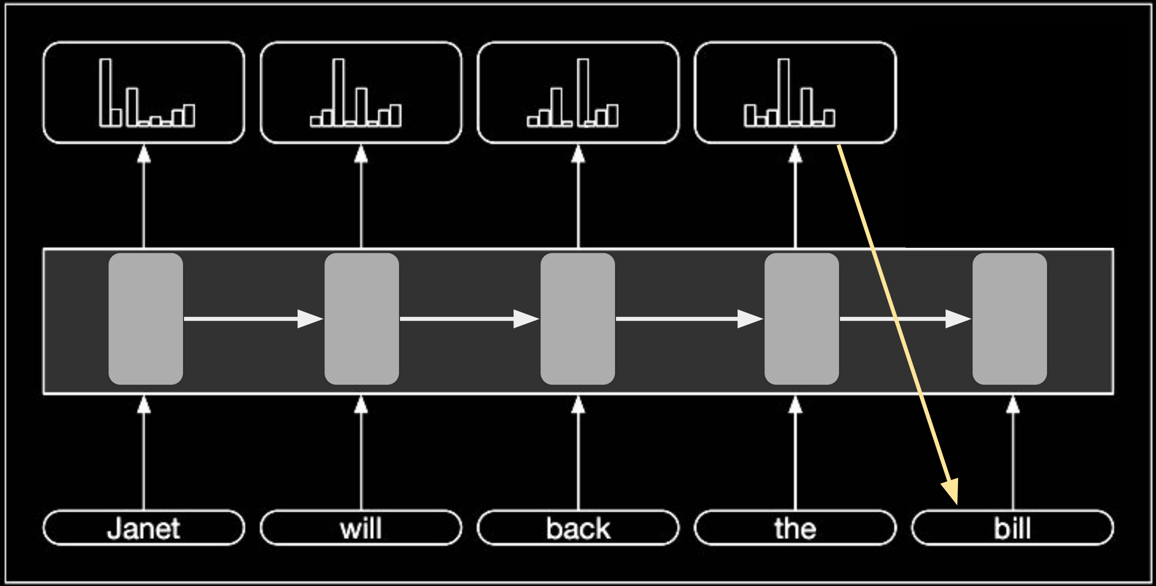
Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*

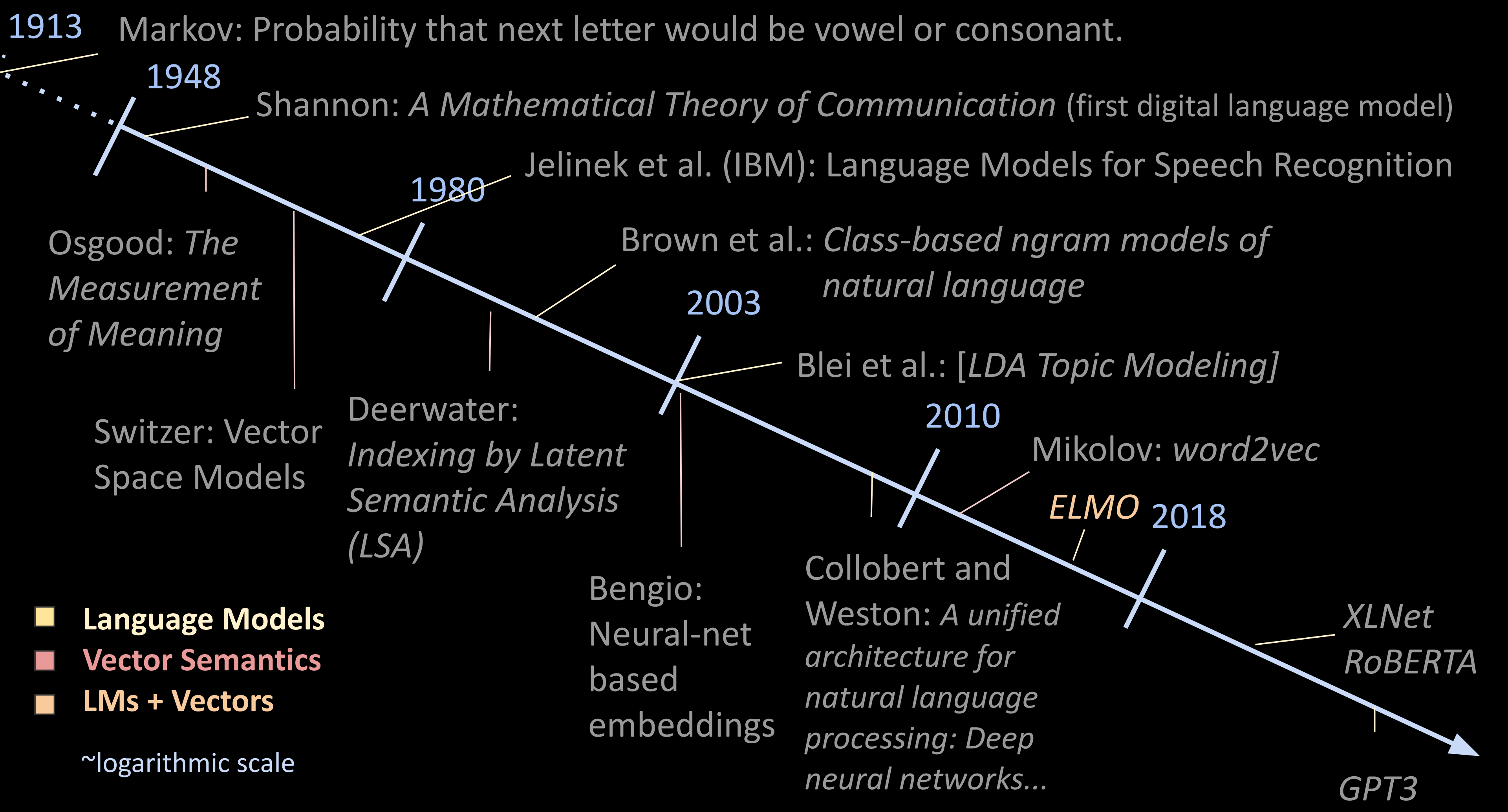


Recurrent Neural Network

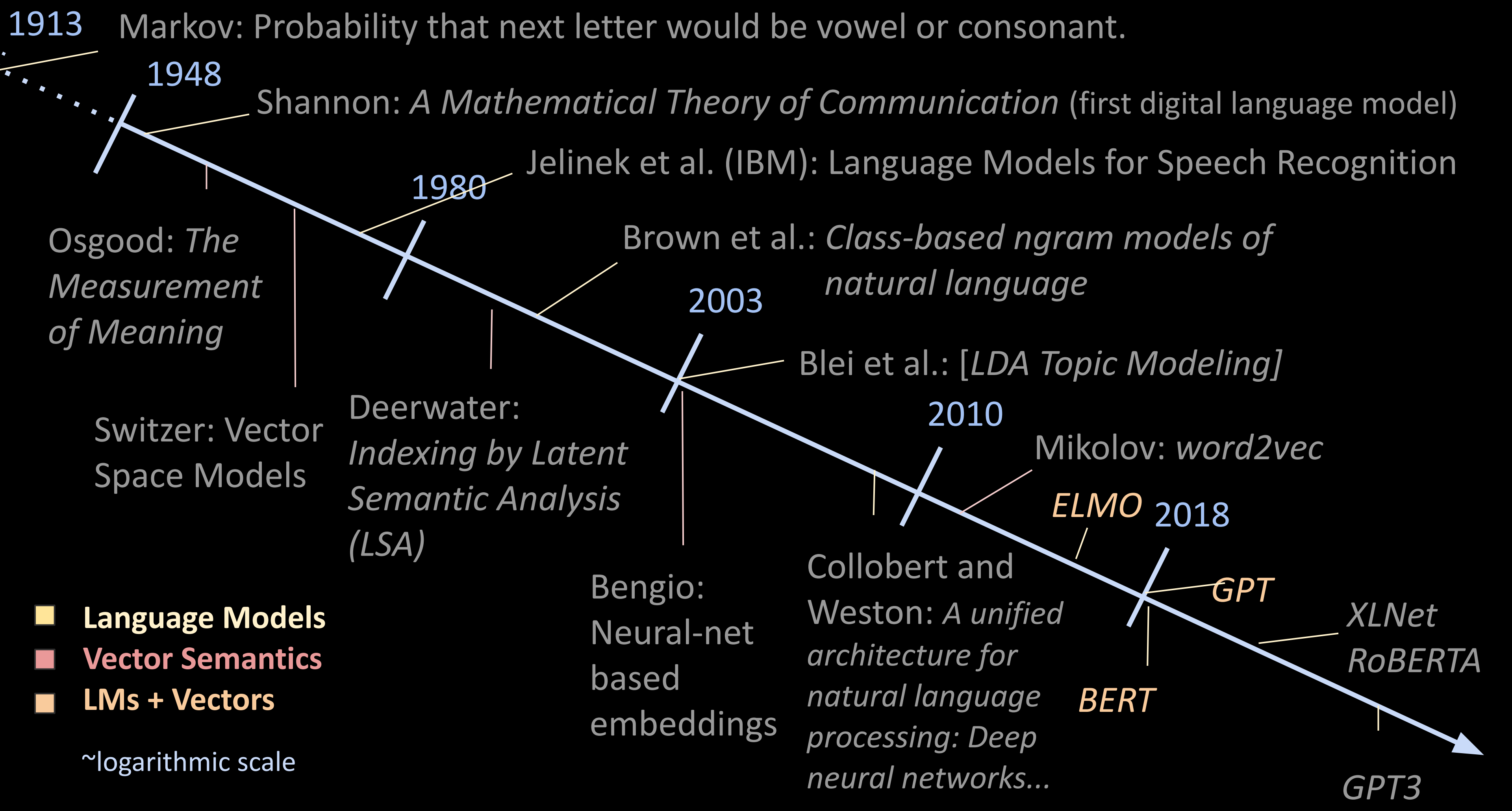


Language modeling
with an RNN

Timeline: *Language Modeling* and *Vector Semantics*



Timeline: *Language Modeling* and *Vector Semantics*



The Transformer: Motivation

Challenges to sequential representation learning

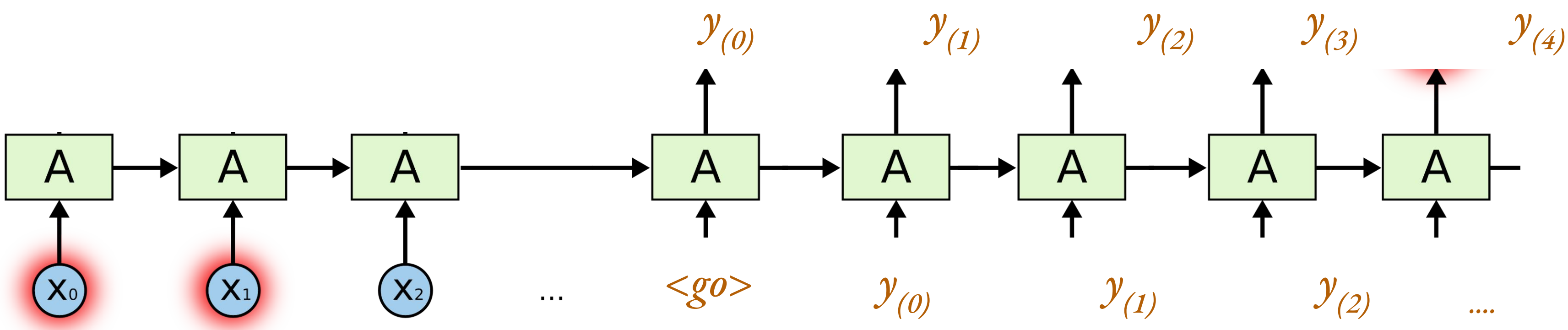
- Capture long-distance dependencies
- Preserving sequential distances / periodicity
- Capture multiple relationships
- Easy to parallelize -- don’t need sequential processing.

The Transformer: Attention-only Models

Challenge:

The ball was kicked by kayla.

- Long distance dependency when translating:



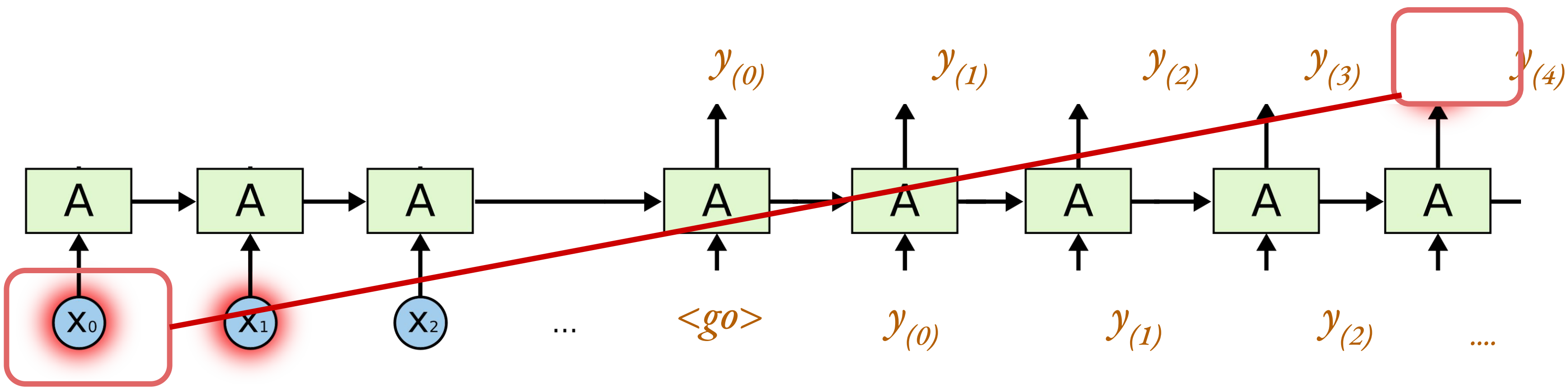
Kayla kicked the ball.

The Transformer: Attention-only Models

Challenge:

The ball was kicked by kayla.

- Long distance dependency when translating:



Kayla kicked the ball.

Transformer Language Models: Uses multiple layers of a **transformer**

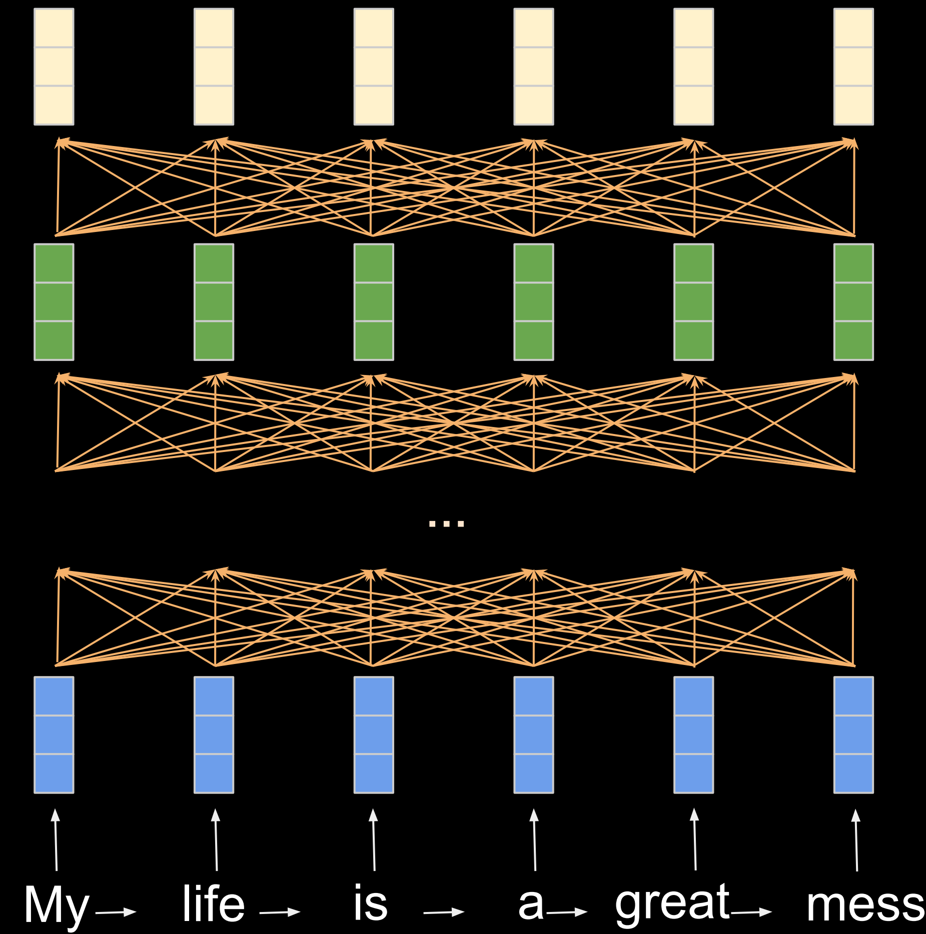
layer k:
(used for language modeling)

layer k-1:
(taken as contextual embedding)

layers 1 to k-2:
(compose embeddings with context)

layer 0:
(input: word-type embeddings)

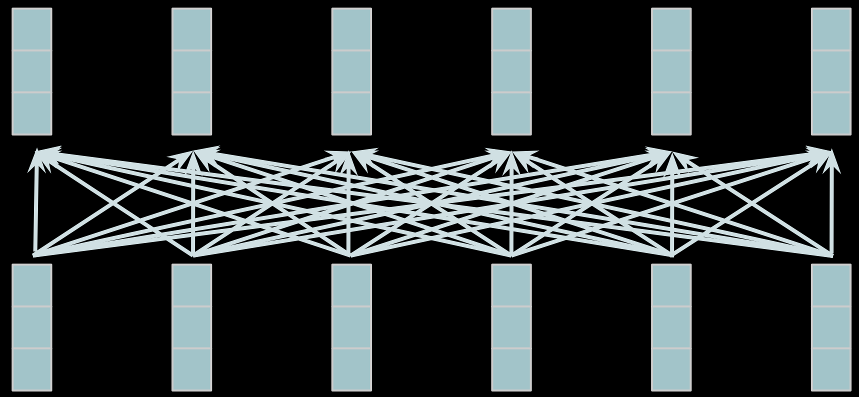
sentence (sequence) input:



(Kjell, Kjell, and Schwartz, 2023)

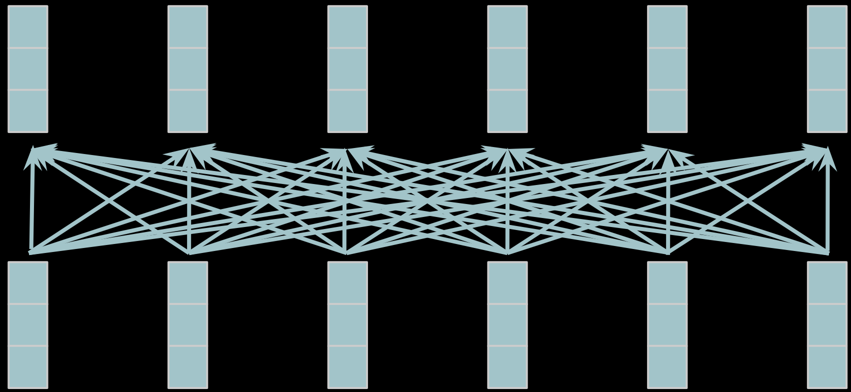
auto-encoder:

- Connections go both directions.
- Task is predict word in middle: p(wi| …, pwi-2, wi-1, wi+1, wi+2…)
- Better for:
  - embeddings
  - fine-tuning (transfer learning)



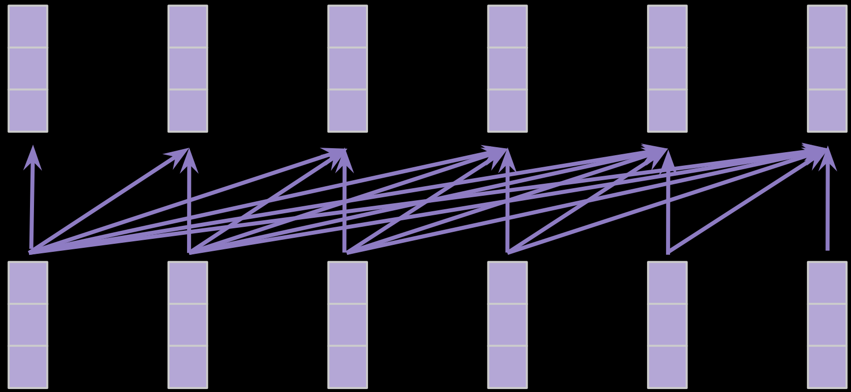
auto-encoder:

- Connections go both directions.
- Task is predict word in middle:
  p(wi| …, pwi-2, wi-1, wi+1, wi+2…)
- Better for:
  - embeddings
  - fine-tuning (transfer learning)



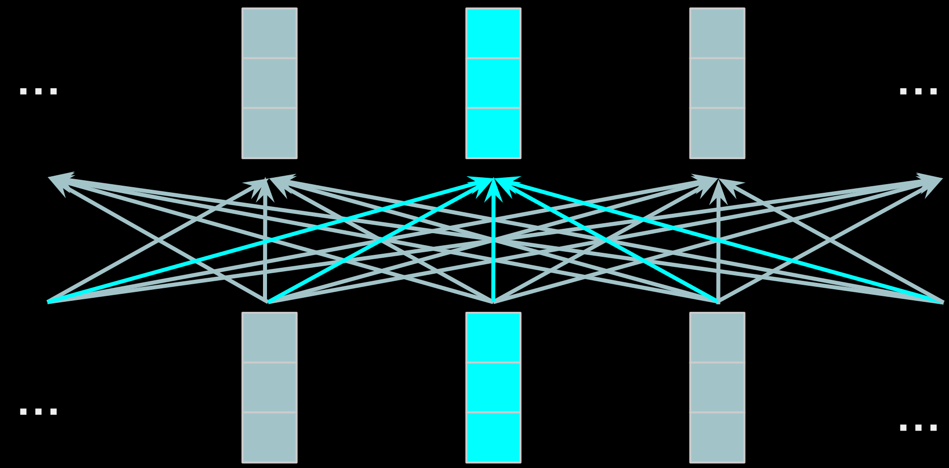
auto-regressor *(generator):*

- Connections go forward only
- Task is predict word next word:
  p(wi| wi-1, wi-2, …)
- Better for:
  - generating text
  - zero-shot learning



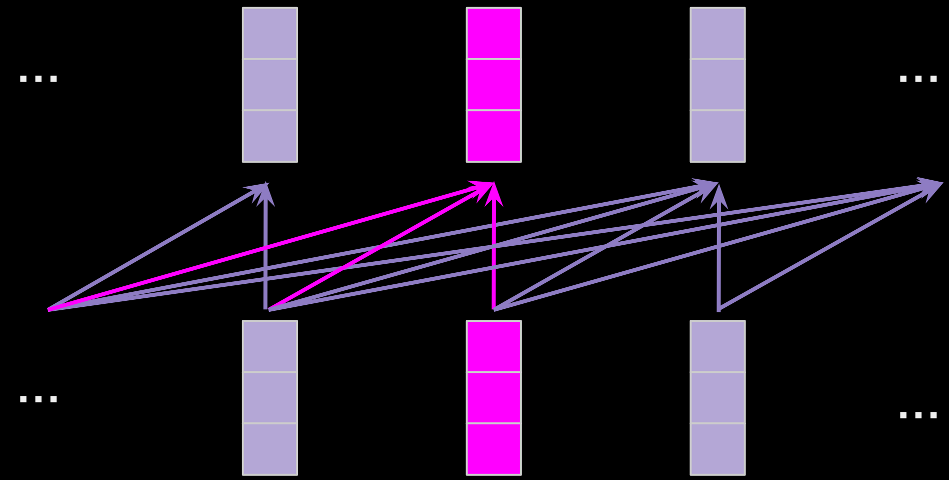
auto-encoder:

- Connections go both directions.
- Task is predict word in middle: p(wi| …, pwi-2, wi-1, wi+1, wi+2…)
- Better for:
  - embeddings
  - fine-tuning (transfer learning)

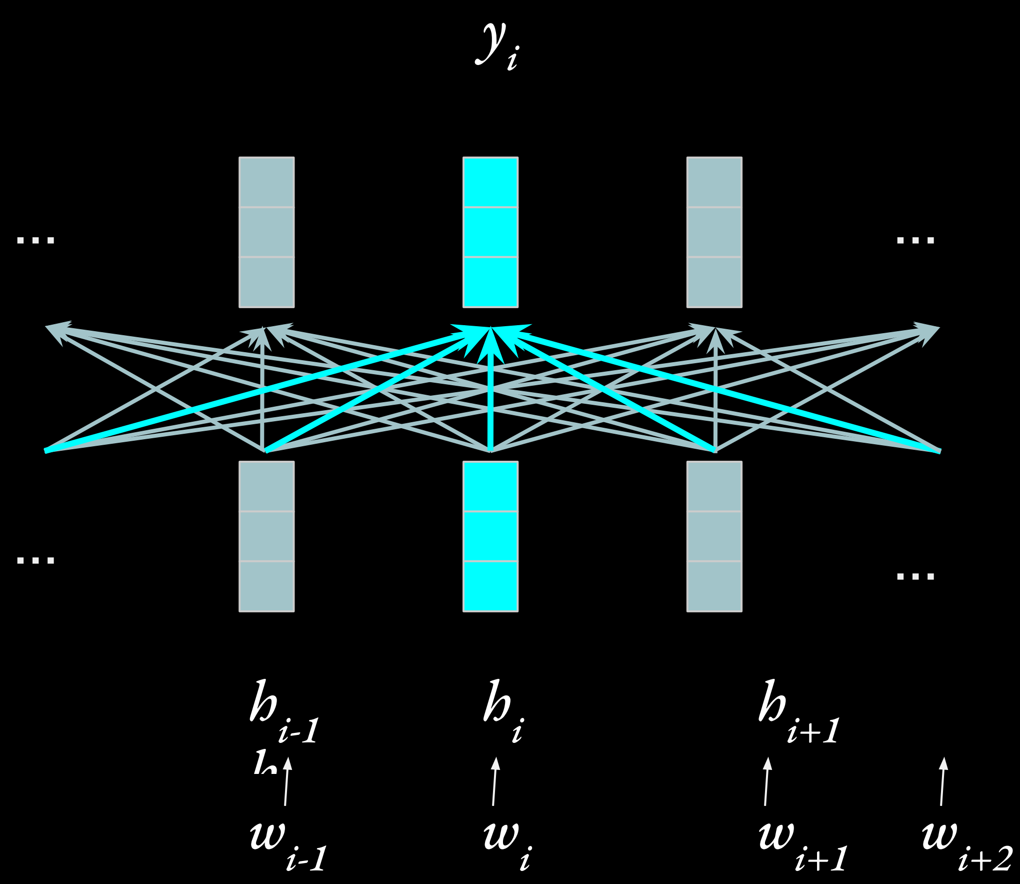


auto-regressor *(generator):*

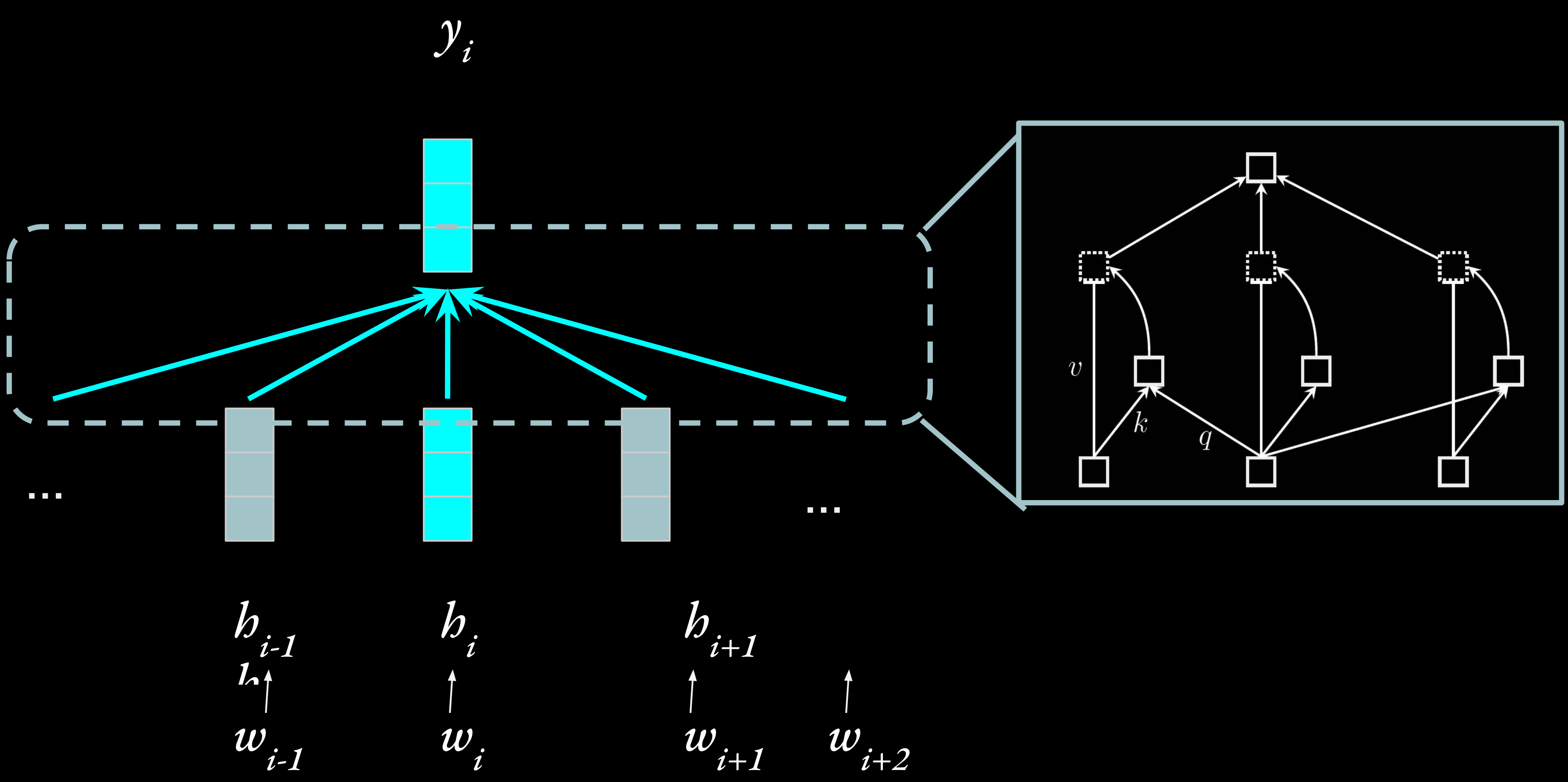
- Connections go forward only
- Task is predict word next word: p(wi| wi-1, wi-2, …)
- Better for:
  - generating text
  - zero-shot learning



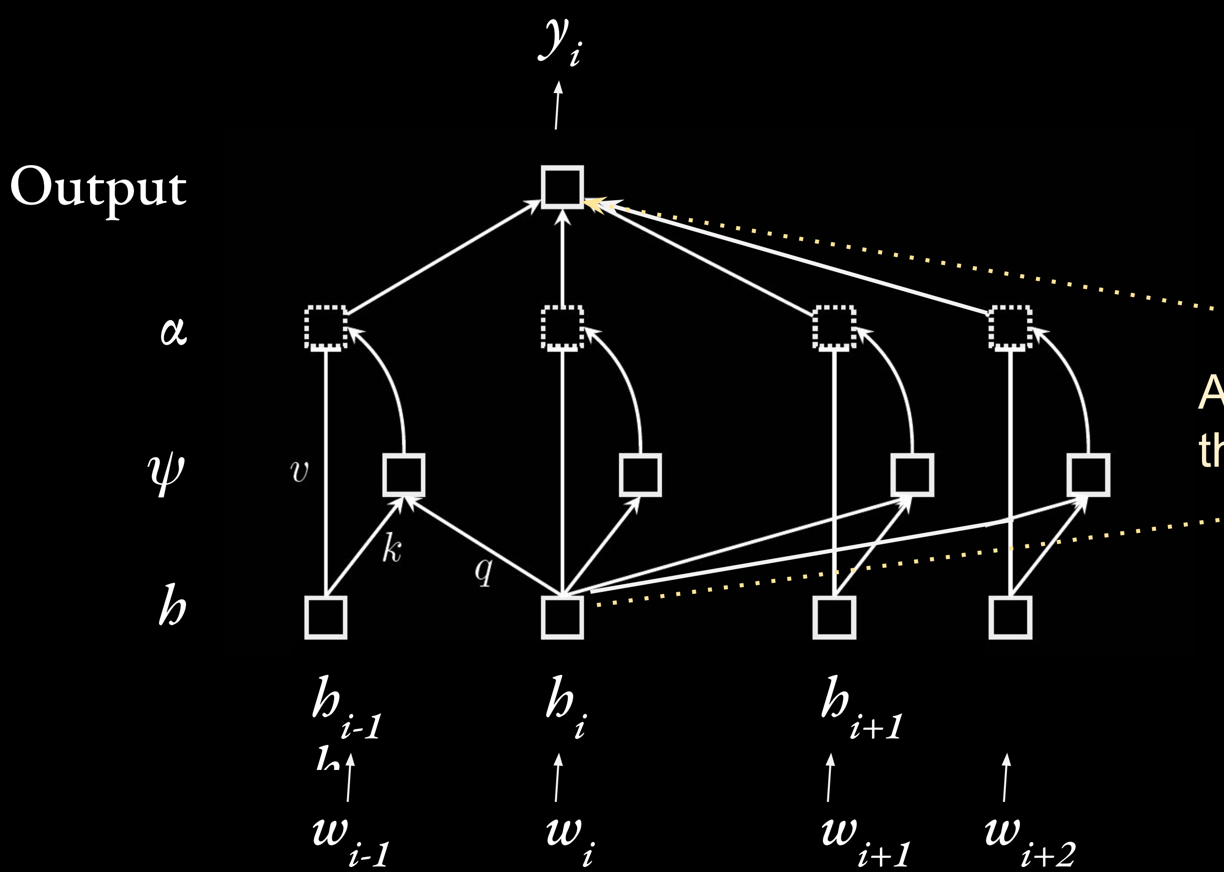
The Transformer's Heart: Self-Attention



The Transformer's Heart: Self-Attention

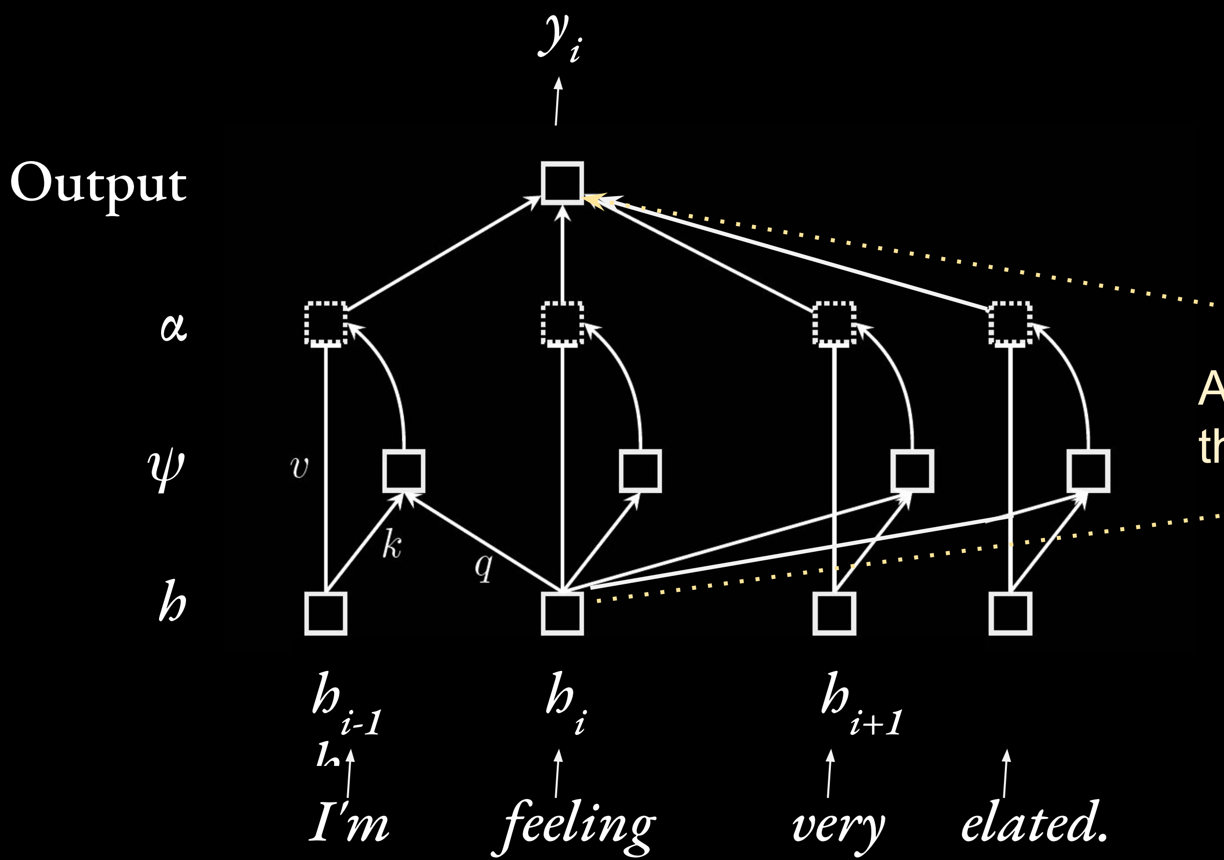


The Transformer's Heart: Self-Attention



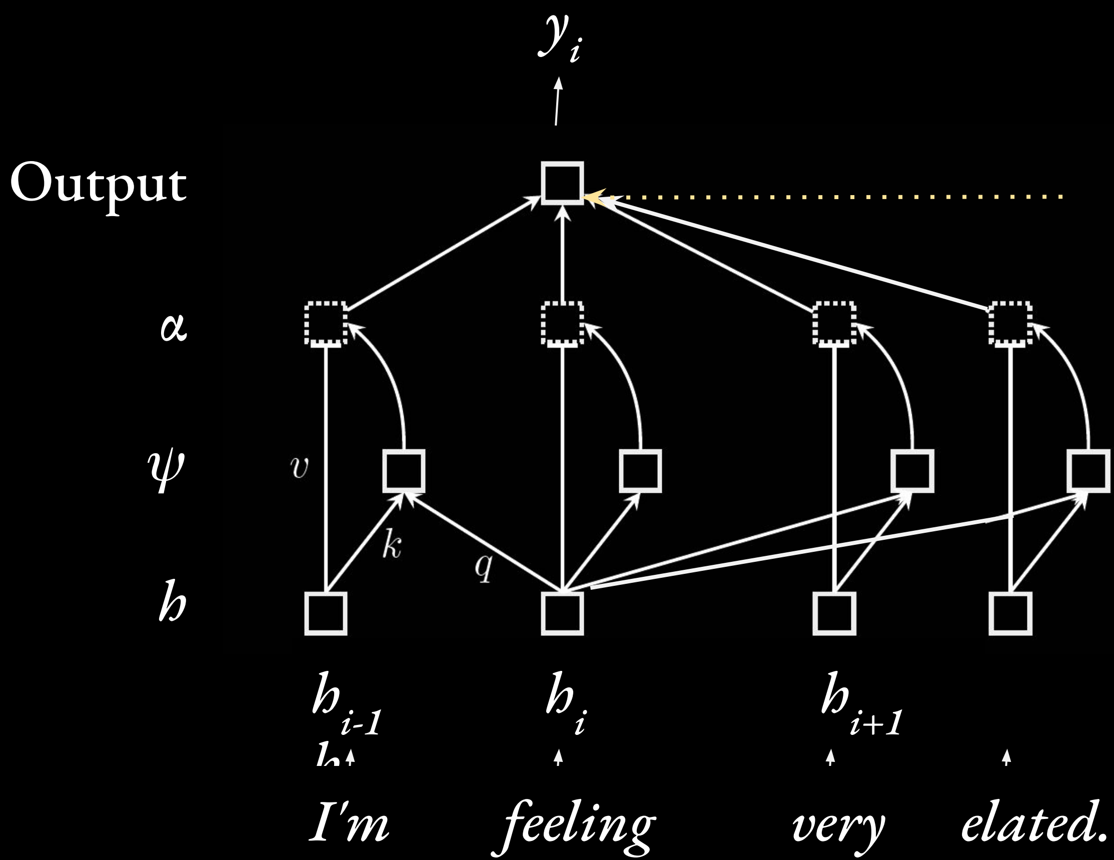
Attend to all other words in the sequence

The Transformer's Heart: Self-Attention



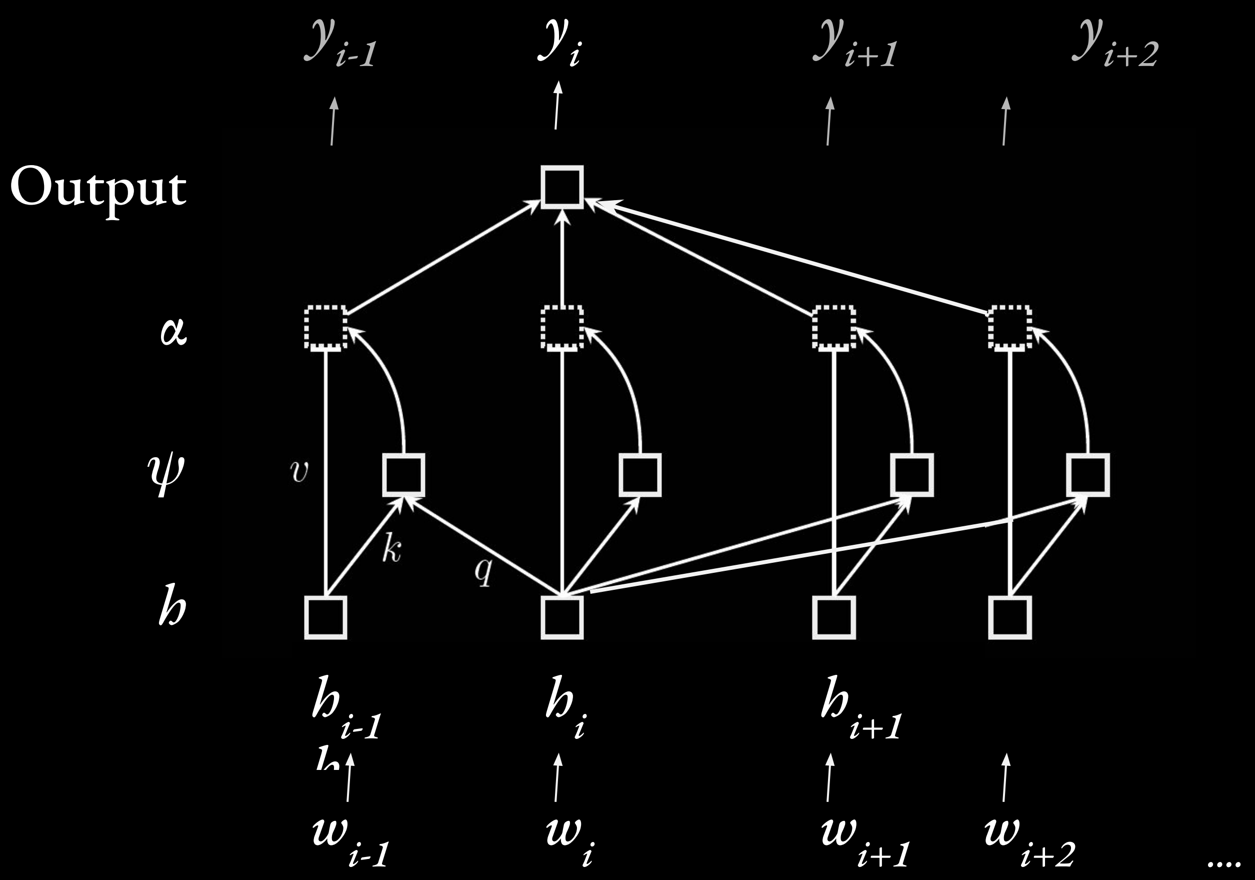
Attend to all other words in the sequence

The Transformer's Heart: Self-Attention

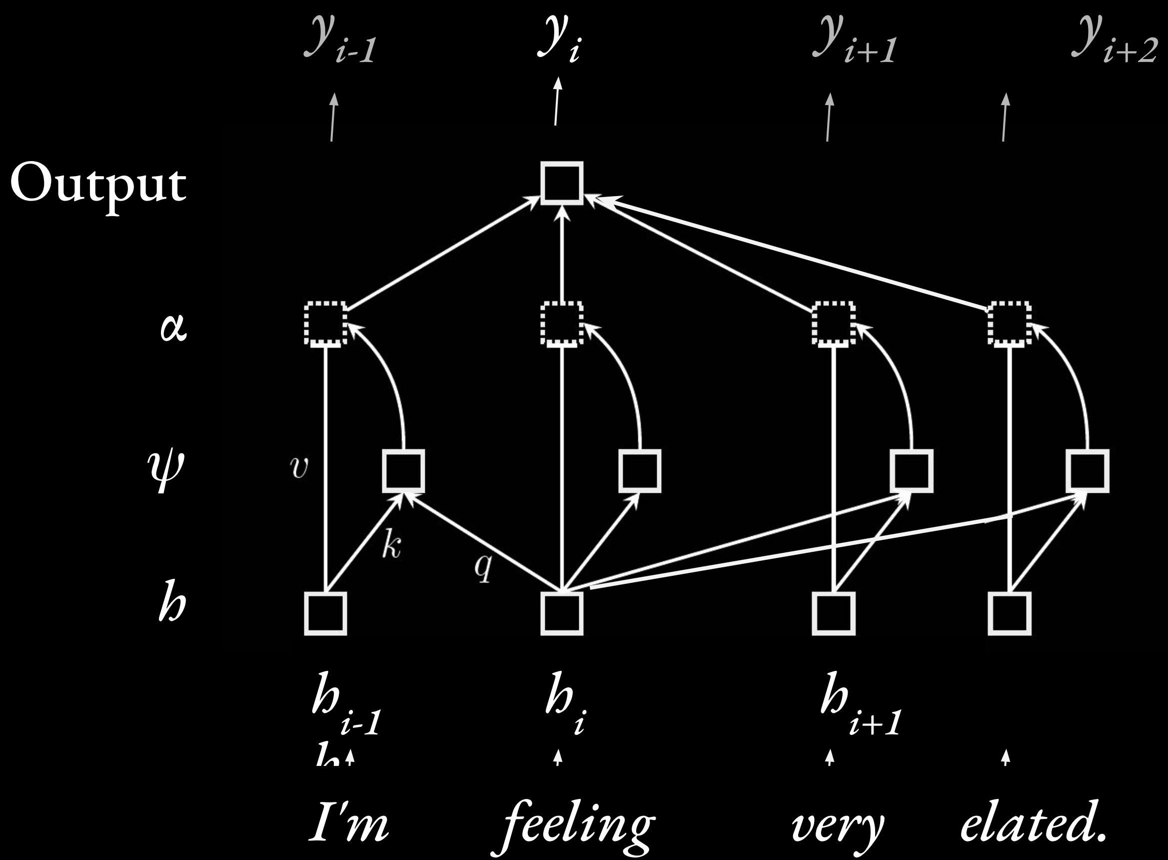


A weighted combination of other words' vectors.

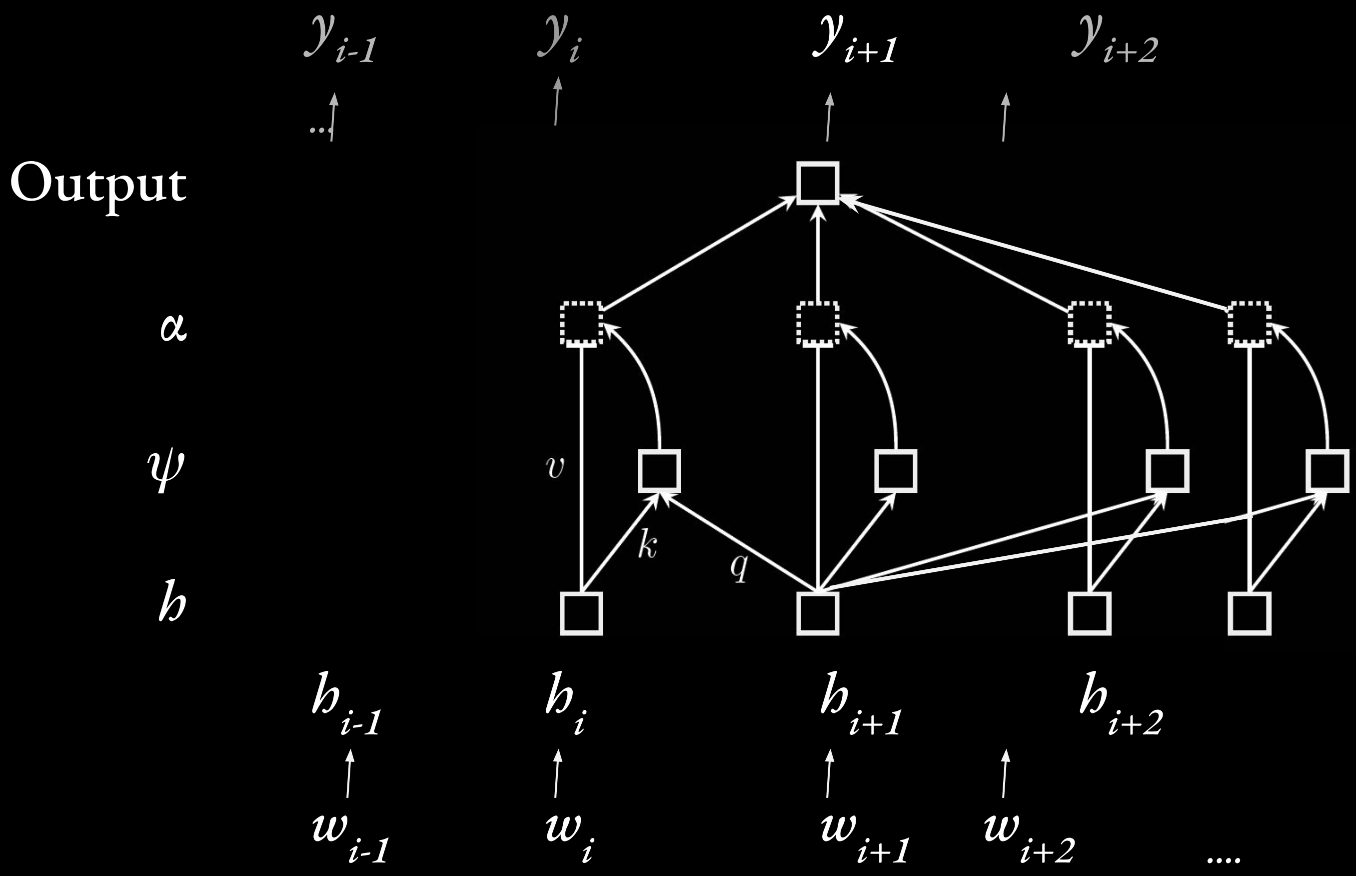
The Transformer's Heart: Self-Attention



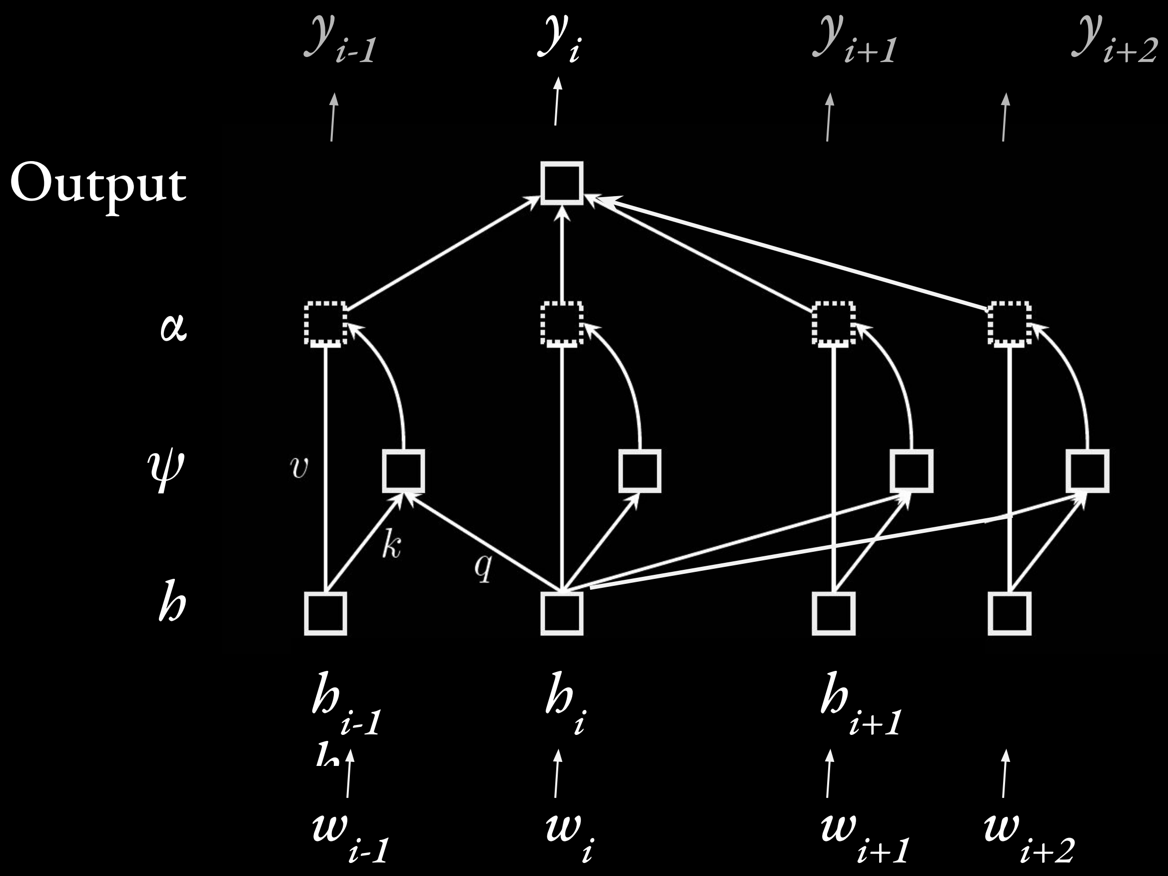
The Transformer's Heart: Self-Attention



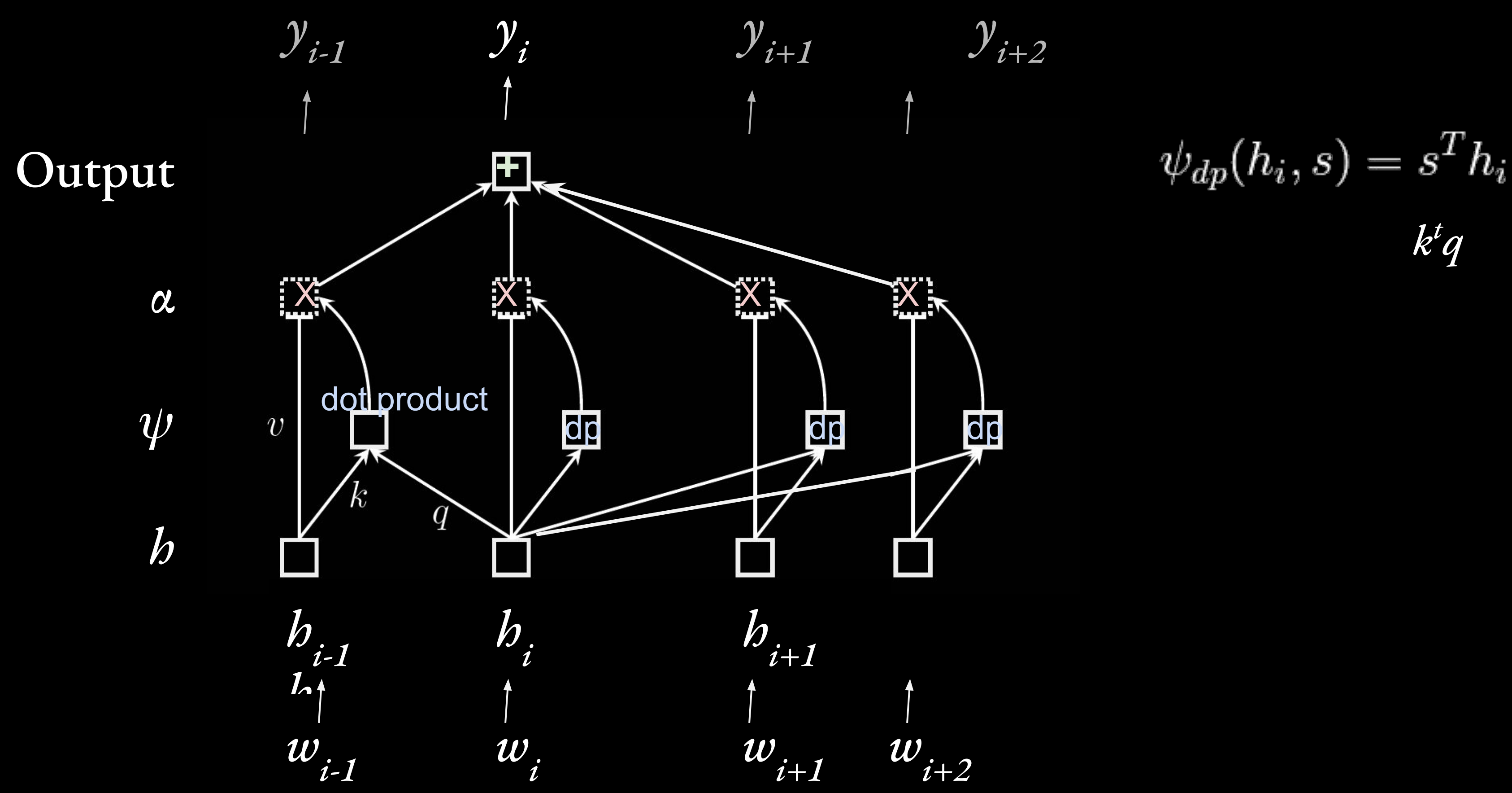
The Transformer's Heart: Self-Attention



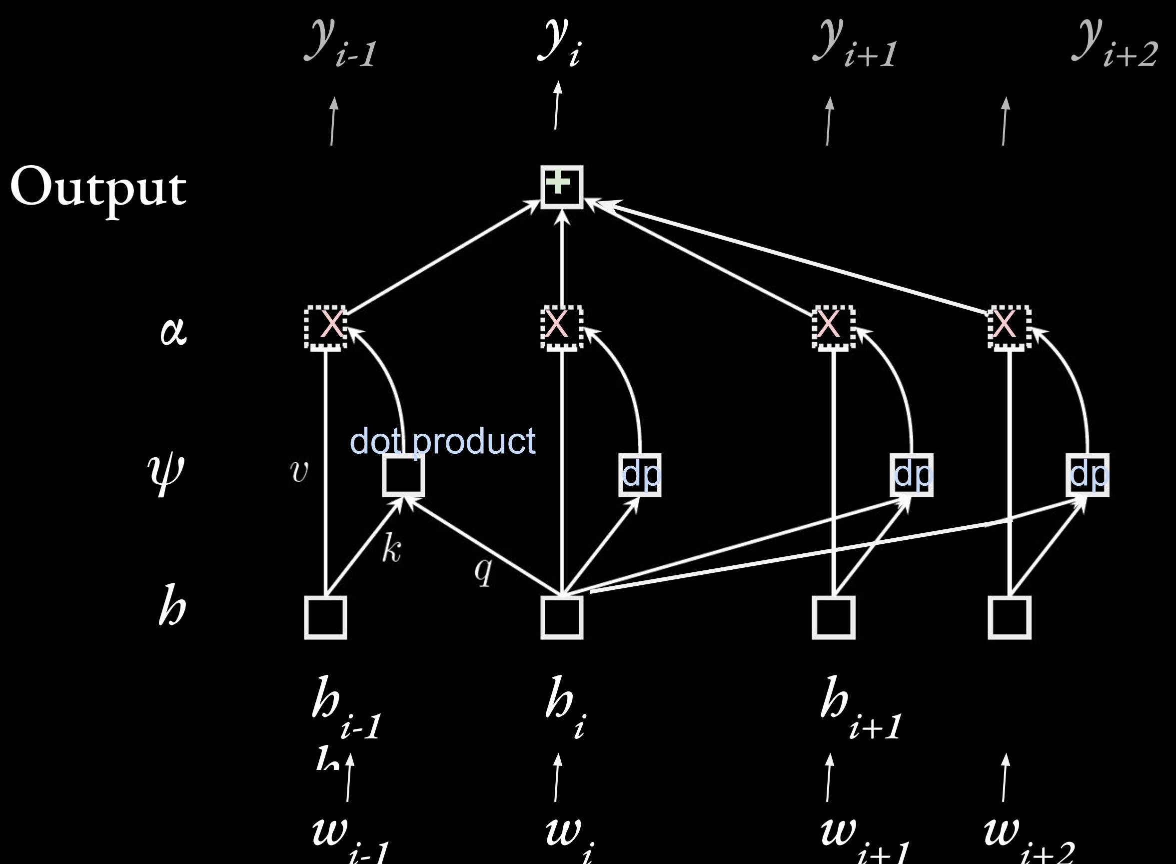
The Transformer's Heart: Self-Attention



The Transformer's Heart: Self-Attention



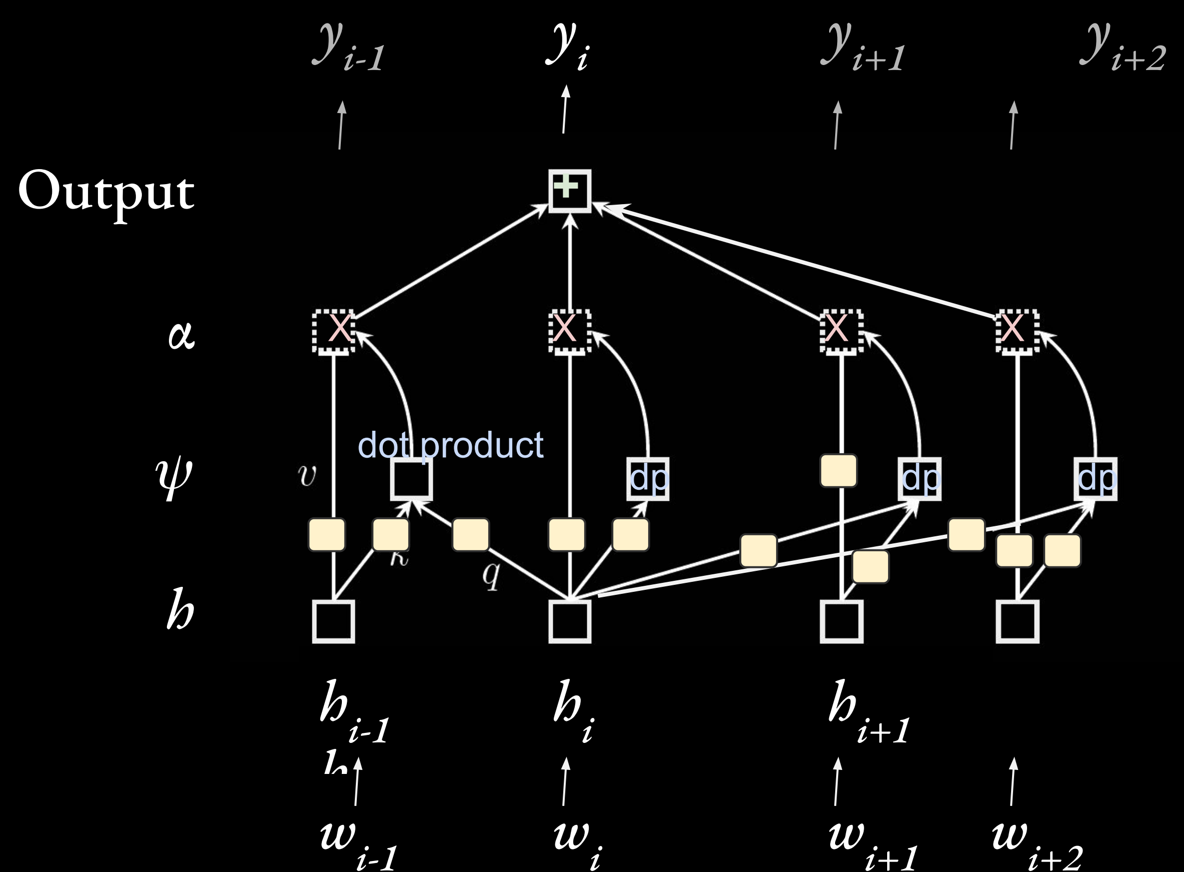
The Transformer's Heart: Self-Attention



scaling parameter

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

The Transformer's Heart: Self-Attention



$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

Linear layer:
$W^T X$

One set of weights for each of for K, Q, and V

Self-Attention in PyTorch

```
import nn.functional as f
class SelfAttention(nn.Module):
    def __init__(self, h_dim:int):
        self.Q = nn.Linear(h_dim, h_dim) #1 head
        self.K = nn.Linear(h_dim, h_dim)
        self.V = nn.Linear(h_dim, h_dim)


    def forward(hidden_states:torch.Tensor):
        v = self.V(hidden_states)
        k = self.K(hidden_states)
        q = self.Q(hidden_states)
        attn_scores = torch.matmul(q, k.T)
        attn_probs = f.Softmax(attn_scores)

        context = torch.matmul(attn_probs, v)
        return context
```

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

Linear layer:
$W^T X$

One set of weights for each of for K, Q, and V

Self-Attention in PyTorch

```
import nn.functional as f
class SelfAttention(nn.Module):
    def __init__(self, h_dim:int):
        self.Q = nn.Linear(h_dim, h_dim) #1 head
        self.K = nn.Linear(h_dim, h_dim)
        self.V = nn.Linear(h_dim, h_dim)
        self.dropout = nn.dropout(p=0.1)

    def forward(hidden_states:torch.Tensor):
        v = self.V(hidden_states)
        k = self.K(hidden_states)
        q = self.Q(hidden_states)
        attn_scores = torch.matmul(q, k.T)
        attn_probs = f.Softmax(attn_scores)
        attn_probs = self.dropout(attn_probs)
        context = torch.matmul(attn_probs, v)
        return context
```

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

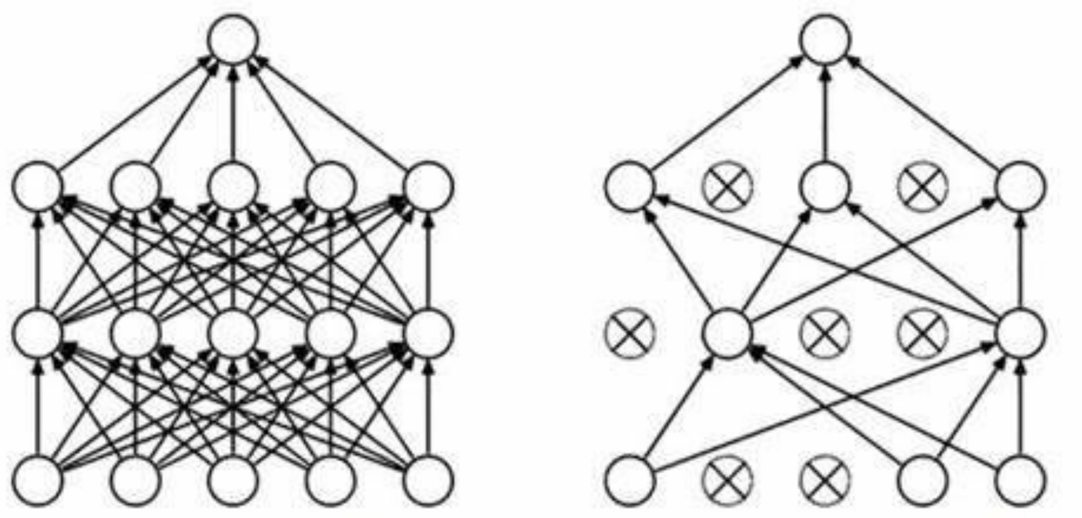
Linear layer:
$W^T X$

One set of weights for each of for K, Q, and V

Self-Attention in PyTorch

```
import nn.functional as f
class SelfAttention(nn.Module):
    def __init__(
        self.Q =
        self.K =
        self.V =
        self.drop

    def forward(h
        v = self.
        k = self.
        q = self.
        attn_scor
        attn_prob
        attn_probs = self.dropout(attn_probs)
        context = torch.matmul(attn_probs, v)
        return context
```



(a) Standard Neural Net

(b) After applying dropout.

$= (k^t q)\,\sigma$

ayer:

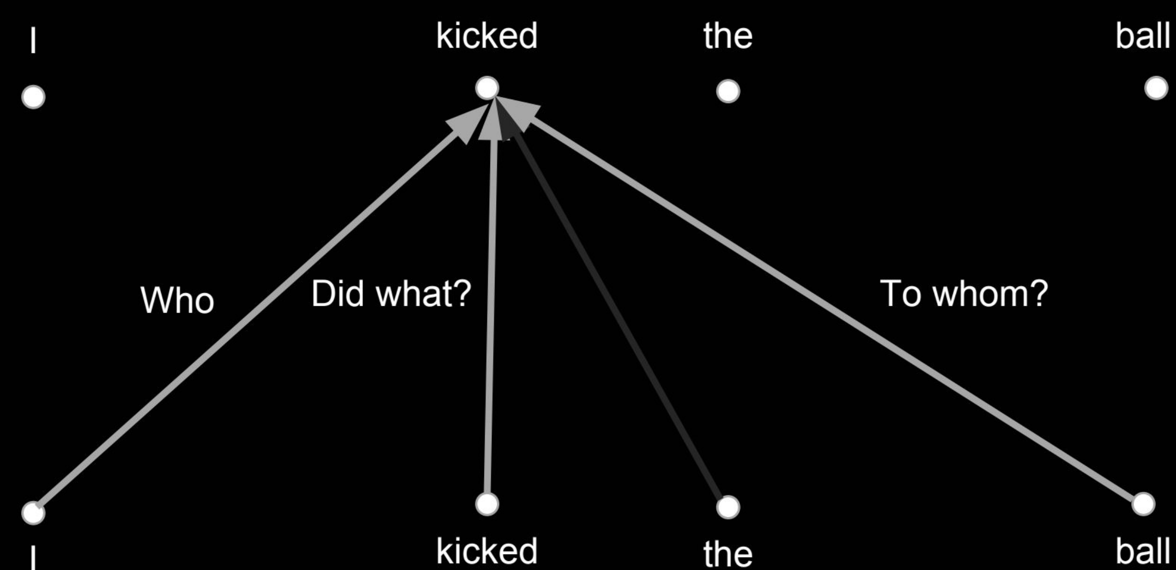
of weights

of for K,

V

The Transformer: Beyond Self-Attention

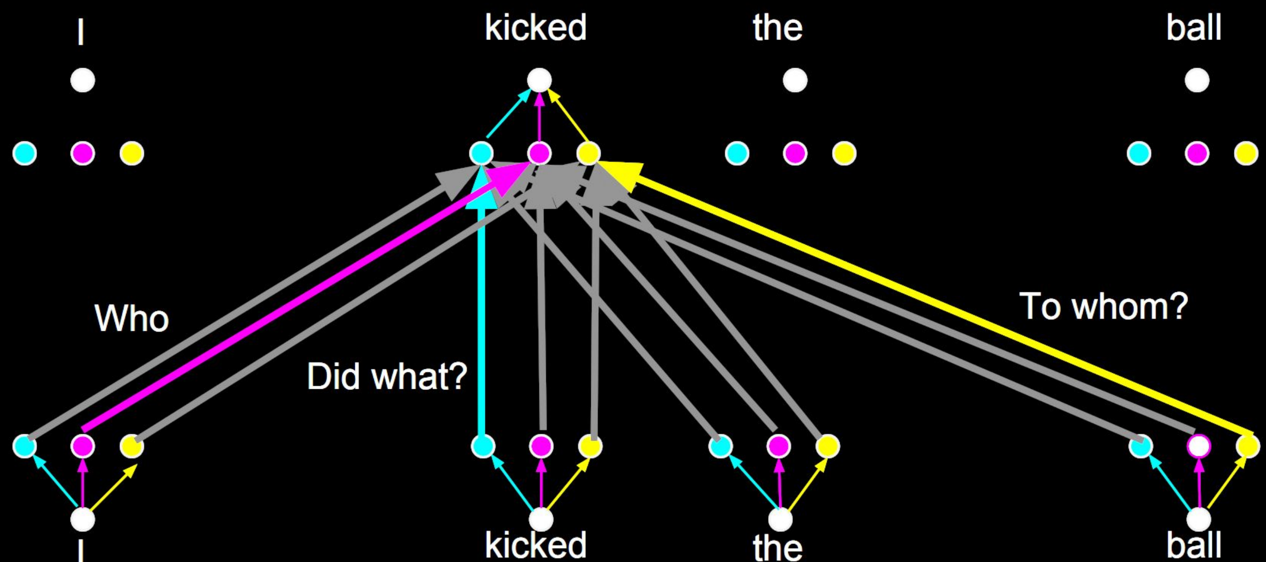
Limitation (thus far): Can't capture multiple types of dependencies between words.



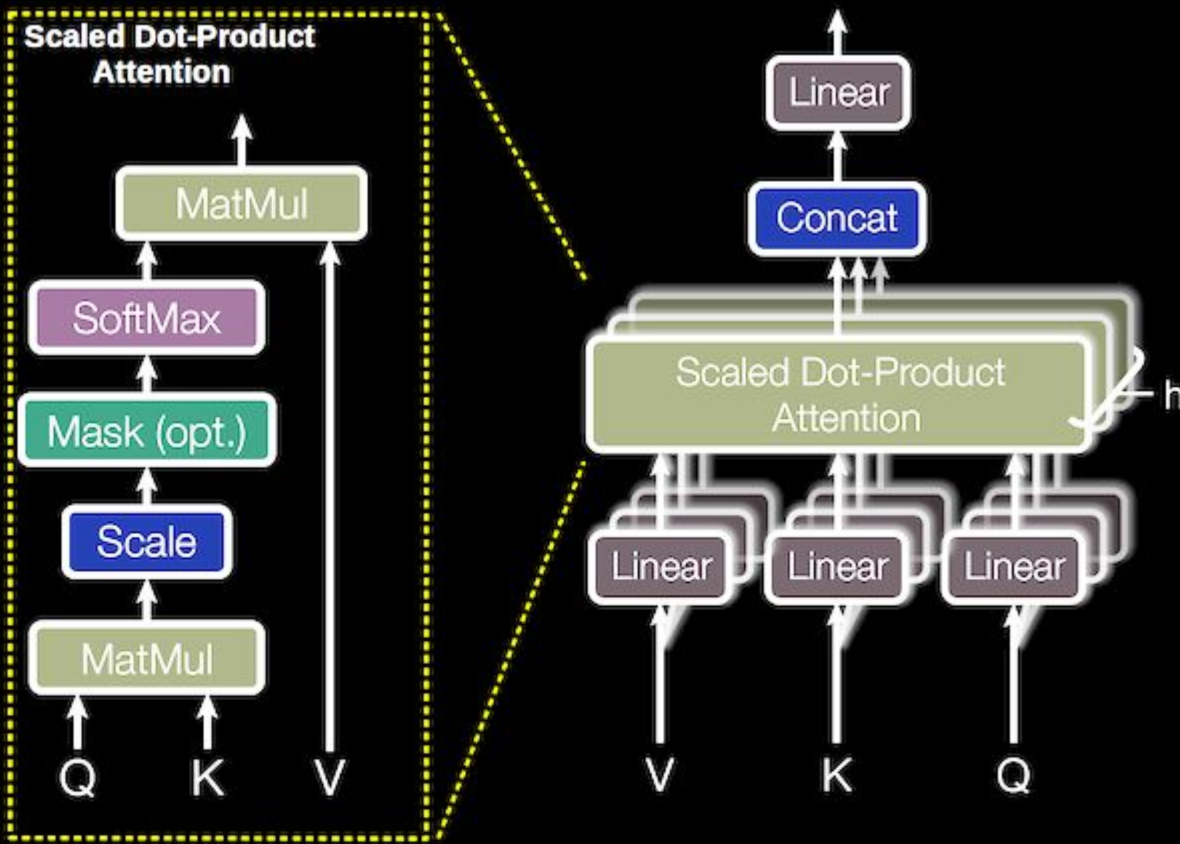
The Transformer: Beyond Self-Attention

Limitation (thus far): Can't capture multiple types of dependencies between words.

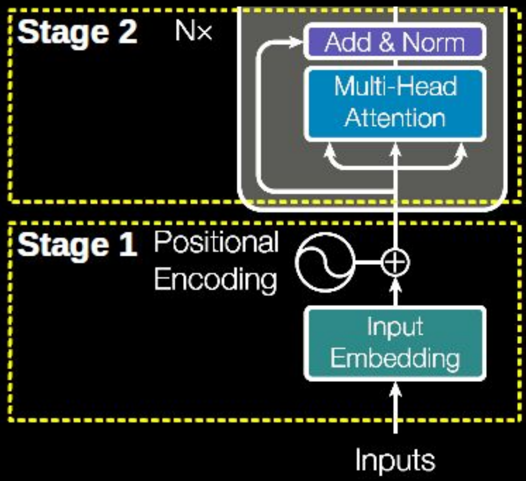
Solution: Multi-head attention



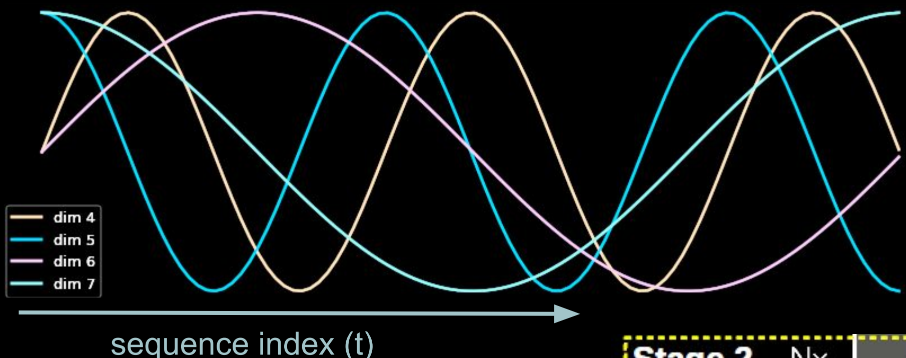
The Transformer: Muli-headed Attention

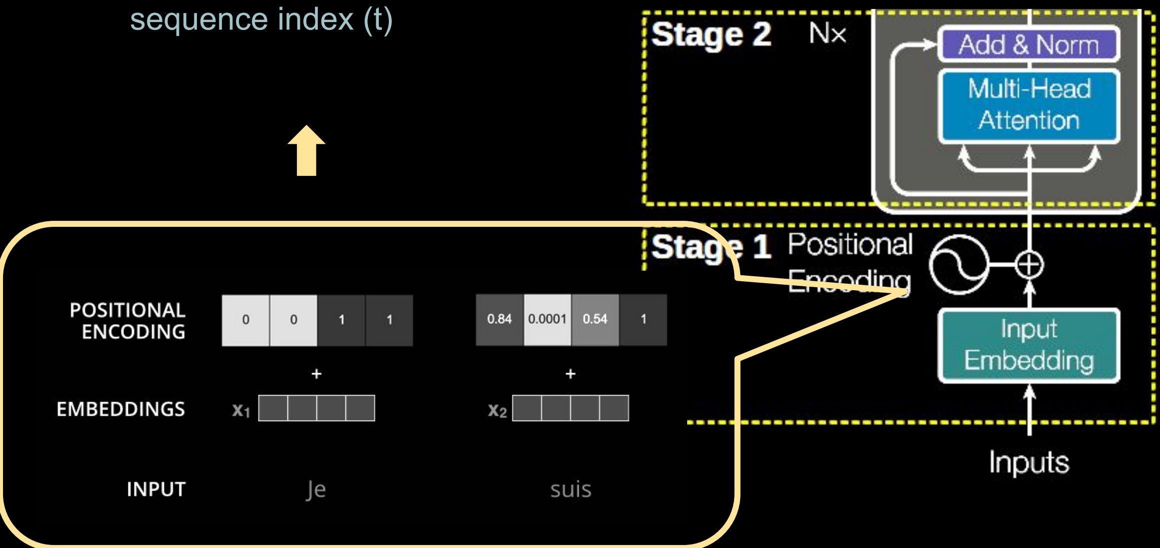


The Transformer

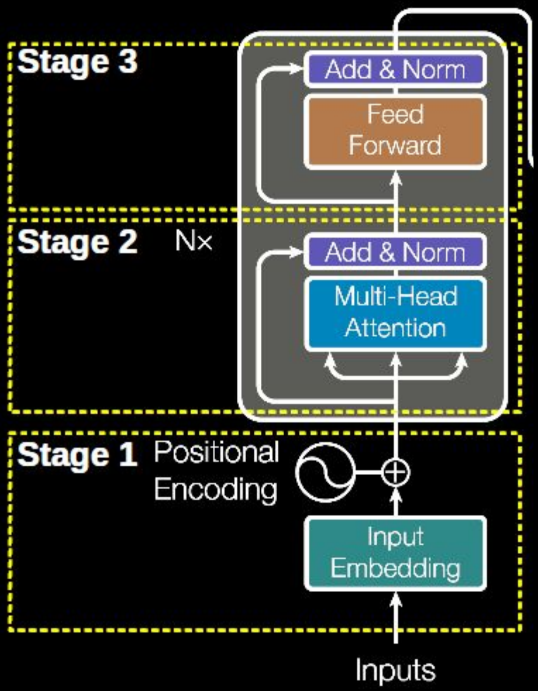


The Transformer

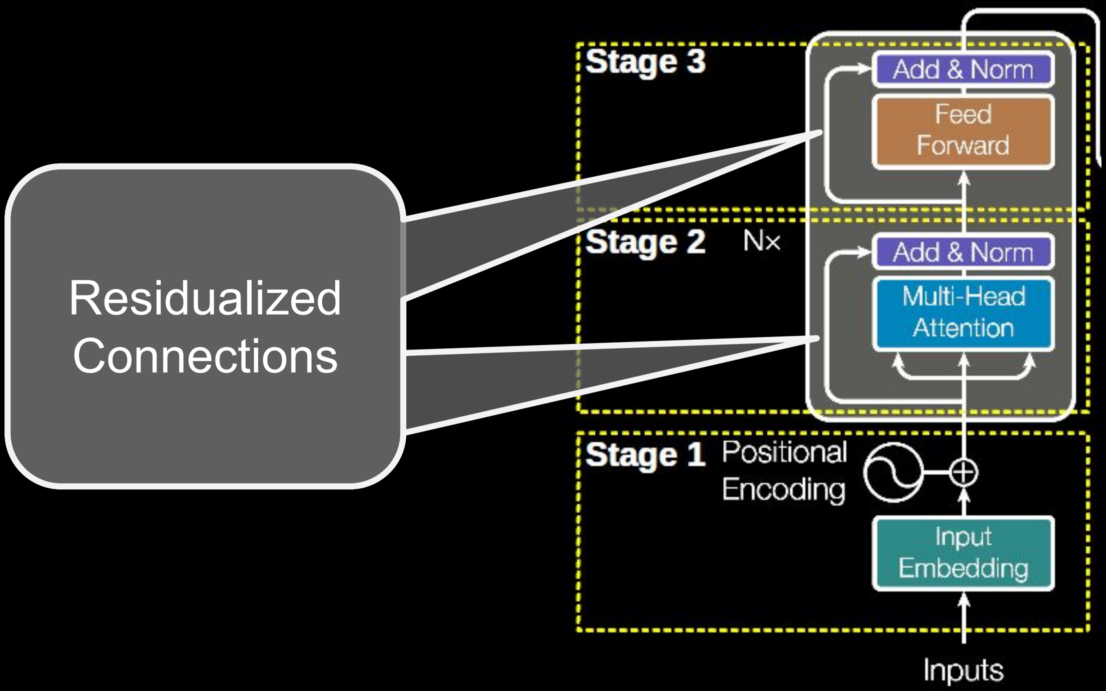




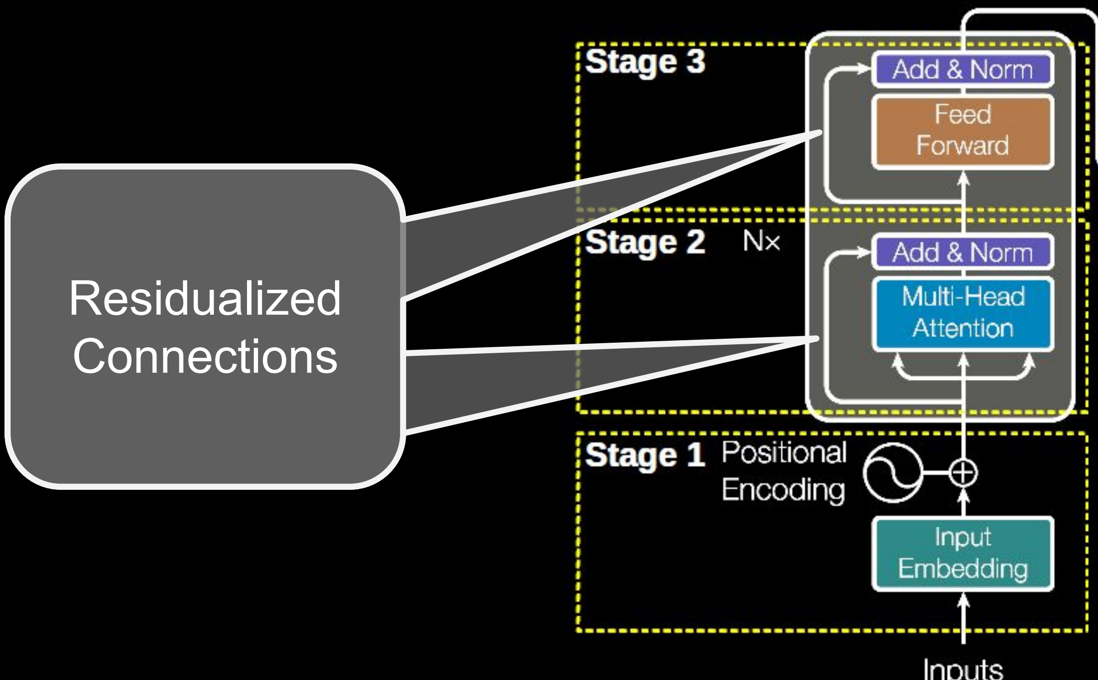
The Transformer



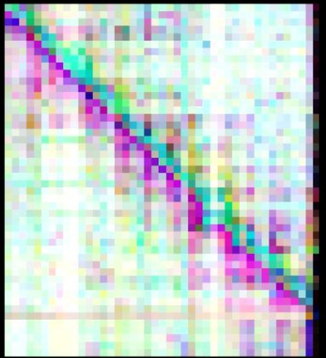
The Transformer



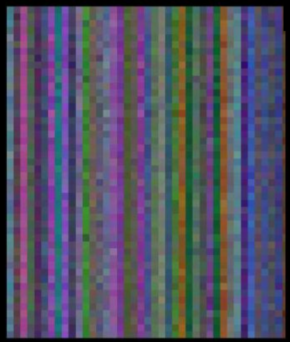
The Transformer



residuals enable positional information to be passed along



With residuals



Without residuals

The Transformer: Motivation

Challenges to sequential representation learning

- Capture long-distance dependencies
- Preserving sequential distances / periodicity
- Capture multiple relationships
- Easy to parallelize -- don’t need sequential processing.

The Transformer: Motivation

Challenges to sequential representation learning

- Capture long-distance dependencies
  Self-attention *treats far away words similar to those close.*
- Preserving sequential distances / periodicity
  Positional embeddings *encode distances/periods.*
- Capture multiple relationships
  Multi-headed attention *enables multiple compositions.*
- Easy to parallelize -- don't need sequential processing.
  Entire layer can be computed at once. Is only matrix multiplications + standardizing.

Transformer (as of 2017)

“WMT-2014” Data Set. BLEU scores:

	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

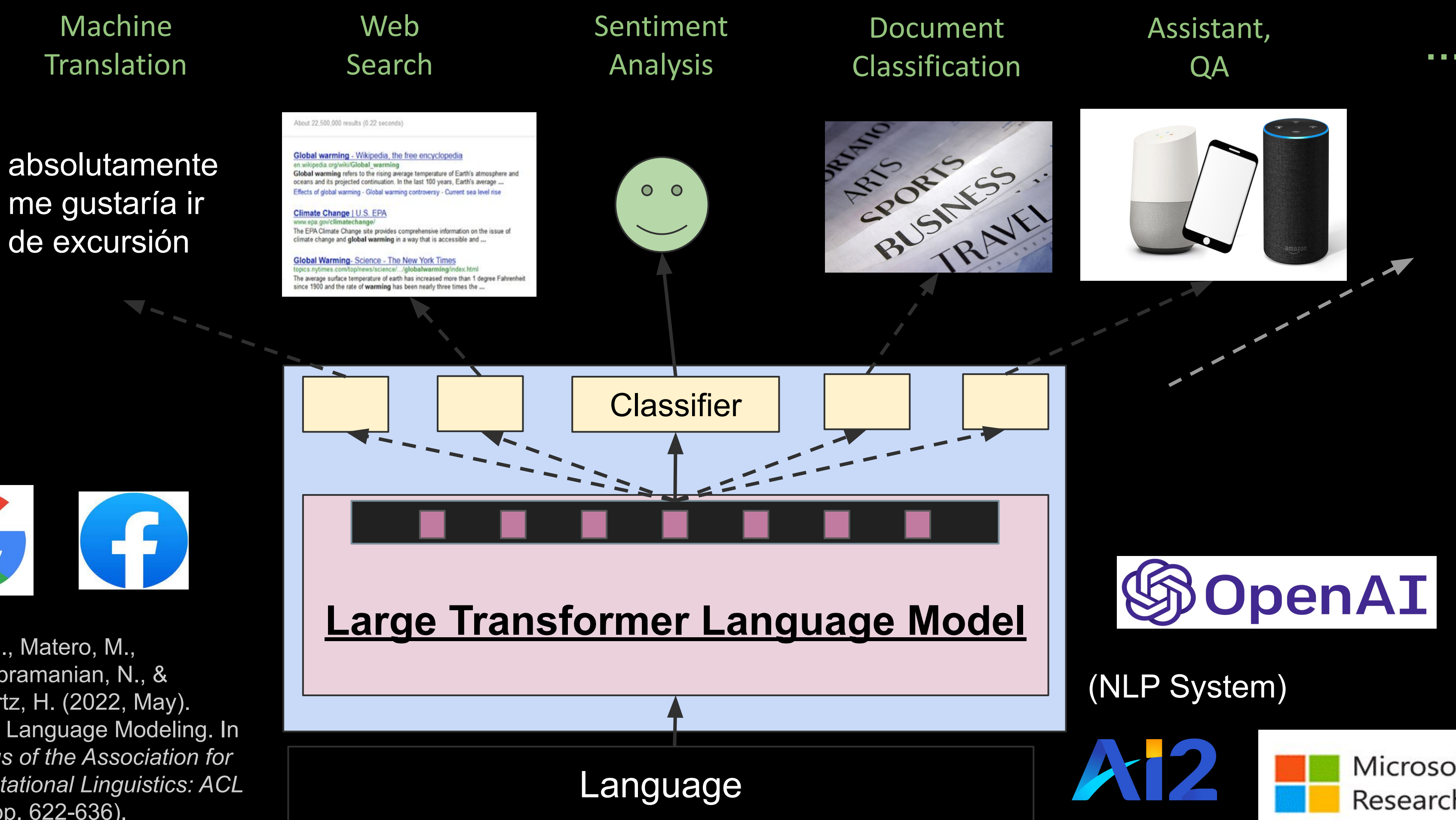
Transformers as of 2023

General Language Understanding Evaluations:

https://gluebenchmark.com/leaderboard

https://super.gluebenchmark.com/leaderboard/

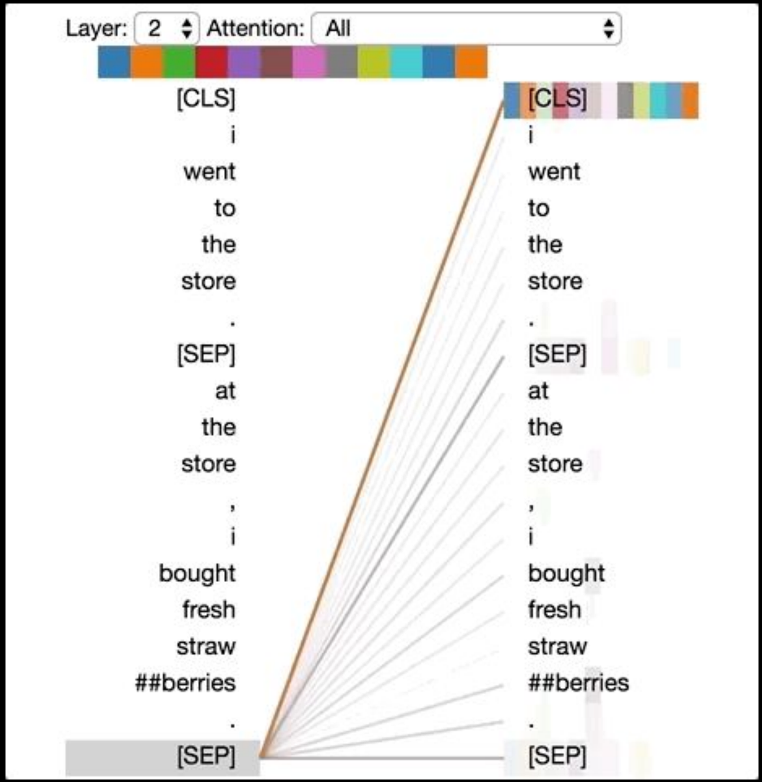
Transformers as of 2023



Soni, N., Matero, M., Balasubramanian, N., & Schwartz, H. (2022, May). Human Language Modeling. In *Findings of the Association for Computational Linguistics: ACL 2022* (pp. 622-636).

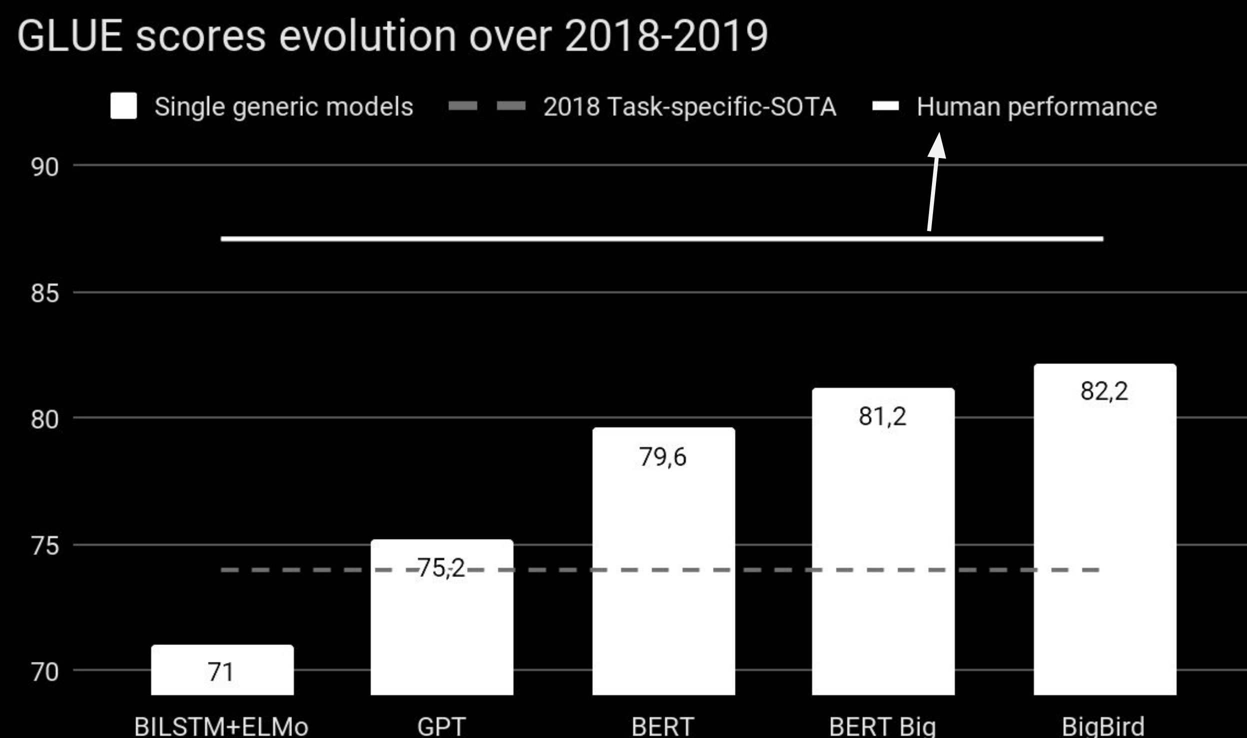
Bert: Attention by Layers

https://colab.research.google.com/drive/1vlOJ1IhdujVjfH857hvYKIdKPTD9Kid8



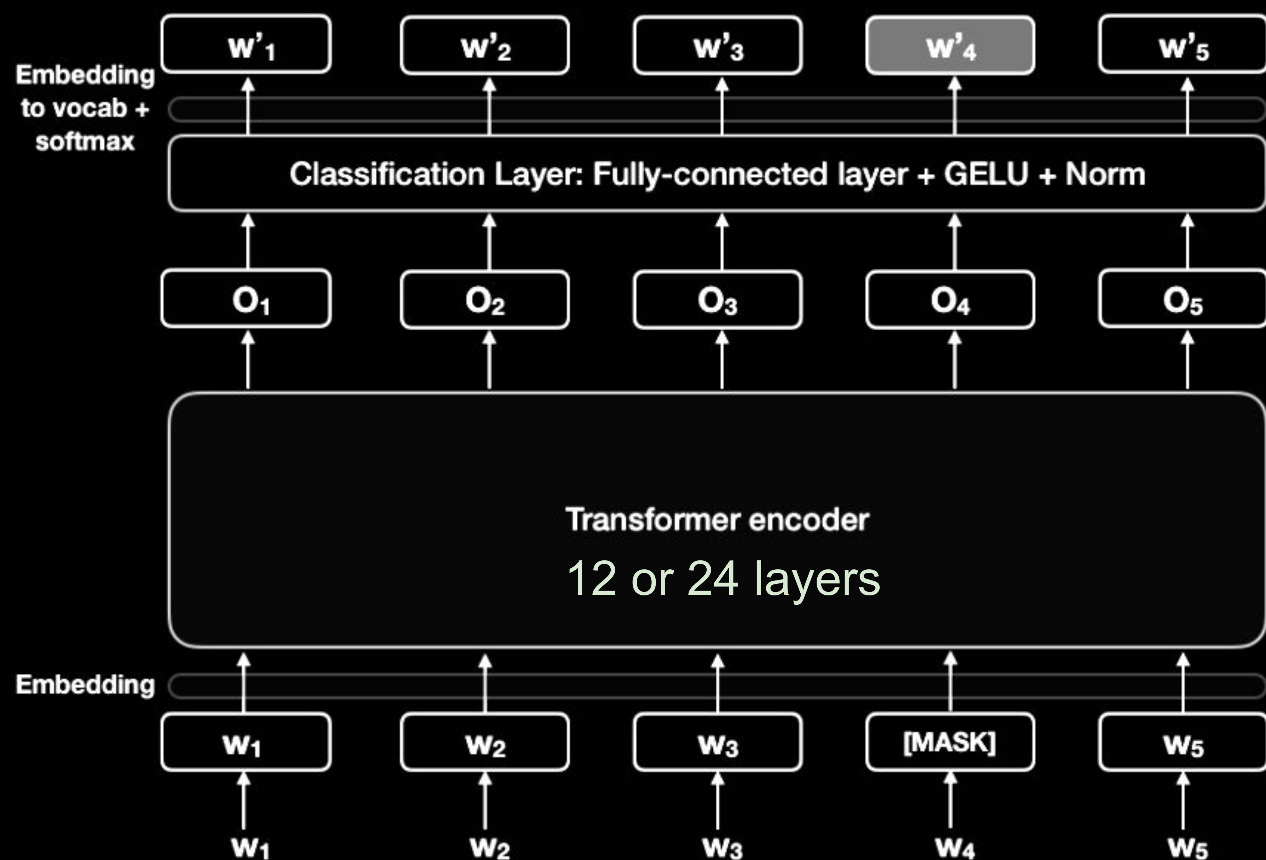
(Vig, 2019)

BERT Performance: e.g. Question Answering

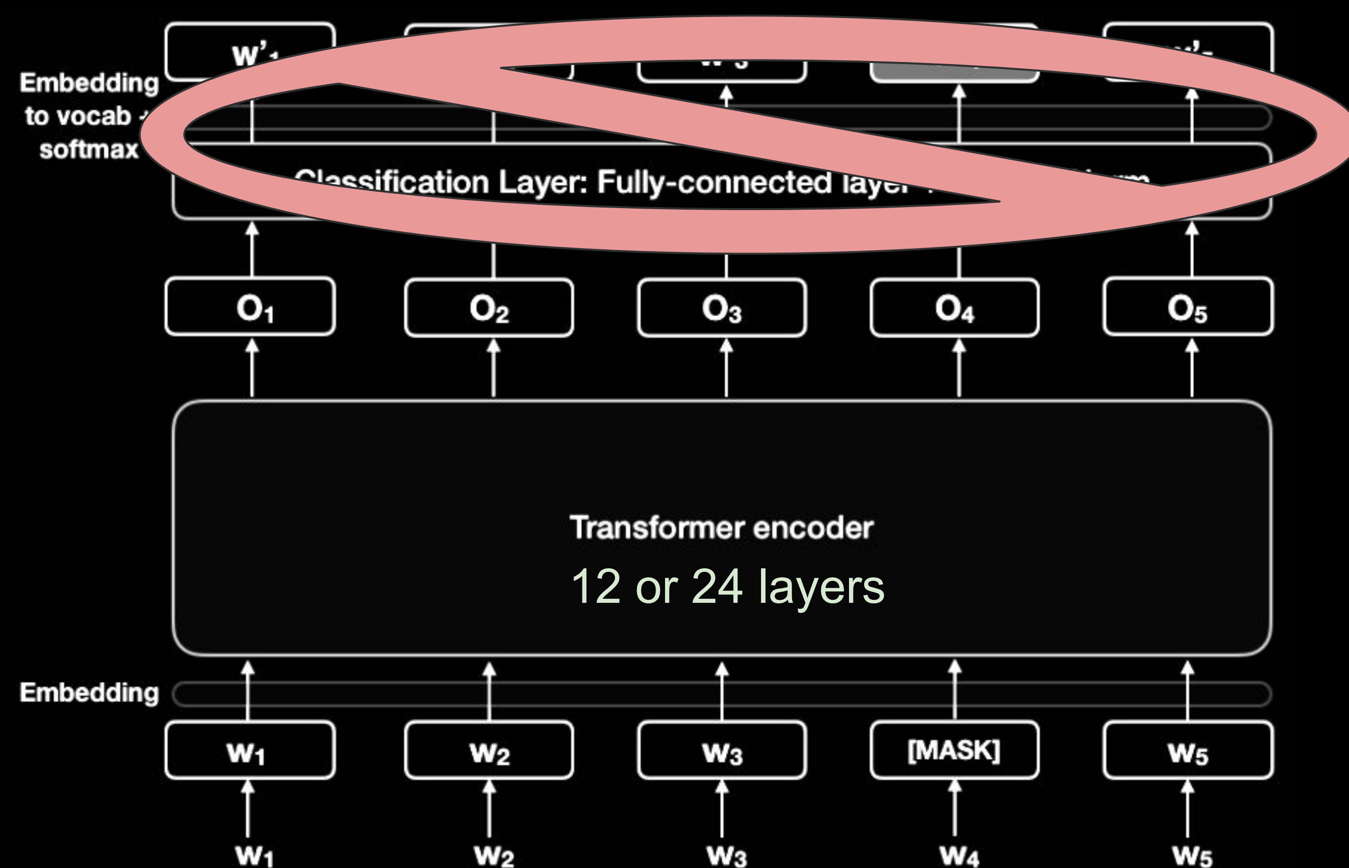


https://rajpurkar.github.io/SQuAD-explorer/

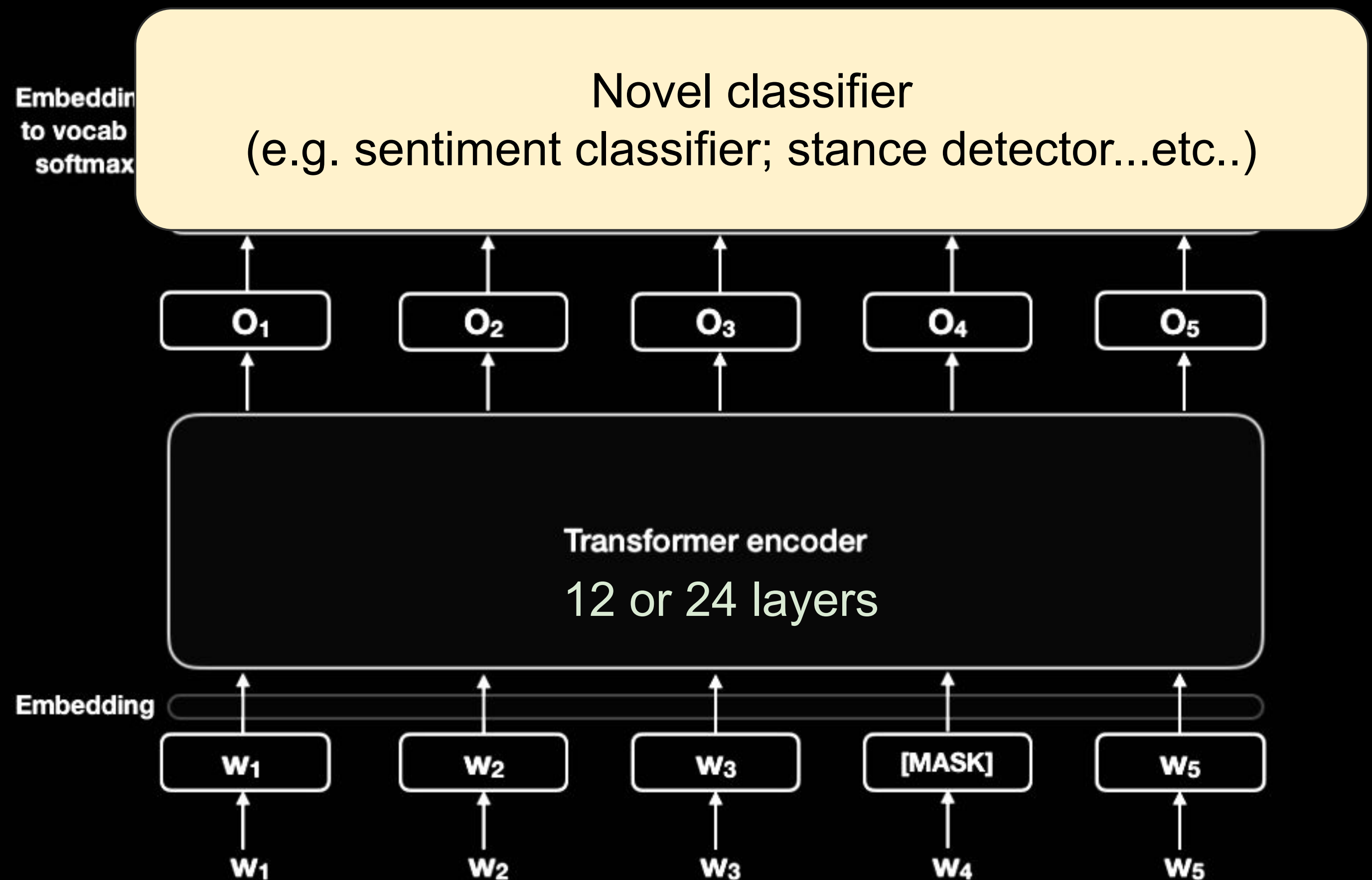
BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



The Transformer: Motivation

Challenges to sequential representation learning

- Capture long-distance dependencies
  Self-attention *treats far away words similar to those close.*
- Preserving sequential distances / periodicity
  Positional embeddings *encode distances/periods.*
- Capture multiple relationships
  Multi-headed attention *enables multiple compositions.*
- Easy to parallelize -- don't need sequential processing.
  Entire layer can be computed at once. Is only matrix multiplications + standardizing.