Large-Scale, Distributed Machine Learning

CSE545 - Spring 2022

Stony Brook University

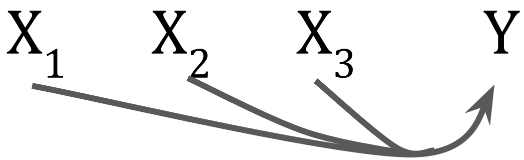
H. Andrew Schwartz

Supervised Learning

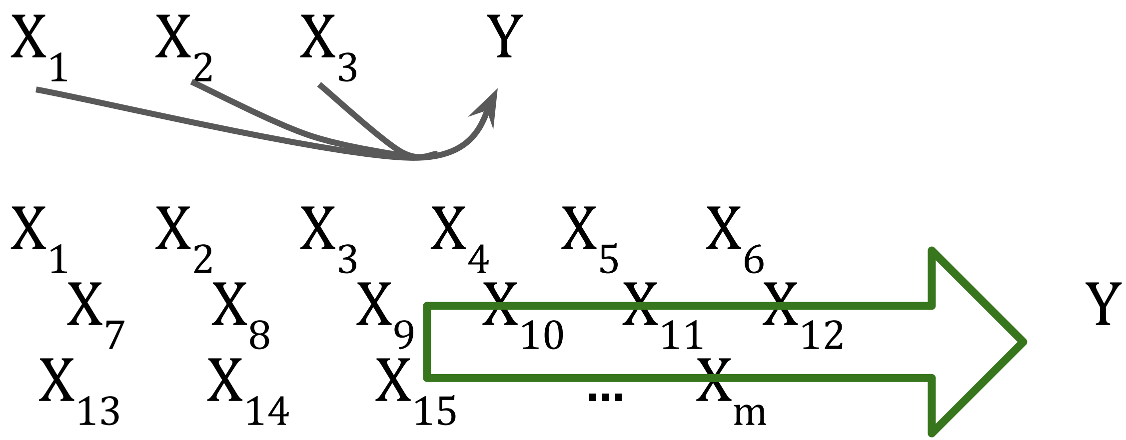
(genes) (health)

X_1 X_2 X_3 Y

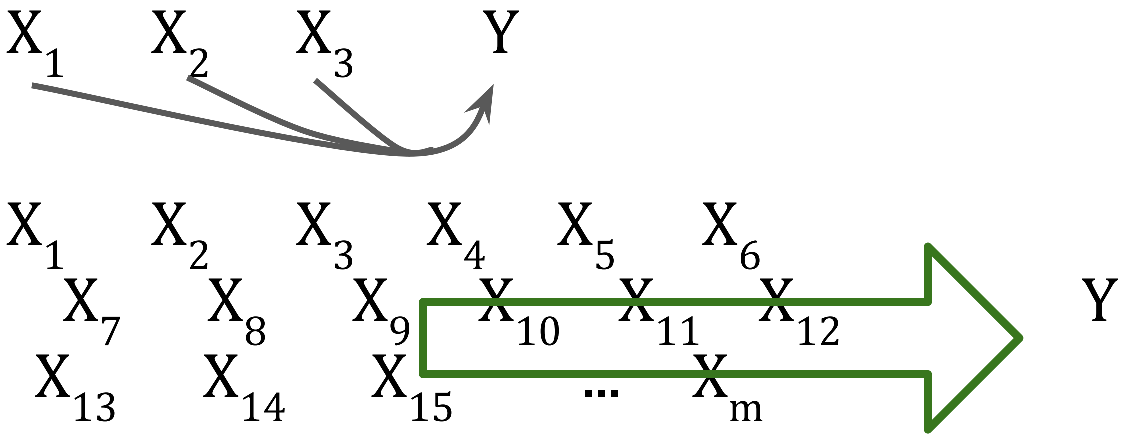
Supervised Learning



Supervised Learning



Supervised Learning



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

Ingredients of a TensorFlow

tensors*
variables - persistent mutable tensors
constants - constant
placeholders - from data

* technically, still *operations*

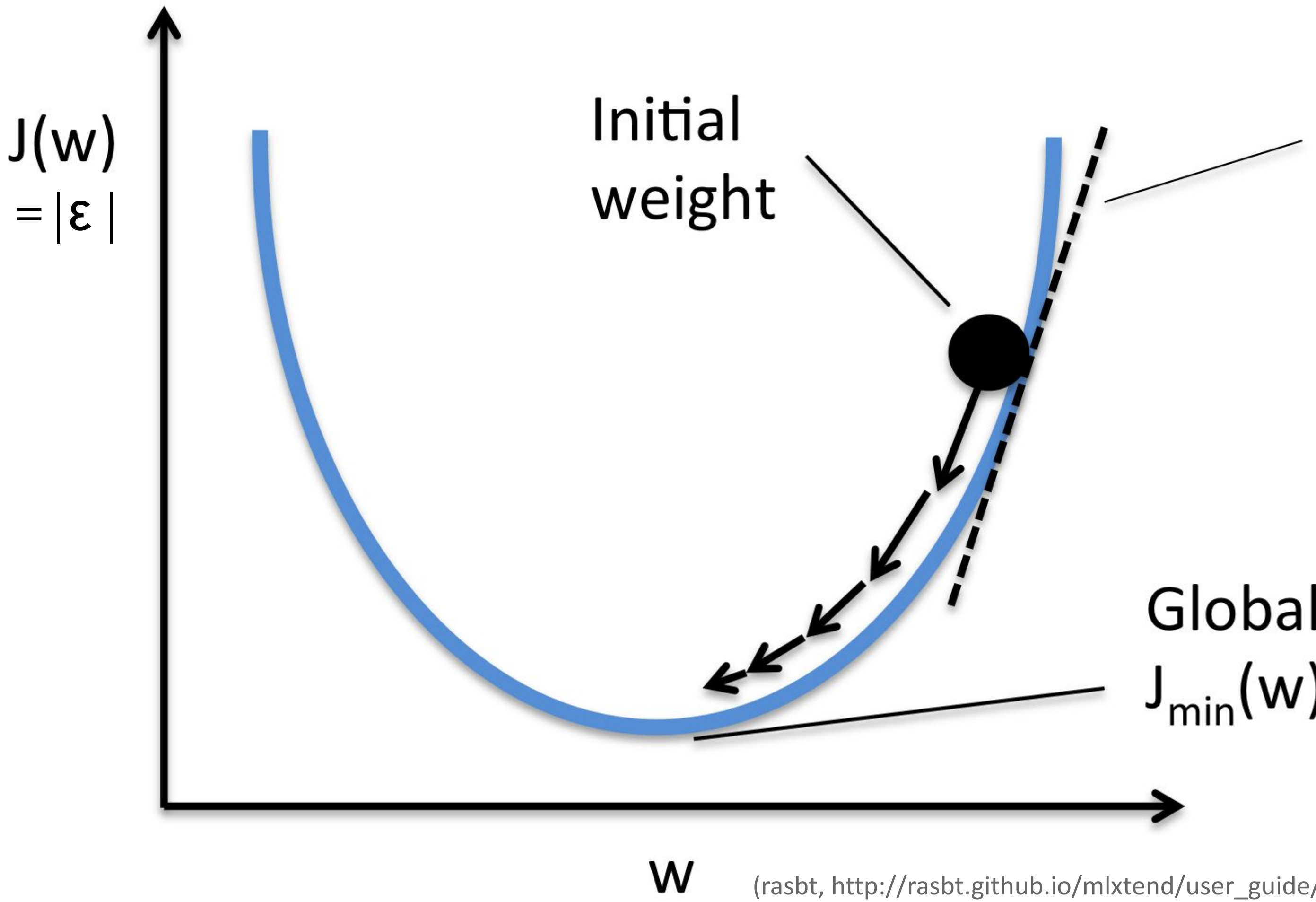
operations
an abstract computation (e.g. matrix multiply, add) executed by device *kernels*

graph

session
defines the environment in which operations *run*. (like a Spark context)

devices
the specific devices (cpus or gpus) on which to run the session.

Review: Gradient Descent



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

Weights Derived from Gradients

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

Weights Derived from Gradients

Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

Weights Derived from Gradients

Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

In standard linear equation:

$$y = mx + b$$

$$\text{let } x' = x + [1, 1, ..., 1]_N^T$$

$$\text{then, } y = mx'$$

(if we add a column of 1s, mx + b is just matmul(m, x))

Weights Derived from Gradients

Linear Regression: Trying to find “betas” that minimize:

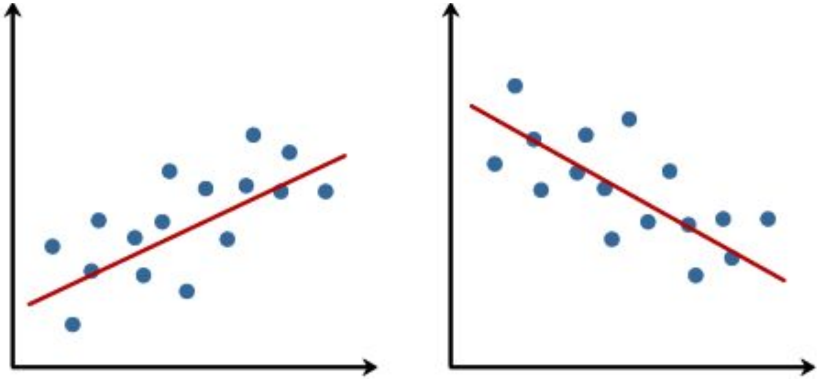
$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$



Weights Derived from Gradients

Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

$\hat{y}_i = X_i\beta$ Thus: $$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

How to update? $\beta_{new} = \beta_{prev} - \alpha * \mathrm{grad}$

Weights Derived from Gradients

Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

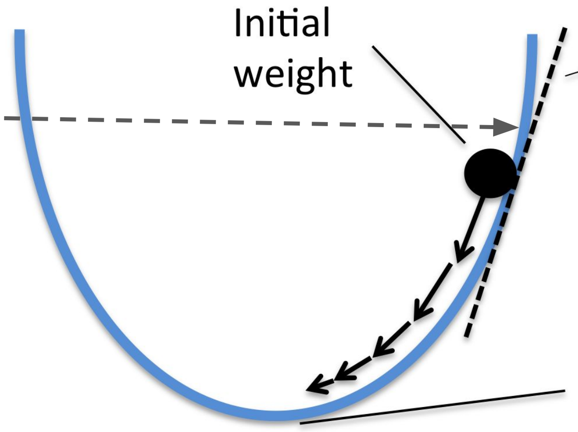
$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

How to update? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

(for gradient descent) "learning rate"



Weights Derived from Gradients

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

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

1. Matrix Solution:

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

Weights Derived from Gradients

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

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

2. Gradient descent solution

(Mirrors many parameter optimization problems.)

1. Matrix Solution:

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

Weights Derived from Gradients

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

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

<u>Gradient descent</u> needs to solve.
(Mirrors many parameter optimization problems.)

TensorFlow has built-in ability to derive gradients given a cost function.

Weights Derived from Gradients

Ridge Regression (L2 Penalized linear regression, $\lambda||\beta||_2^2$)

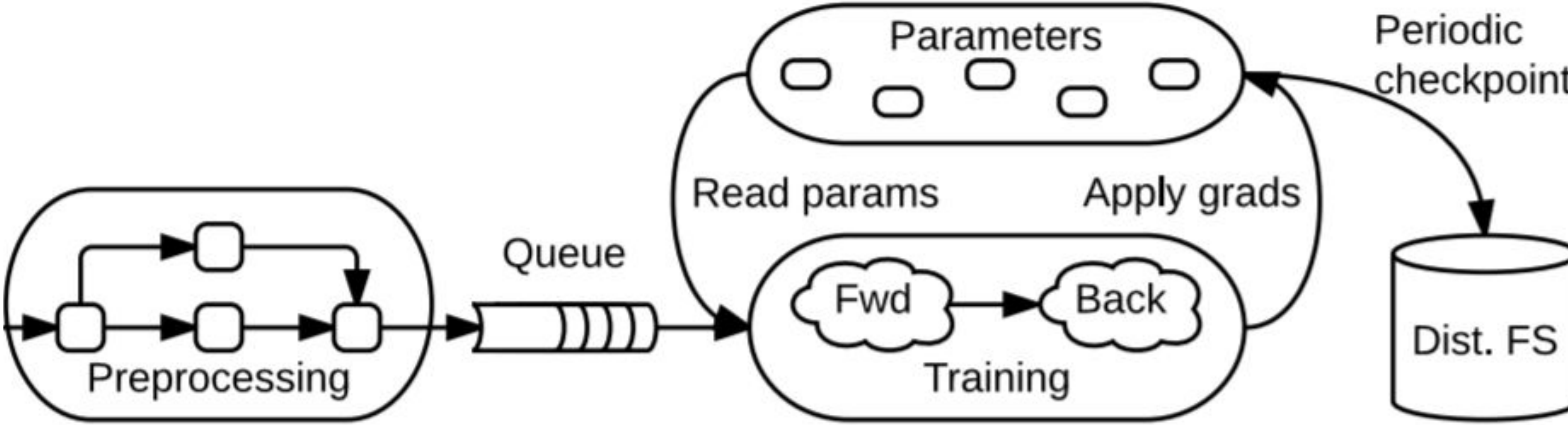
$$\hat{\beta}^{ridge} = argmin_{\beta}\{\boxed{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{m}\beta_j^2}\}$$

Gradient descent needs to solve.
(Mirrors many parameter optimization problems.)

TensorFlow has built-in ability to derive gradients given a cost function.

```
tf.gradients(cost, [params])
```

Weights Derived from Gradients



TensorFlow has built-in ability to derive gradients given a cost function.

```
tf.gradients(cost, [params])
```

Options for distribution

1. **Distribute copies of entire dataset**
   a. Train over all with different hyperparameters
   b. Train different folds per worker node.

Options for distribution

1. **Distribute copies of entire dataset**
   a. Train over all with different hyperparameters
   b. Train different folds per worker node.

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

Options for distribution

1. **Distribute copies of entire dataset**
    a. Train over all with different hyperparameters
    b. Train different folds per worker node.

2. **Distribute data**
    a. Each node finds parameters for subset of data
    b. Needs mechanism for updating parameters
        i. Centralized parameter server
        ii. Distributed All-Reduce

3. **Distribute model or individual operations** (e.g. matrix multiply)

Options for distribution

1. **Distribute copies of entire dataset**
    a. Train over all with different hyperparameters
    b. Train different folds per worker node.

    Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
    a. Each node finds parameters for subset of data
    b. Needs mechanism for updating parameters
        i. Centralized parameter server
        ii. Distributed All-Reduce

    Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

    Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Options for distribution

Done often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
   a. Train over all with different hyperparameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Options for distribution

Done often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
   a. Train over all with different hyperparameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Preferred method for big data or very complex models (i.e. models with many internal parameters).

   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Options for distribution

Done often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
   a. Train over all with different hyperparameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Data Parellelism

   Preferred method for big data or very complex models (i.e. models with many internal parameters).

   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Model Parellelism

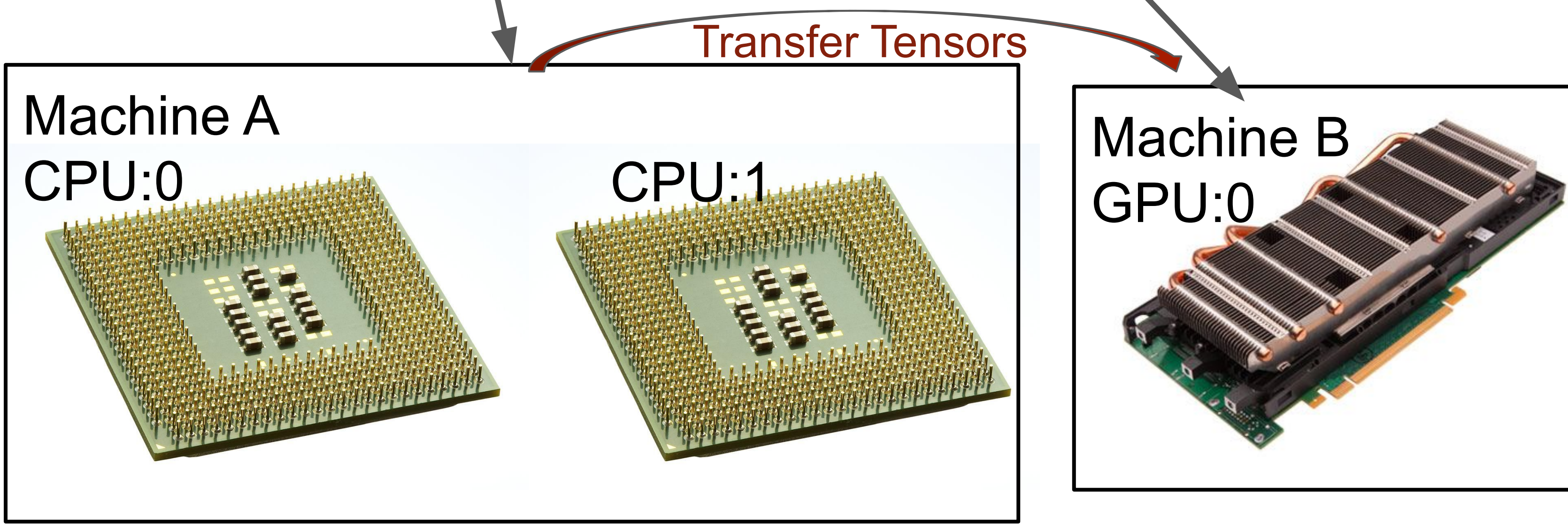
   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Model Parallelism

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```



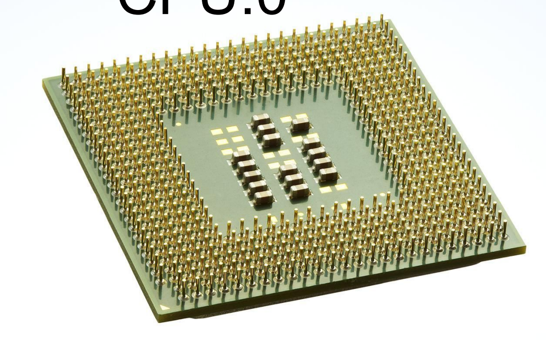
Data Parallelism

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

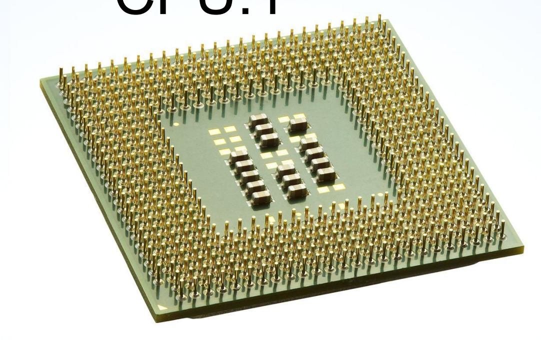
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

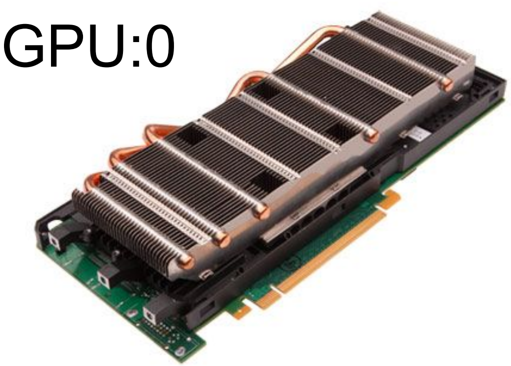
CPU:0



CPU:1



GPU:0



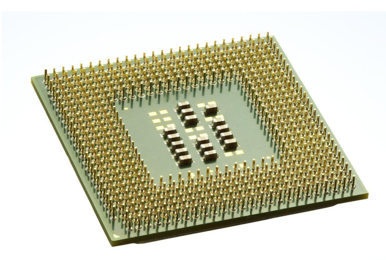
Data Parallelism

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

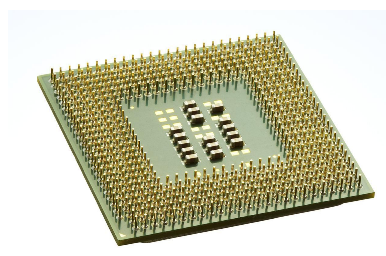
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

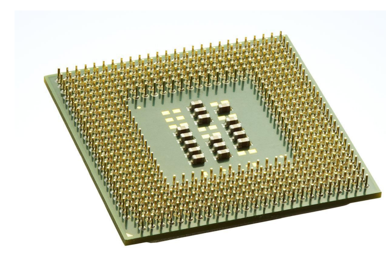
worker:0



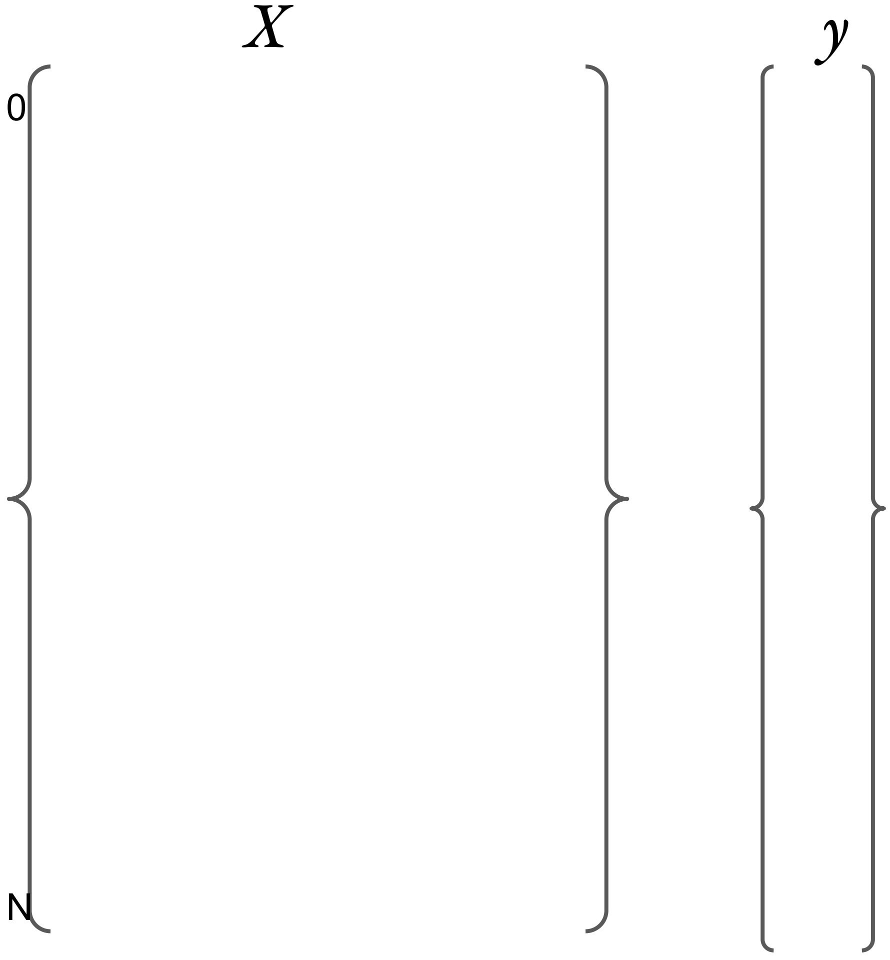
worker:1



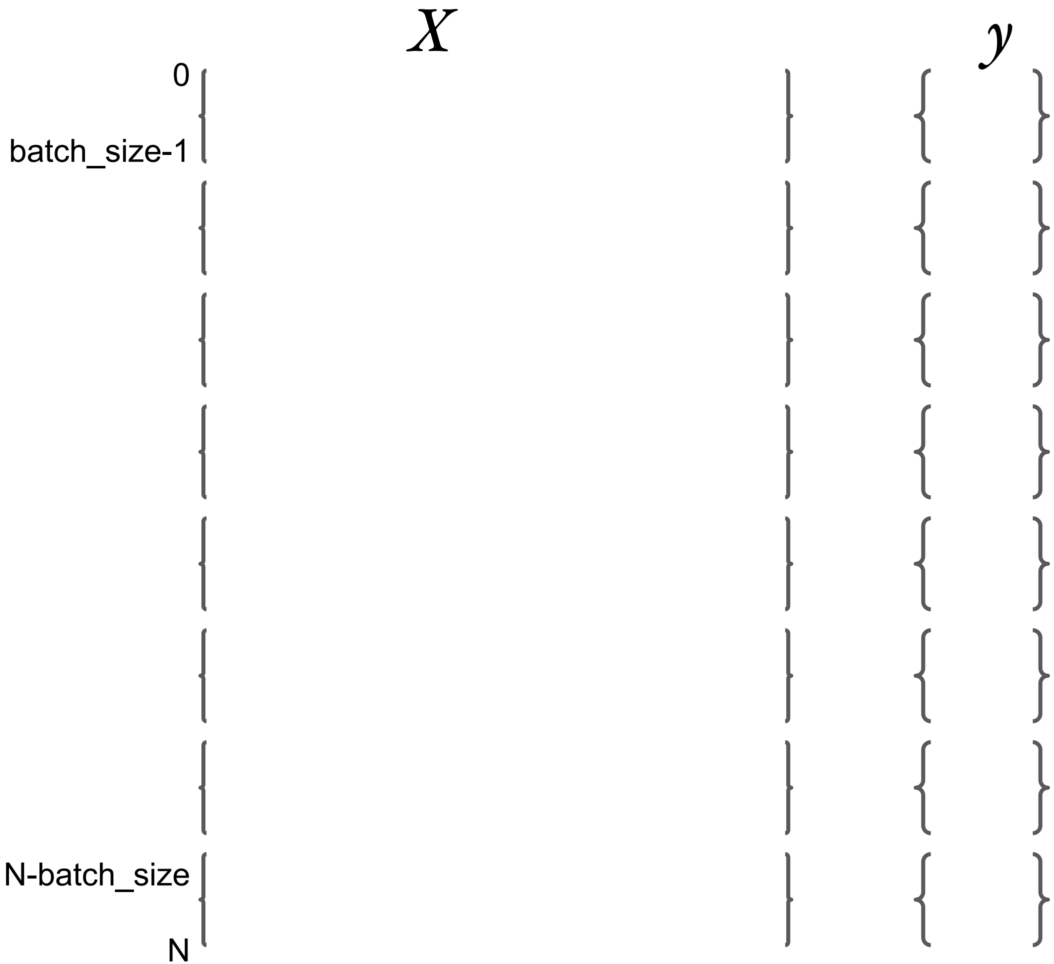
worker:2



Distributing Data

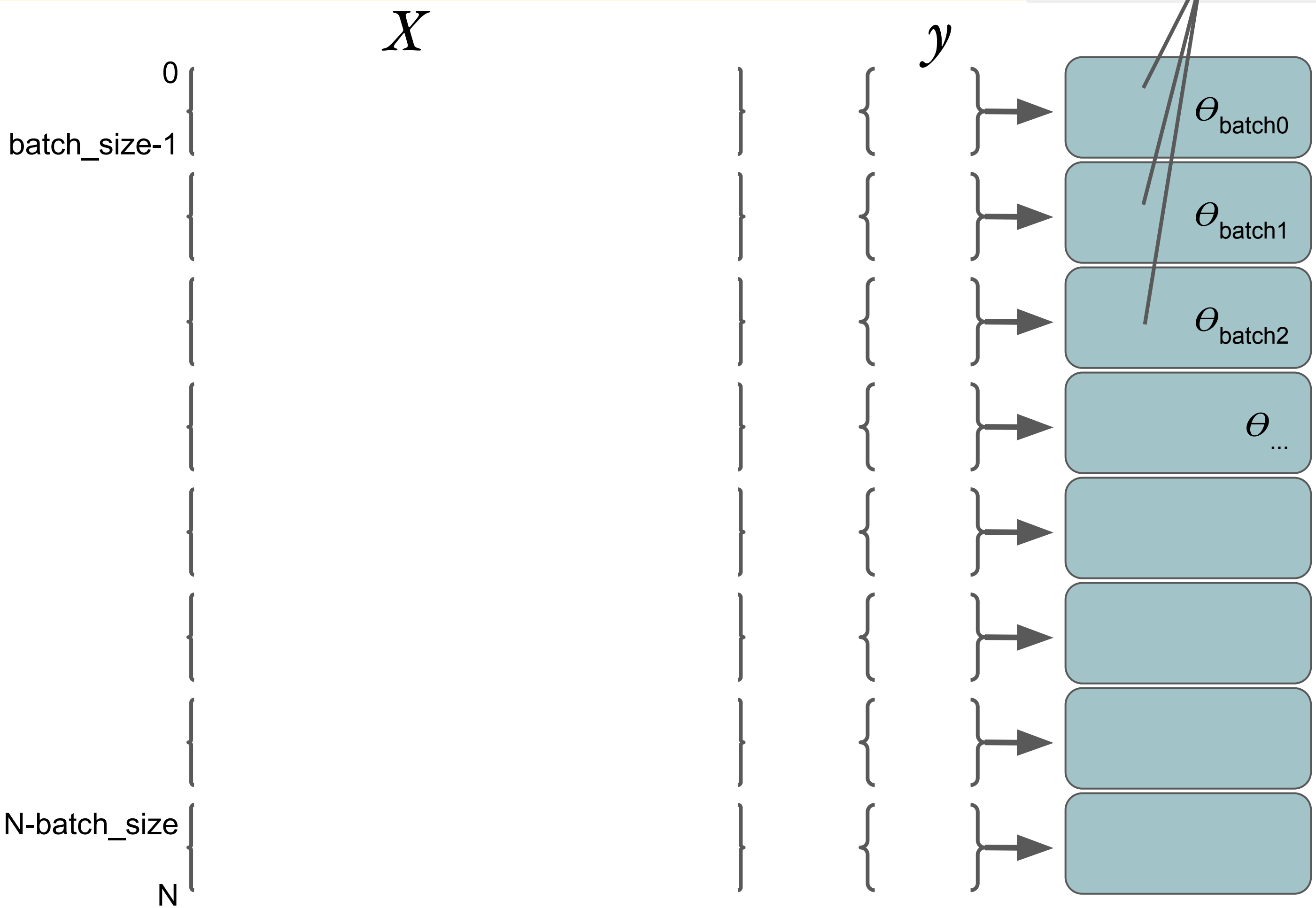


Distributing Data

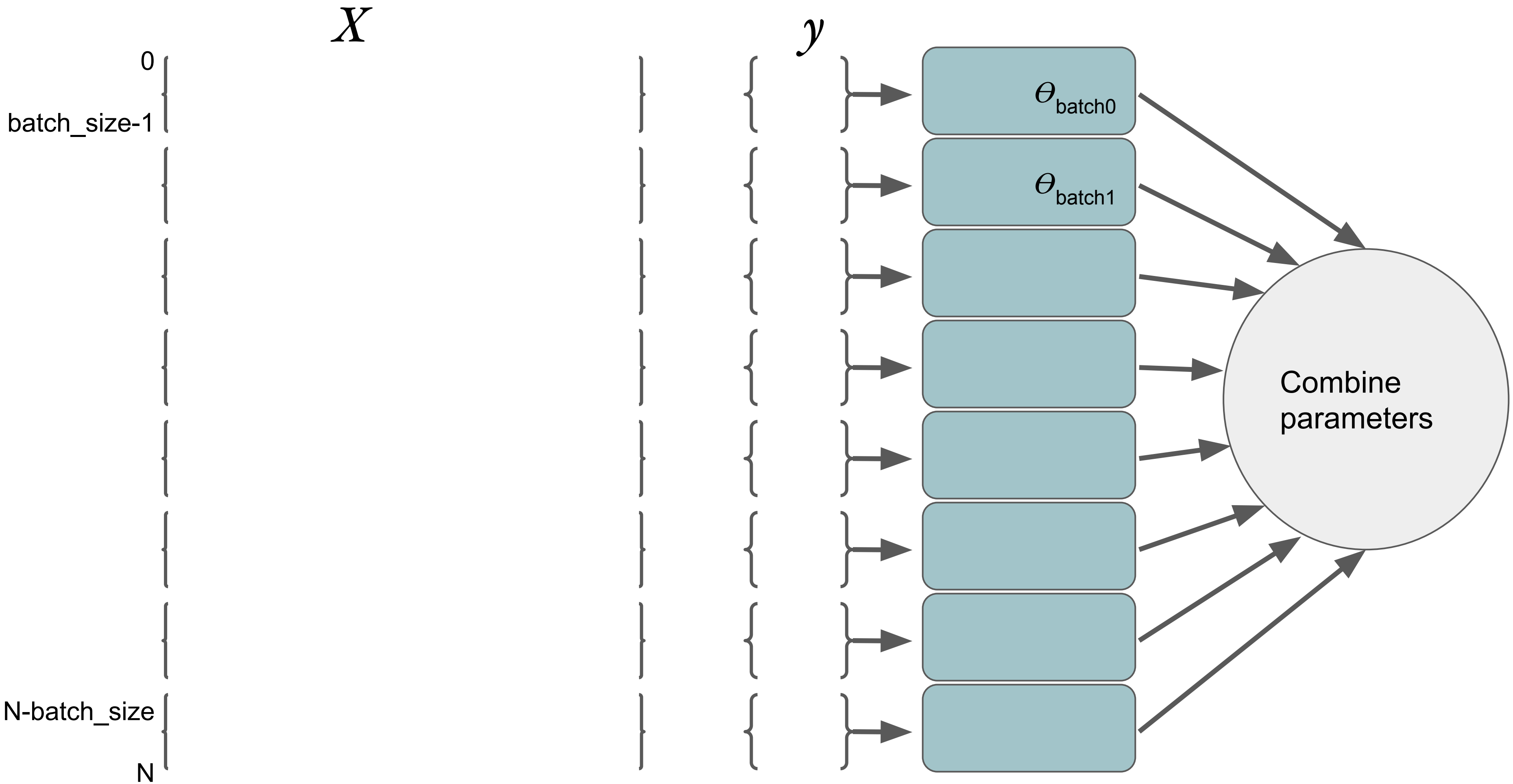


Distributing Data

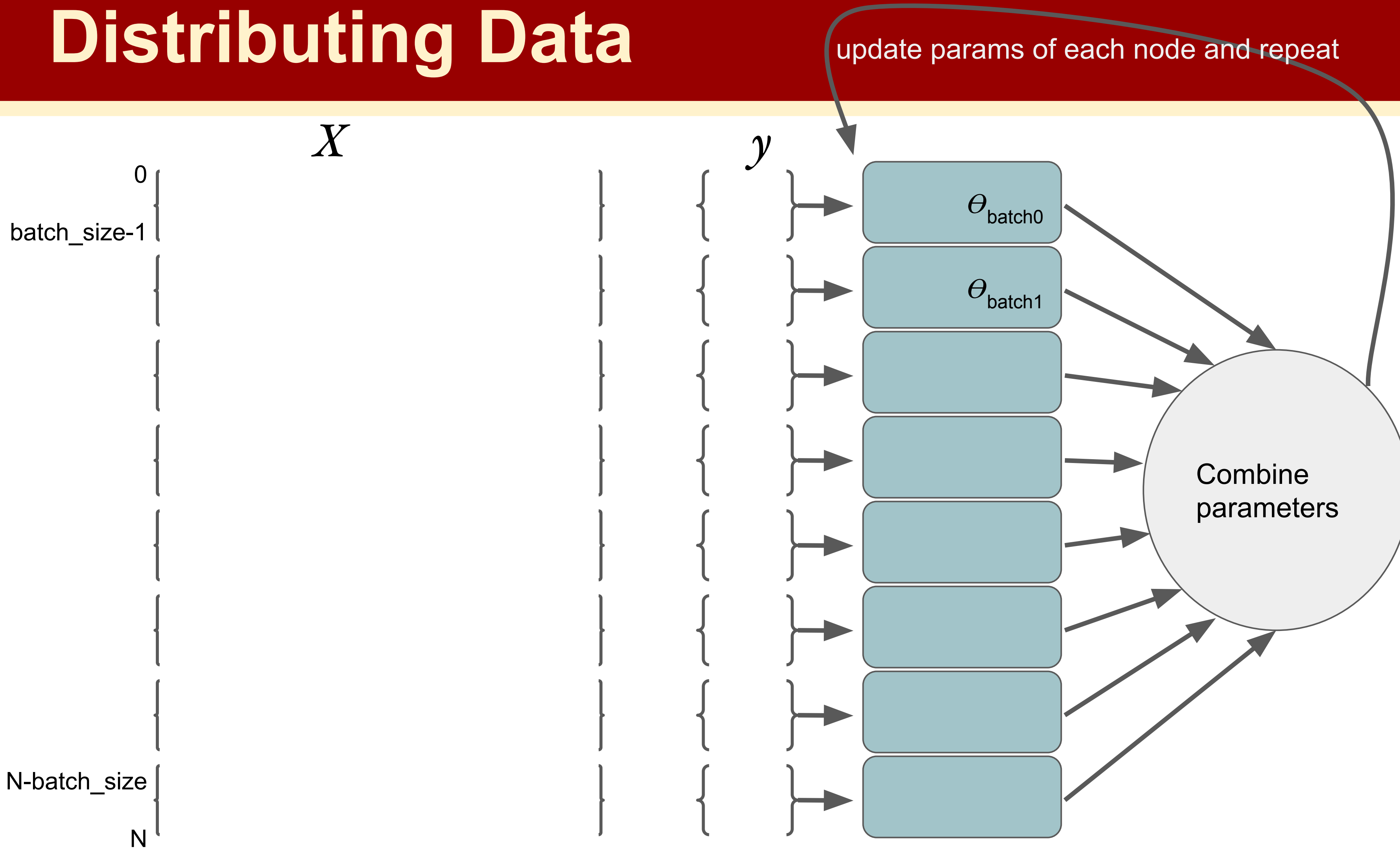
learn parameters (i.e. weights), given graph with cost function and *optimizer*



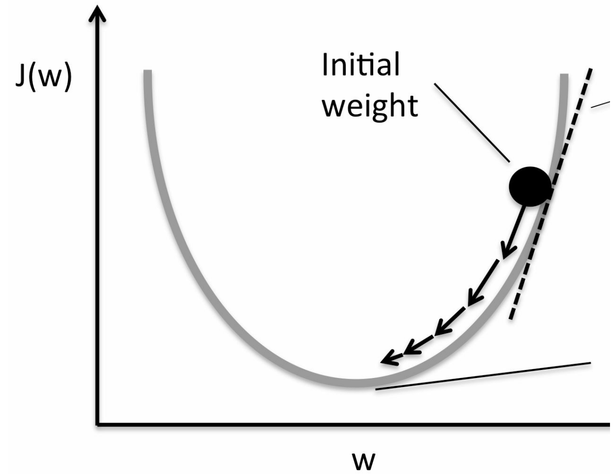
Distributing Data



Distributing Data



Gradient Descent for Linear Regression



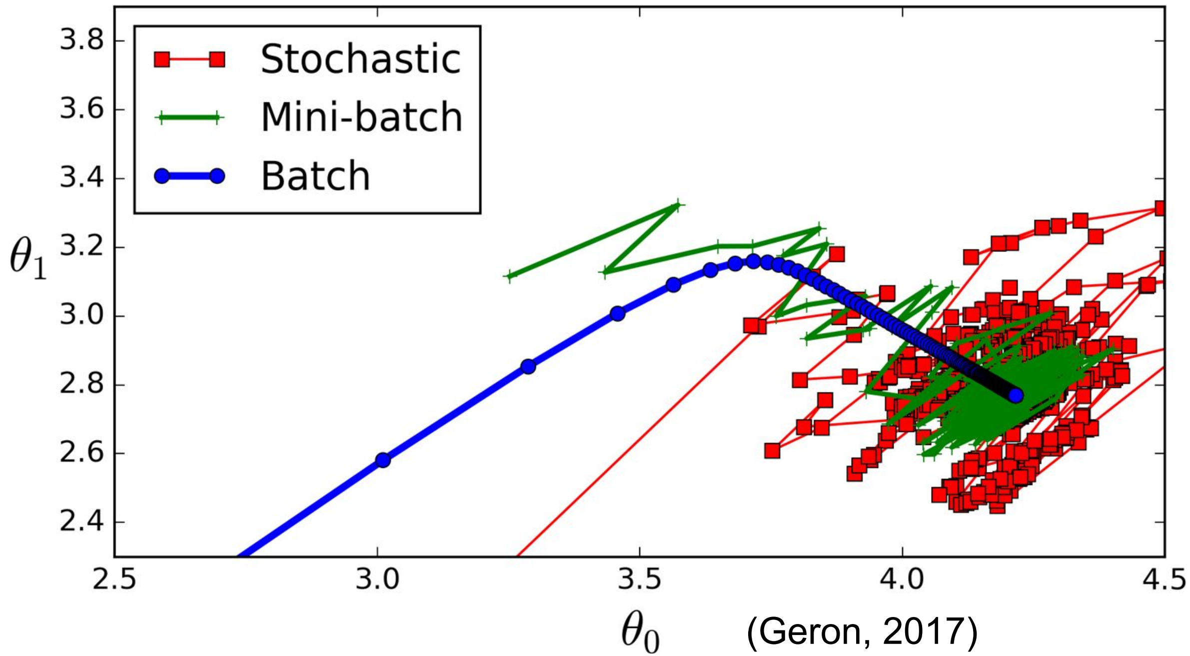
(Geron, 2017)

Gradient Descent for Linear Regression

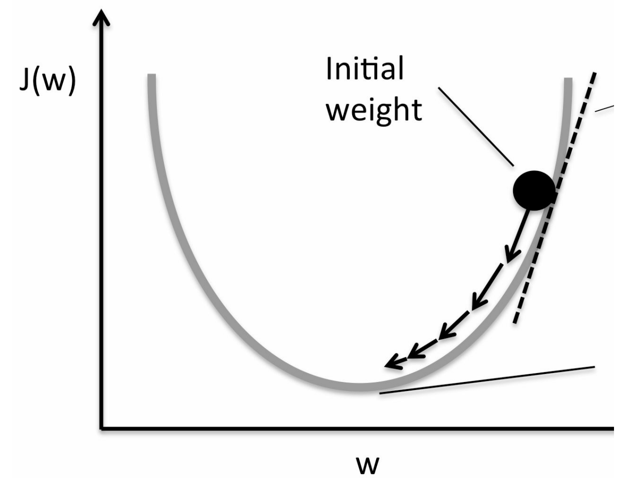
Batch Gradient Descent

Stochastic Gradient Descent: One example at a time

Mini-batch Gradient Descent: k examples at a time.



(Geron, 2017)

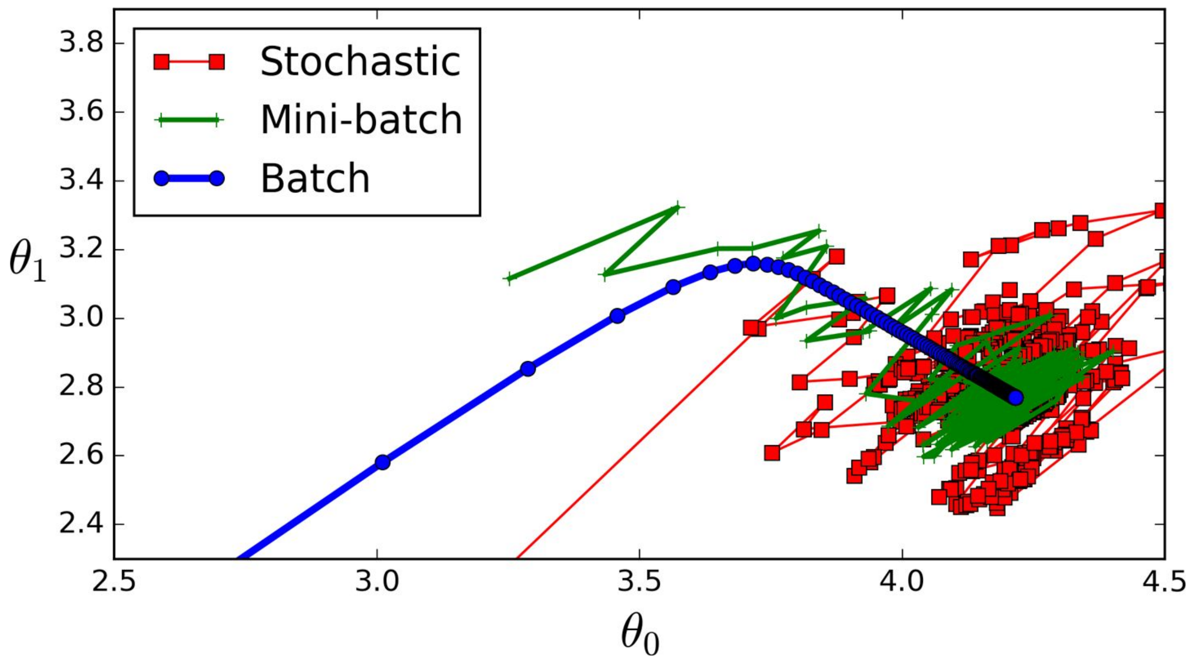


Gradient Descent for Linear Regression

Batch Gradient Descent

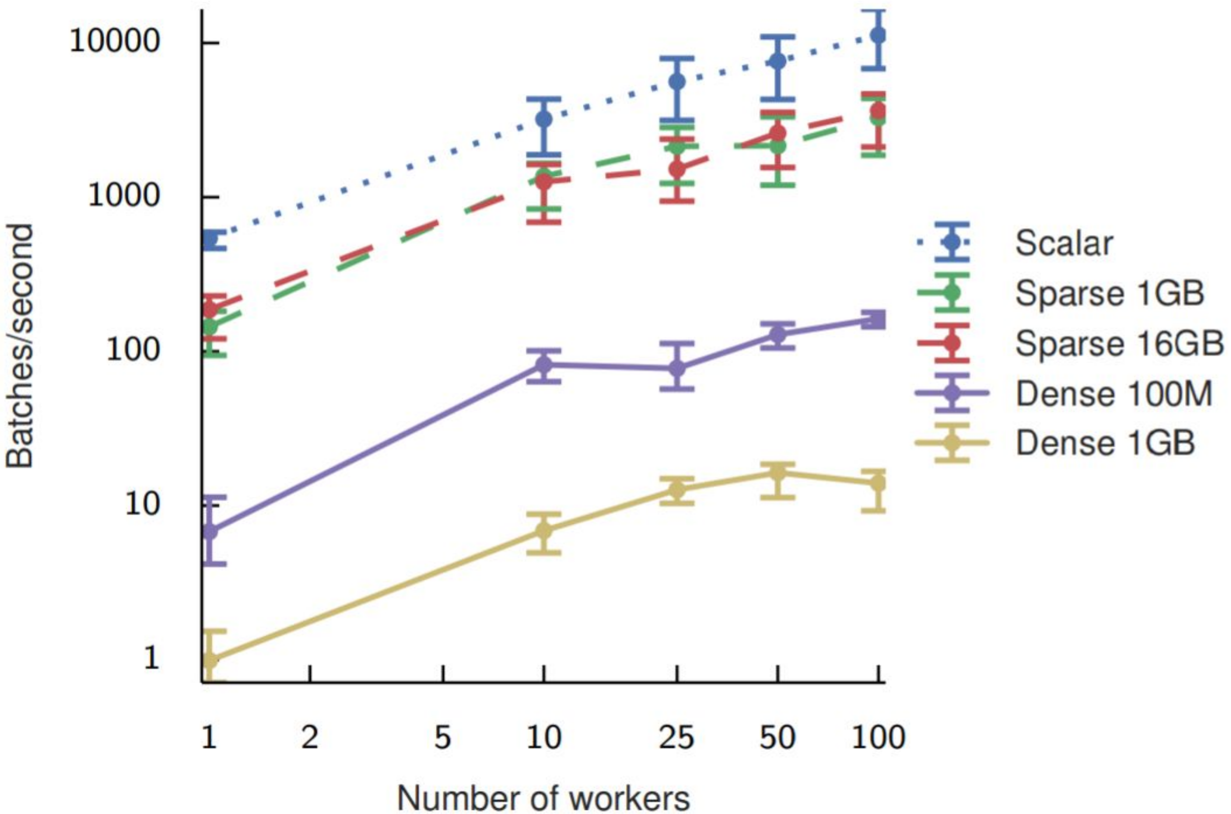
Stochastic Gradient Descent: One example at a time

Mini-batch Gradient Descent: k examples at a time.



(Geron, 2017)

Distributed TensorFlow



Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265-283).

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn't work with Model Parallelism)

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

next slides

discussed previously

Parallelisms:

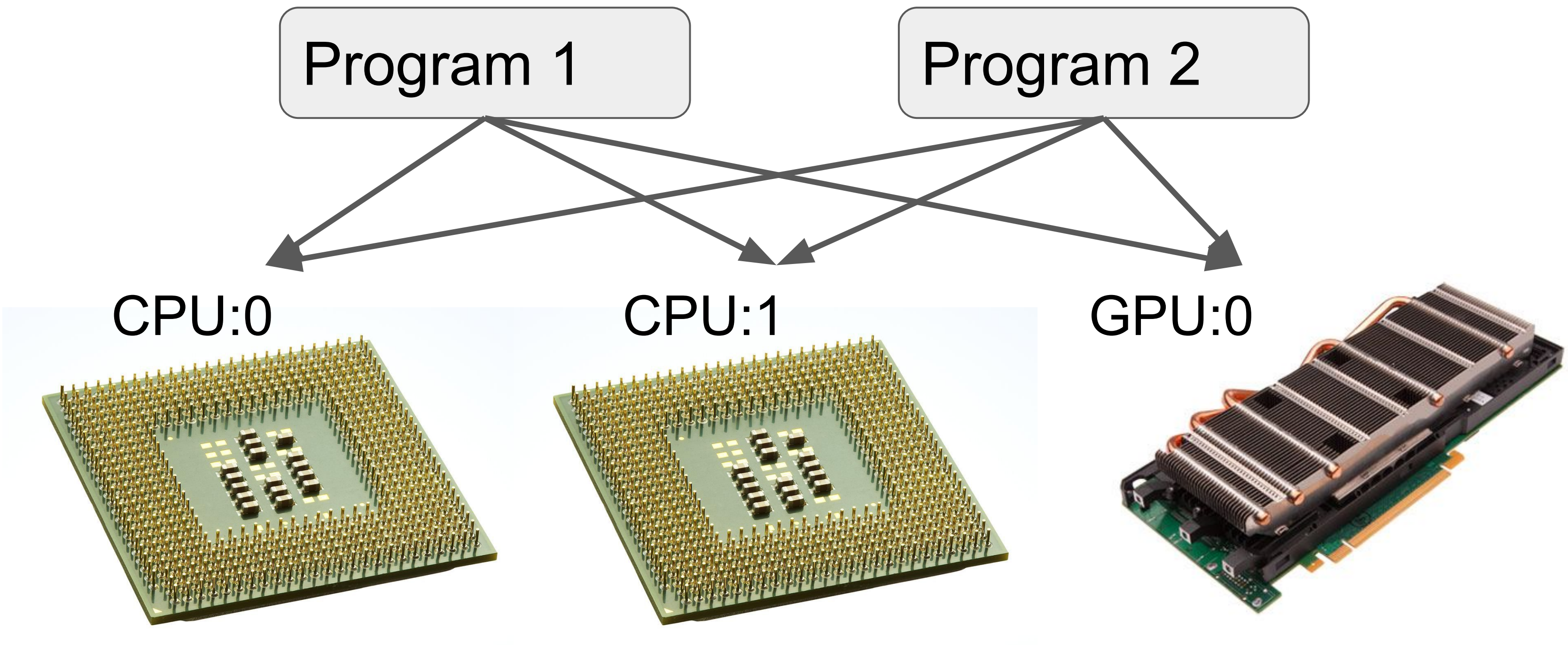
- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn't work with Model Parallelism)

Local Distribution

Multiple devices on single machine



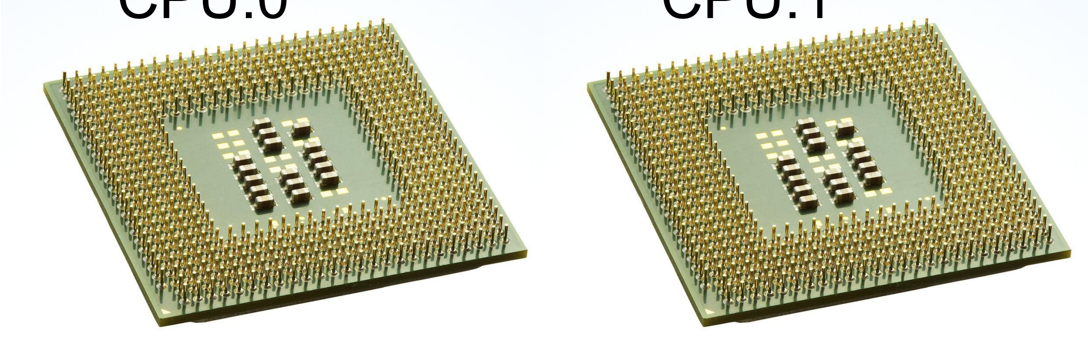
Local Distribution

Multiple devices on single machine

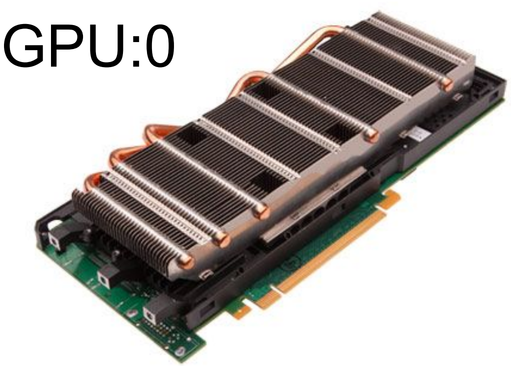
```
with tf.device("/cpu:1")
    beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
    y_pred=tf.matmul(beta,X)
```

CPU:0 CPU:1



GPU:0

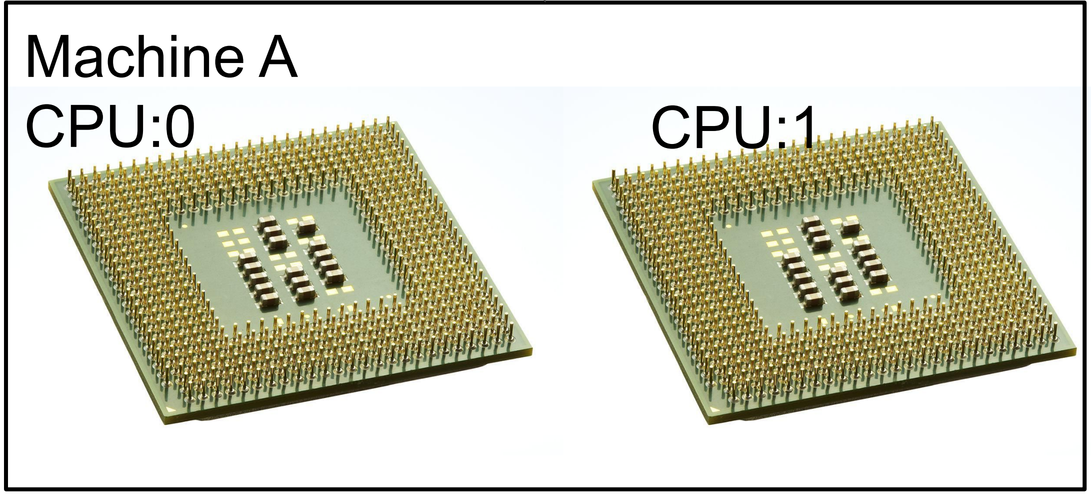


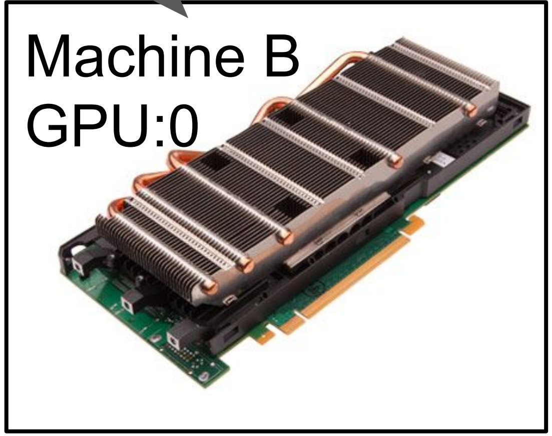
Cluster Distribution

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```





Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Parallelisms

Model Parallelism

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

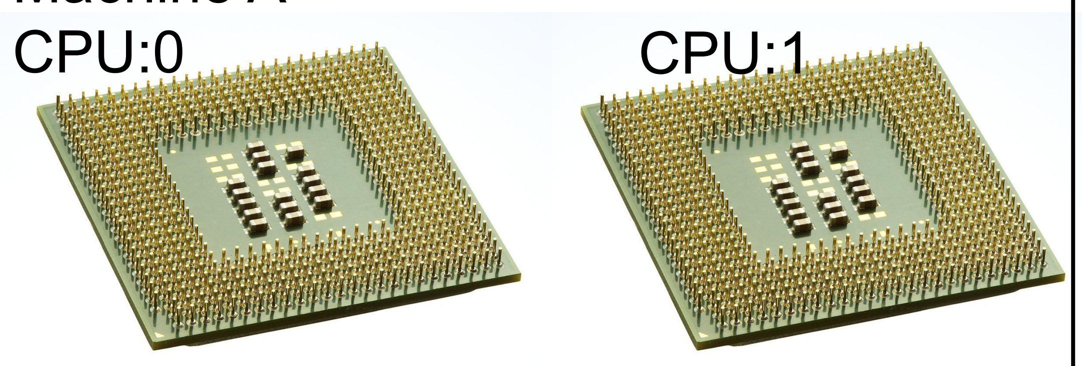
```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```

Transfer Tensors

Machine A

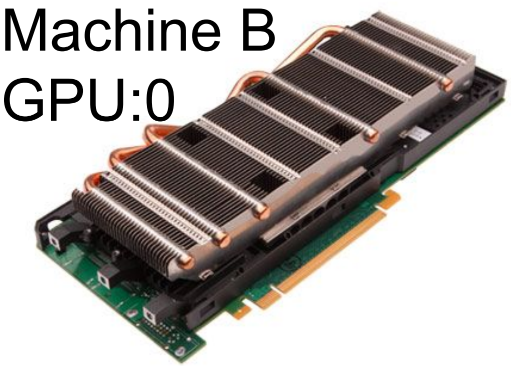
CPU:0

CPU:1



Machine B

GPU:0



Parallelisms

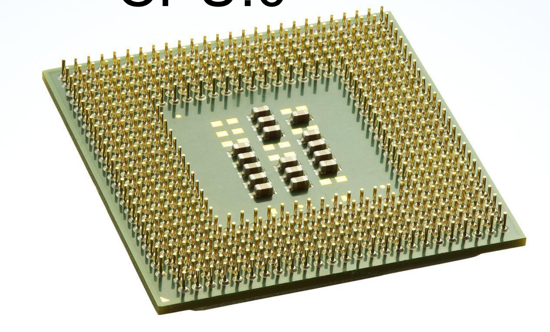
Data Parallelism

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

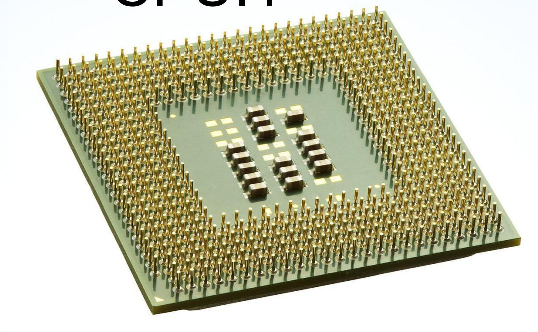
```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

```
...
  beta=tf.Variable(...)
  pred=tf.matmul(beta,X)
```

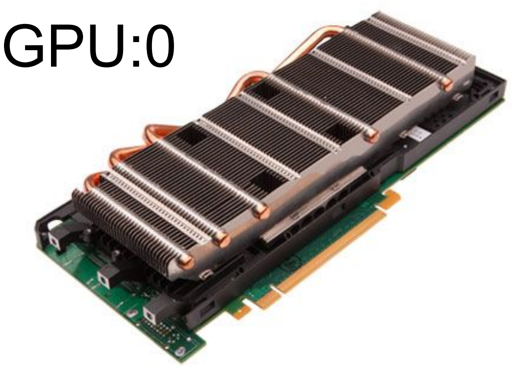
CPU:0



CPU:1



GPU:0



Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn't work with Model Parallelism)

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

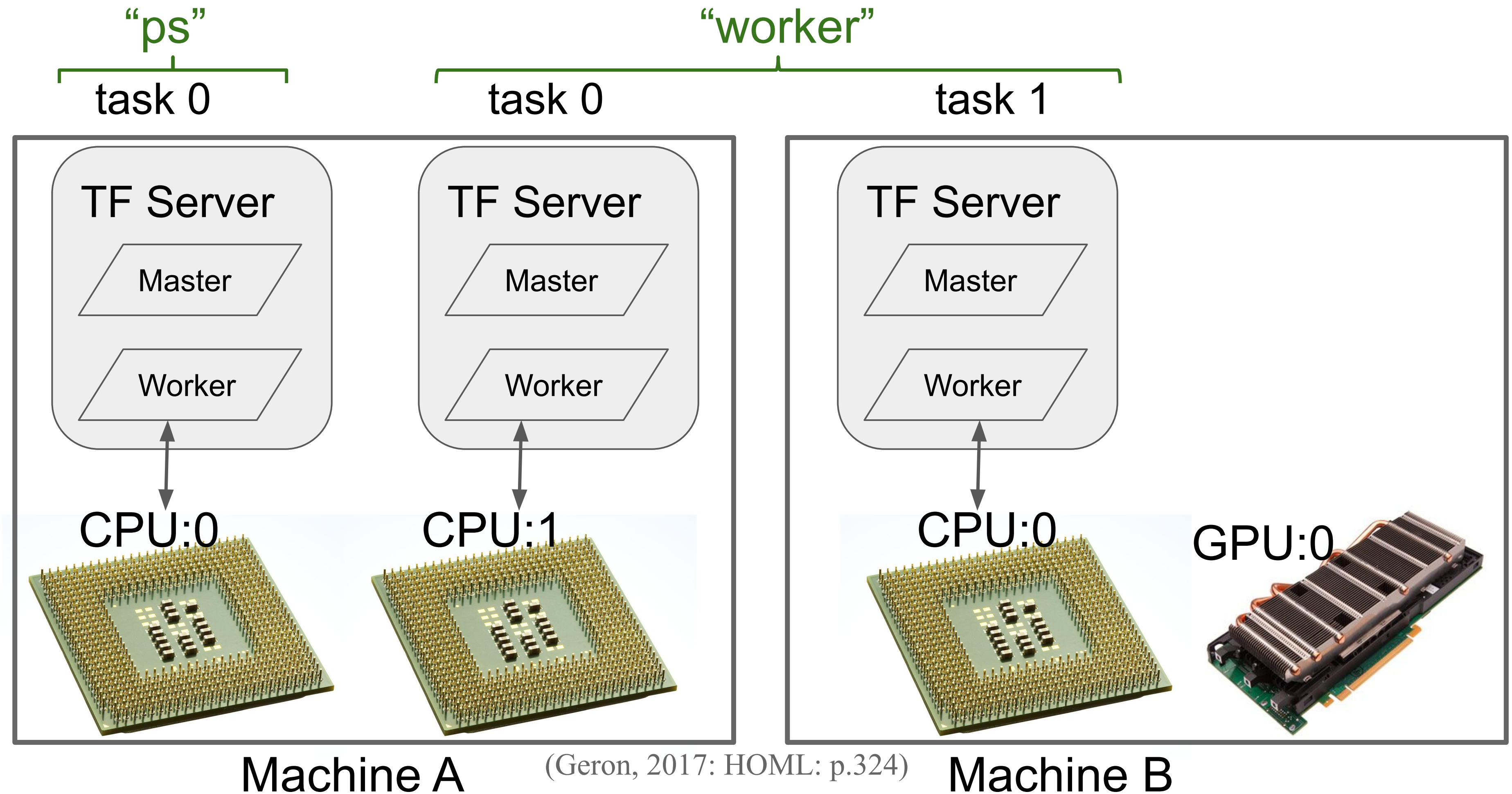
Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

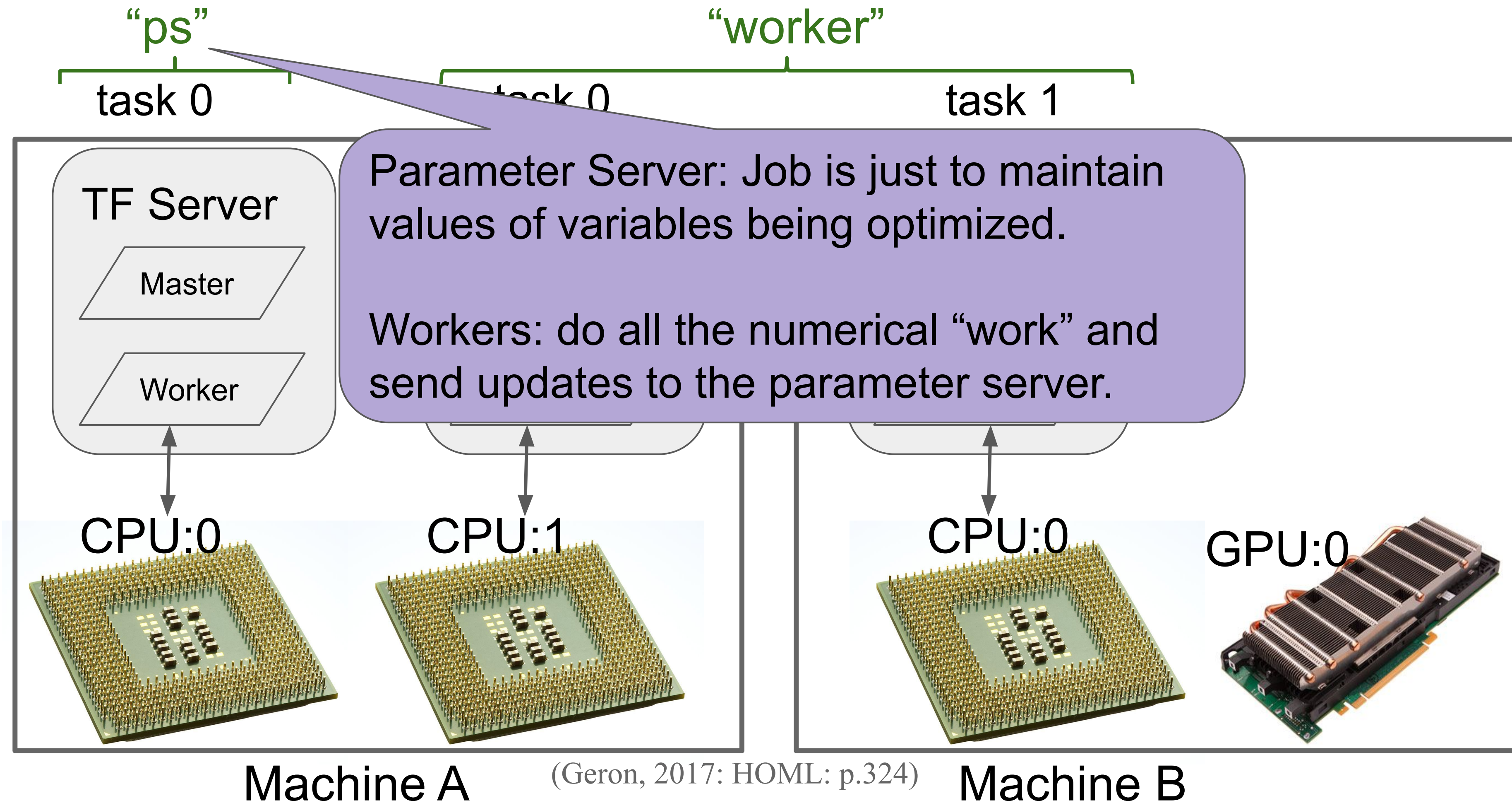
- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)

Asynchronous Parameter Server



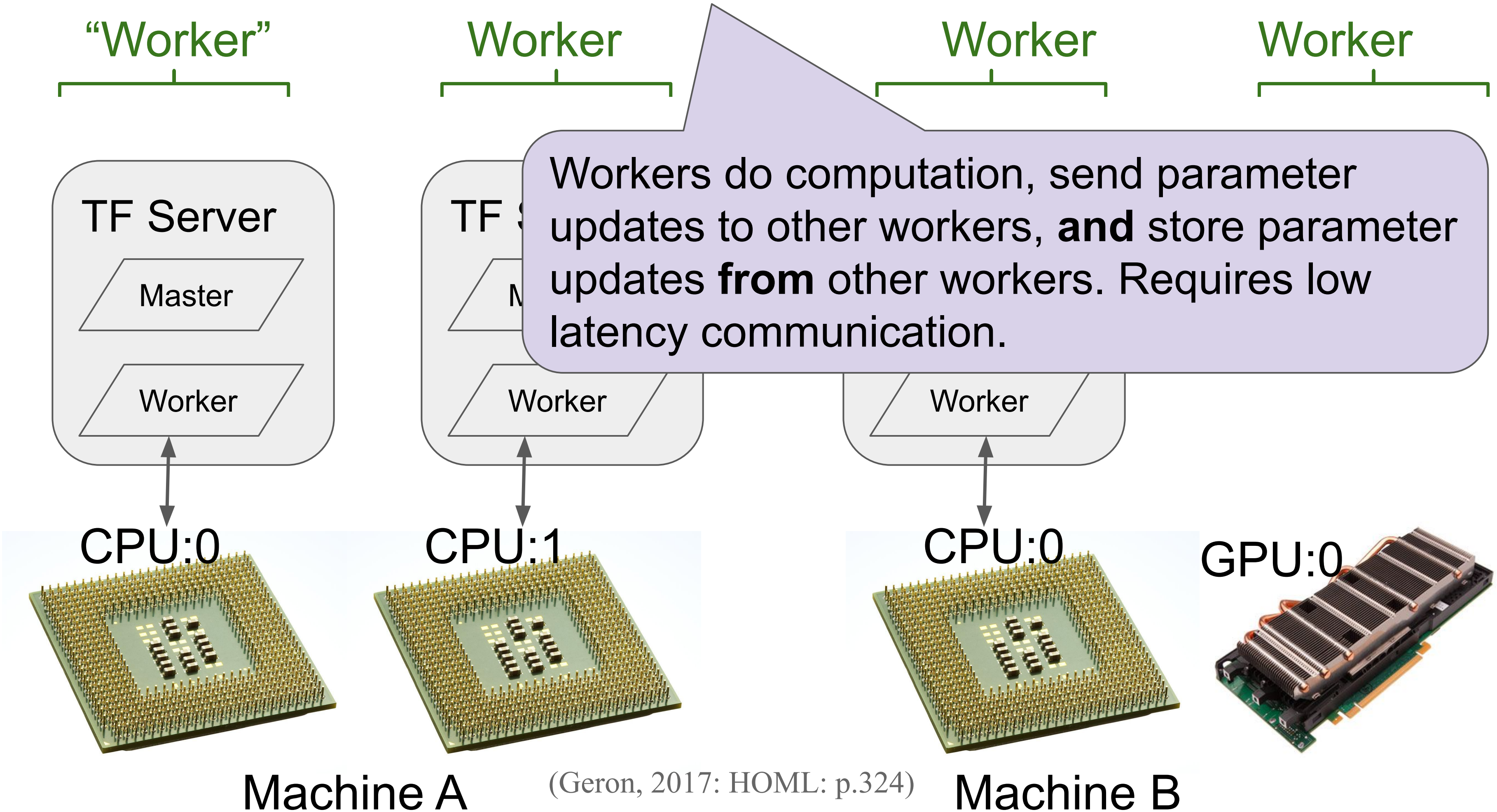
(Geron, 2017: HOML: p.324)

Asynchronous Parameter Server



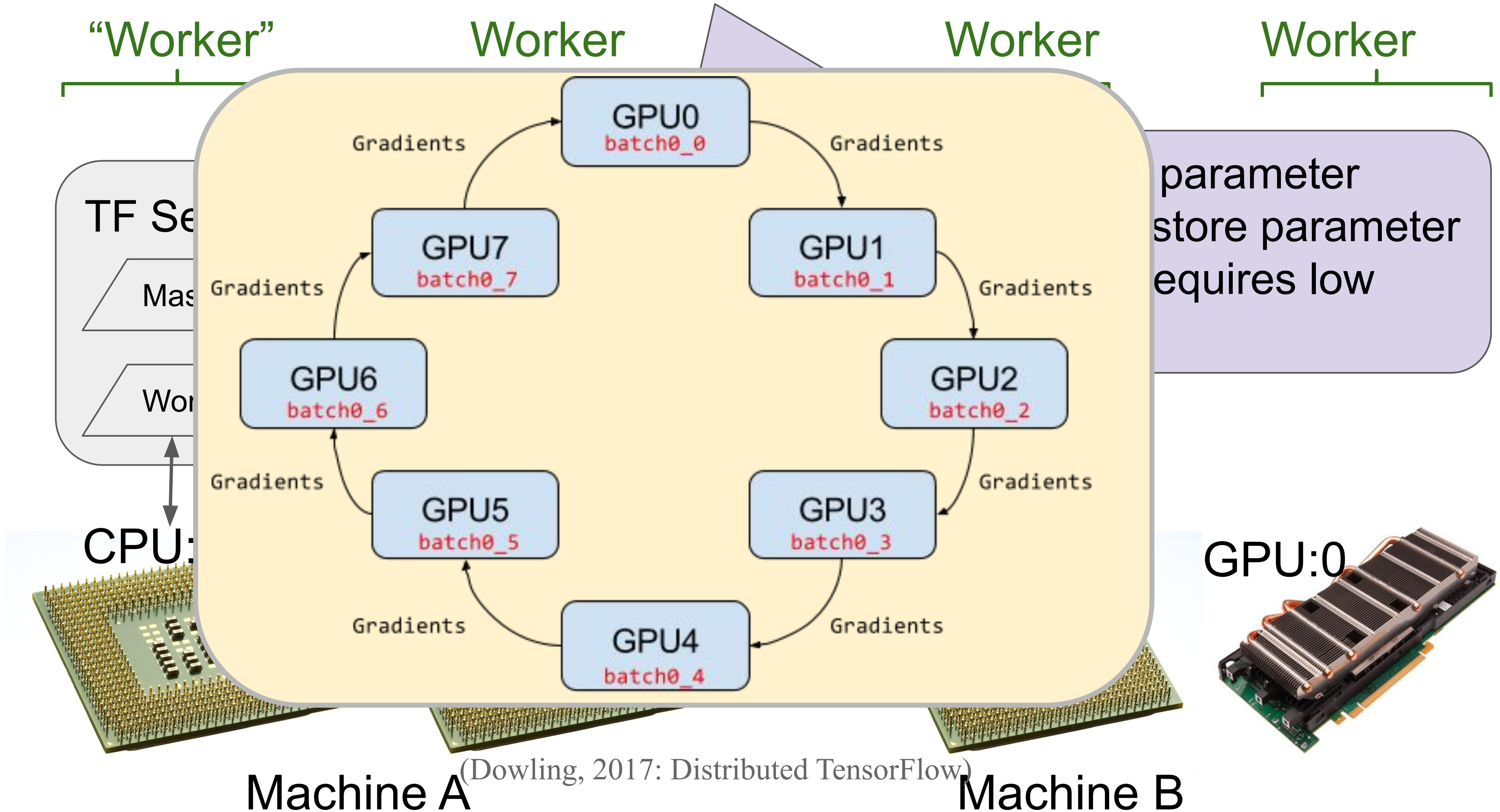
(Geron, 2017: HOML: p.324)

Synchronous All Reduce



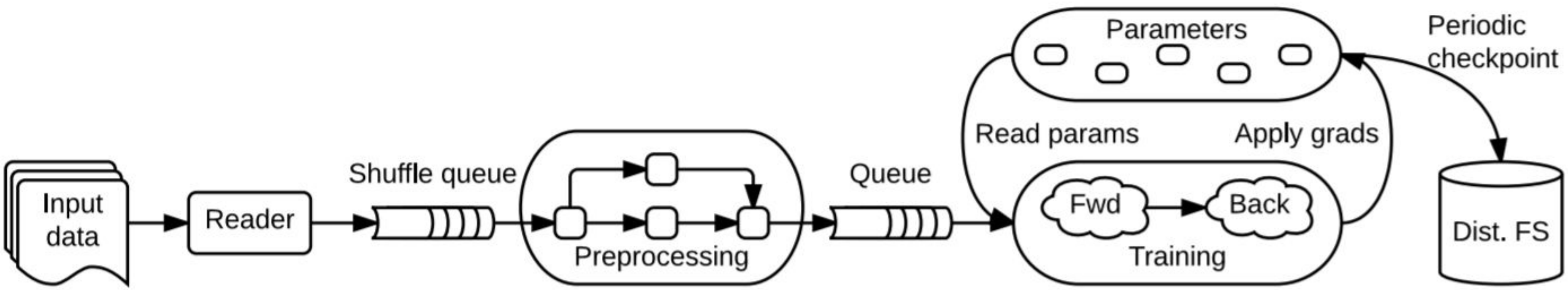
(Geron, 2017: HOML: p.324)

Synchronous All Reduce



(Dowling, 2017: Distributed TensorFlow)

Distributed TF: Full Pipeline



Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). TensorFlow: A System for Large-Scale Machine Learning. In *OSDI* (Vol. 16, pp. 265-283).

Review: Distributed ML

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

Done very often in practice. Not talked about much because it's mostly as easy as it sounds.

Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

Data Parellelism

Preferred method for big data or very complex models (i.e. models with many internal parameters).

Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

Model Parellelism

Pro: Parameters can be localized Con: High communication for transferring Intermediar data.

Post-Exam2 Topics:

1. Research Ethics
2. Useful Plots
3. Machine Learning Cross Validation
4. Convolutional Neural Networks
5. Recurrent Neural Networks
6. Transformer Networks

Post-Exam2 Topics:

1. **Research Ethics**
2. Useful Plots
3. Machine Learning Cross Validation
4. Convolutional Neural Networks
5. Recurrent Neural Networks
6. Transformer Networks

Ethics in Big Data

Bias

Privacy

Ethical Research Practice

Ethics in Big Data

Types of bias:

- **Outcome Disparity:** Predicted distribution given A,
  are dissimilar from ideal distribution given A
  - Selection bias
  - Label bias
  - Over-amplification

- **Error Disparity:** Predicts less accurate for authors of given demographics.

- **Semantic Bias:** Representations of meaning store demographic associations.

Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Two Examples

model
accuracy

The WSJ Effect

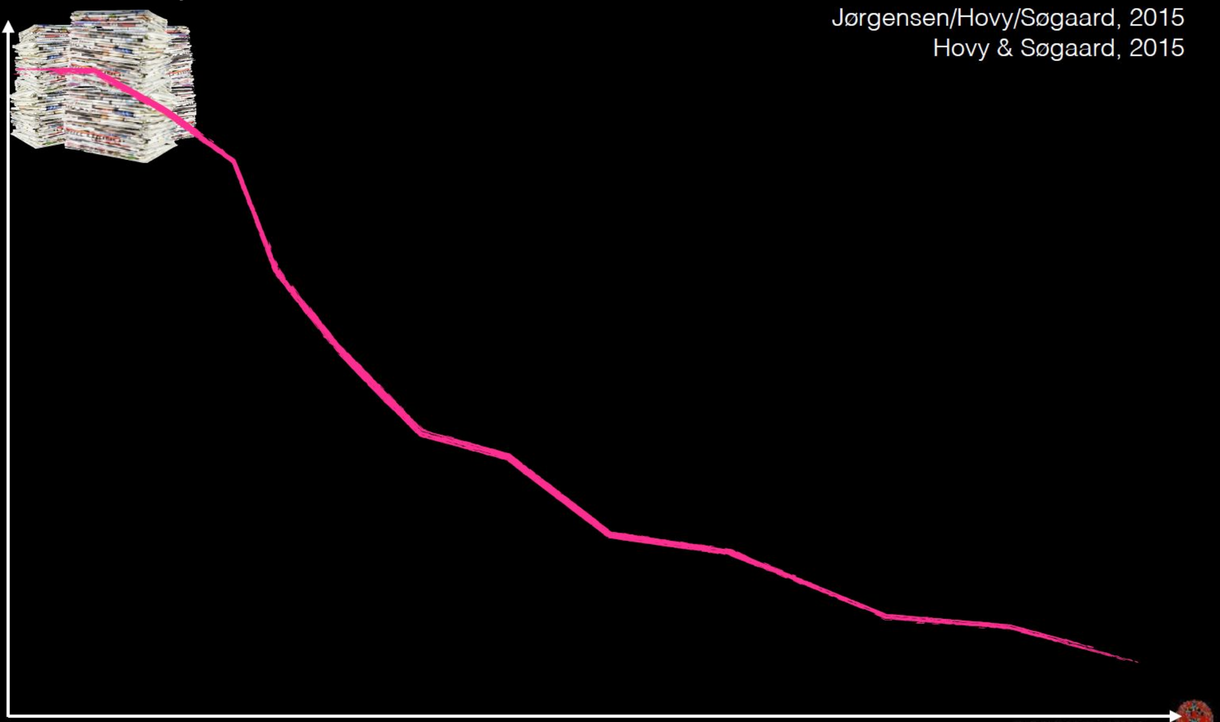
Jørgensen/Hovy/Søgaard, 2015
Hovy & Søgaard, 2015

distance from “standard” WSJ author demographics

Two Examples

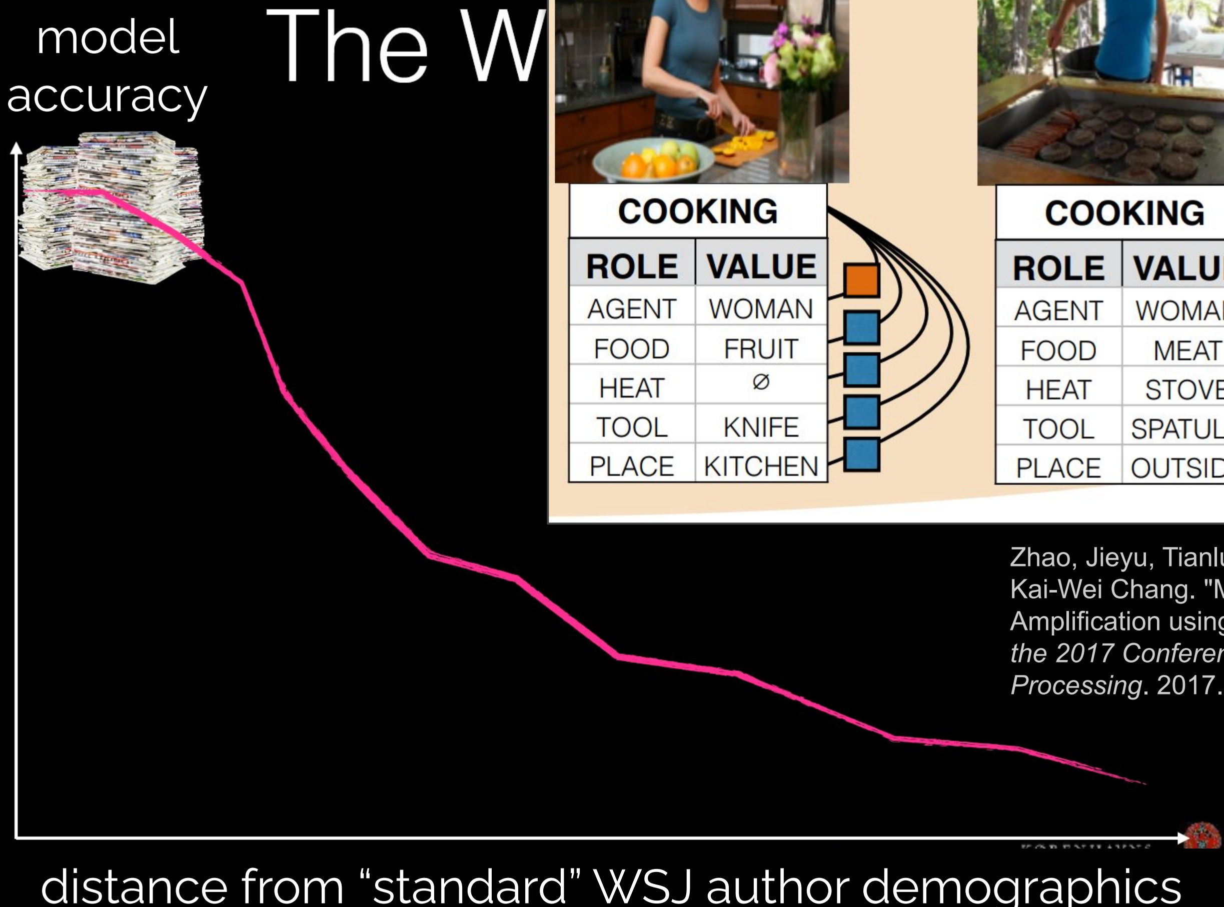
The WSJ Effect

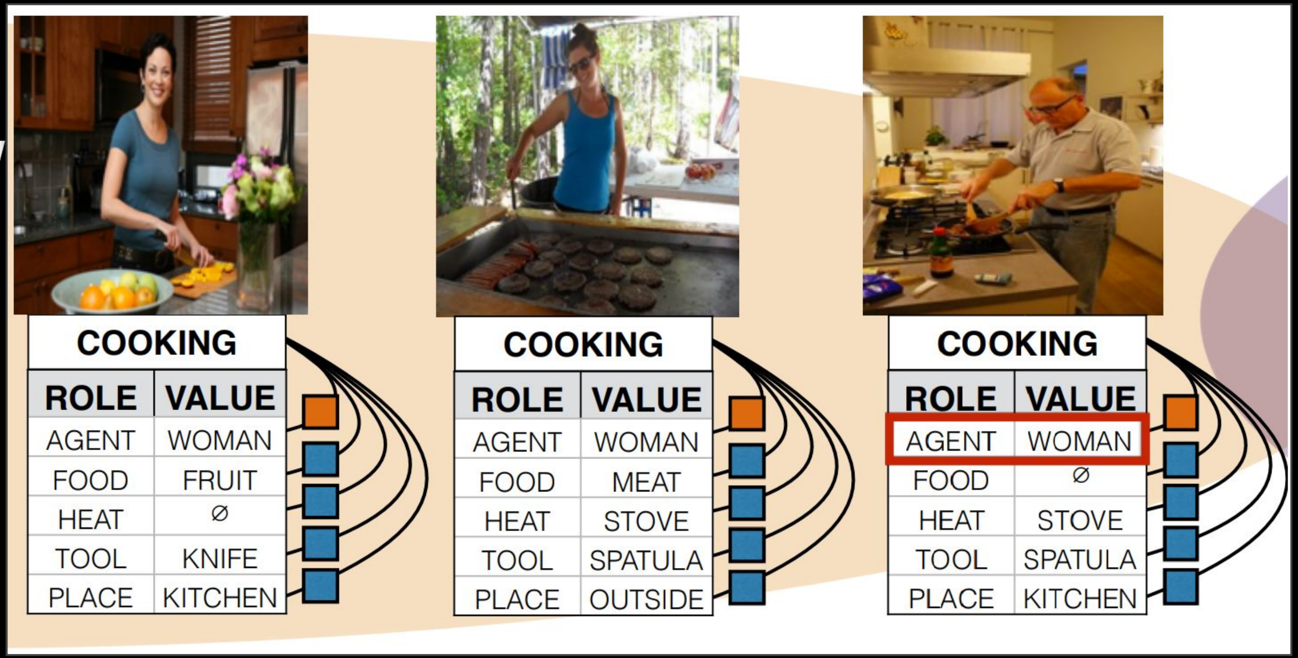
Jørgensen/Hovy/Søgaard, 2015
Hovy & Søgaard, 2015



Two Examples

The W

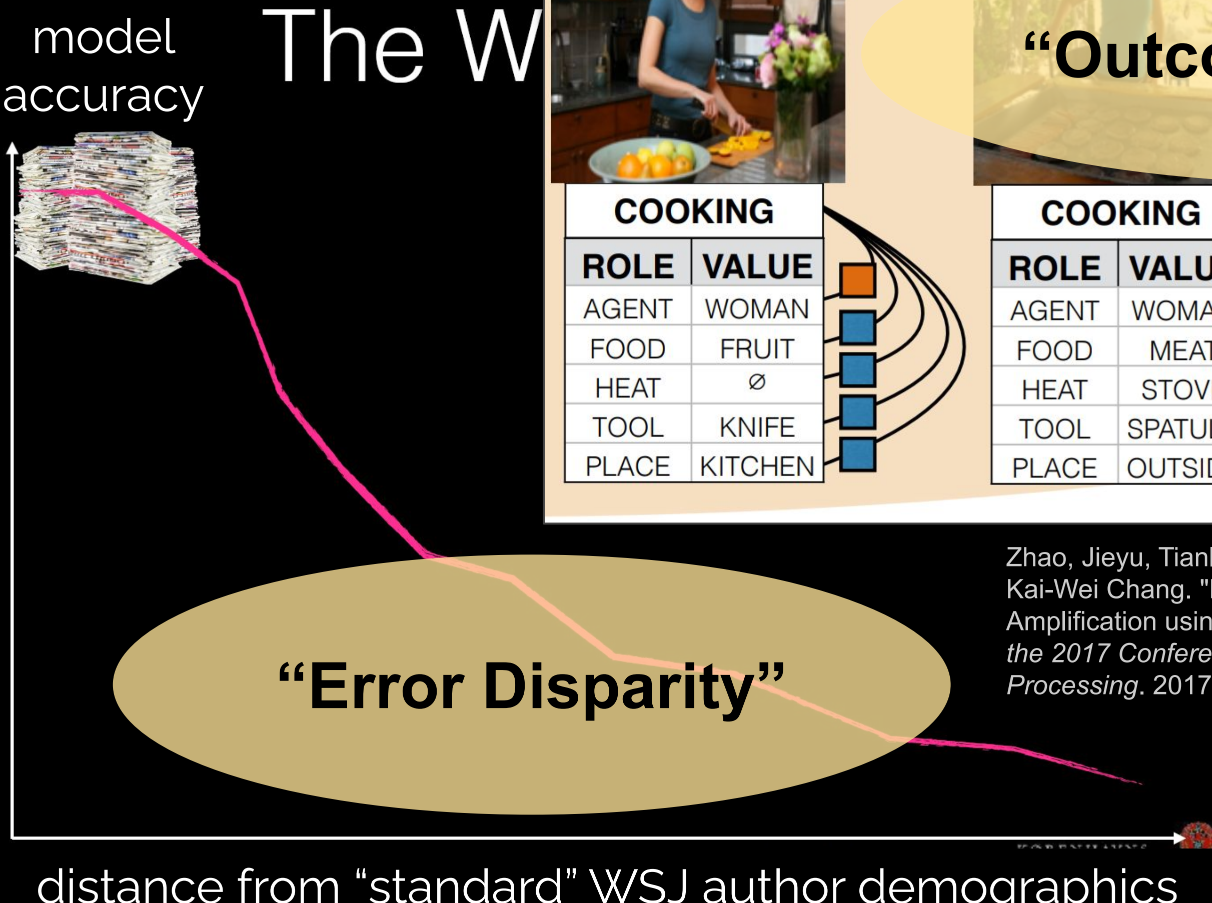


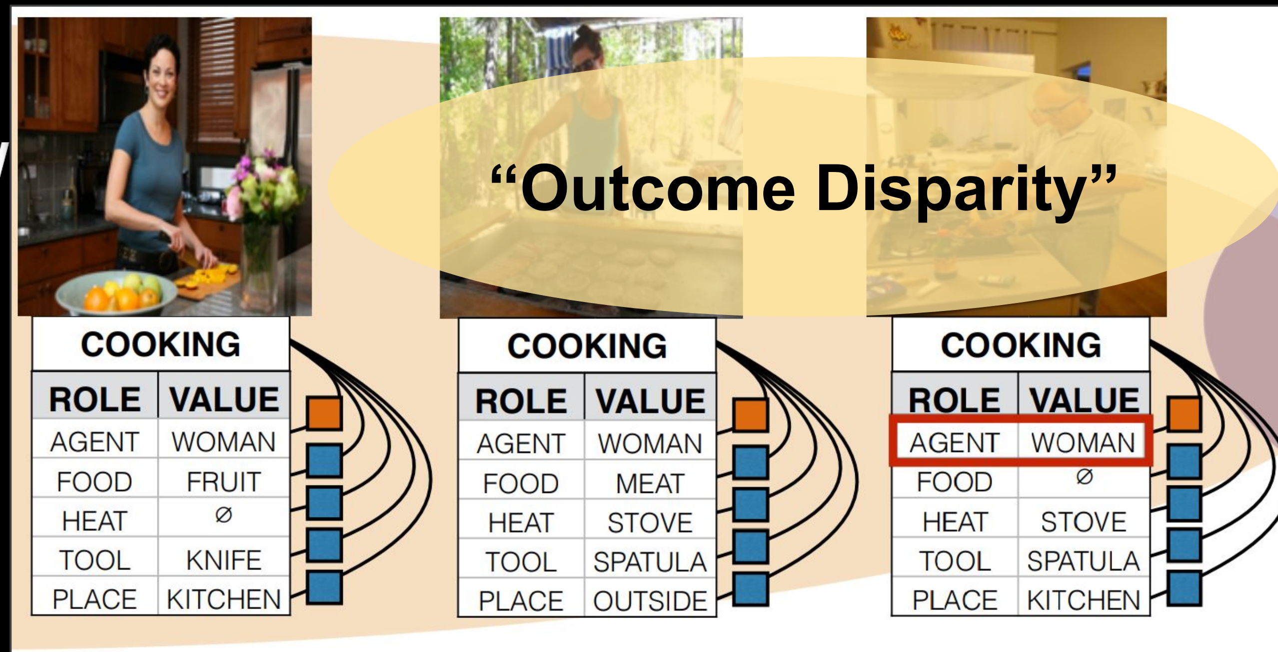


Zhao, Jieyu, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. "Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017.

Two Examples

The W





Zhao, Jieyu, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. "Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017.

Our data and models are (human) biased.

“Outcome Disparity”

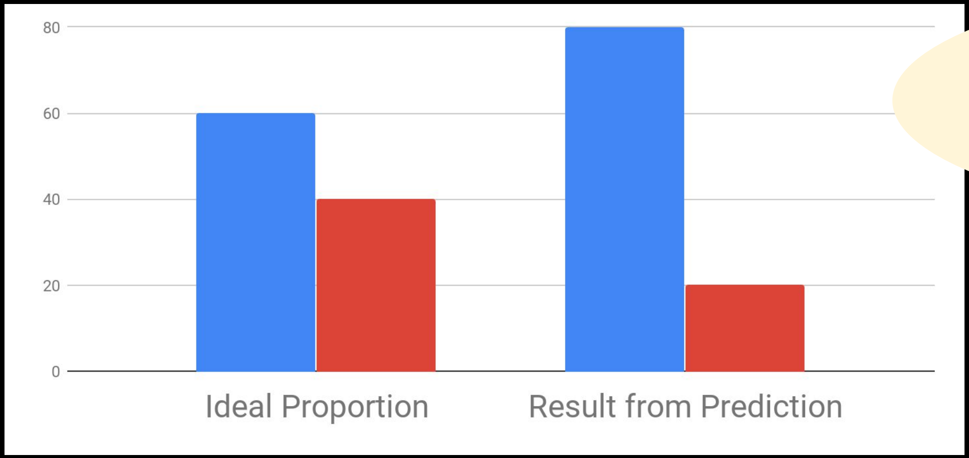
Person-level

attribute = 1

attribute = 2

“Error Disparity”

Our data and models are (human) biased.



“Outcome Disparity”

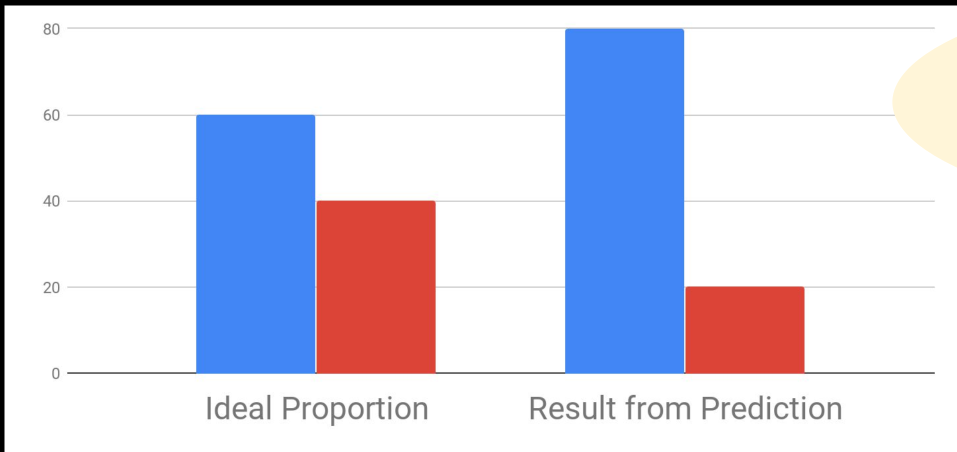
Person-level

attribute = 1

attribute = 2

“Error Disparity”

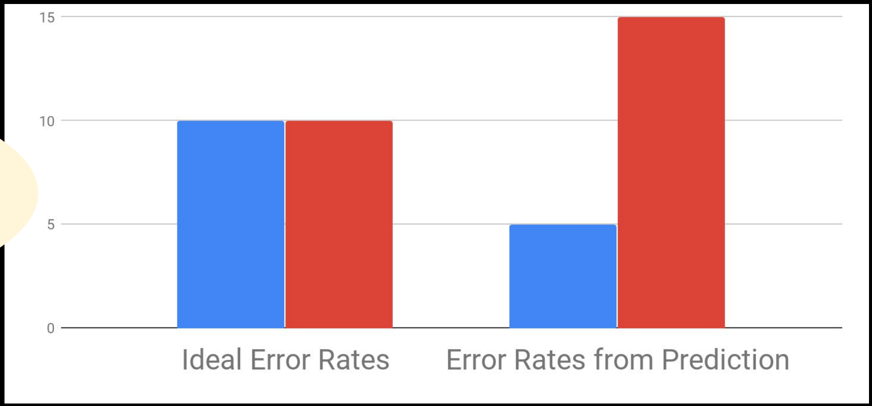
Our data and models are (human) biased.



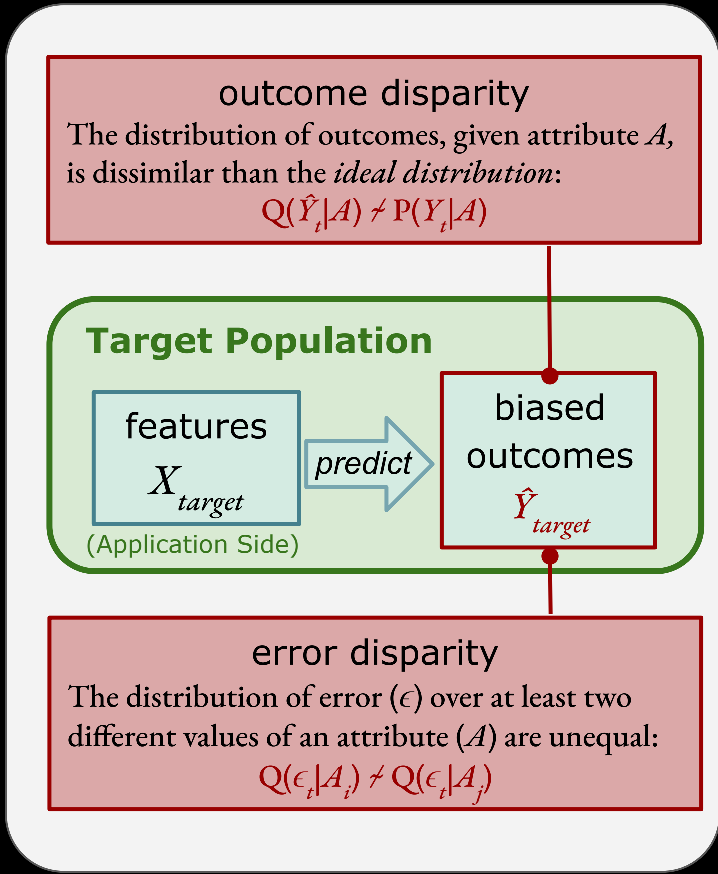
"Outcome Disparity"

Person-level
attribute = 1
attribute = 2

"Error Disparity"

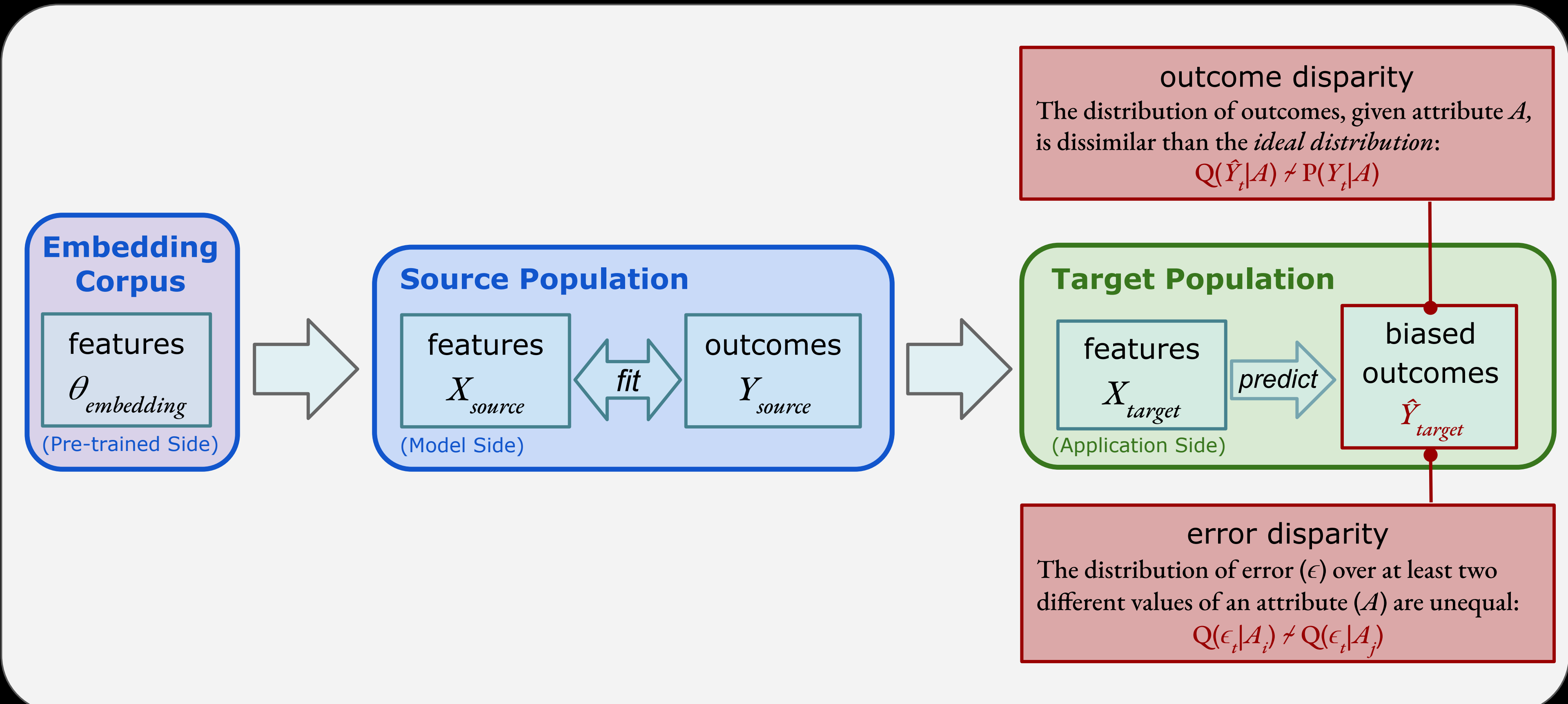


Predictive Bias Framework



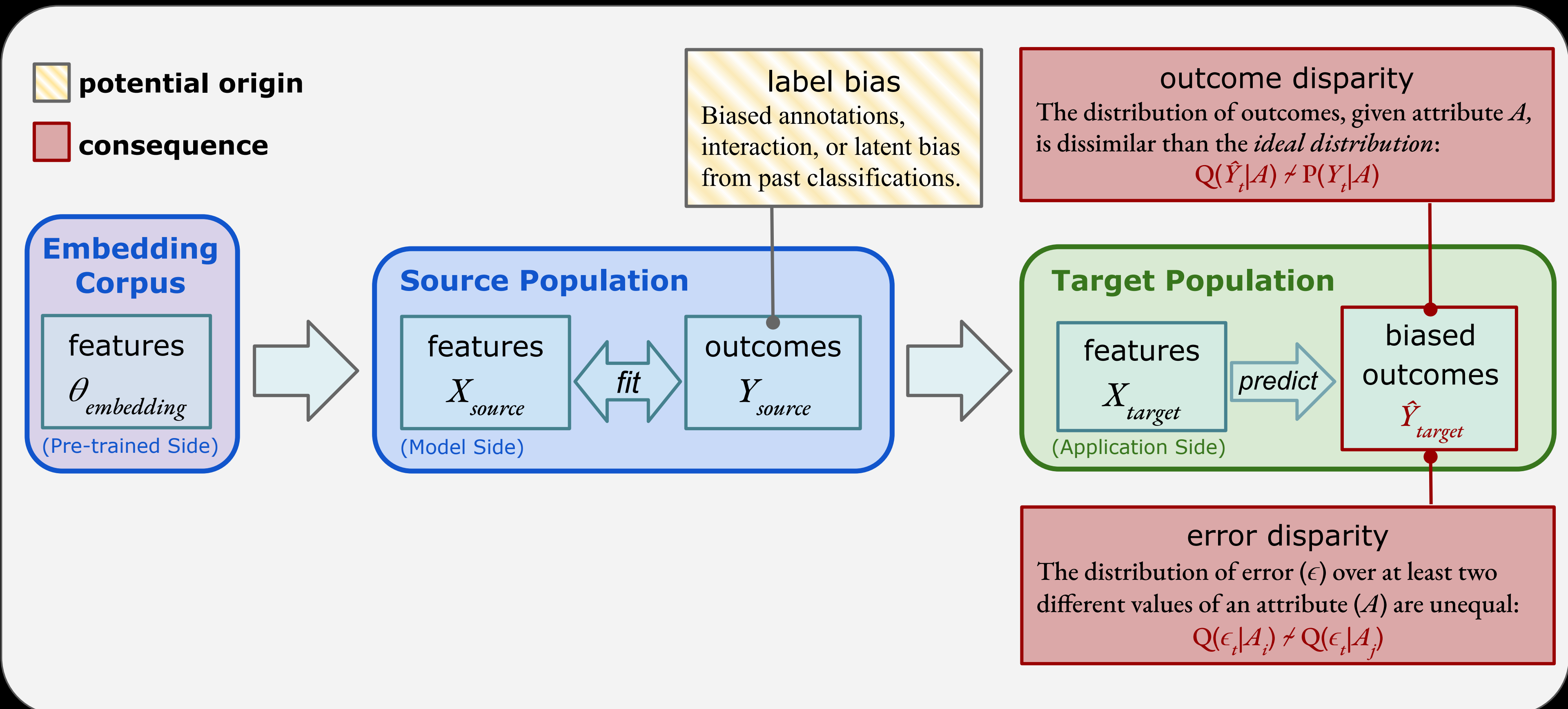
Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Predictive Bias Framework



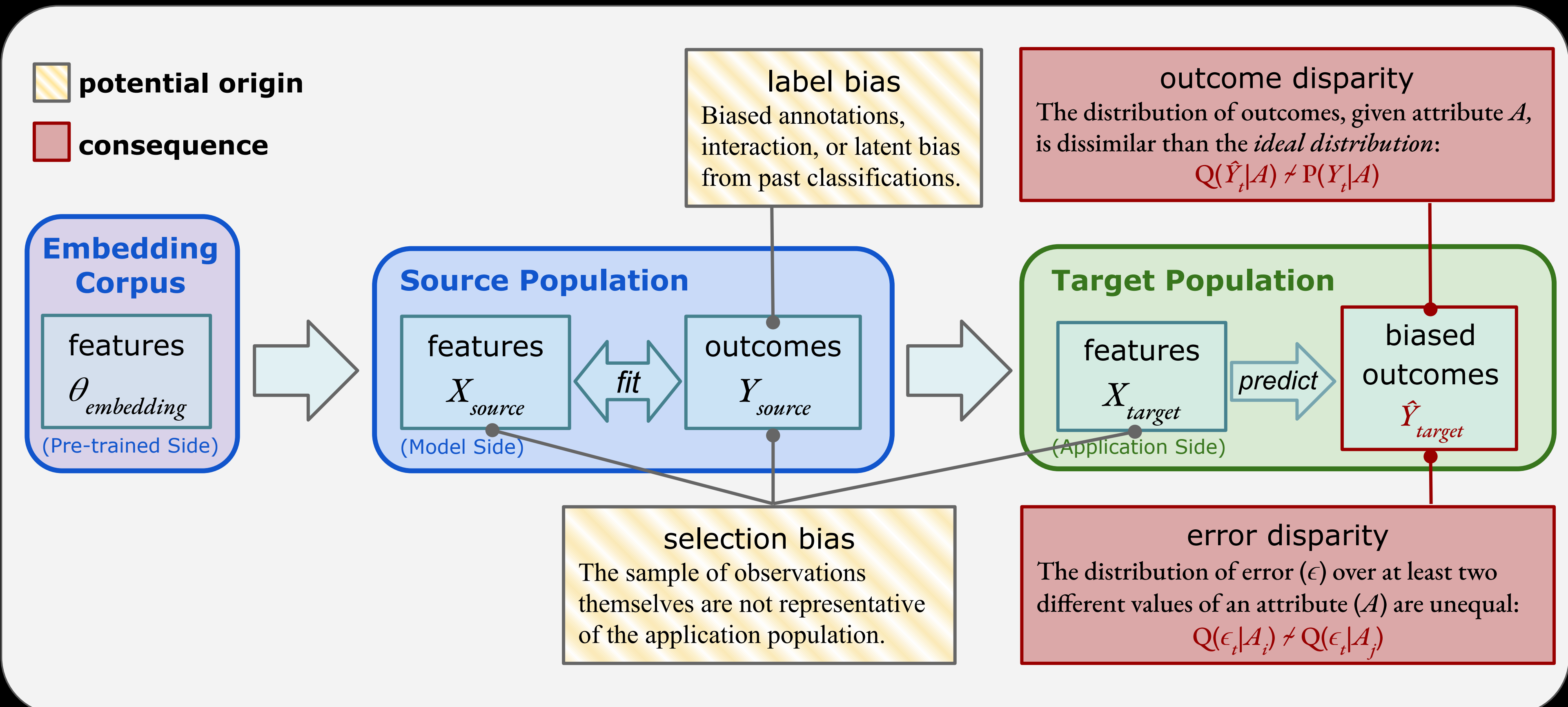
Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Predictive Bias Framework



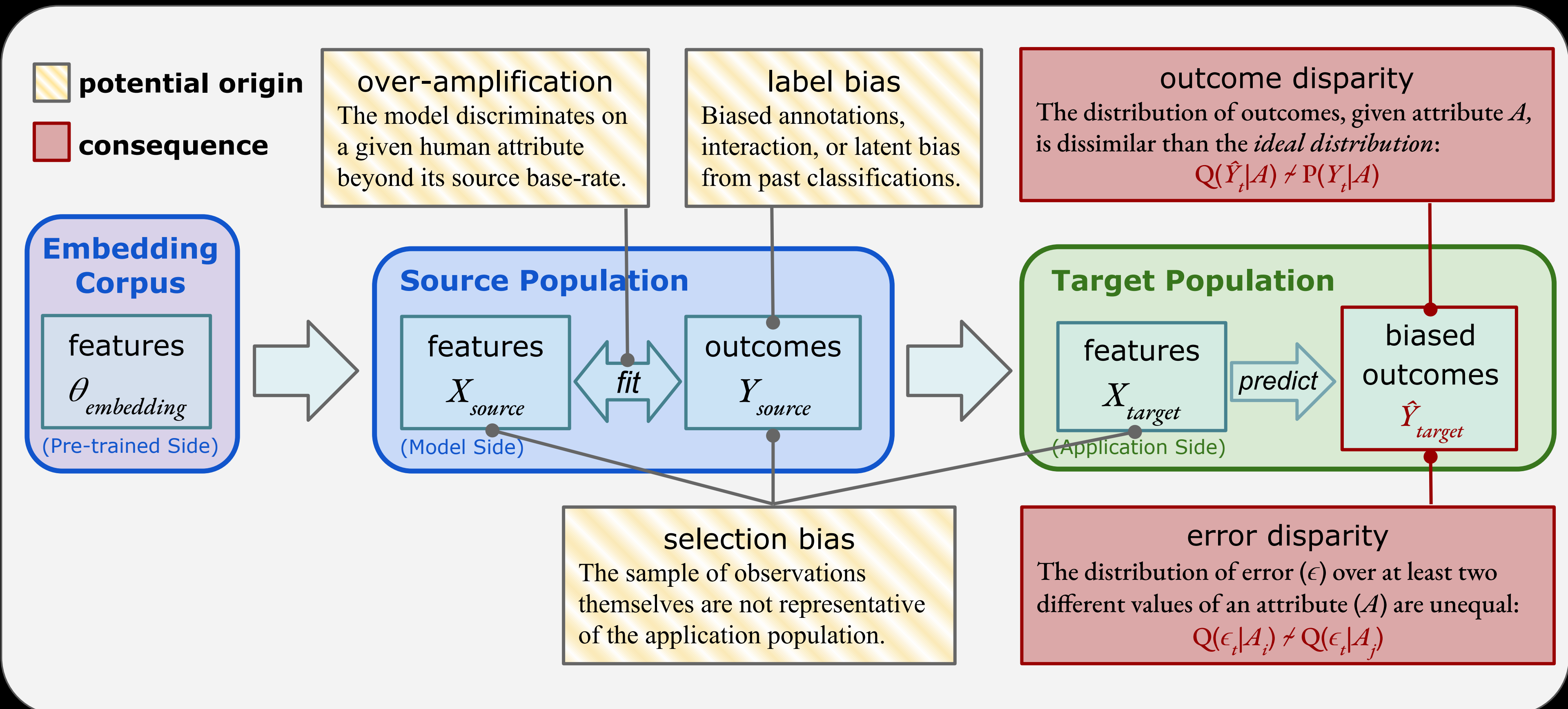
Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Predictive Bias Framework



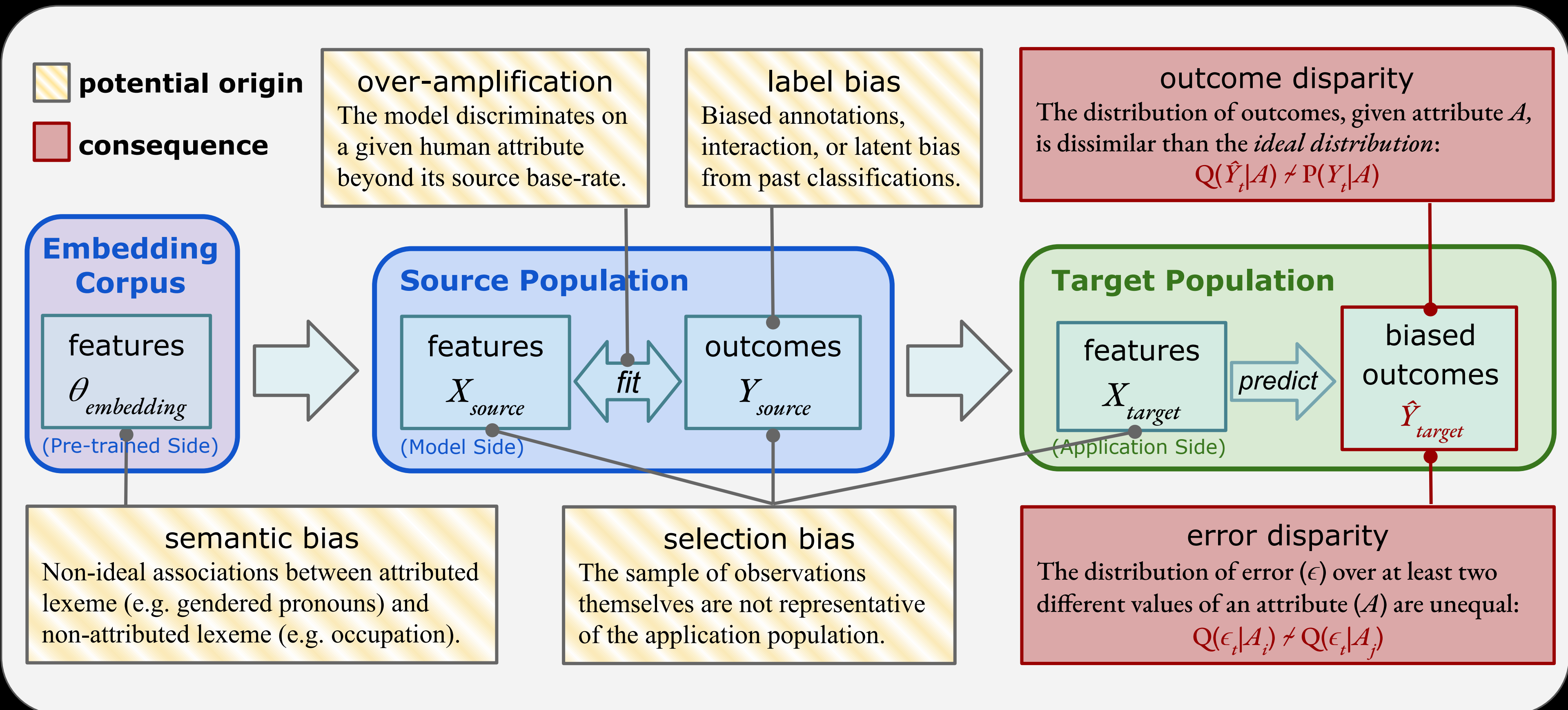
Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Predictive Bias Framework



Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Predictive Bias Framework



Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Predictive Bias Framework

E.g. Coreference resolution:
connecting entities to references (i.e. pronouns).

"*The doctor told Mary that she had run some blood tests.*"

semantic bias
Non-ideal associations between attributed lexeme (e.g. gendered pronouns) and non-attributed lexeme (e.g. occupation).

selection bias
The sample of observations themselves are not representative of the application population.

error disparity
The distribution of error (ϵ) over at least two different values of an attribute (A) are unequal:
$Q(\epsilon_t|A_i) \nsim Q(\epsilon_t|A_j)$

Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Ethics in Big Data

Types of bias:

- **Outcome Disparity:** Predicted distribution given A, are dissimilar from ideal distribution given A
  - Selection bias
  - Label bias
  - Over-amplification
- **Error Disparity:** Predicts less accurate for authors of given demographics.
- **Semantic Bias:** Representations of meaning store demographic associations.

Shah, D., Schwartz, H. A., Hovy, D. (2020). Predictive Biases in Natural Language Processing Models: A Conceptual Framework and Overview. *In ACL-2020: Proceedings of the Association for Computational Linguistics.*

Ethics in Big Data

Privacy

- Risk Categories:
  - Revealing unintended private information
  - Targeted persuasion

Ethics in Big Data

Privacy

- Risk Categories:
  - Revealing unintended private information
  - Targeted persuasion

- Mitigation strategies:
  - Informed consent -- let participants know
  - Do not share / secure storage
  - *Federated learning* -- separate and obfuscate to the point of preserving privacy
  - Transparency in information targeting

    "You are being shown this ad because …"

Ethics in Big Data

Human Subjects Research

Observational versus Interventional

Ethics in Big Data

Human Subjects Research

Observational versus Interventional

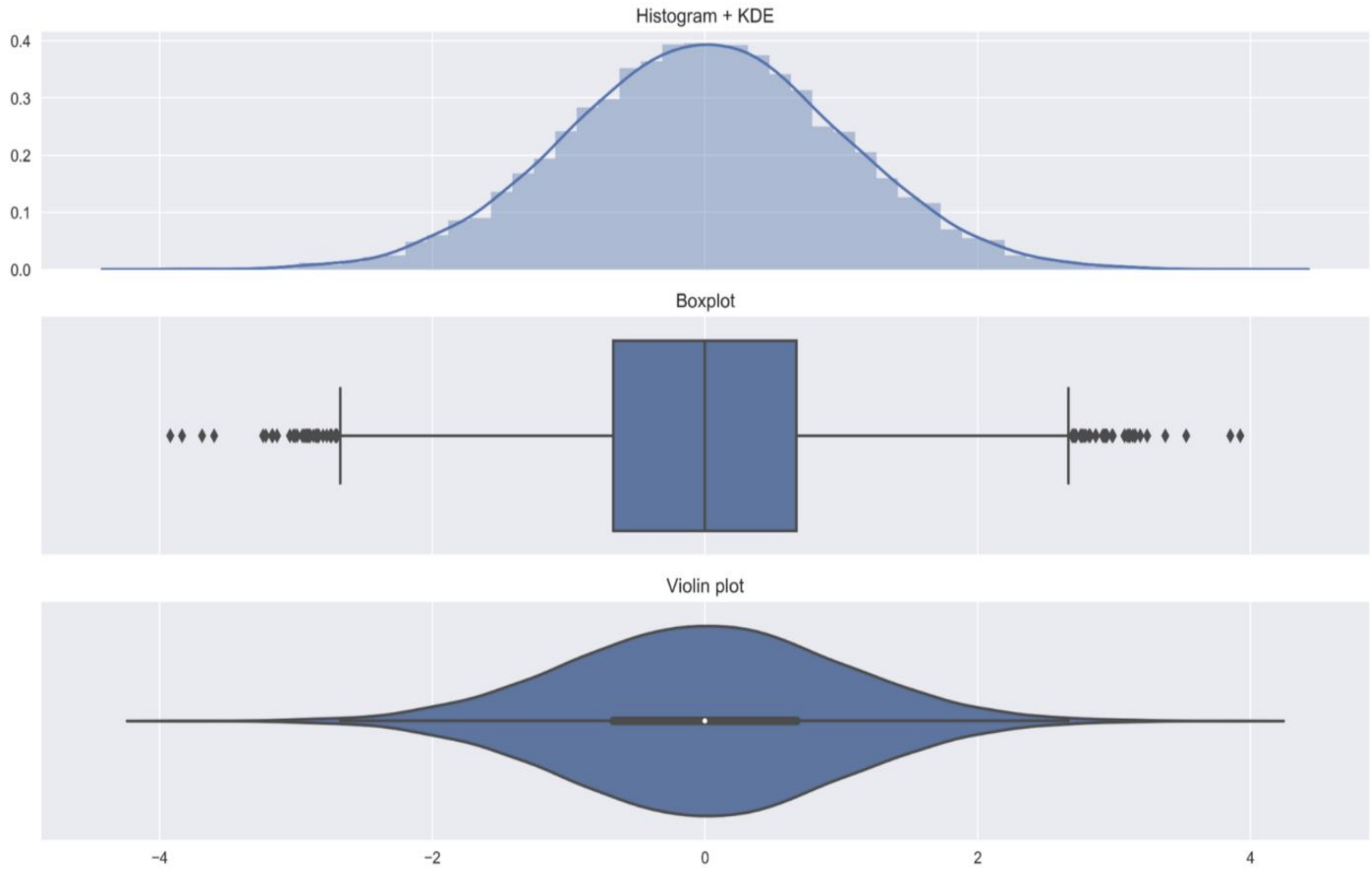
(The Belmount Report, 1979)

(i) Distinction of research from practice.
(ii) Risk-Benefit criteria
(iii) Appropriate selection of human subjects for participation in research
(iv) Informed consent in various research settings.

Post-Exam2 Topics:

1. Research Ethics
2. **Useful Plots**
3. Machine Learning Cross Validation
4. Convolutional Neural Networks
5. Recurrent Neural Networks
6. Transformer Networks

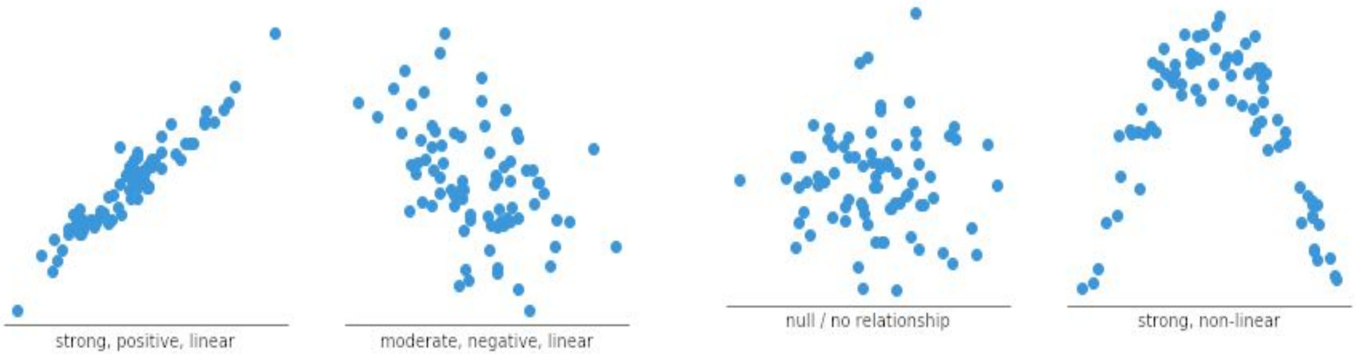
Useful Plots: For distributions



(Lewinson, 2019)

Useful Plots: Correlation

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



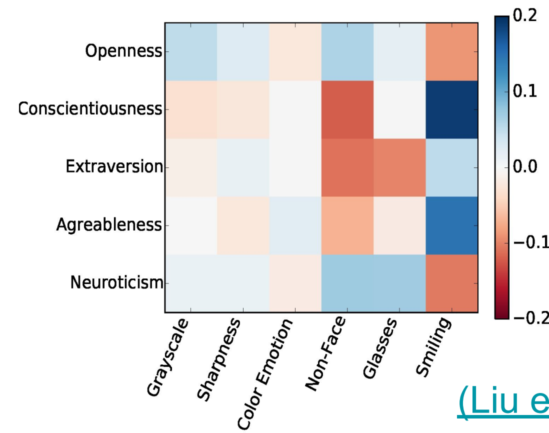
(Chartio)

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).

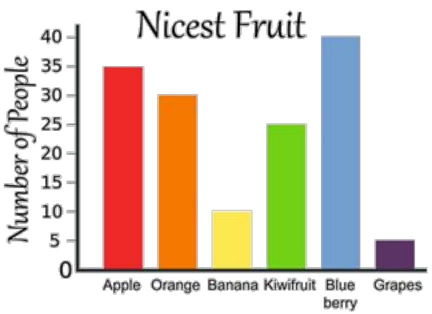
https://doi.org/10.1371/journal.pone.0201703.g003



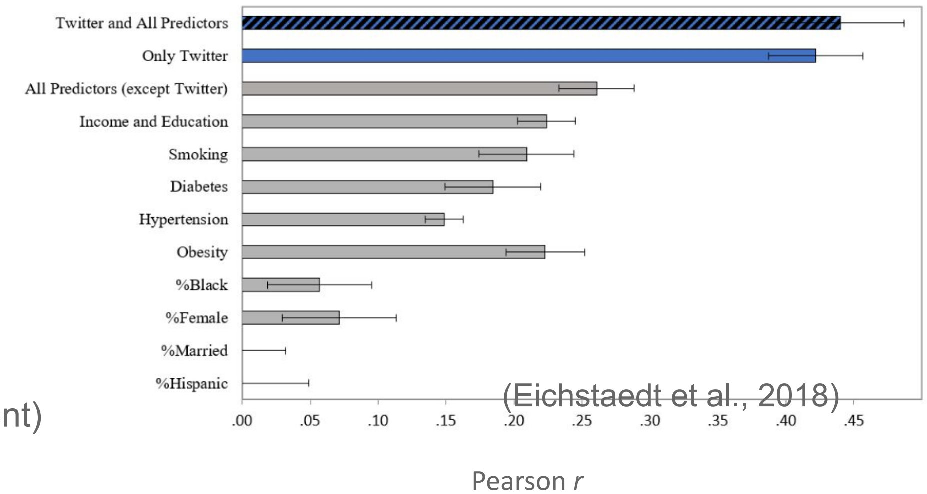
(Liu et al., 2016)

Useful Plots: Any Values

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

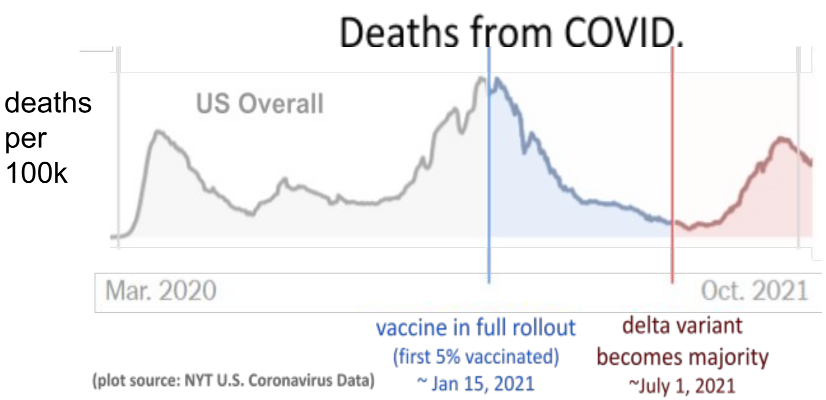


(Science sEDiment)



(Eichstaedt et al., 2018)

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



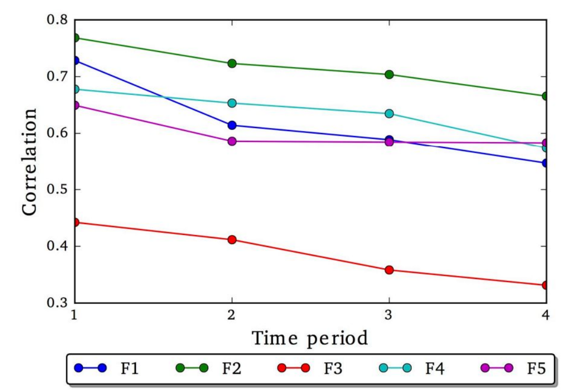


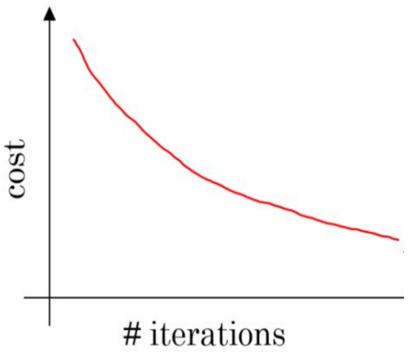
Fig 6. Test re-test validity of our learned factors.

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

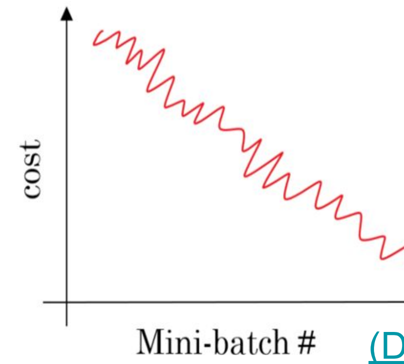
Useful Plots: Prediction

Learning Curve: for plotting error from gradient descent.

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

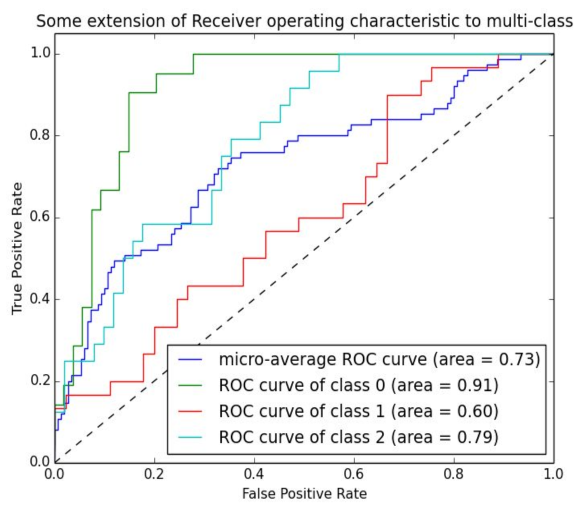


for a model with non-convex optimization (i.e. most deep learning)

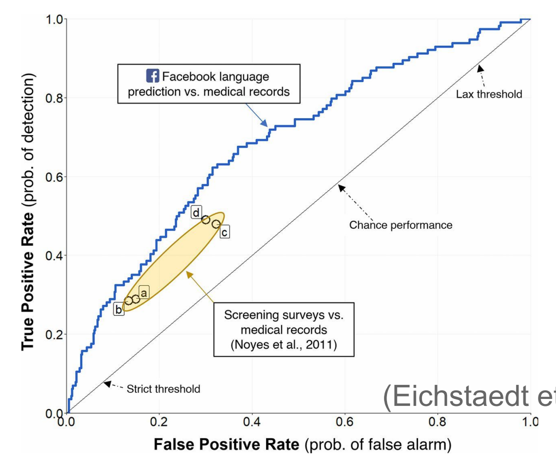


[(Dabura, 2017)]

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



[(PLOT_ROC)]



(Eichstaedt et al., 2018)

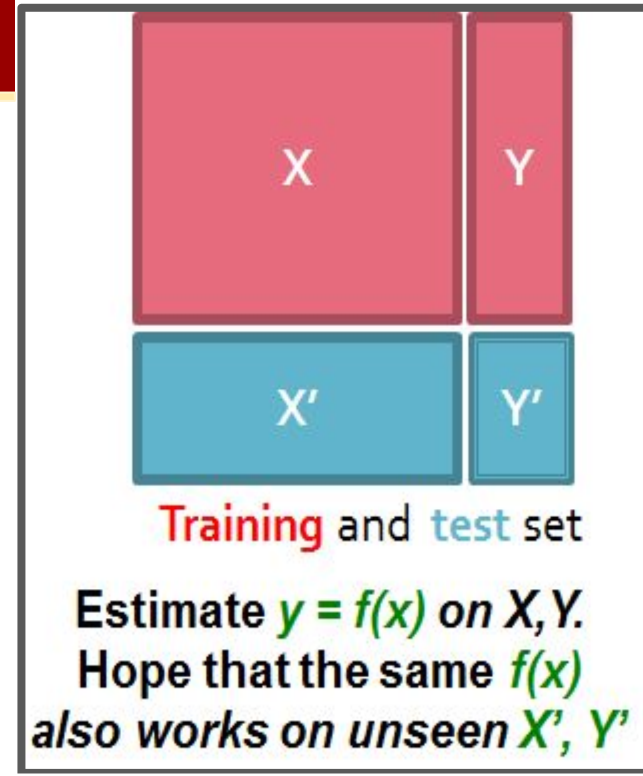
Post-Exam2 Topics:

1. Research Ethics
2. Useful Plots
3. **Machine Learning Cross Validation**
4. Convolutional Neural Networks
5. Recurrent Neural Networks
6. Transformer Networks

Supervised Learning

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

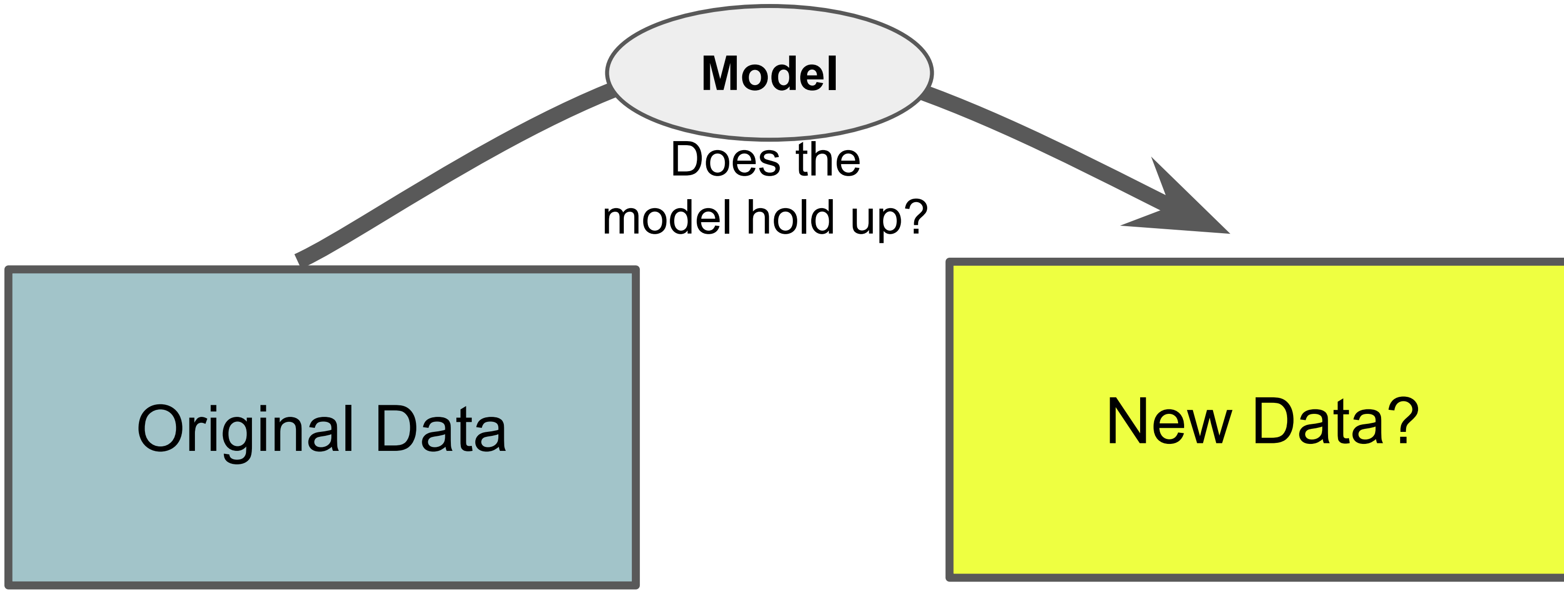
$X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6$
$X_7 \quad X_8 \quad X_9 \quad X_{10} \quad X_{11} \quad X_{12}$
$X_{13} \quad X_{14} \quad X_{15} \quad \ldots \quad X_m$



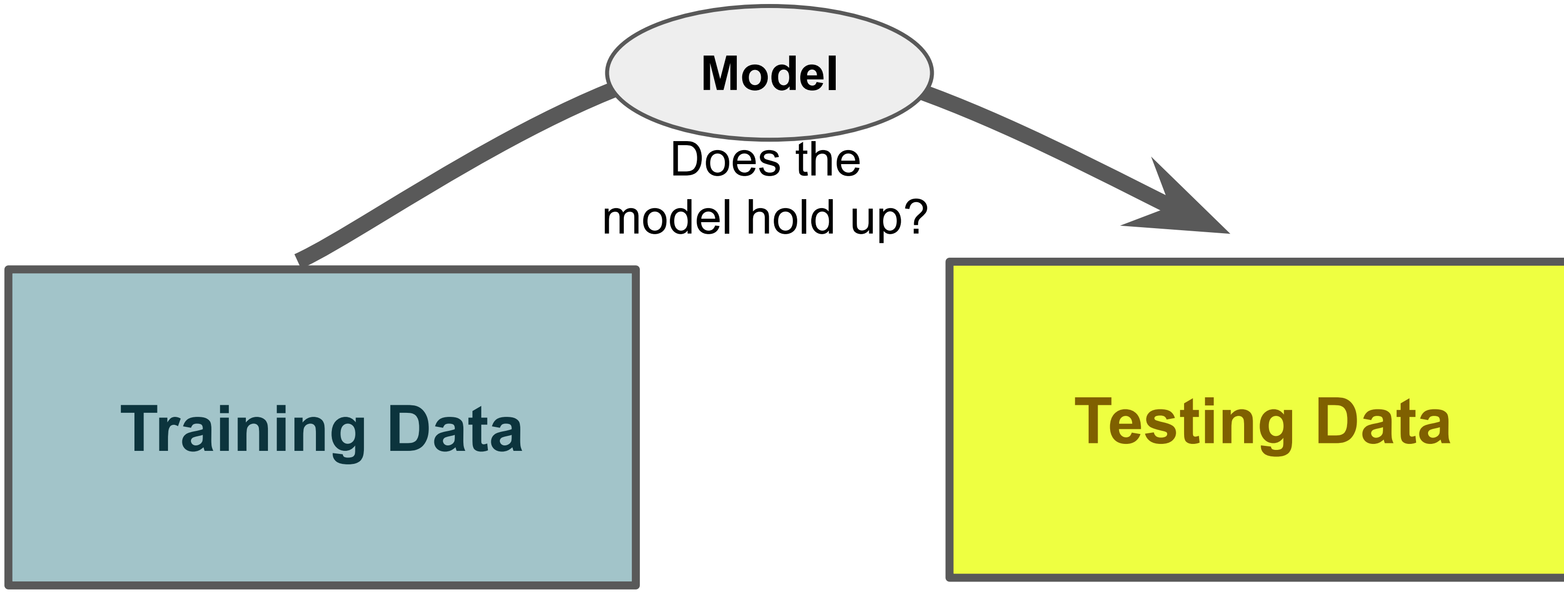
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

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

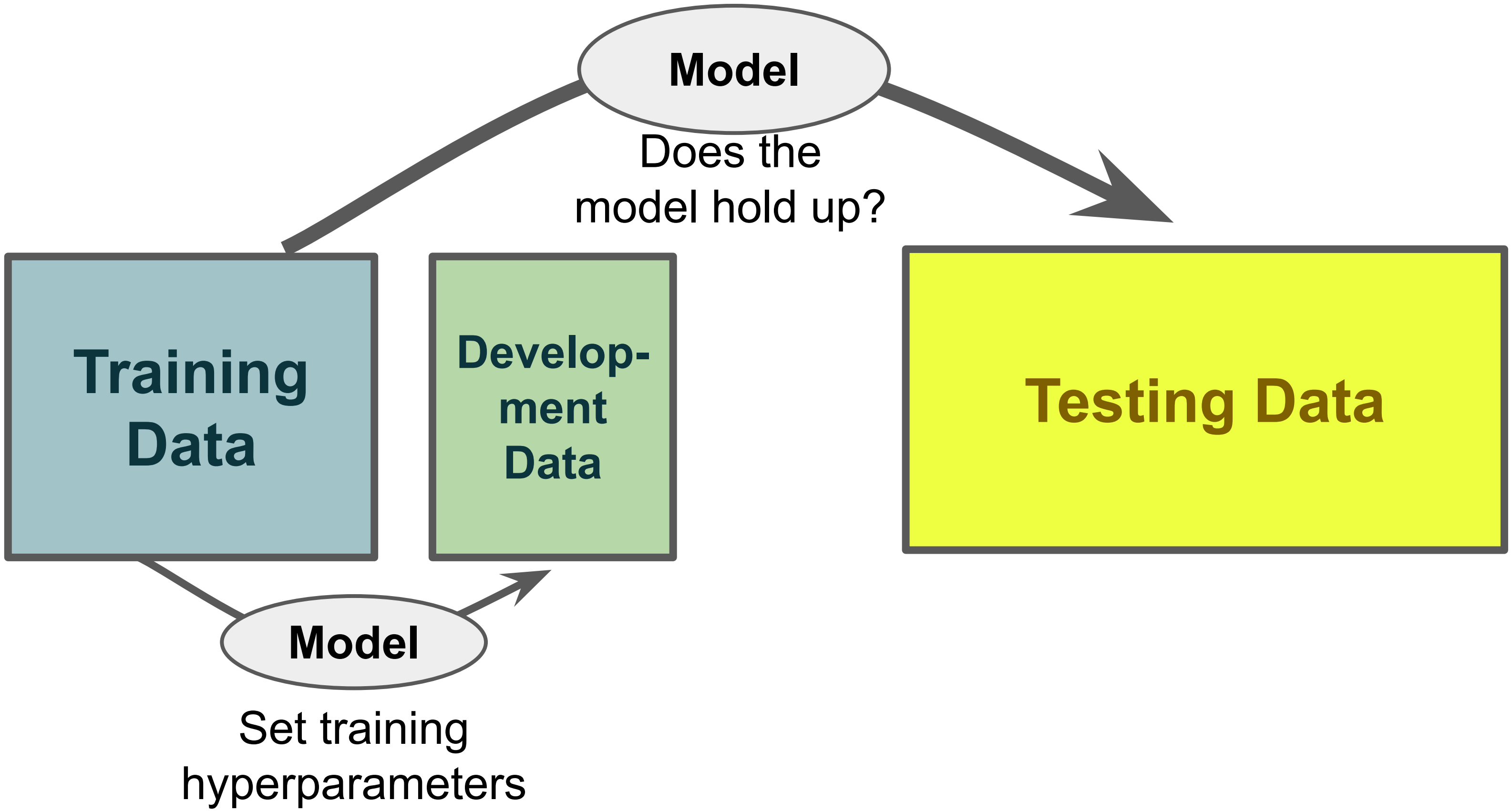
Common Goal: Generalize to new data



Common Goal: Generalize to new data

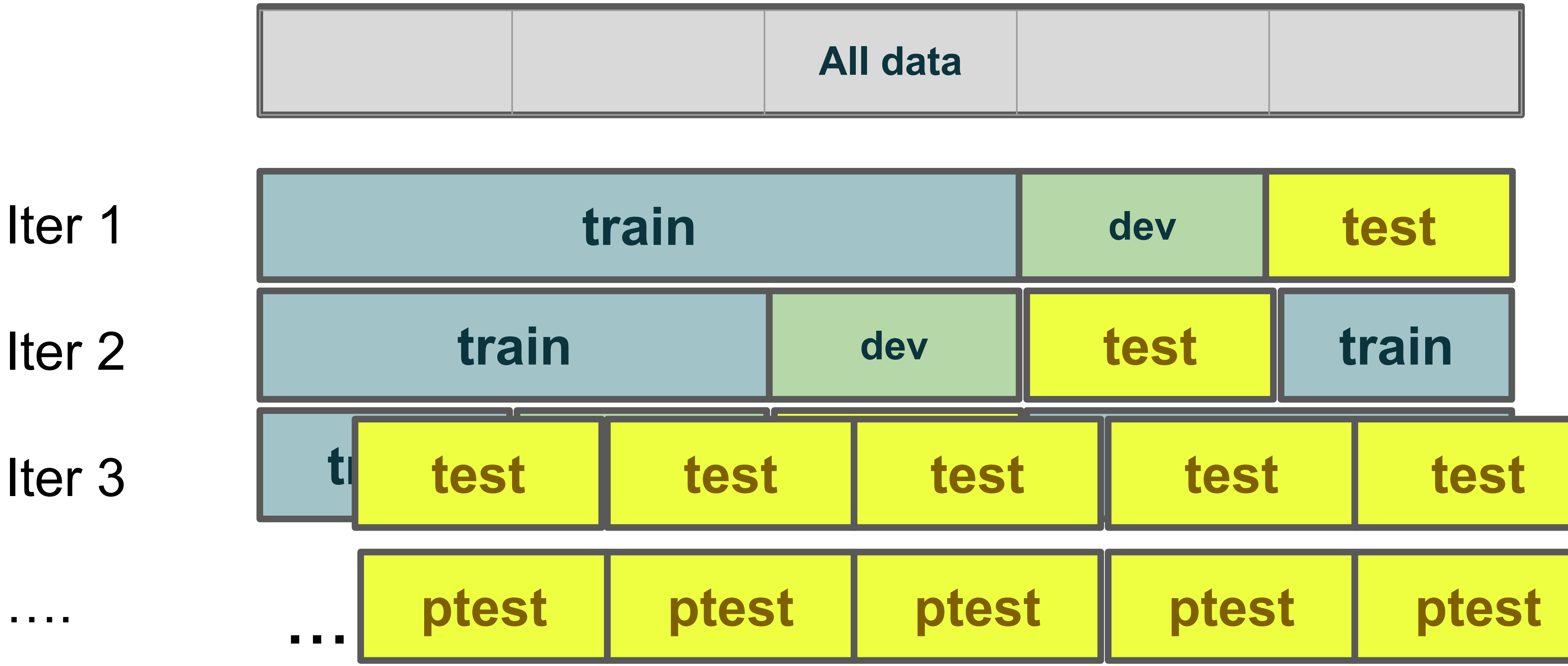


ML: GOAL


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



observed dep variable	test	test	test	test	test
estimated dep variable	ptest	ptest	ptest	ptest	ptest

Review: Distributed ML

Done very often in practice. Not talked about much because it's mostly as easy as it sounds.

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

   Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

   Data Parellelism

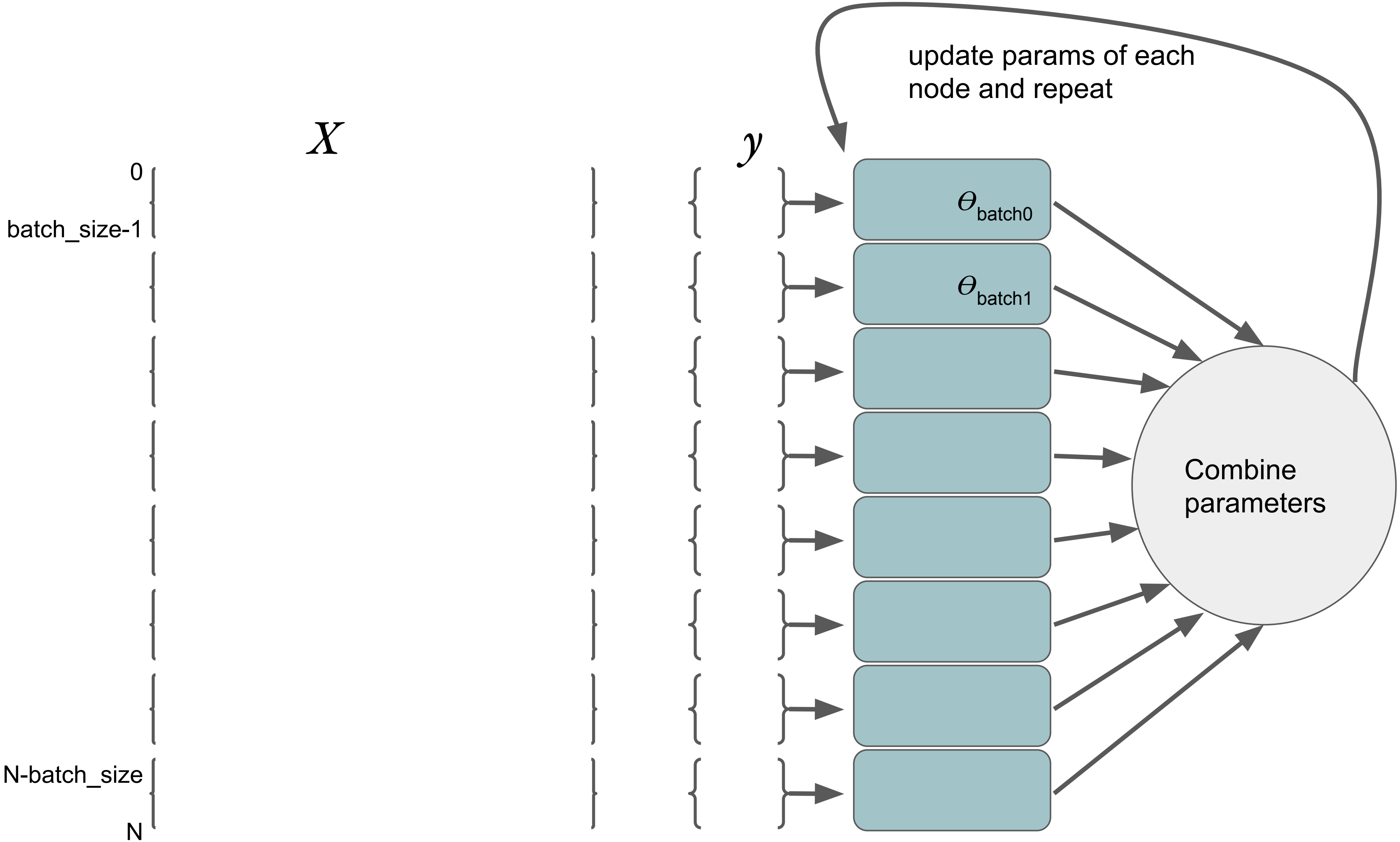
   Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

Preferred method for big data or very complex models (i.e. models with many internal parameters).

3. **Distribute model or individual operations** (e.g. matrix multiply)

   Model Parellelism

   Pro: Parameters can be localized Con: High communication for transferring Intermediar data.


update params of each node and repeat
X
y
0
batch_size-1
N-batch_size
N
θ batch0
θ batch1
Combine parameters

Post-Exam2 Topics:

1. Research Ethics
2. Useful Plots
3. Machine Learning Cross Validation
4. **Recurrent Neural Networks**
5. Convolutional Neural Networks
6. Transformer Networks

From Linear Models to Neural Nets

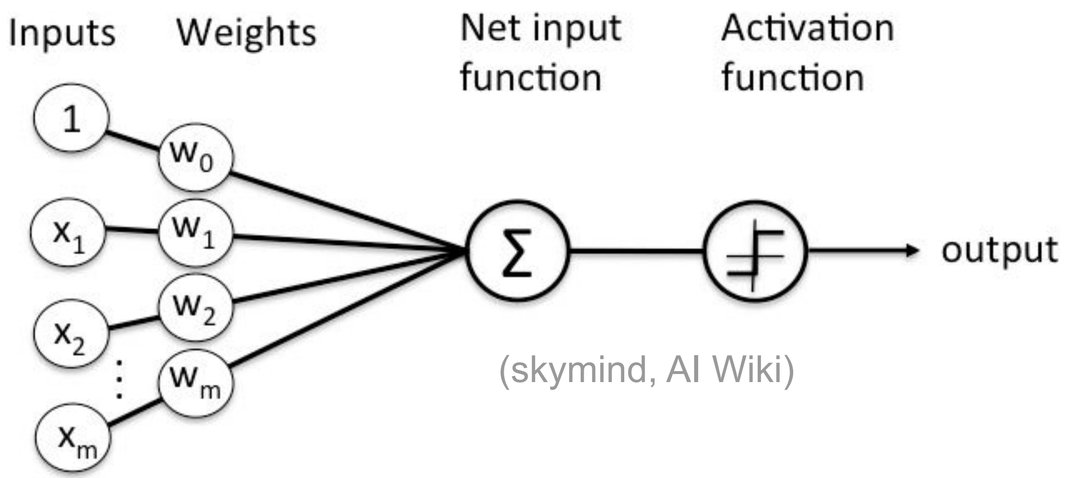
Linear Regression: $y = wX$

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

From Linear Models to Neural Nets

Linear Regression: $y = wX$

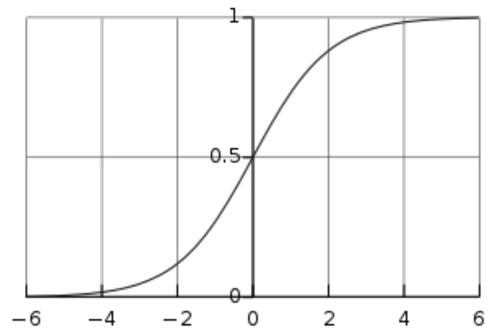
Neural Network Nodes: $output = f(wX)$



(skymind, AI Wiki)

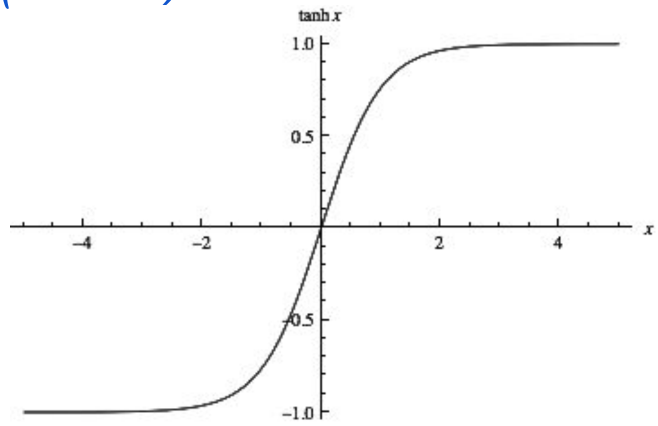
Common Activation Functions

$z = wX$

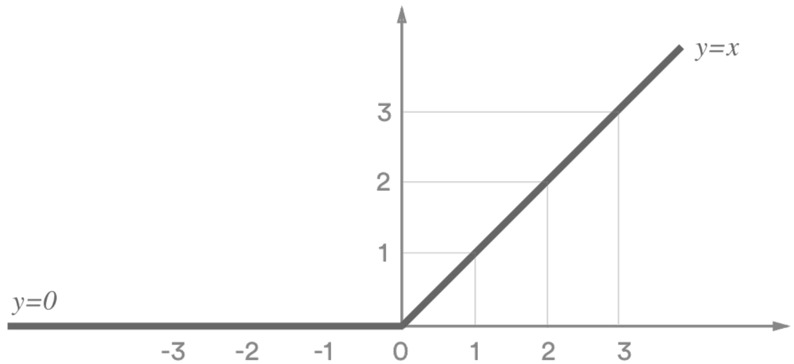


Logistic: $\sigma(z) = 1 / (1 + e^{-z})$

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



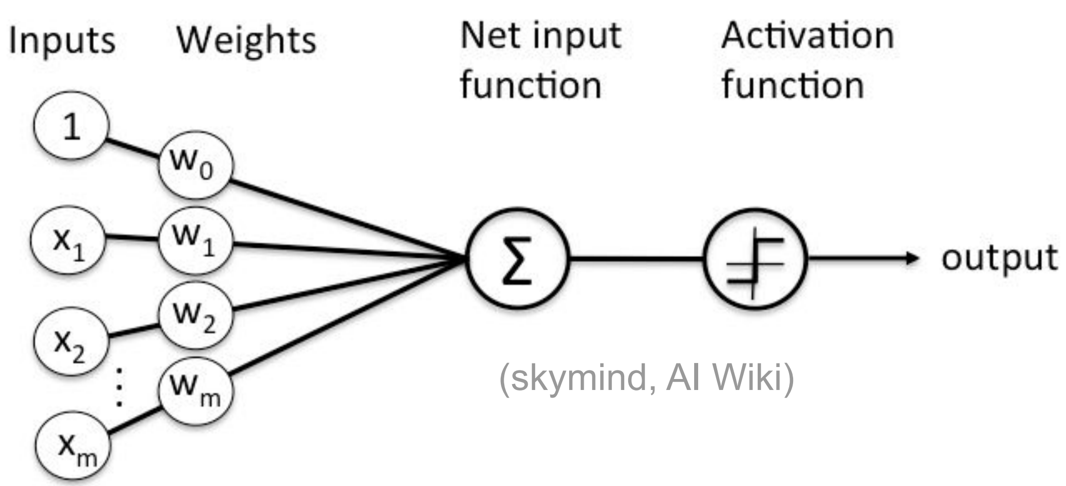
Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$



From Linear Models to Neural Nets

Linear Regression: $y = wX$

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

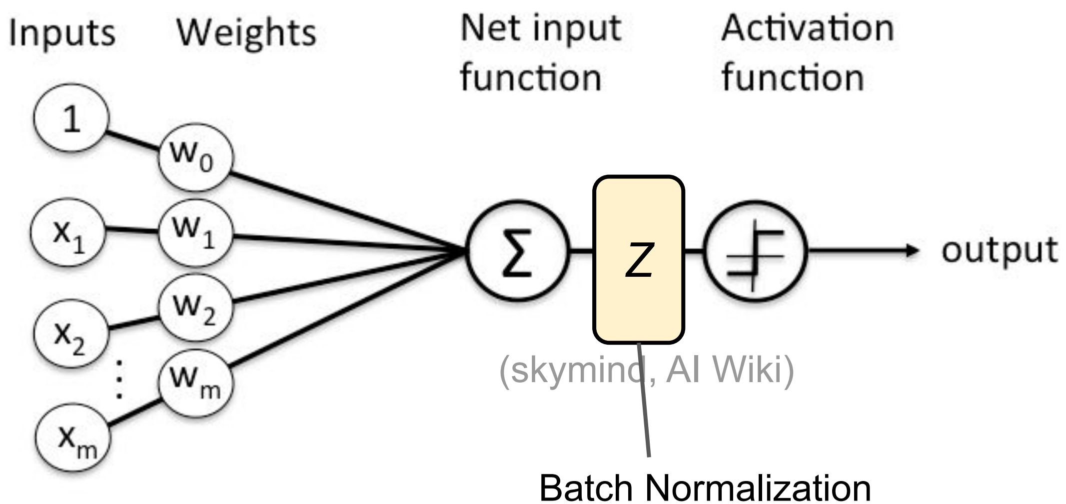


(skymind, AI Wiki)

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: $\mathcal{B} = \{x_{1\ldots m}\}$;
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: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

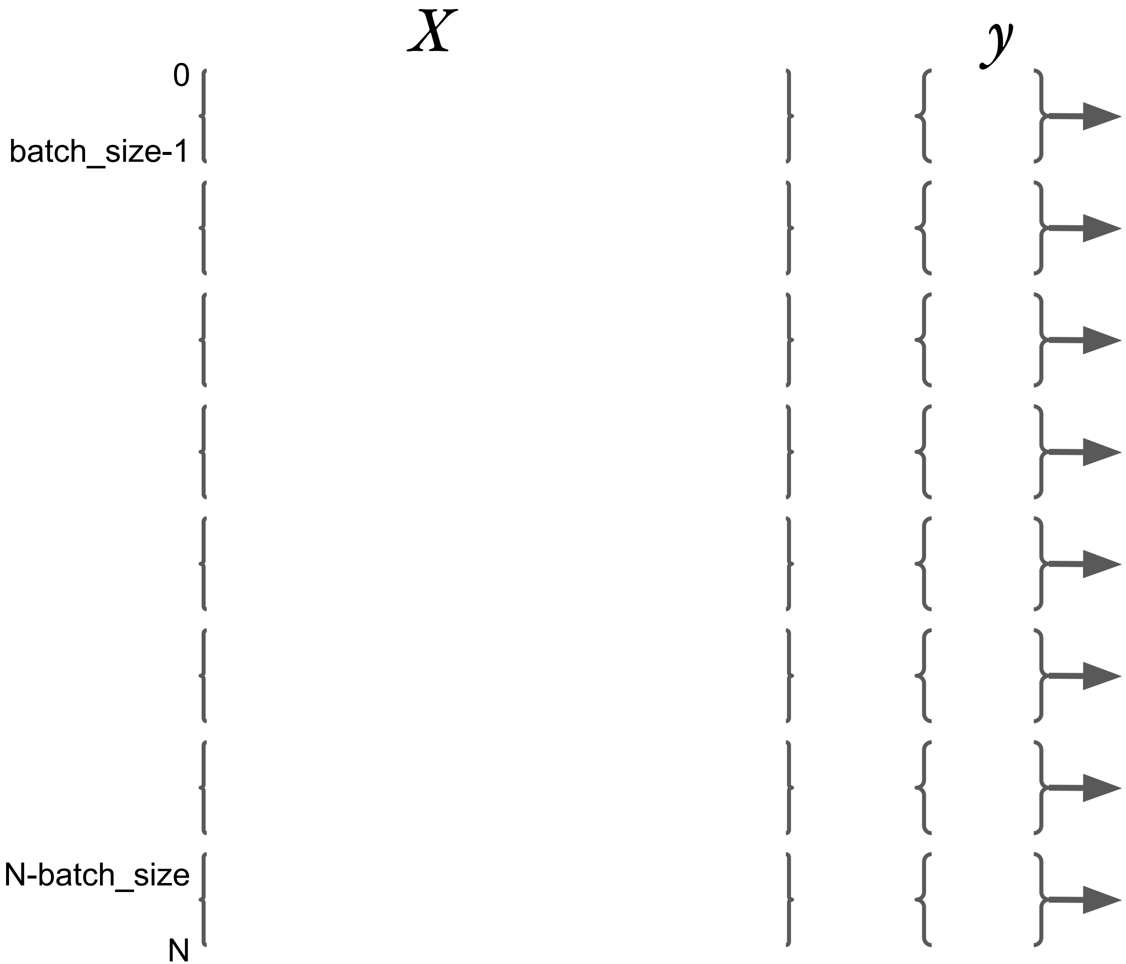
$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

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

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

(Ioffe and Szegedy, 2015)

Batch Normalization



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

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

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

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

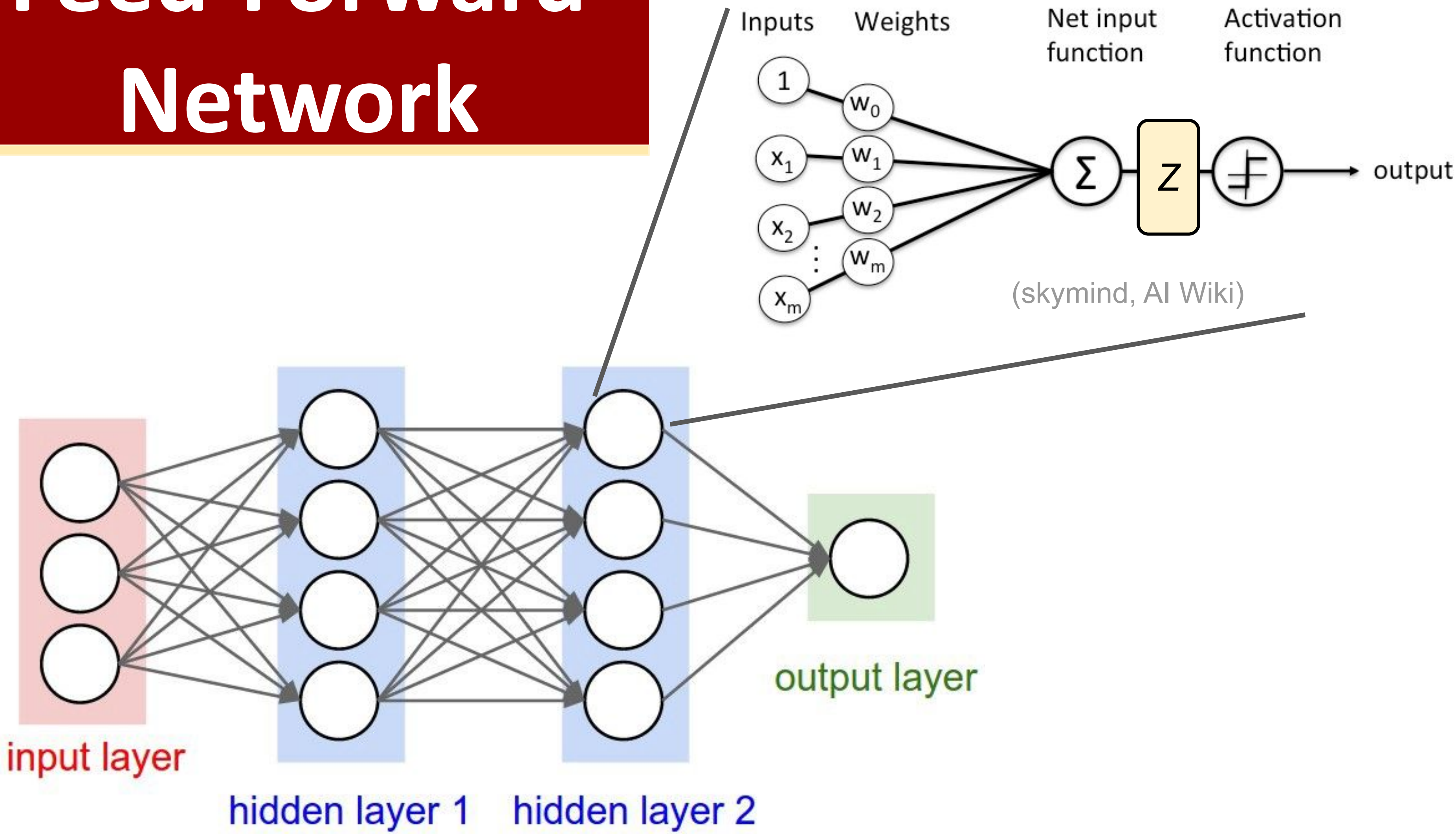
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

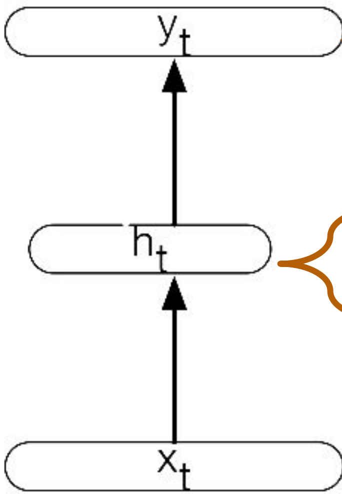
Why?

- 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.

Feed-Forward Network



Recurrent Neural Network



$y_{(t)} = f(h_{(t)}W)$

Activation Function

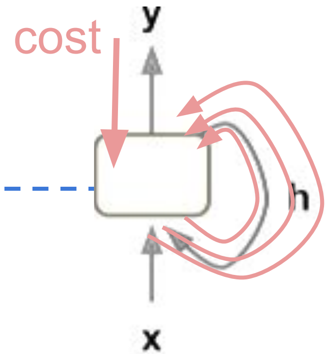
“hidden layer”

$h_{(t)} = g(h_{(t-1)}U + x_{(t)}V)$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

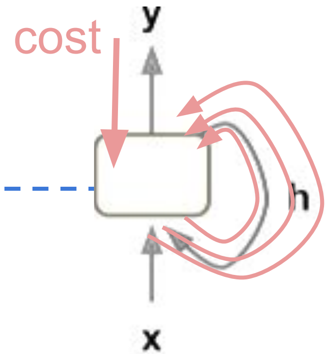
RNN: Optimization



Backward Propagation through Time

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden state
    y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
...
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

RNN: Optimization



Backward Propagation through Time

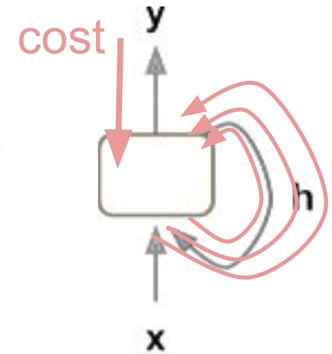
```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,
state
    y(i) = tf.softmax(tf.matmul
...
cost = tf.reduce_mean(-tf.redu
```

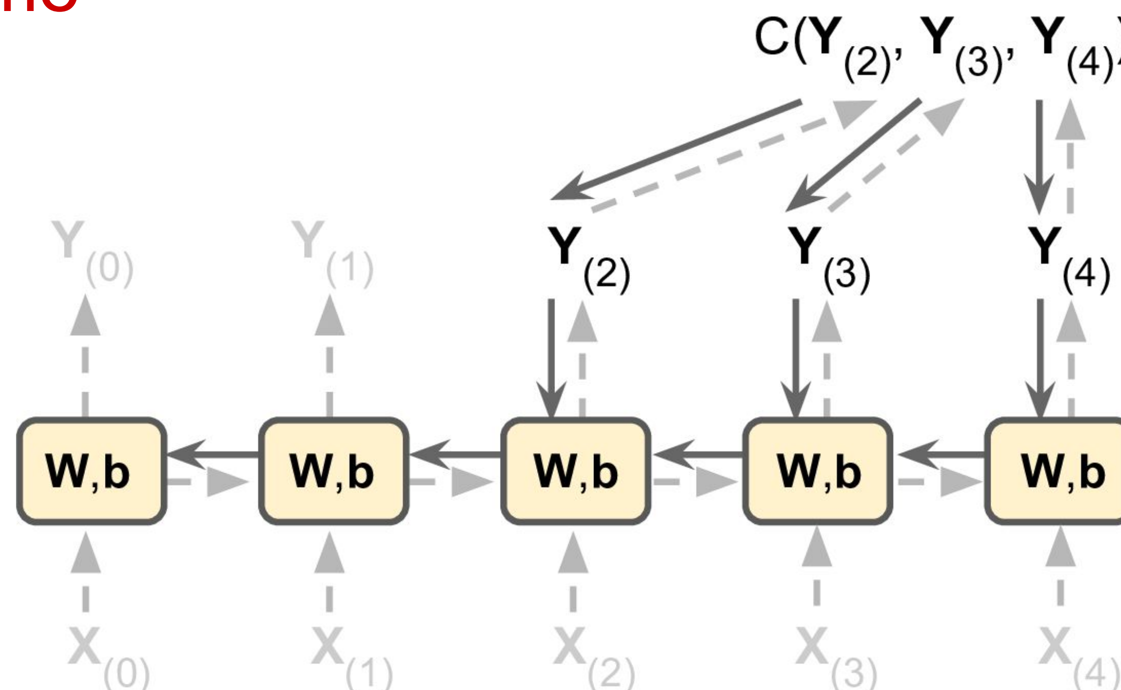
To find the gradient for the overall graph, we use **back propogation,** which *essentially* chains together the gradients for each node (function) in the graph.

With many recursions, the gradients can vanish or explode (become too large or small for floating point operations).

RNN: Optimization

Backward Propagation through Time

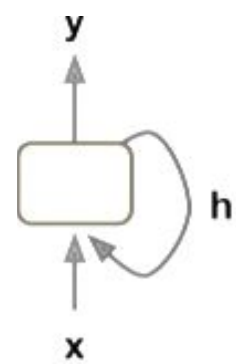




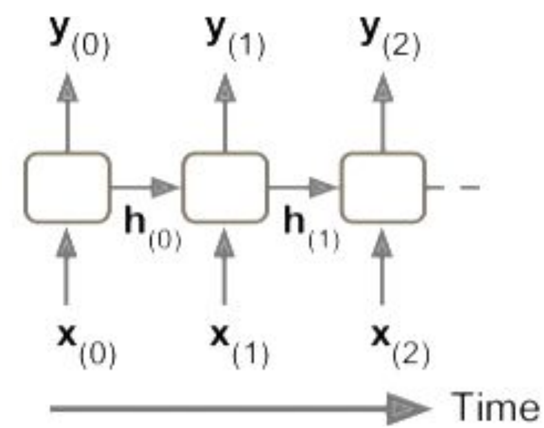
(Geron, 2017)

How to Addressing Vanishing Gradient?

Dominant approach: Use Long Short Term Memory Networks (LSTM)



RNN model

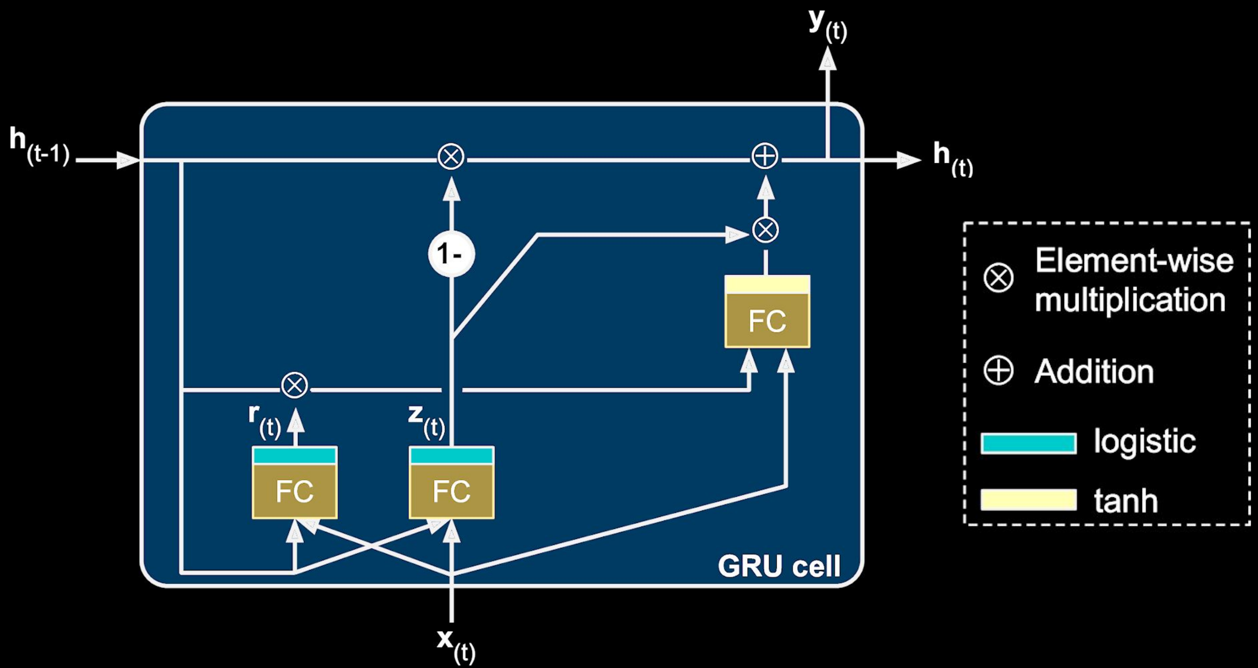


"unrolled" depiction

(Geron, 2017)

RNN: The GRU

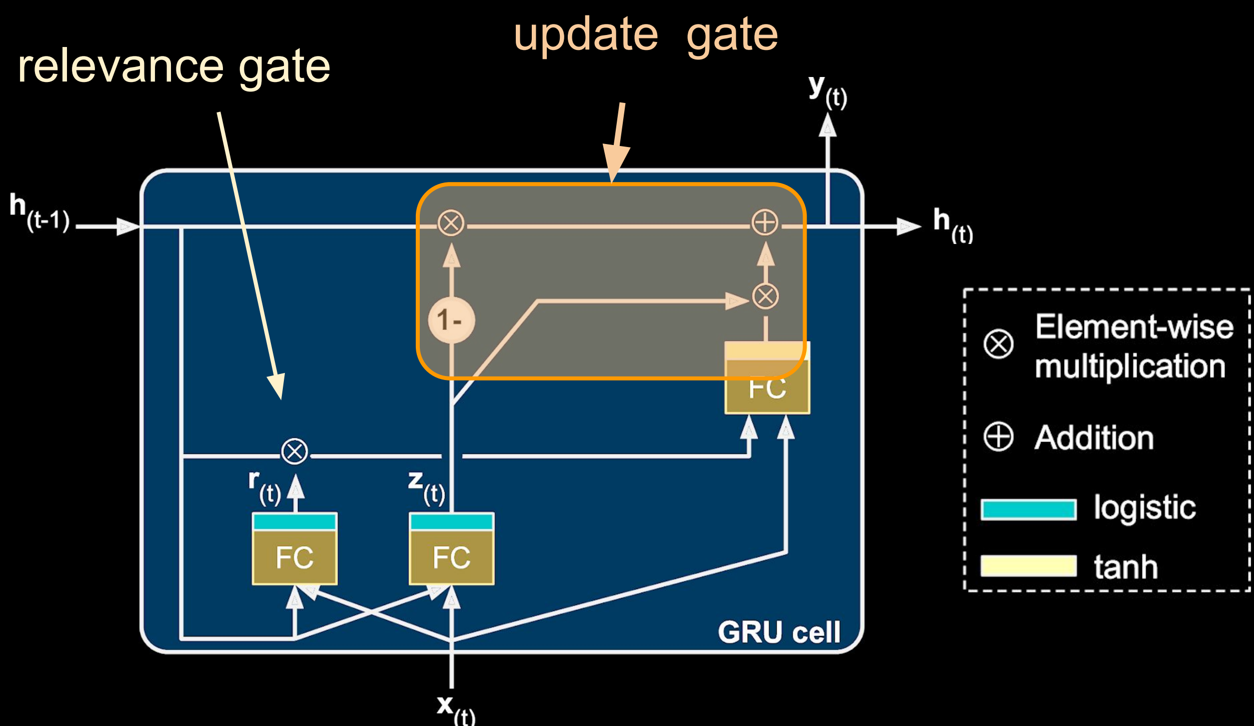
Gated Recurrent Unit



(Geron, 2017)

RNN: The GRU

Gated Recurrent Unit



(Geron, 2017)

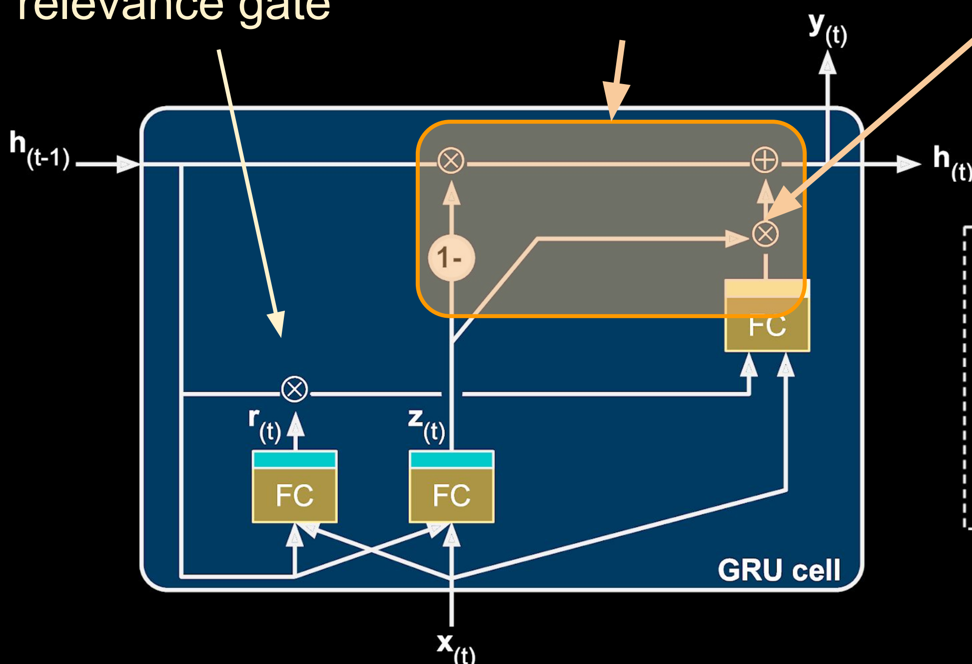
RNN: The GRU

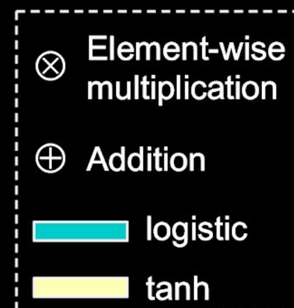
Gated Recurrent Unit

relevance gate

update gate

A candidate for updating h,
sometimes called: h~



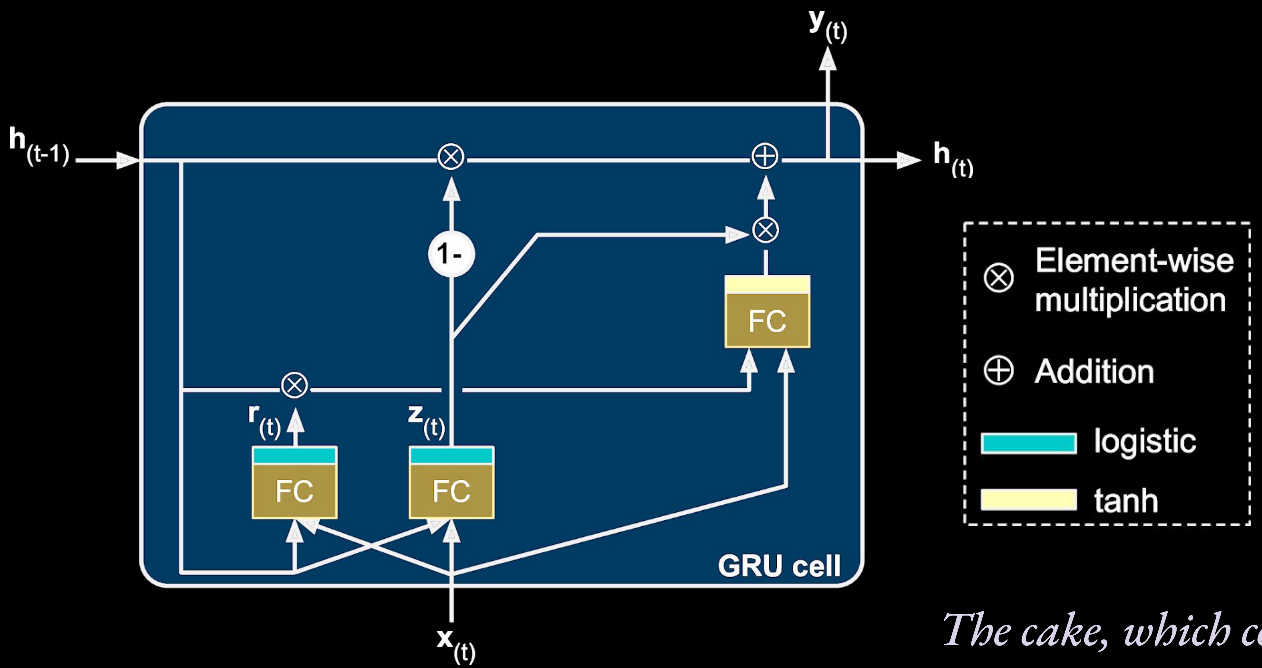


(Geron, 2017)

RNN: The GRU

Gated Recurrent Unit

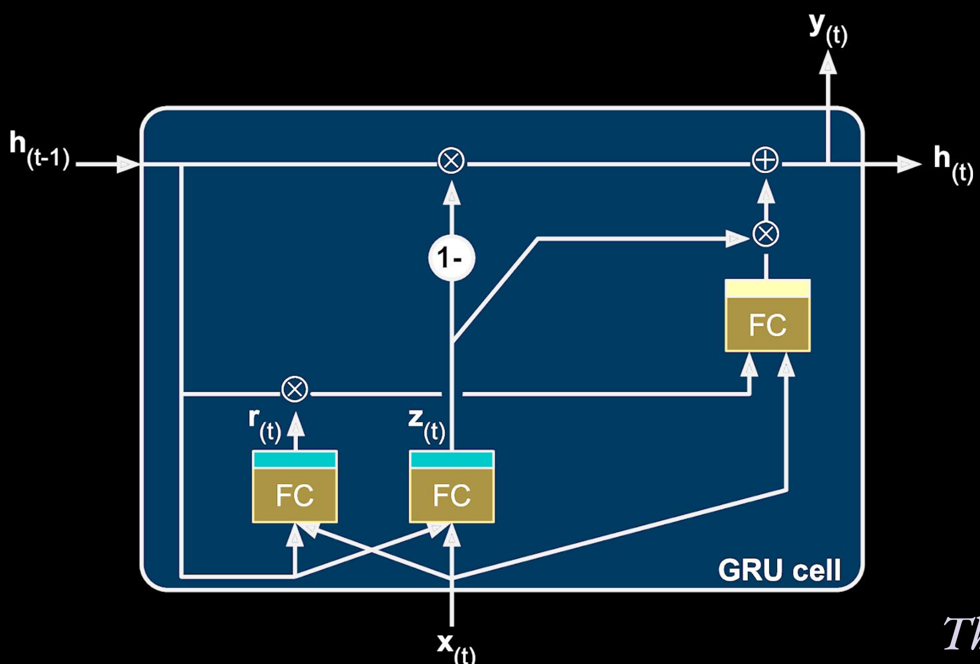
$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g\right)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The cake, which contained candles, was eaten.

What about the gradient?

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g\right)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The gates (i.e. multiplications based on a logistic) often end up keeping the hidden state exactly (or nearly exactly) as it was. Thus, for most dimensions of h,

$$h_{(t)} \approx h_{(t-1)}$$

The cake, which contained candles, was eaten.

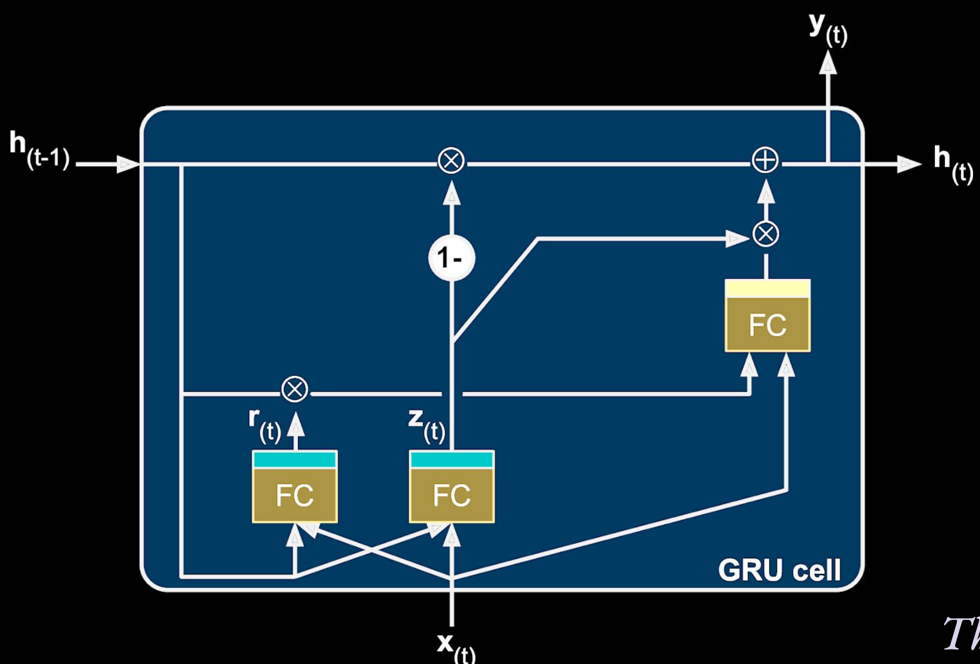
What about the gradient?

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{z})$$

$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{r})$$

$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_{g}\right)$$

$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The gates (i.e. multiplications based on a logistic) often end up keeping the hidden state exactly (or nearly exactly) as it was. Thus, for most dimensions of h,

$$h_{(t)} \approx h_{(t-1)}$$

This tends to keep the gradient from vanishing since the same values will be present through multiple times in backpropagation through time. (The same idea applies to LSTMs but is easier to see here).

The cake, which contained candles, was eaten.

The GRU (LSTM): Zoomed out

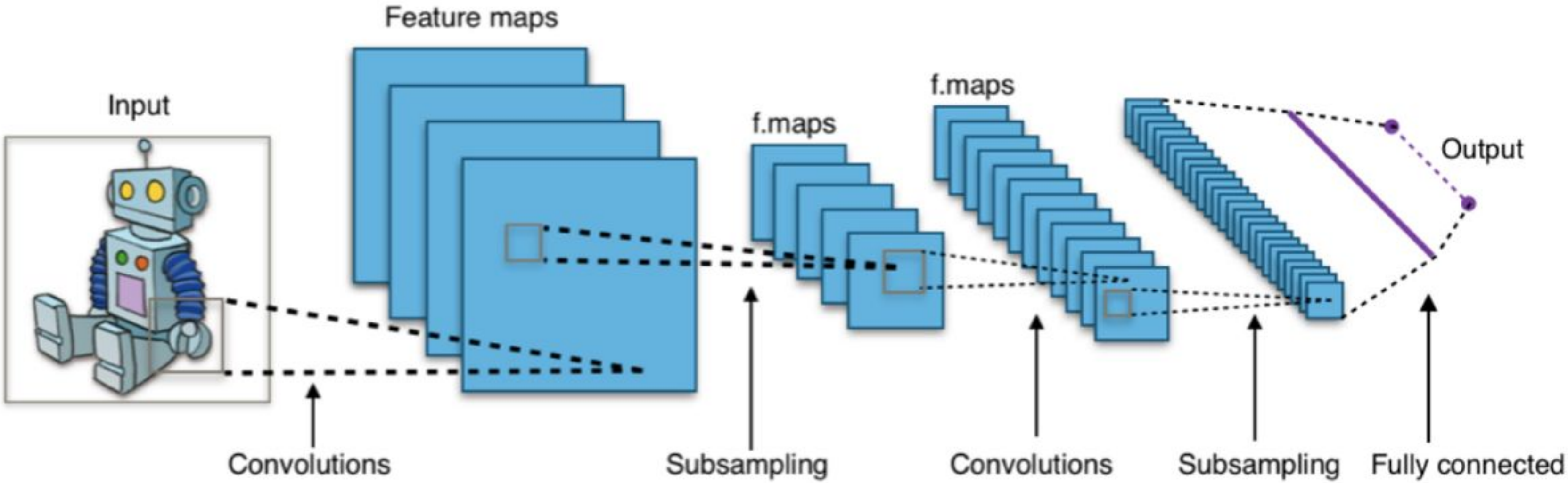
Take-Aways

- Simple RNNs are powerful models but they are difficult to train:
  - Just two functions $h_{(t)}$ and $y_{(t)}$ where $h_{(t)}$ is a combination of $h_{(t-1)}$ and $x_{(t)}$.
  - Exploding and vanishing gradients make training difficult to converge.
- LSTM (e.g. GRU cells) solve
  - Hidden states pass from one time-step to the next, allow for long-distance dependencies.
  - Gates are used to keep hidden states from changing rapidly (and thus keeps gradients under control).
  - To train: mini-batch stochastic gradient descent over cross-entropy cost

Post-Exam2 Topics:

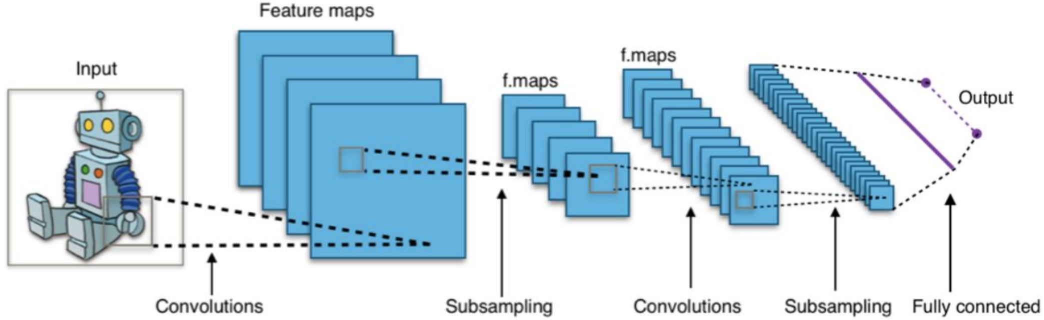
1. Research Ethics
2. Useful Plots
3. Machine Learning Cross Validation
4. Recurrent Neural Networks
5. **Convolutional Neural Networks**
6. Transformer Networks

Convolutional Neural Networks



(wikipedia)

Convolution Layer



“Convolution”

3	1	1	2	8	4
1	0	7	3	2	6
2	3	5	1	1	3
1	4	1	2	6	5
3	2	1	3	7	2
9	2	6	2	5	1

Original image 6x6

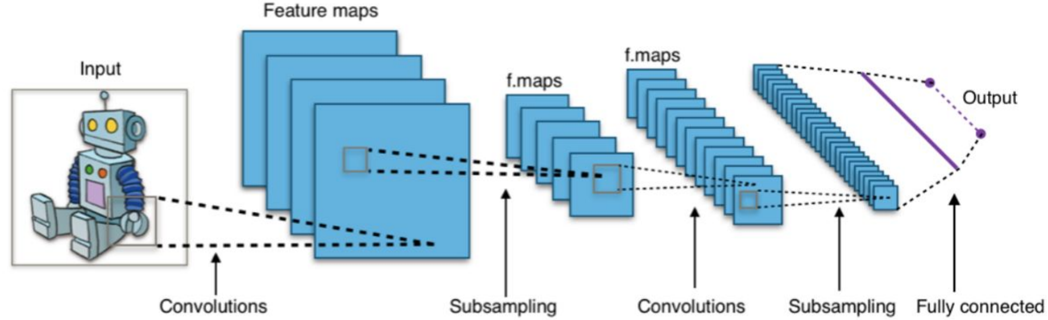
✖

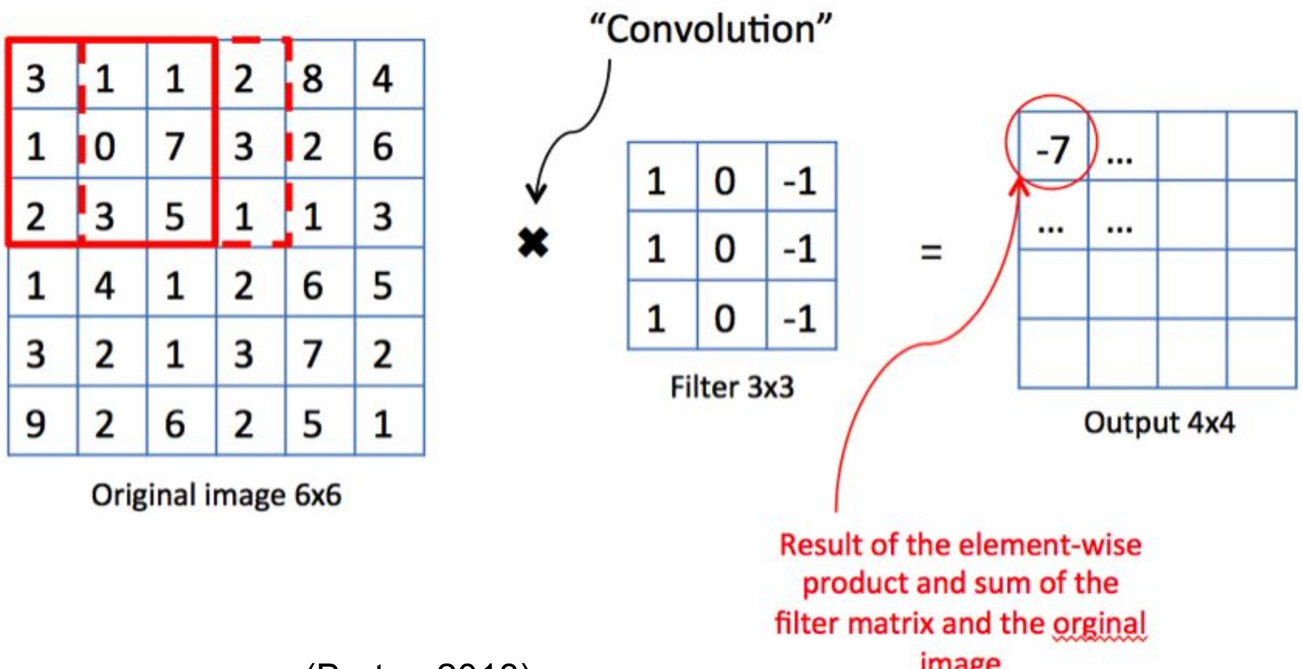
1	0	-1
1	0	-1
1	0	-1

Filter 3x3

(Barter, 2018)

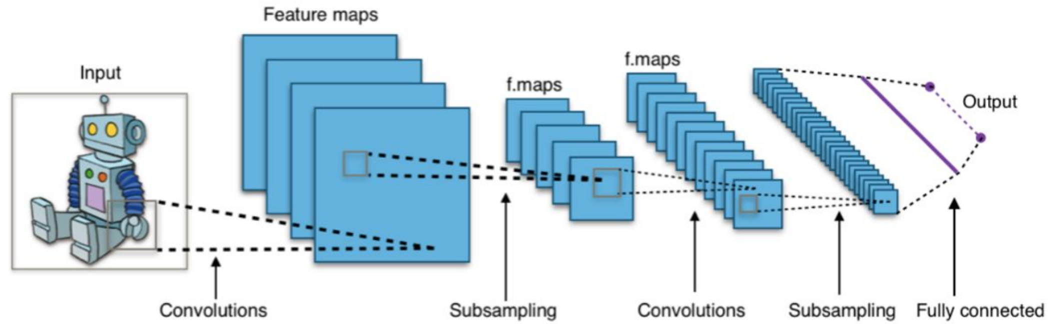
Convolution Layer

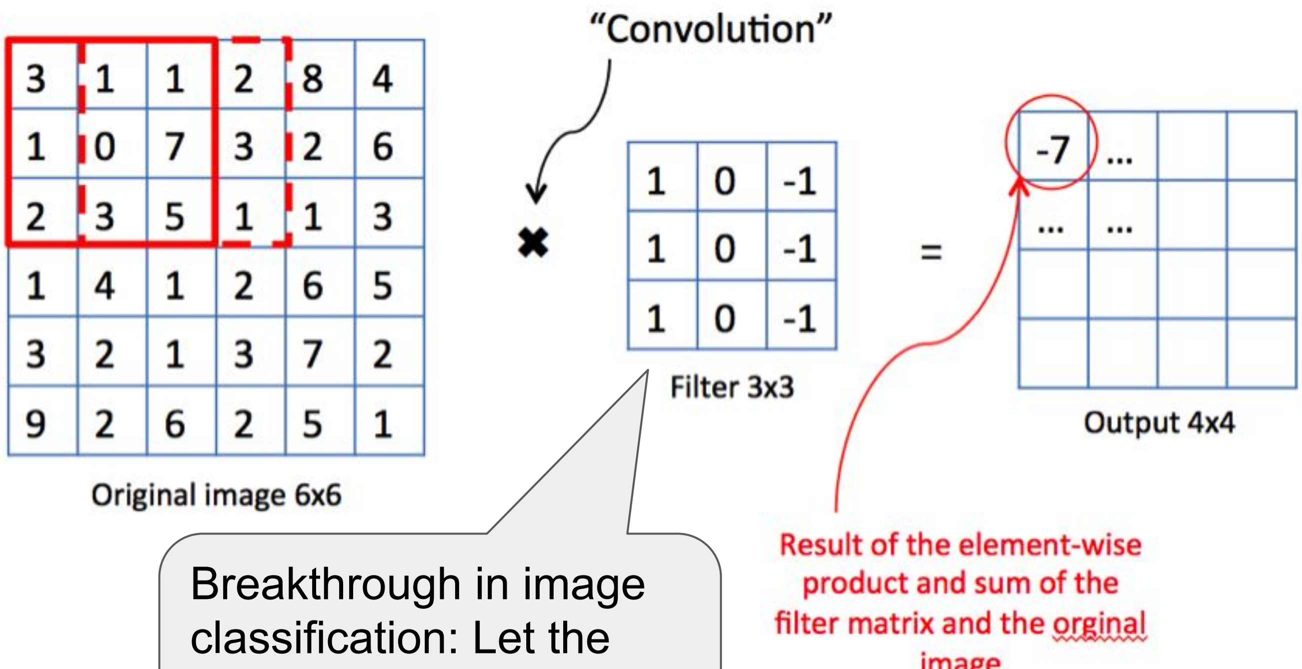




(Barter, 2018)

Convolution Layer

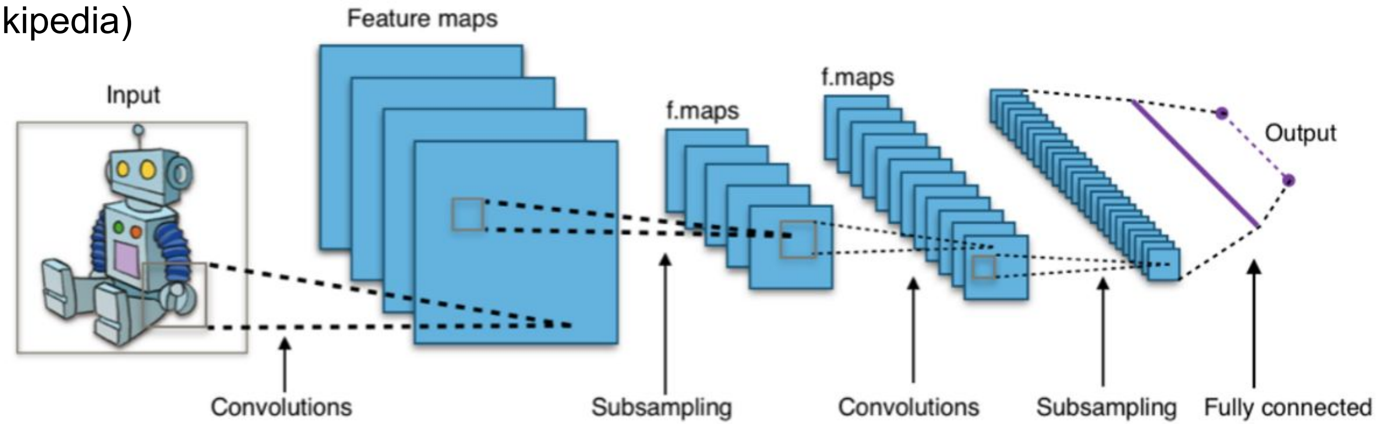




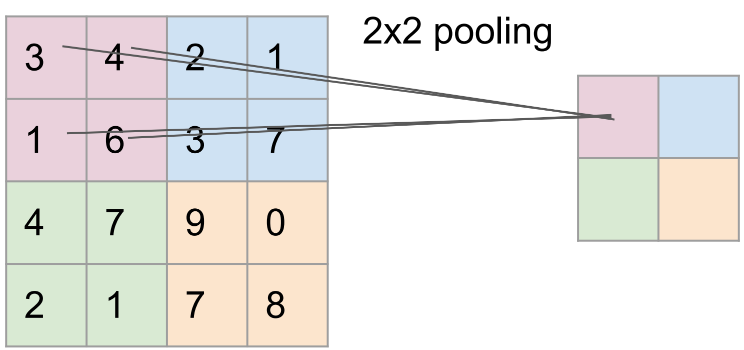
Breakthrough in image classification: Let the model automatically learn the filter weights!

Subsampling (Pooling)

(wikipedia)

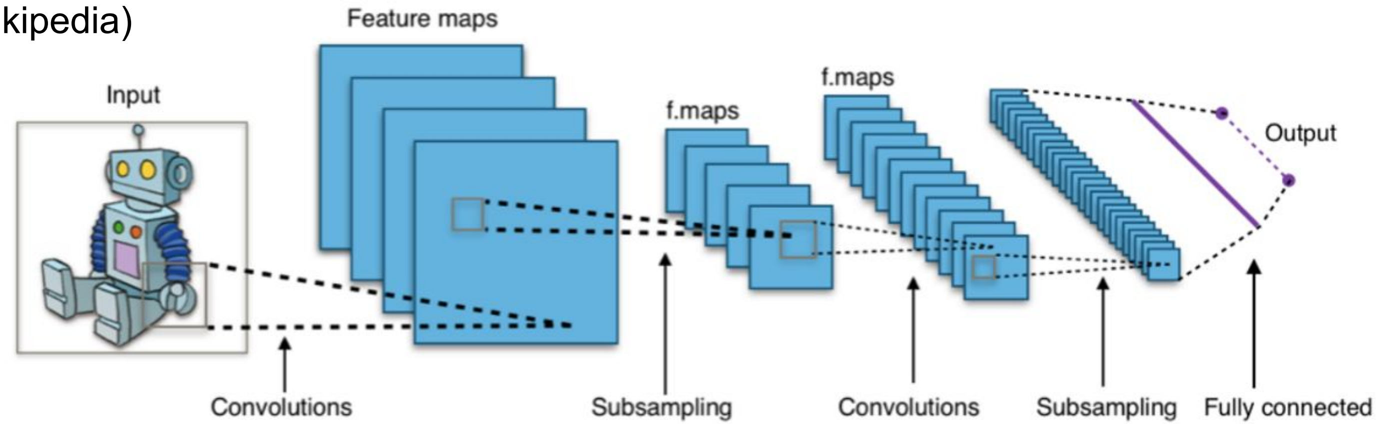


Subsampling -- reducing total grid size (i.e. reducing parameters for next layer)

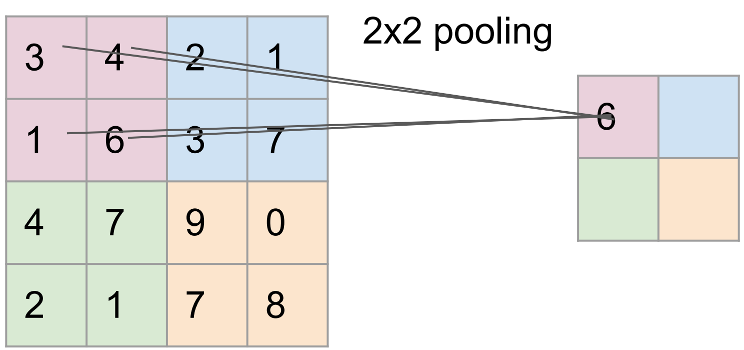


Subsampling (Pooling)

(wikipedia)



Subsampling -- reducing total grid size (i.e. reducing parameters for next layer)

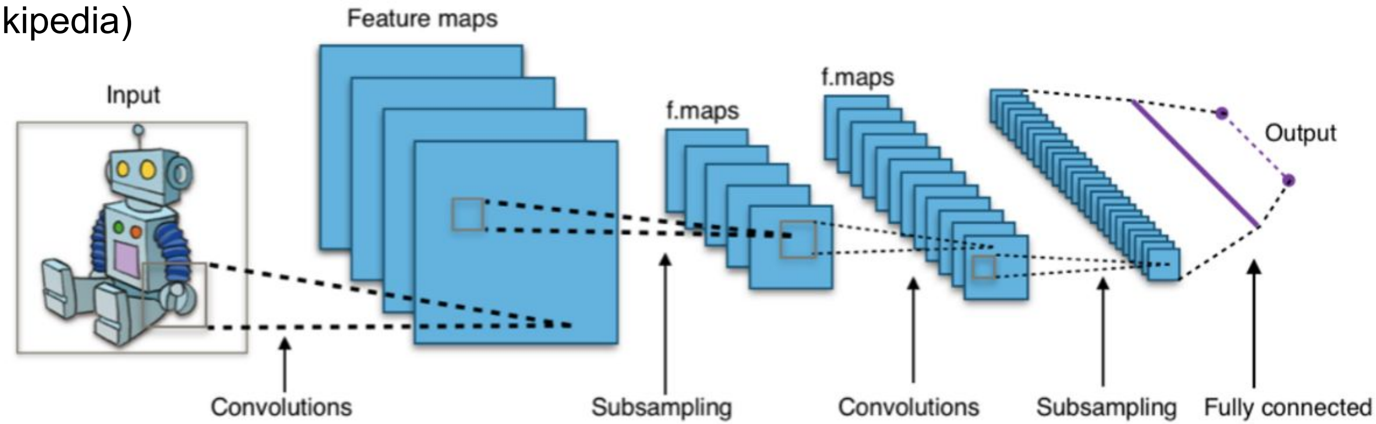


Types of pooling

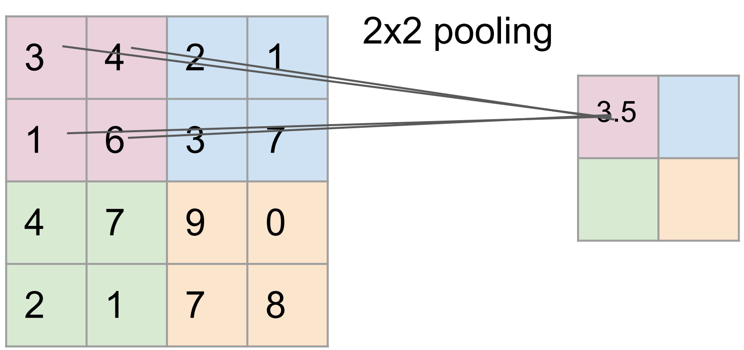
- **max**
- avg

Subsampling (Pooling)

(wikipedia)



Subsampling -- reducing total grid size (i.e. reducing parameters for next layer)



Types of pooling

- max
- **avg**

Standard Training Loss Function

```
RNN_cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
            #where did this come from?
```

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

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"

Standard Training Loss Function

```
RNN_cost = tf.reduce_mean(-tf.reduce_sum(y*tf.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: $\ell(\beta) = \sum_{i=1}^{N} y_i log\, p(x_i) + (1 - y_i) log\, (1 - p(x_i))$

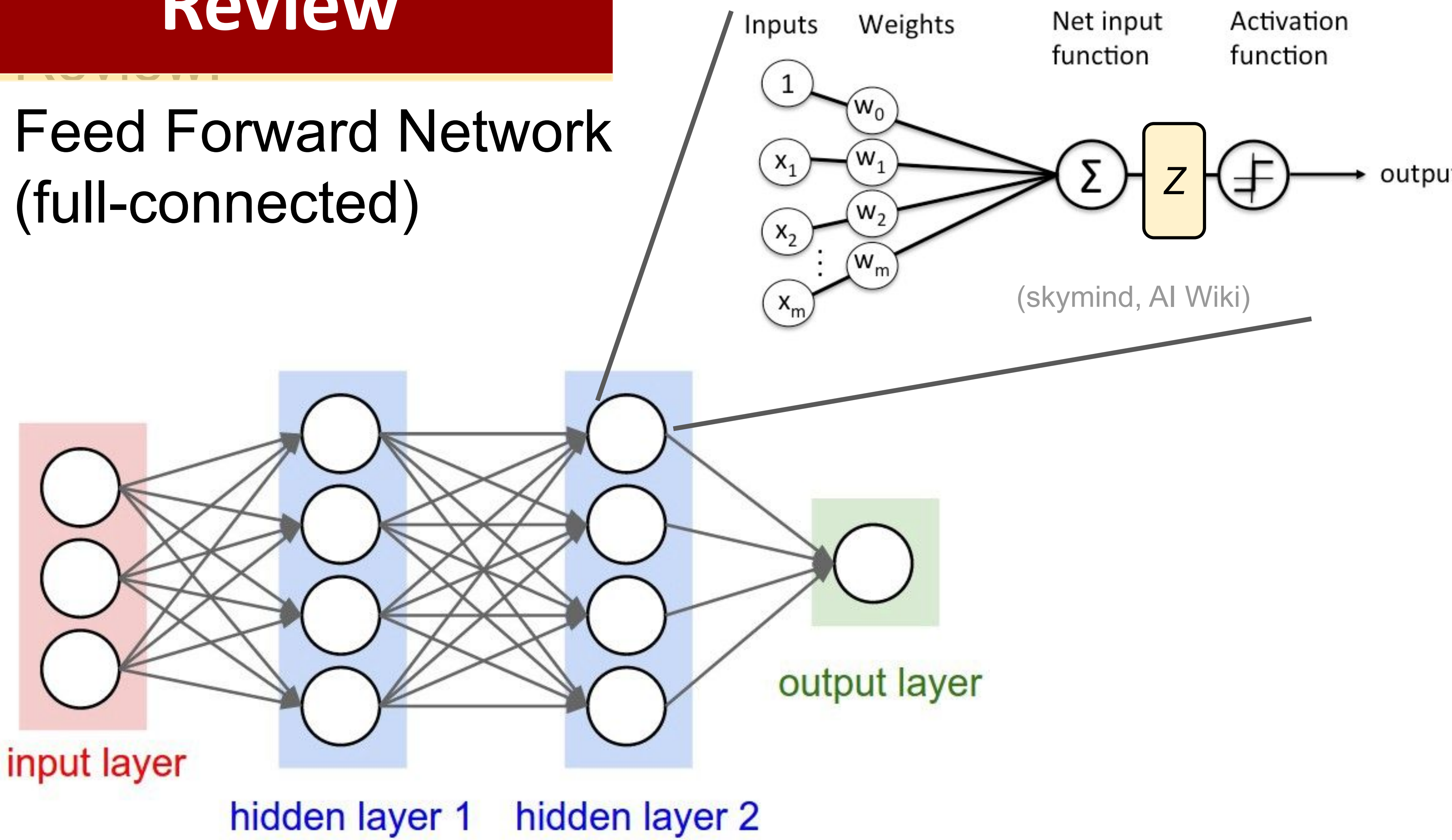
Log Loss: $J(\beta) = -\frac{1}{N} \sum_{i=1}^{N} y_i log\, p(x_i) + (1 - y_i) log\, (1 - p)(x_i))$

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)

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”

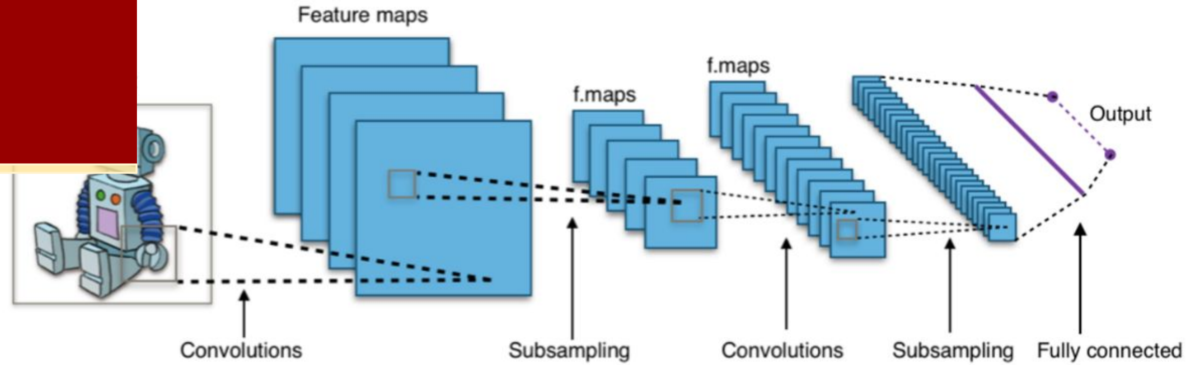
Review

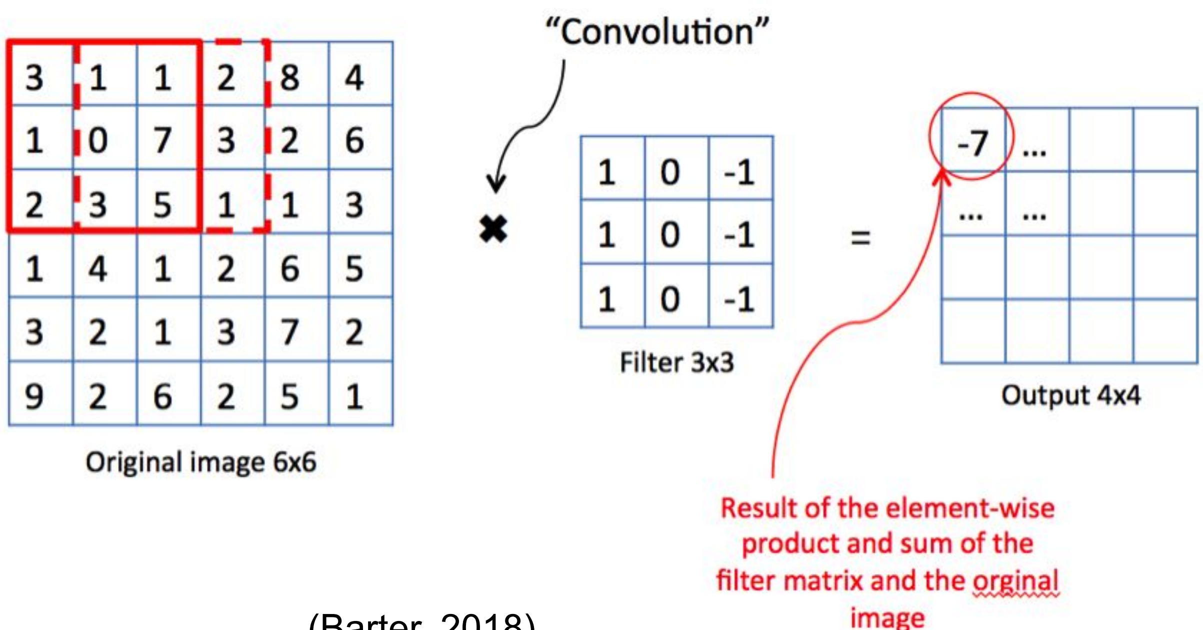
Feed Forward Network
(full-connected)



Review

Convolutional NN





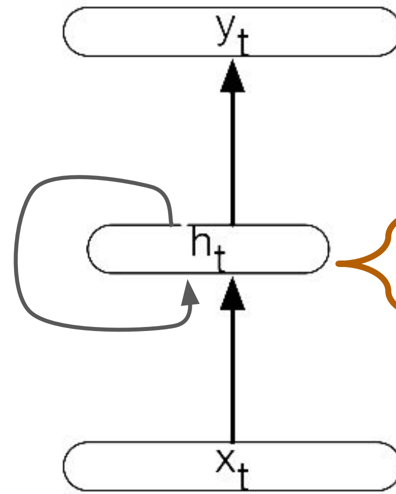
(Barter, 2018)

Post-Exam2 Topics:

1. Research Ethics
2. Useful Plots
3. Machine Learning Cross Validation
4. Recurrent Neural Networks
5. Convolutional Neural Networks
6. **Transformer Networks**

Review

Recurrent Neural Network



"hidden layer"

$$y_{(t)} = f(h_{(t)}W)$$

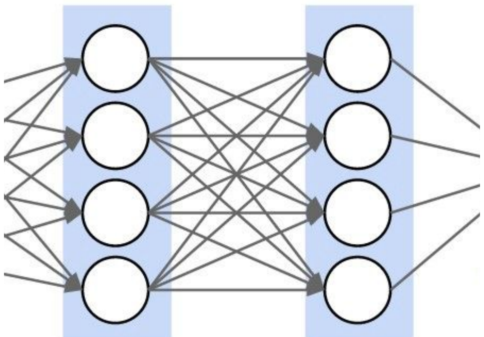
Activation Function

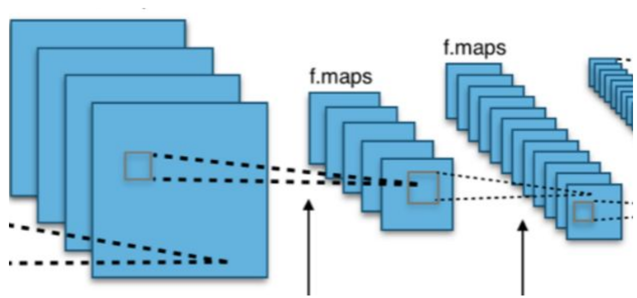
$$h_{(t)} = g(h_{(t\text{-}1)}U + x_{(t)}V)$$

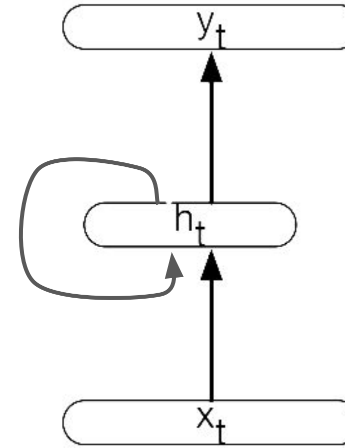
Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

FFN **CNN** **RNN**

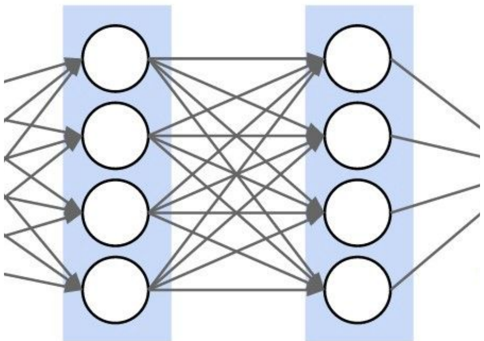


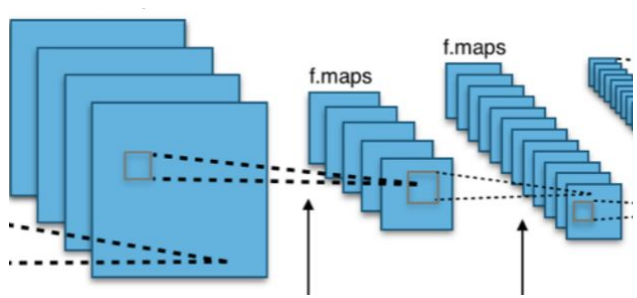


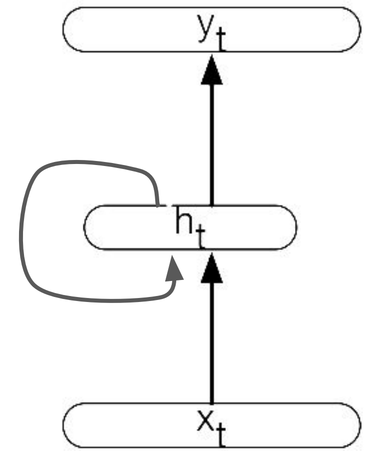


Can model computation (e.g. matrix operations for a single input) be parallelized?

FFN **CNN** **RNN**

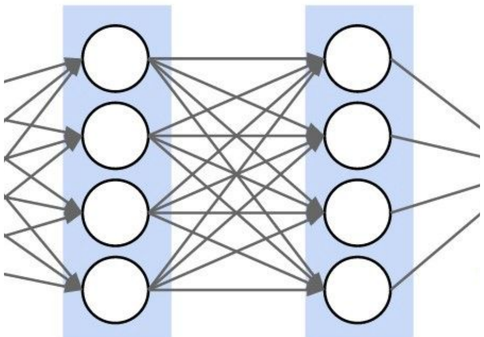


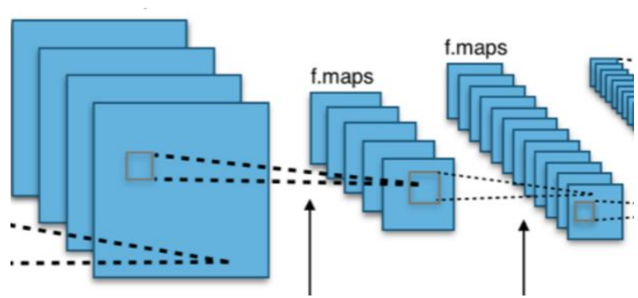


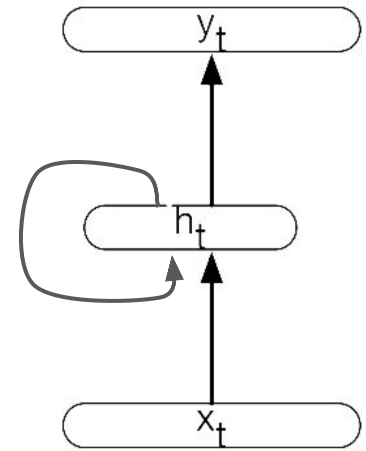


Can model computation (e.g. matrix operations for a single input) be parallelized?

FFN **CNN** **RNN**

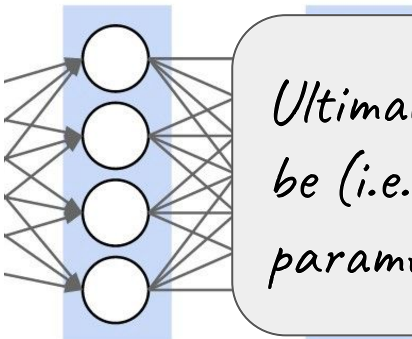


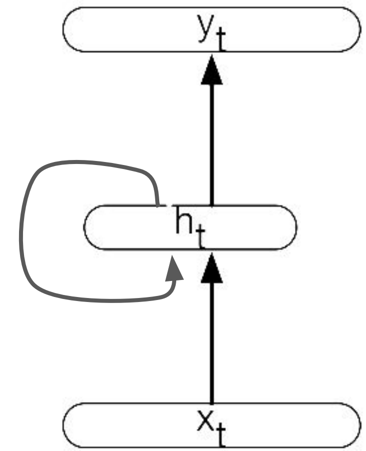




Can model computation (e.g. matrix operations for a single input) be parallelized?

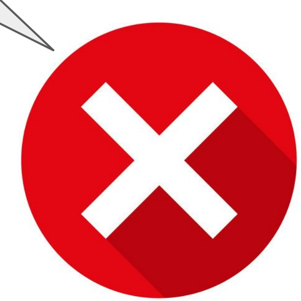
FFN **CNN** **RNN**





Ultimately limits how complex the model can be (i.e. it's total number of paramers/weights) as compared to a CNN.

Can model computation (e.g. matrix operations for a singl ut) be parallelized?



The Transformer: Attention-only Models

Can handle sequences and long-distance dependencies, but….

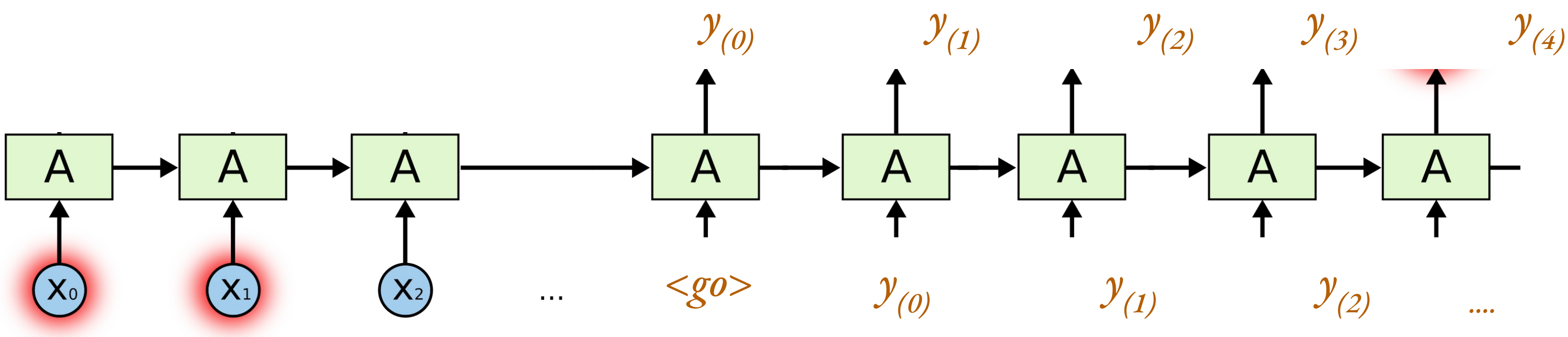
- Don’t want complexity of LSTM/GRU cells
- Constant num edges between input steps
- Enables “interactions” (i.e. adaptations) between words
- **Easy to parallelize -- don’t need sequential processing.**

The Transformer: Attention-only Models

Challenge:

The ball was kicked by kayla.

- Long distance dependency when translating:



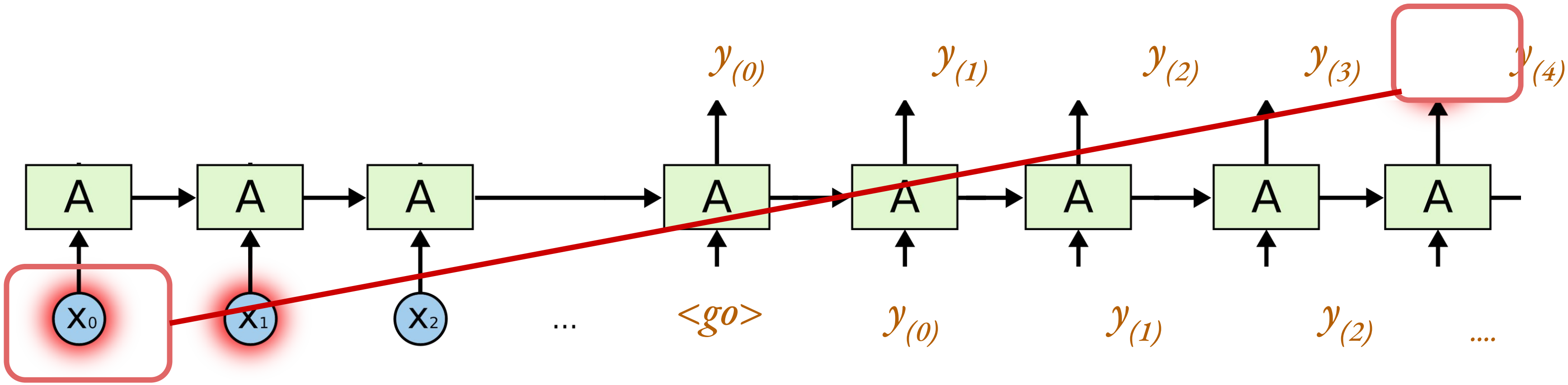
Kayla kicked the ball.

The Transformer: Attention-only Models

Challenge:

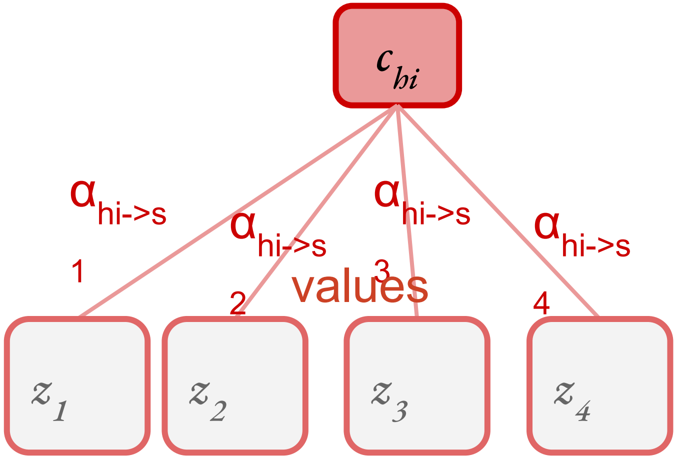
The ball was kicked by kayla.

- Long distance dependency when translating:



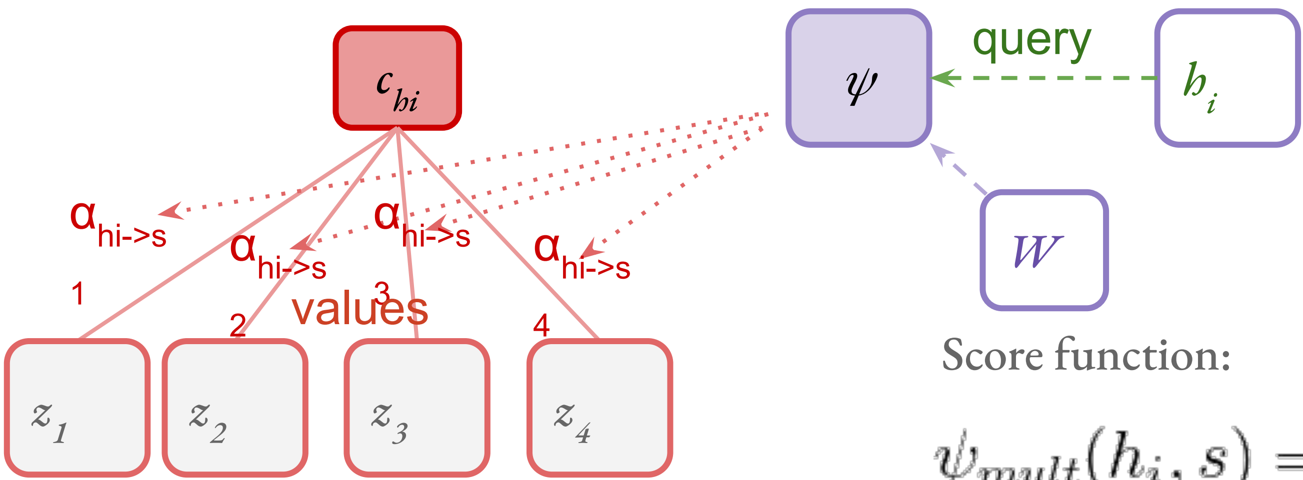
Kayla kicked the ball.

Attention



c_{hi}
$\alpha_{hi\text{->}s_1}$
$\alpha_{hi\text{->}s_2}$
$\alpha_{hi\text{->}s_3}$
$\alpha_{hi\text{->}s_4}$
values
z_1
z_2
z_3
z_4

Attention

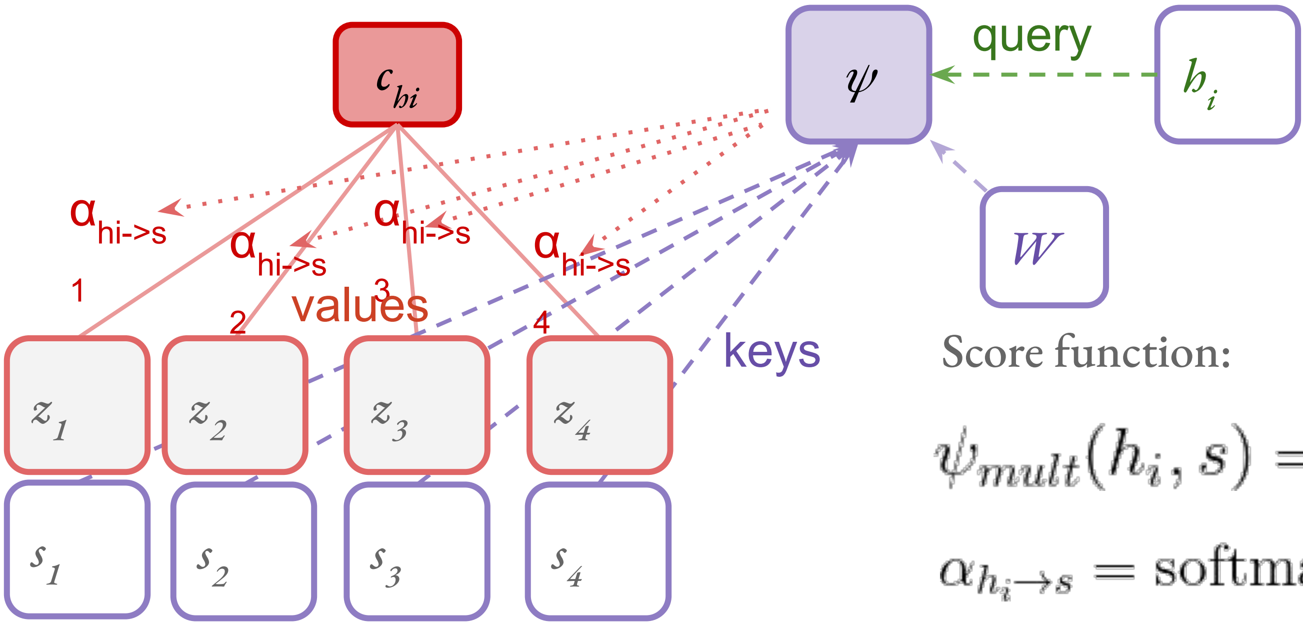


Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

Attention



Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

$$c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \to s_n} z_n$$

The Transformer: Attention-only Models

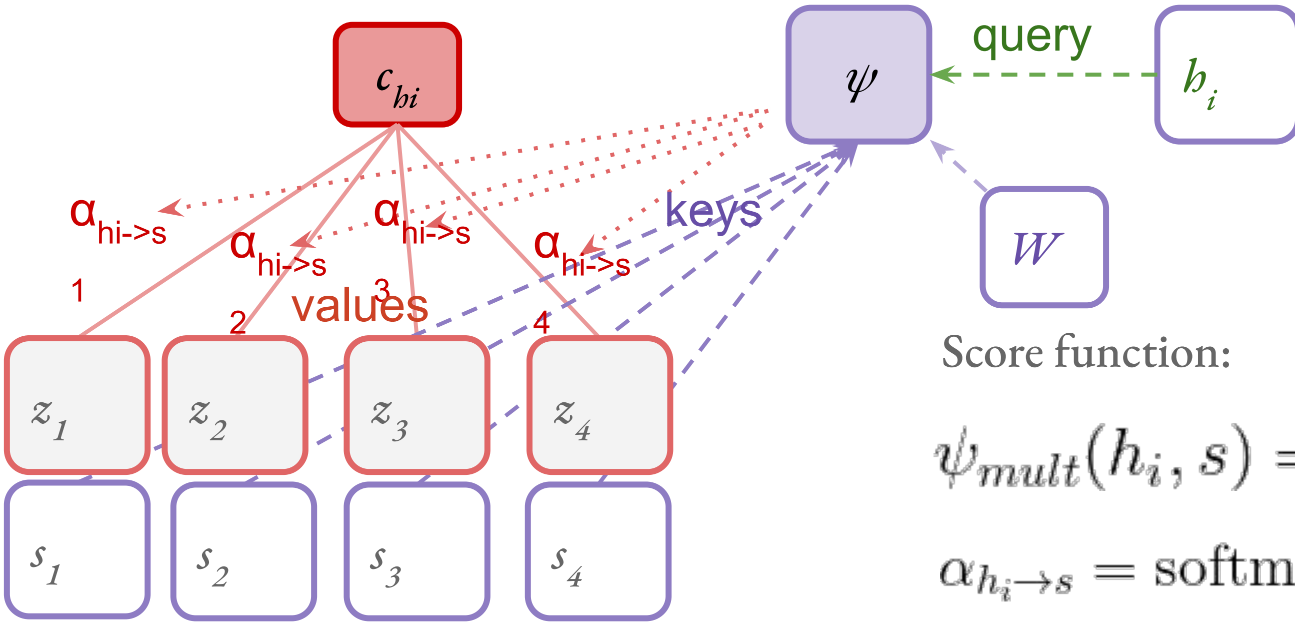
Challenge:

- Long distance dependency when translating:

Attention came about for encoder decoder models.

Then self-attention was introduced:

Attention



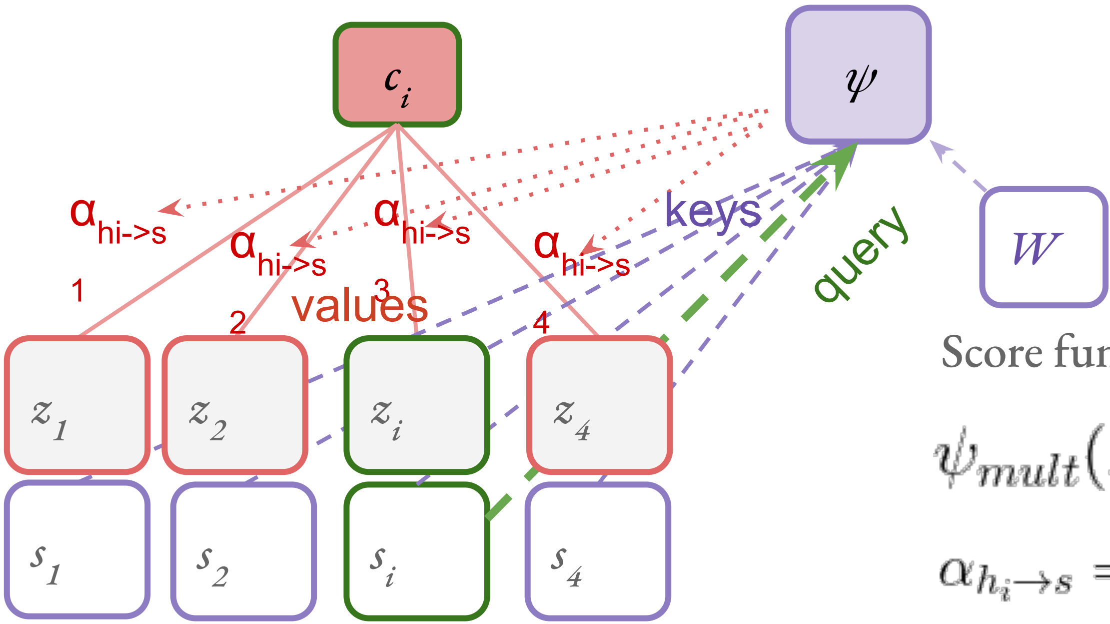
Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

$$c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \to s_n} z_n$$

Attention



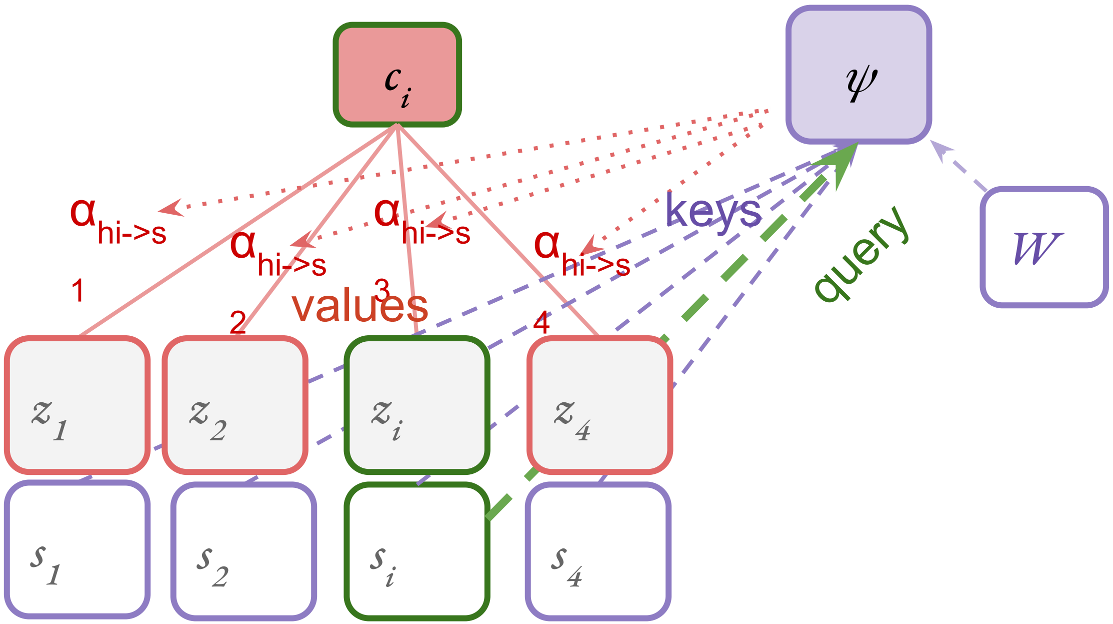
Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

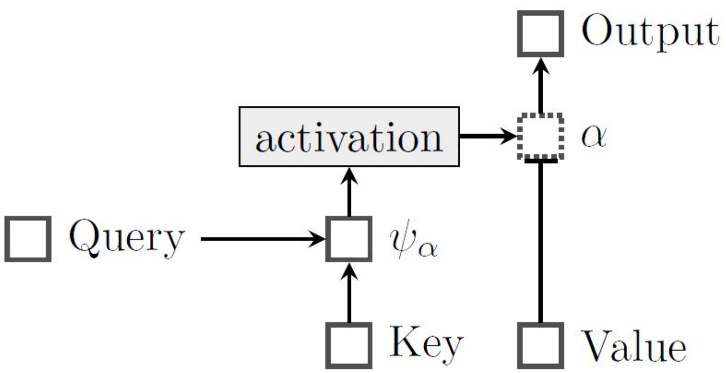
$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

$$c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \to s_n} z_n$$

Attention



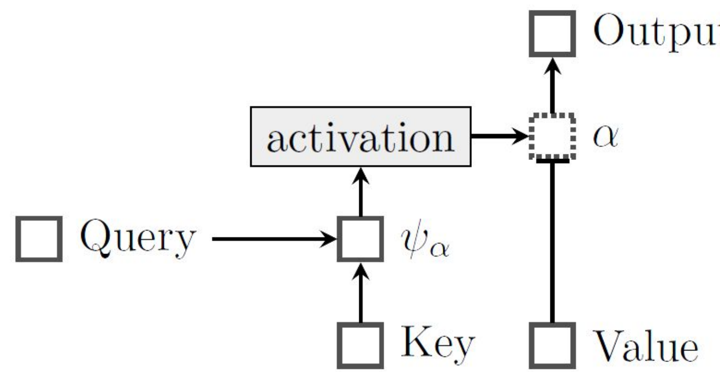
Attention as weighting a value based on a query and key:



(Eisenstein, 2018)

The Transformer: Attention-only Models

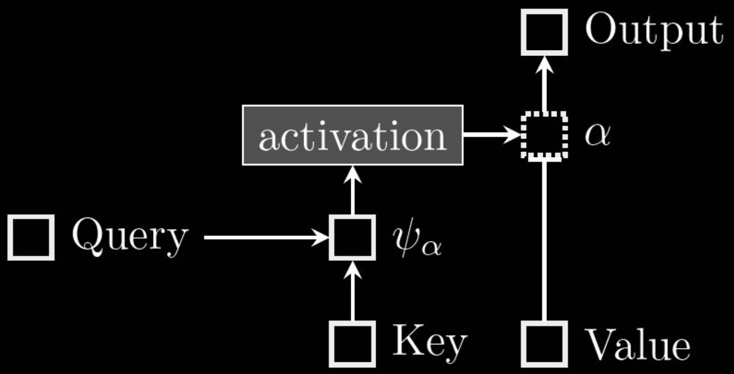
Attention as weighting a value based on a query and key:



(Eisenstein, 2018)

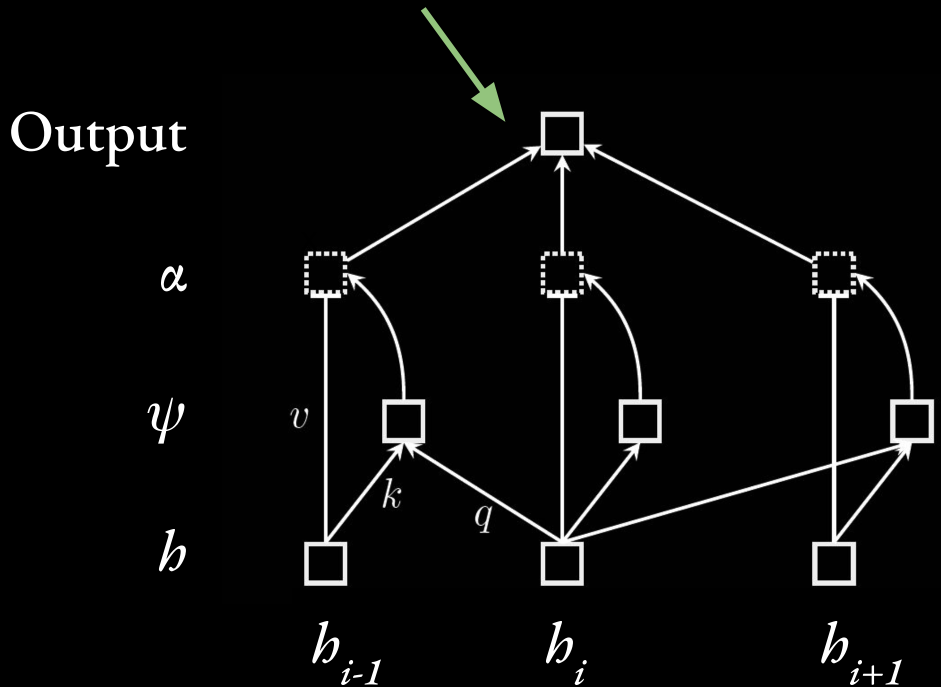
The Transformer: Attention-only Models

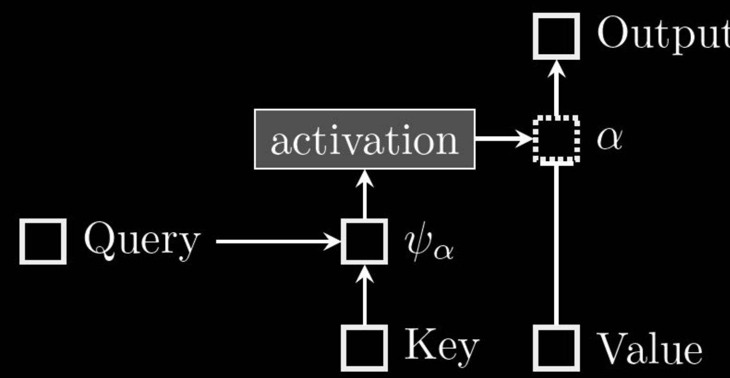
CSE 545 Supplemental Lecture
Will begin at 2:00pm



(Eisenstein, 2018)

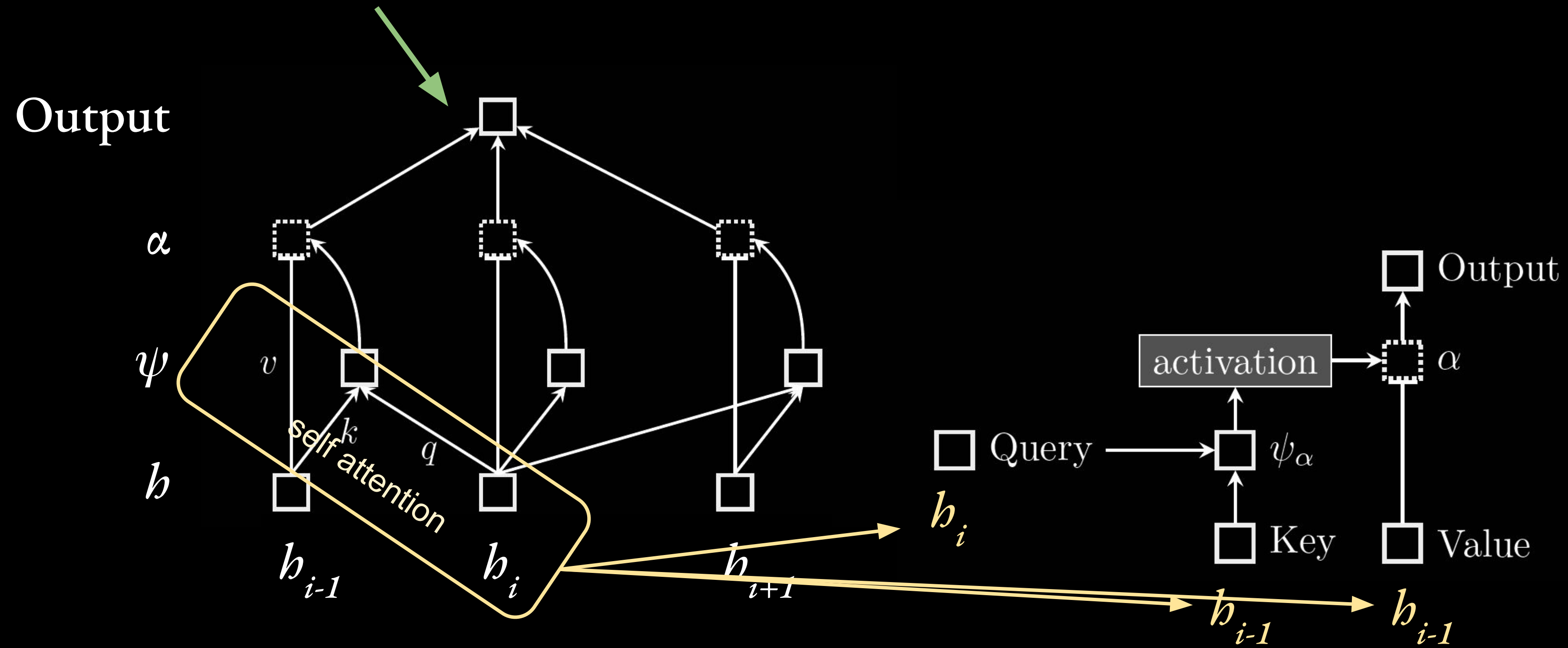
The Transformer: Attention-only Models





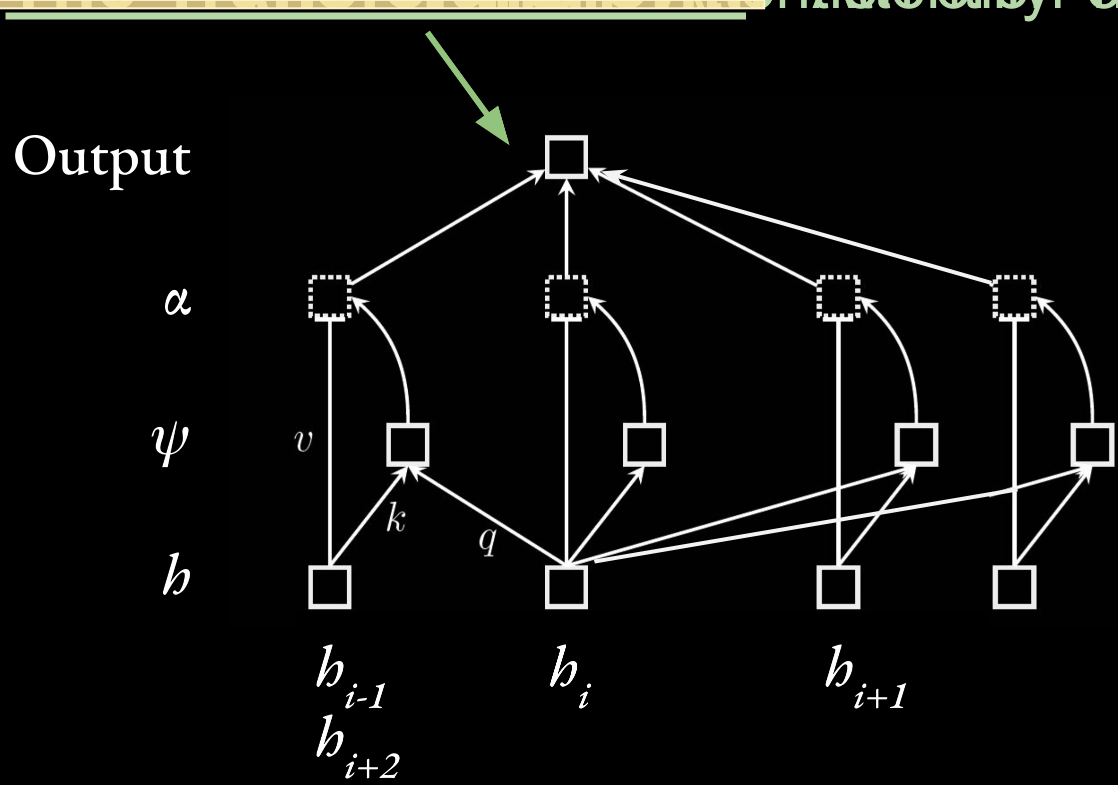
(Eisenstein, 2018)

The Transformer: Attention-only Models

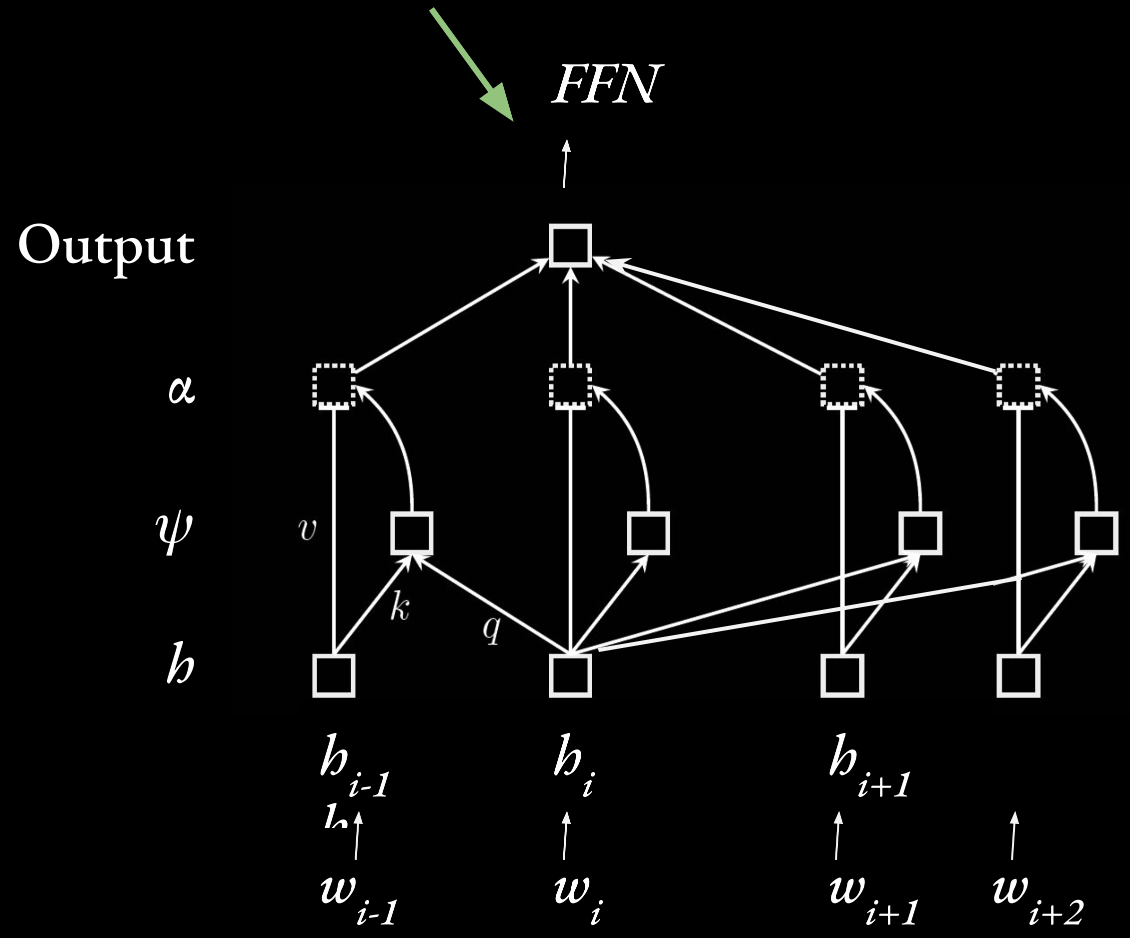


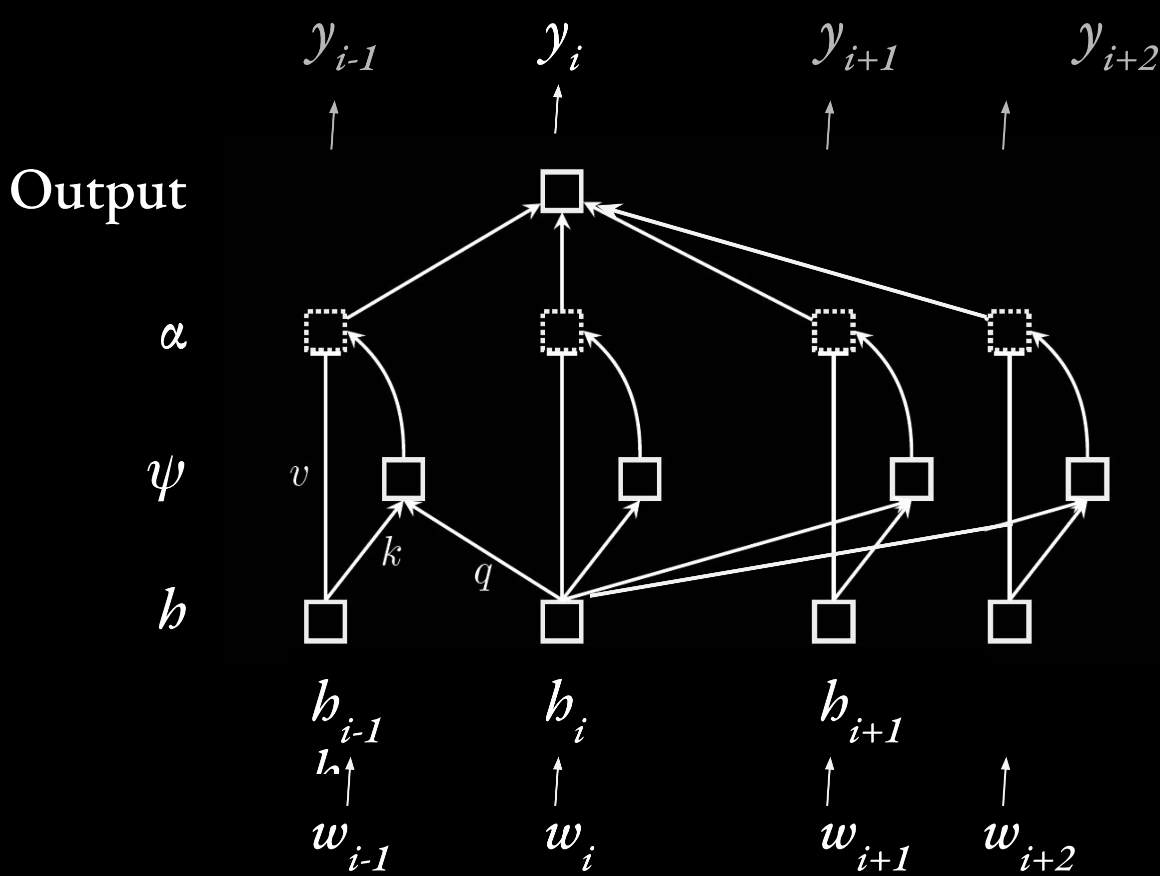
(Eisenstein, 2018)

The Transformer: Attention-only Models



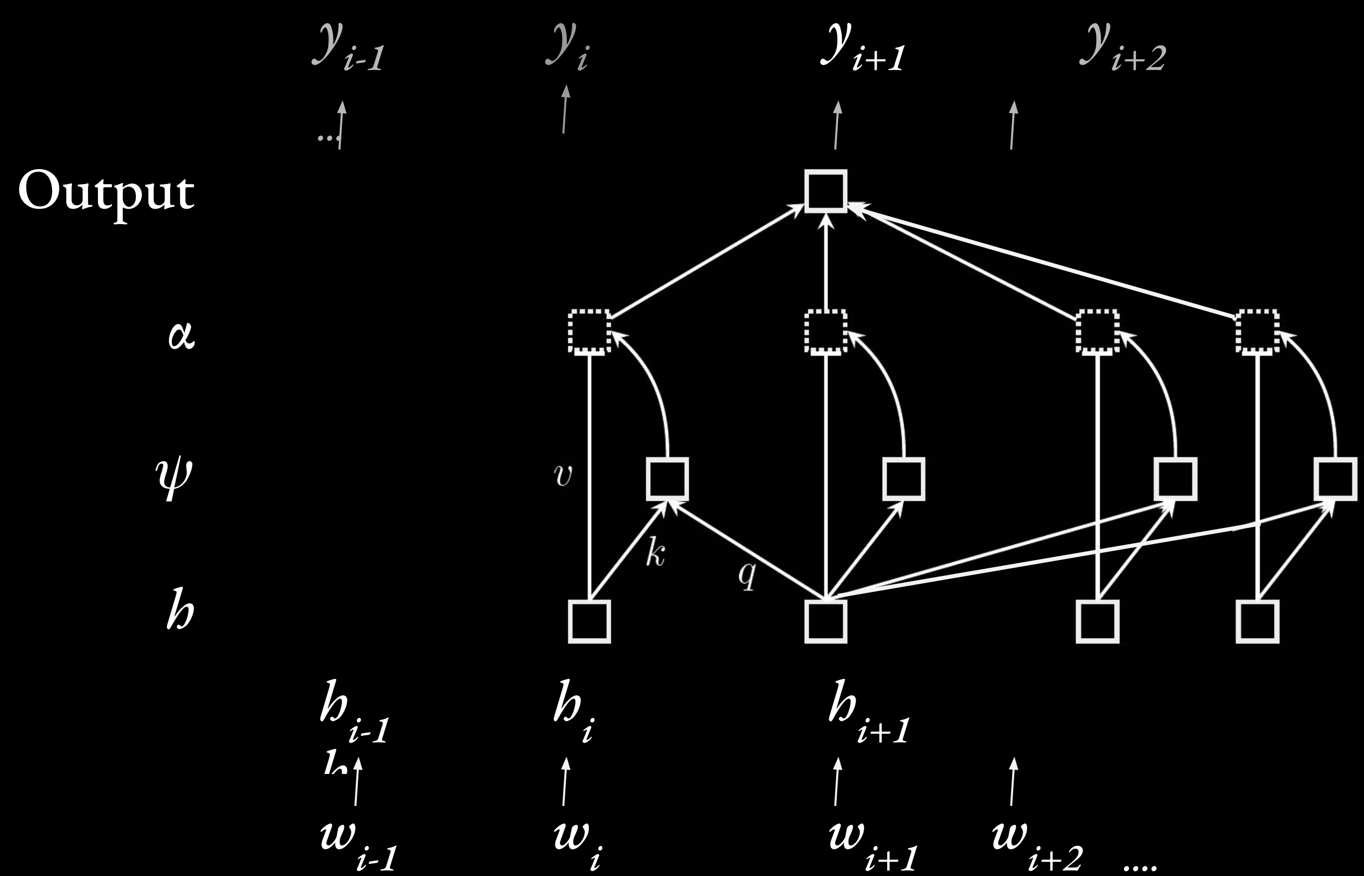
The Transformer: Attention-only Models



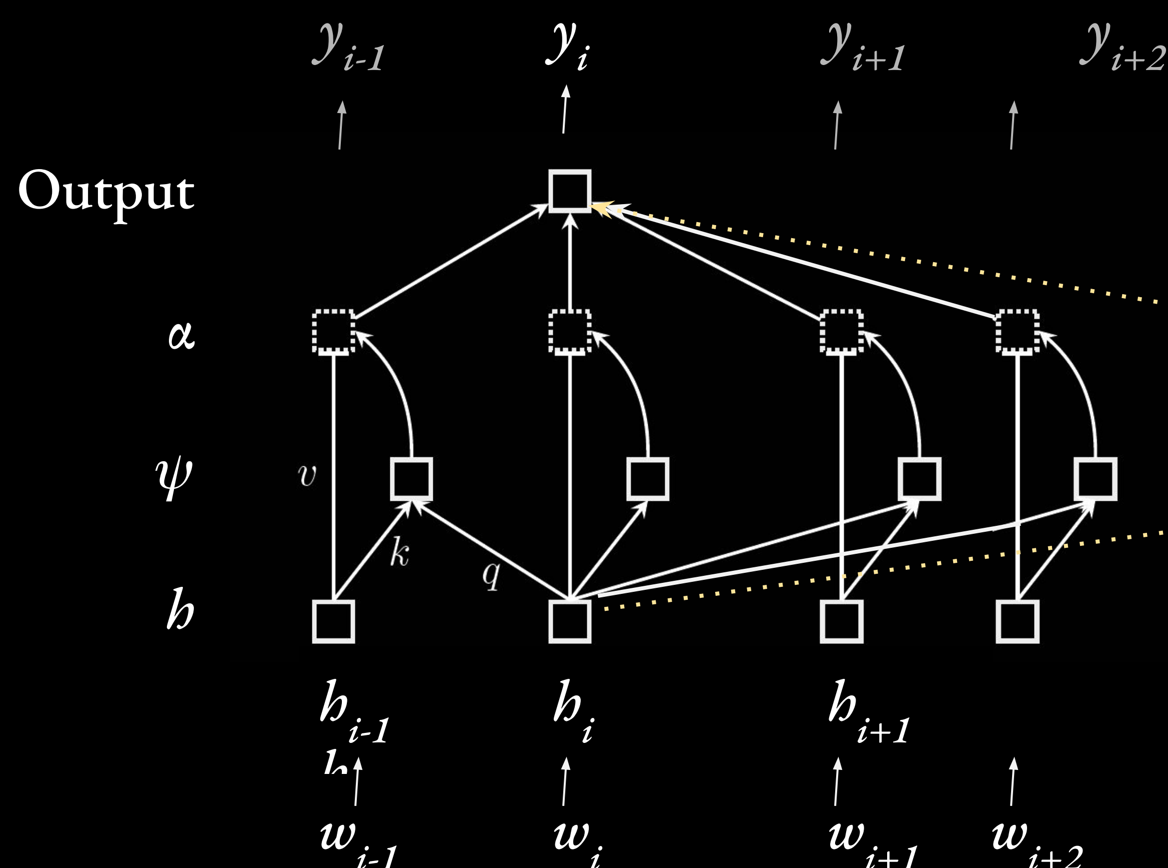


y_{i-1}
y_i
y_{i+1}
y_{i+2}
Output
α
ψ
v
k
q
h
h_{i-1}
h_i
h_{i+1}
w_{i-1}
w_i
w_{i+1}
w_{i+2}

The Transformer: “Attention-only” models

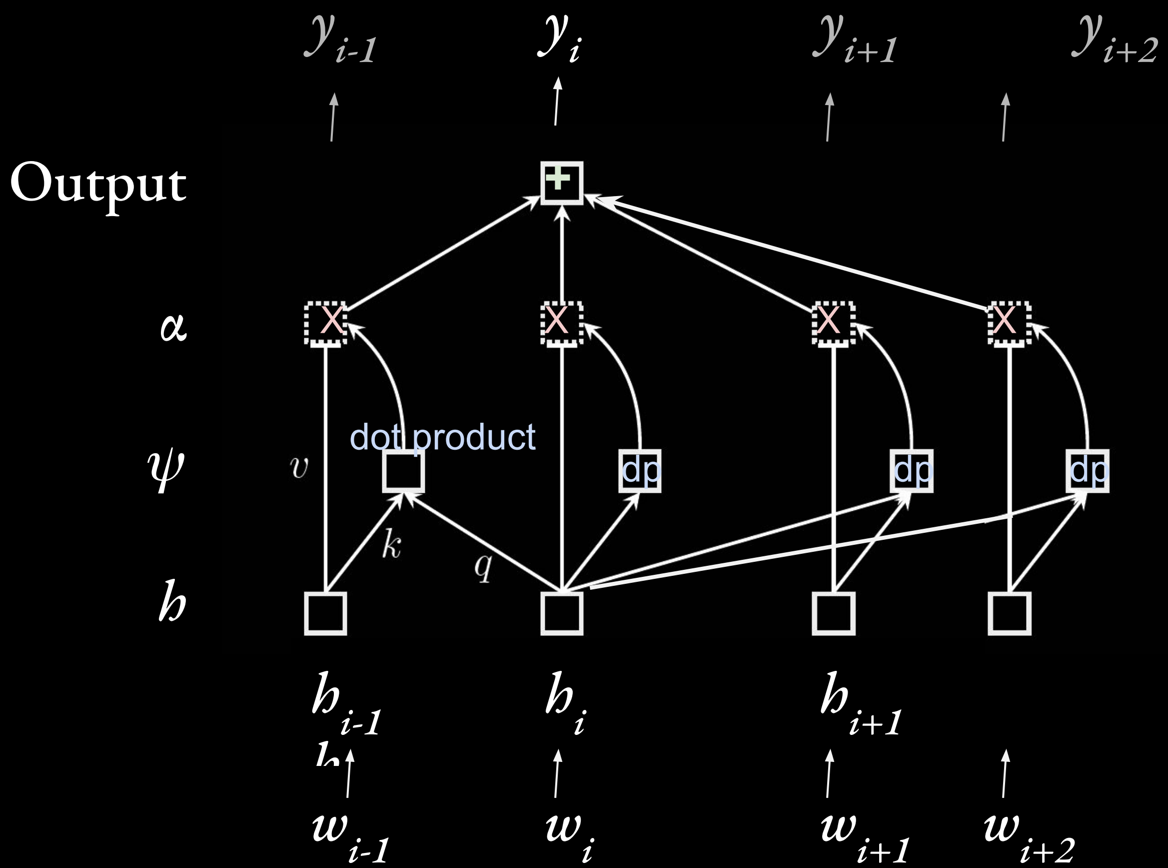


The Transformer: "Attention-only" models



Attend to all hidden states in your "neighborhood".

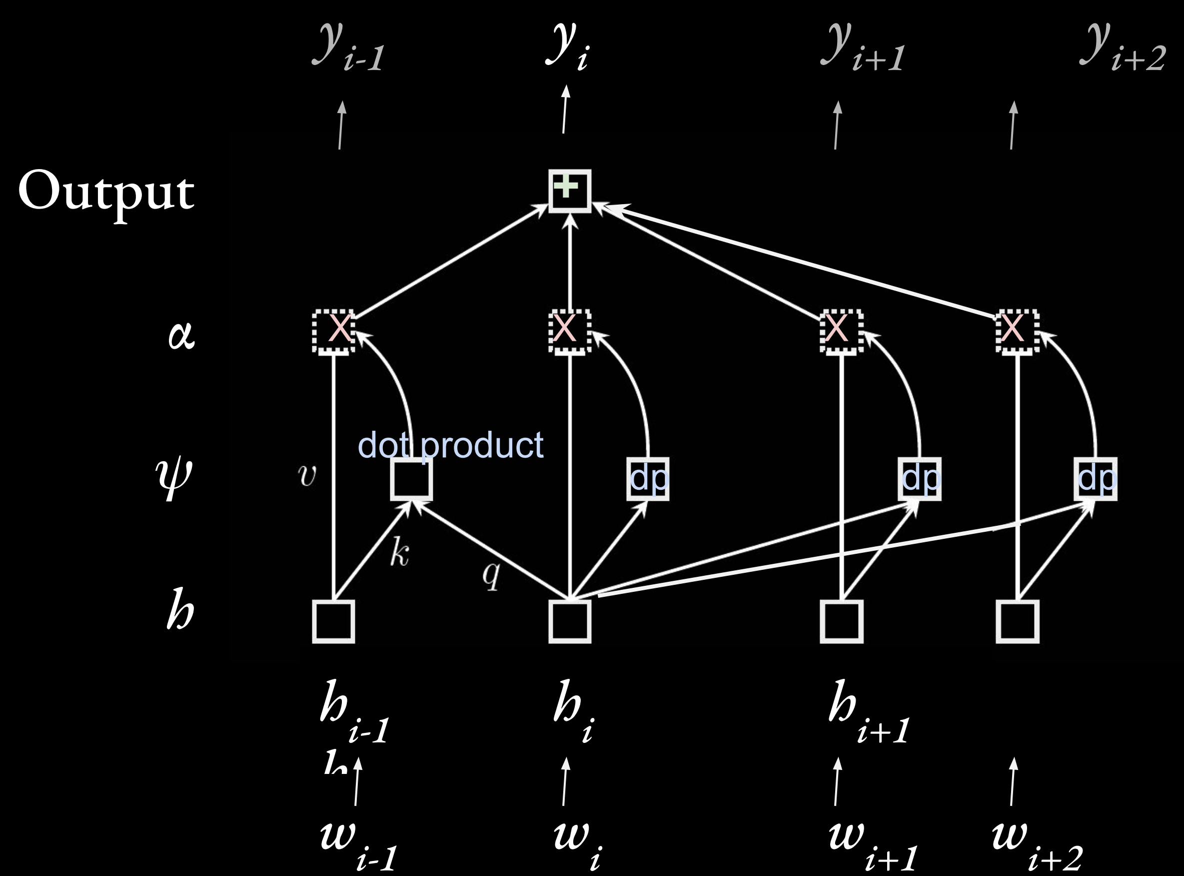
The Transformer: "Attention-only" models



$$\psi_{dp}(h_i, s) = s^T h_i$$

$k^t q$

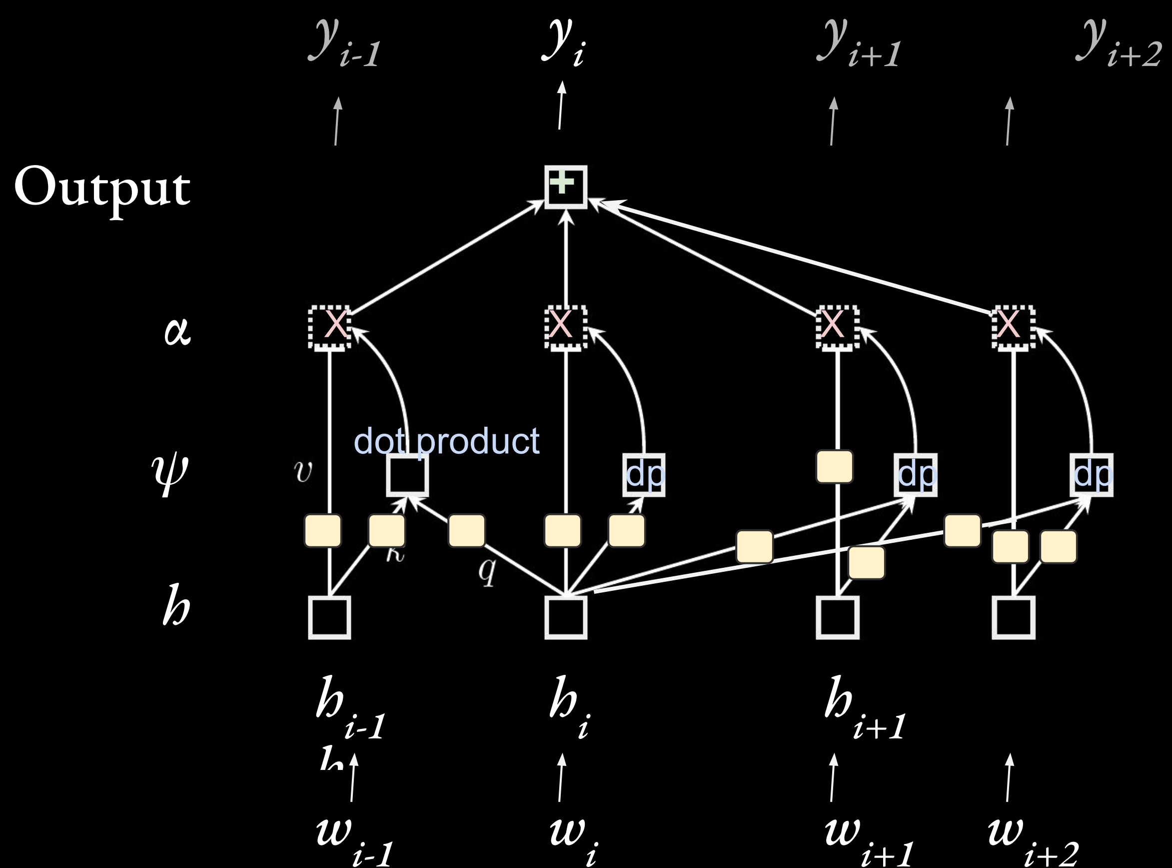
The Transformer: “Attention-only” models



scaling parameter

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

The Transformer: “Attention-only” models



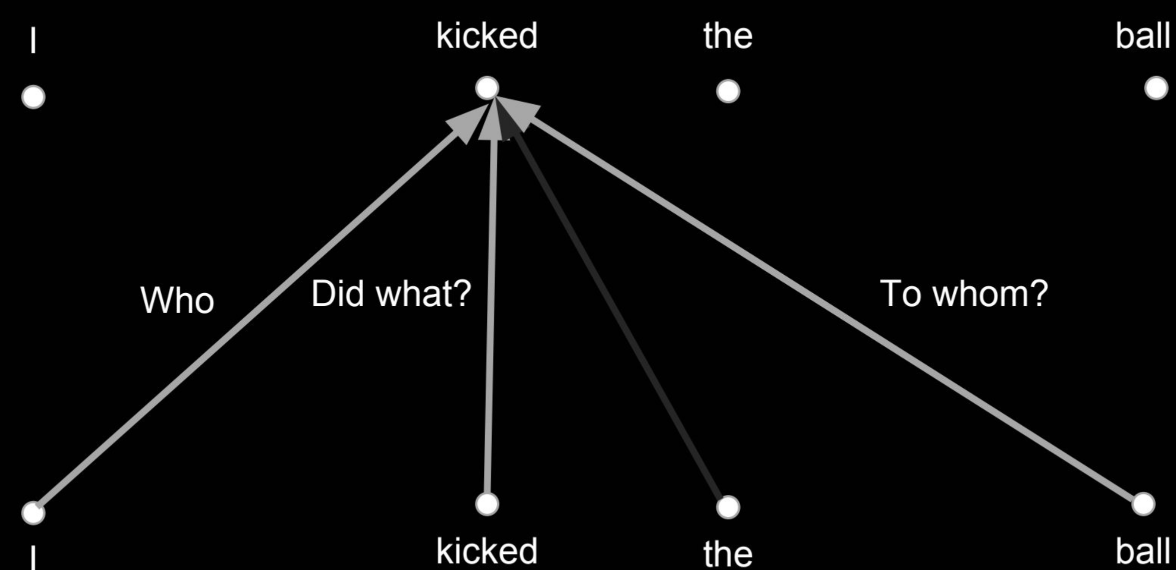
$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

Linear layer:
$W^T X$

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

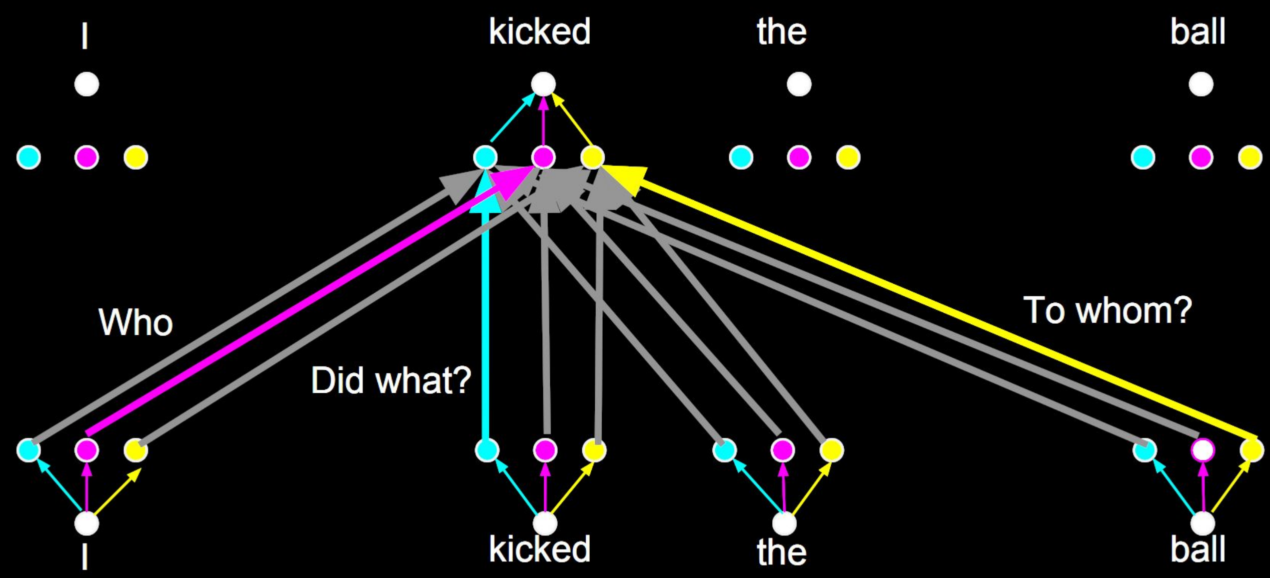
The Transformer

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

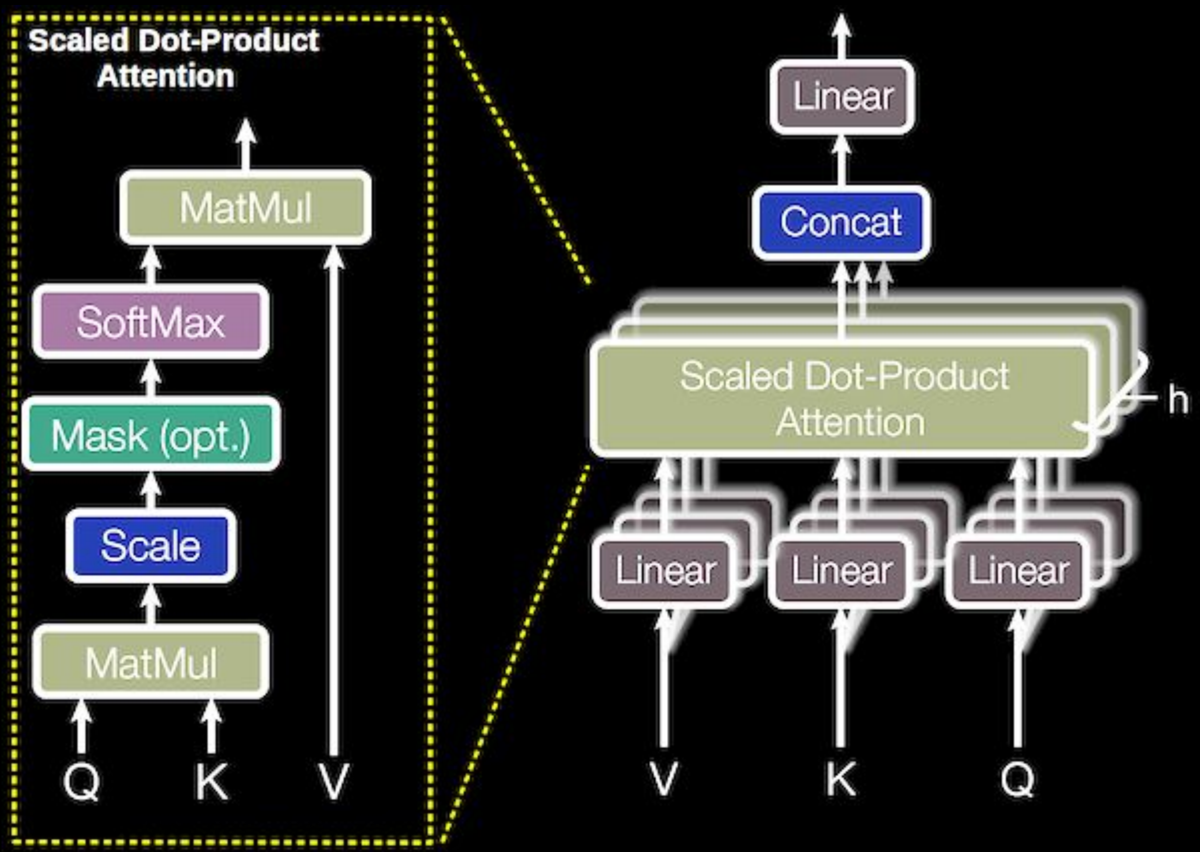


The Transformer

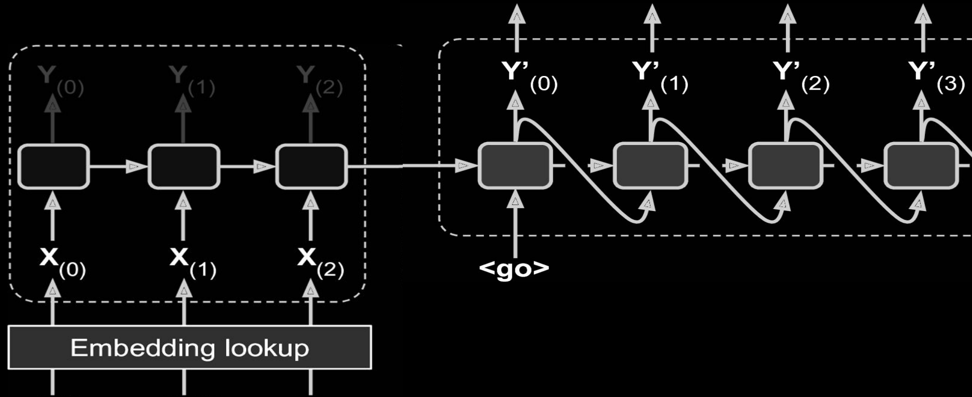
Solution: Multi-head attention

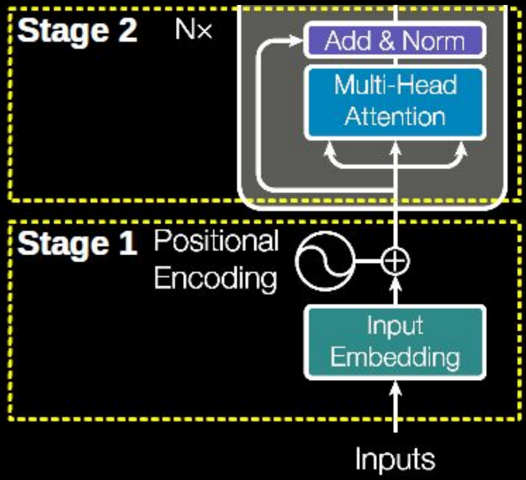


Multi-head Attention

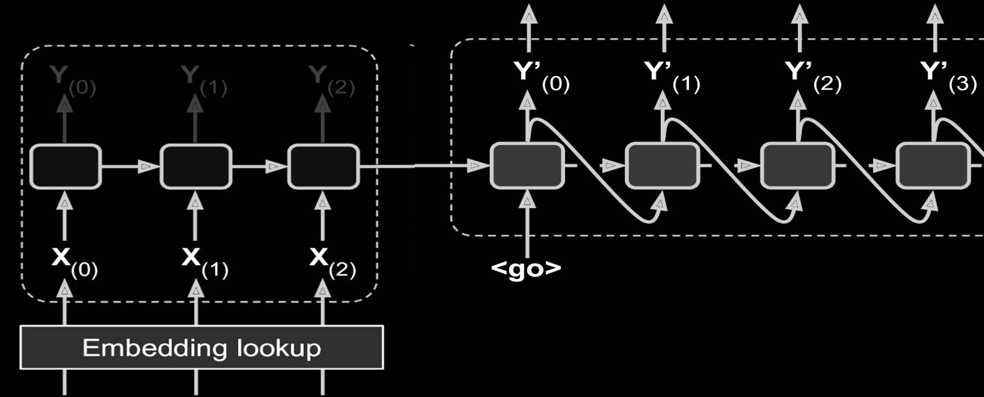


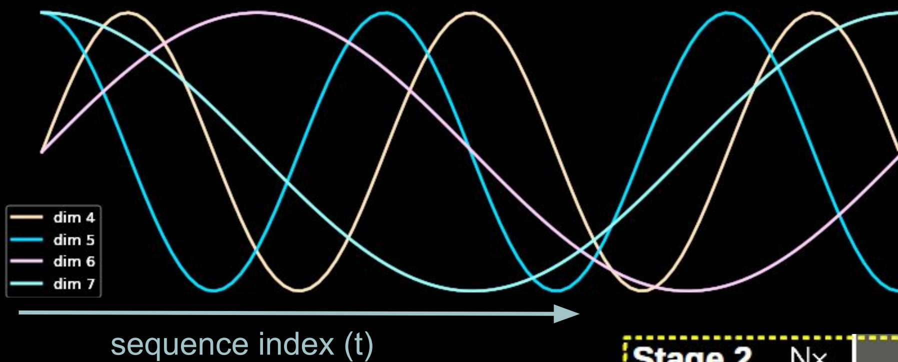
Transformer for Encoder-Decoder

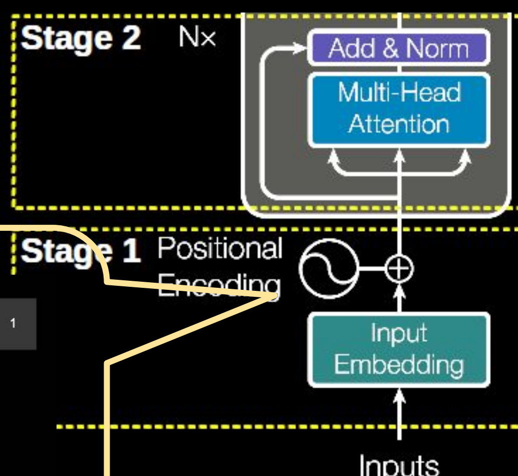




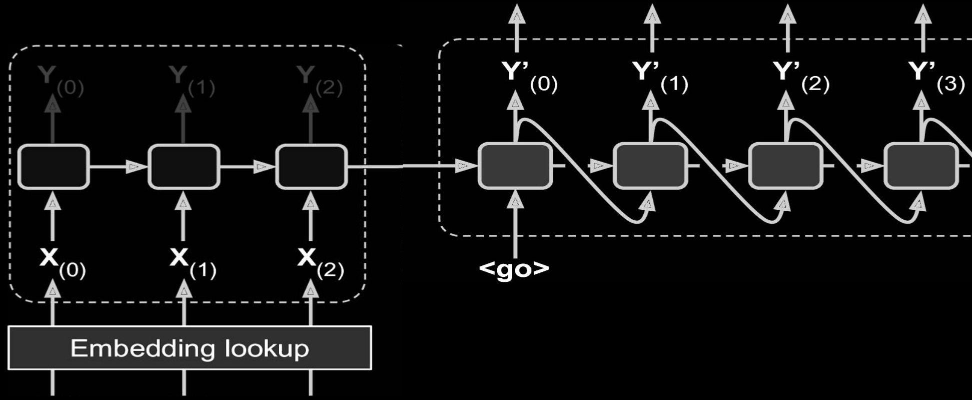
Transformer for Encoder-Decoder

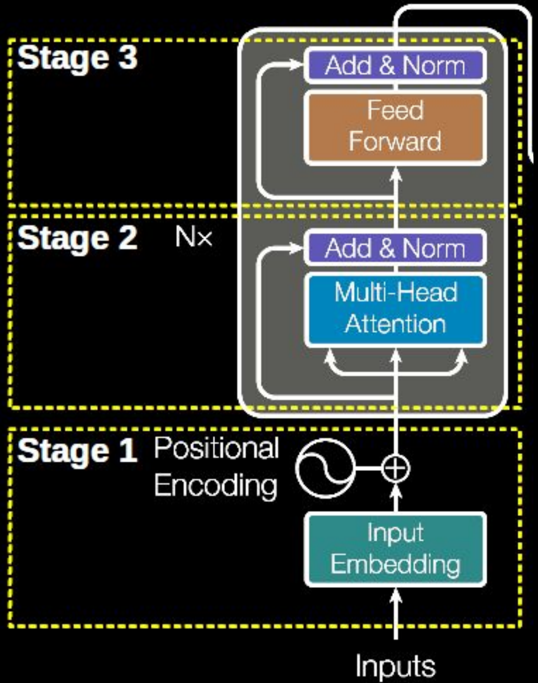




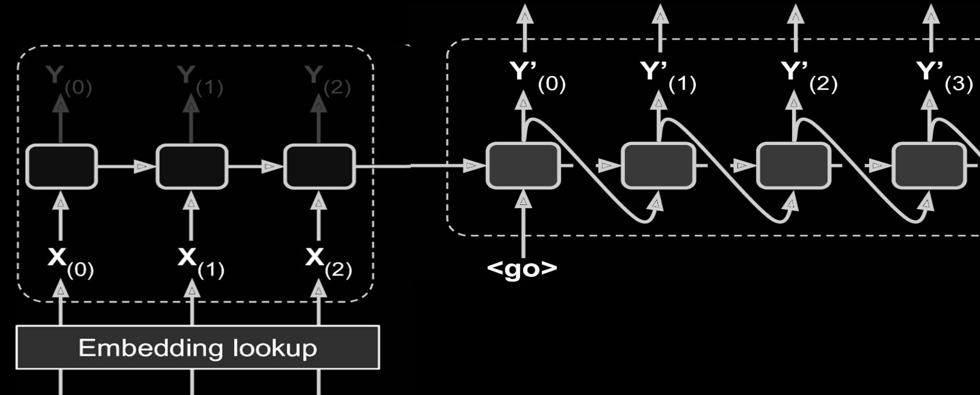


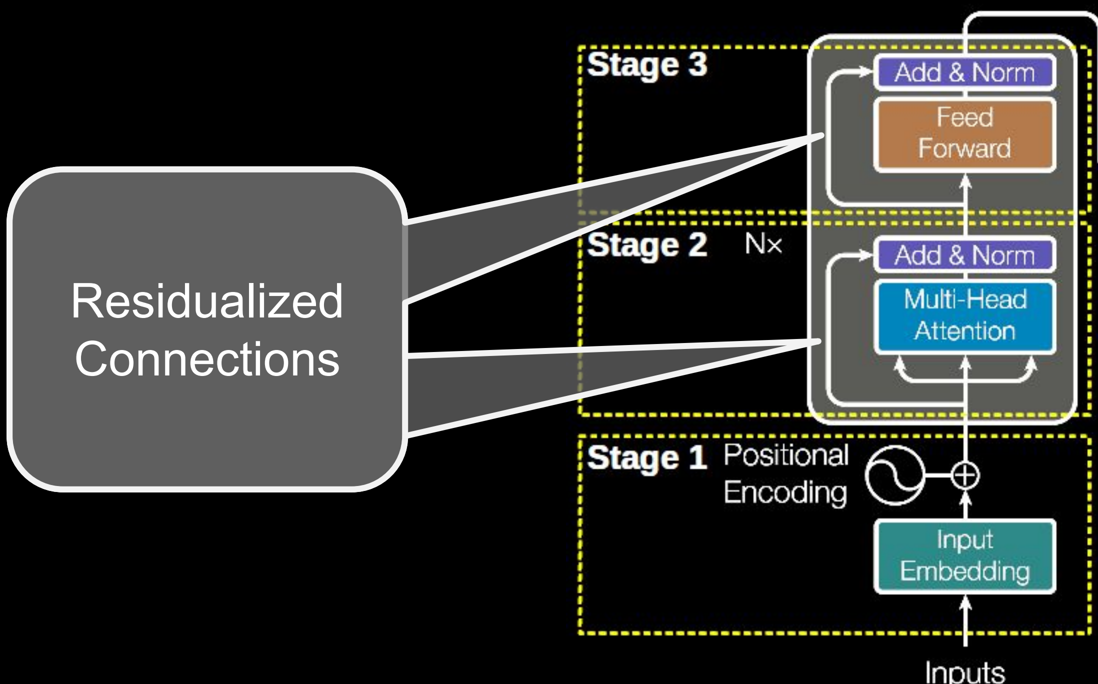
Transformer for Encoder-Decoder



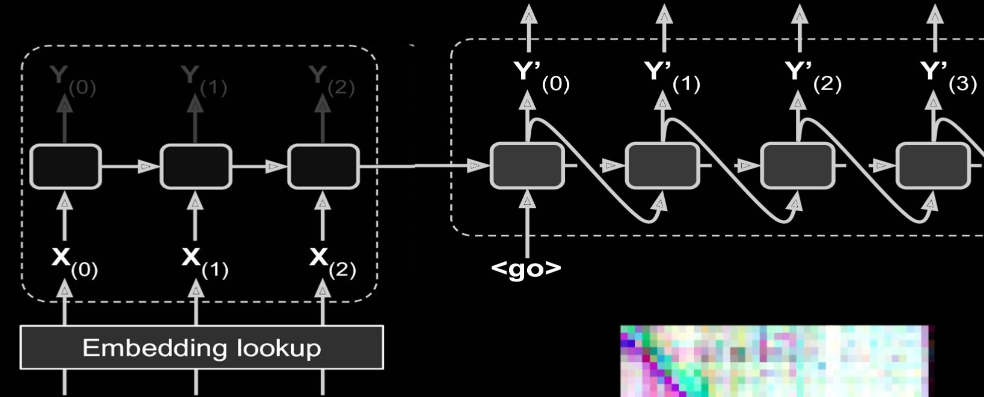


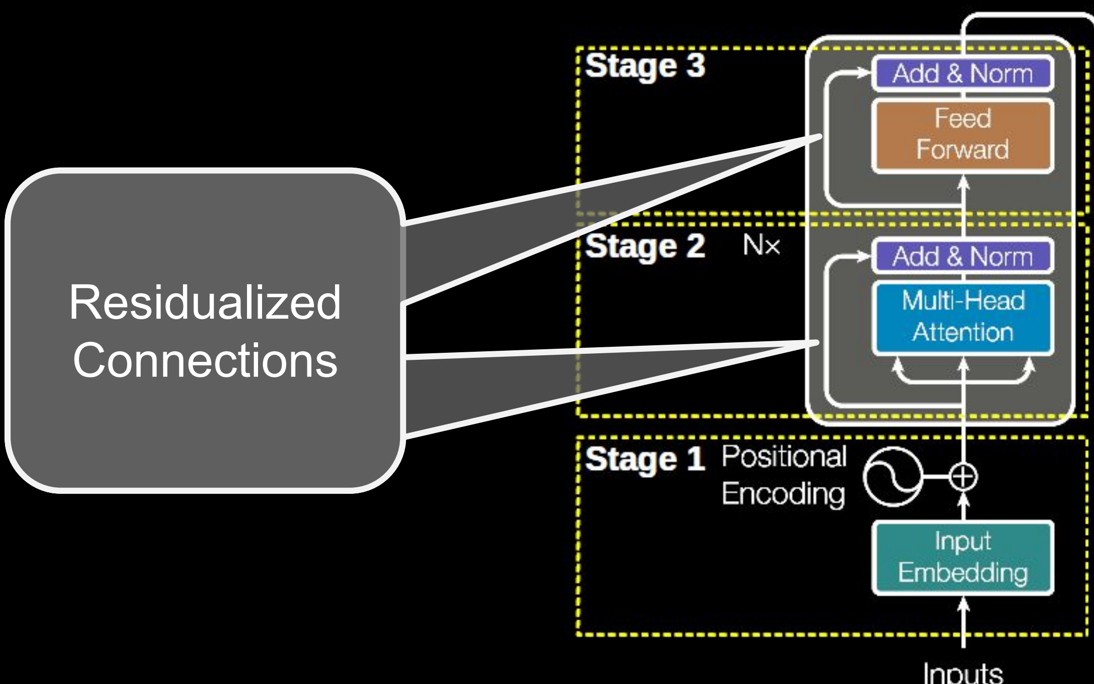
Transformer for Encoder-Decoder



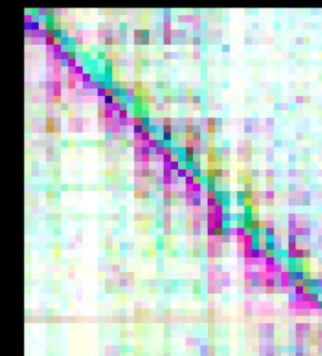


Transformer for Encoder-Decoder

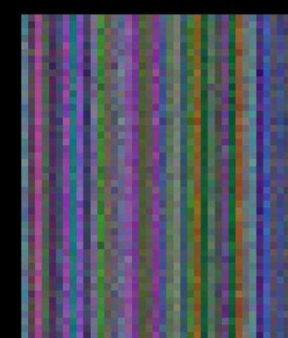




residuals enable positional information to be passed along

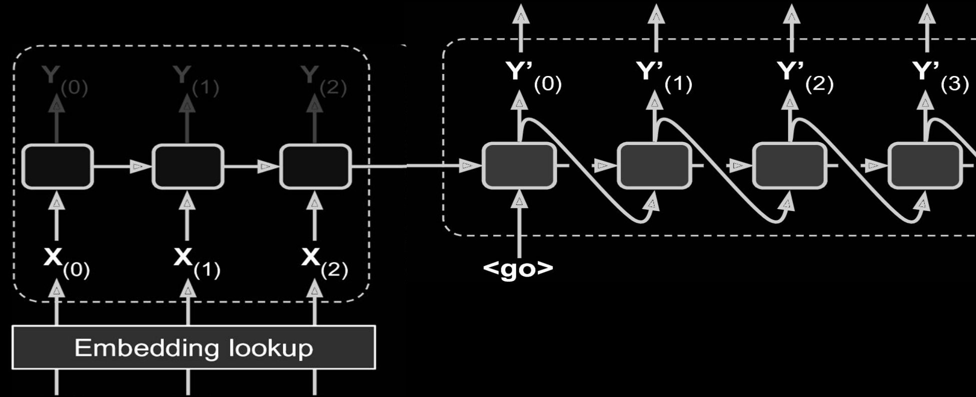


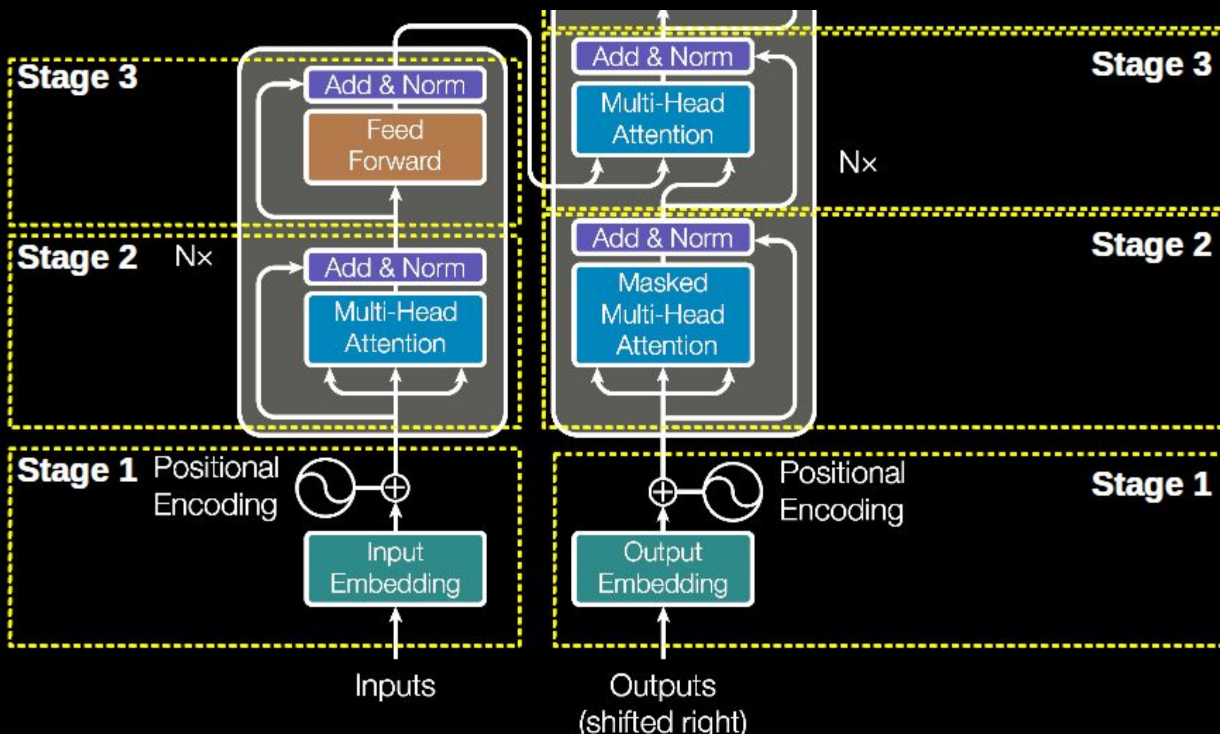
With residuals



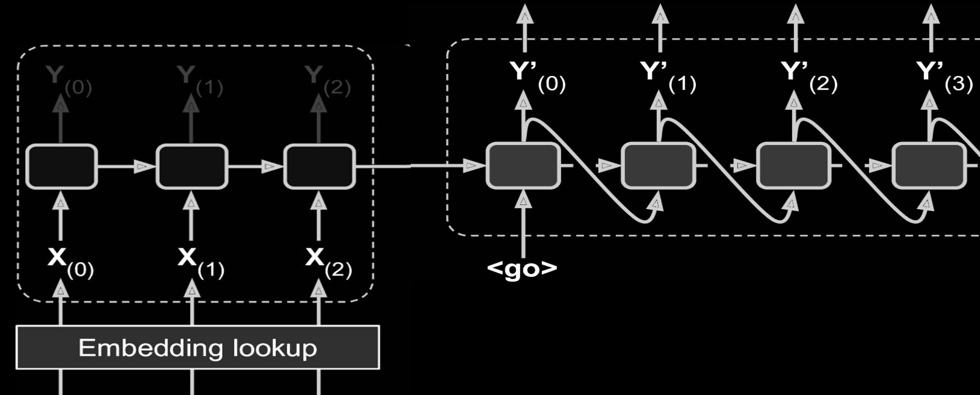
Without residuals

Transformer for Encoder-Decoder

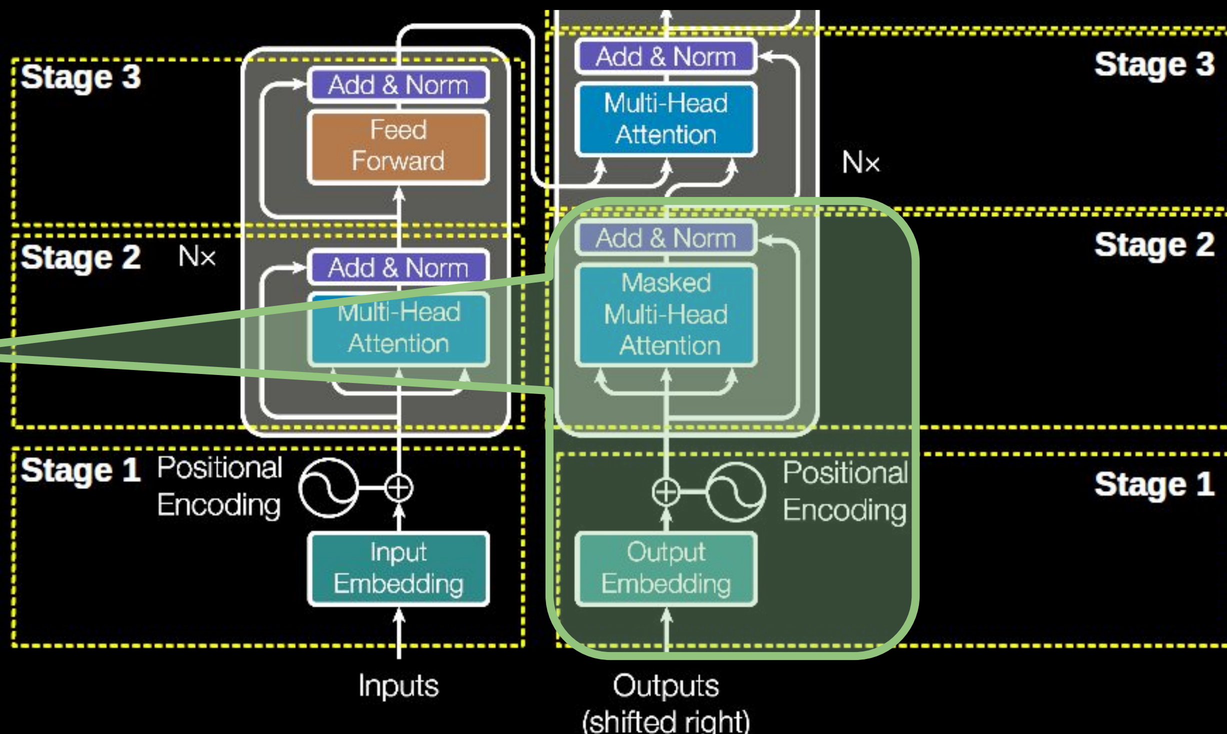




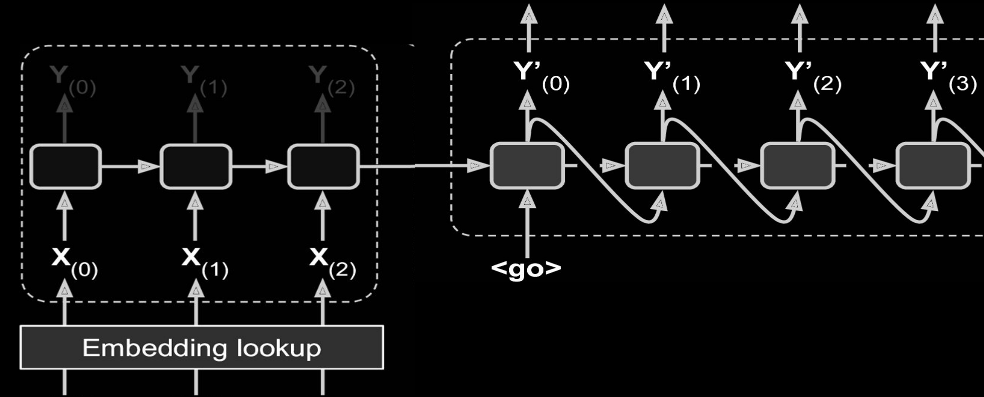
Transformer for Encoder-Decoder



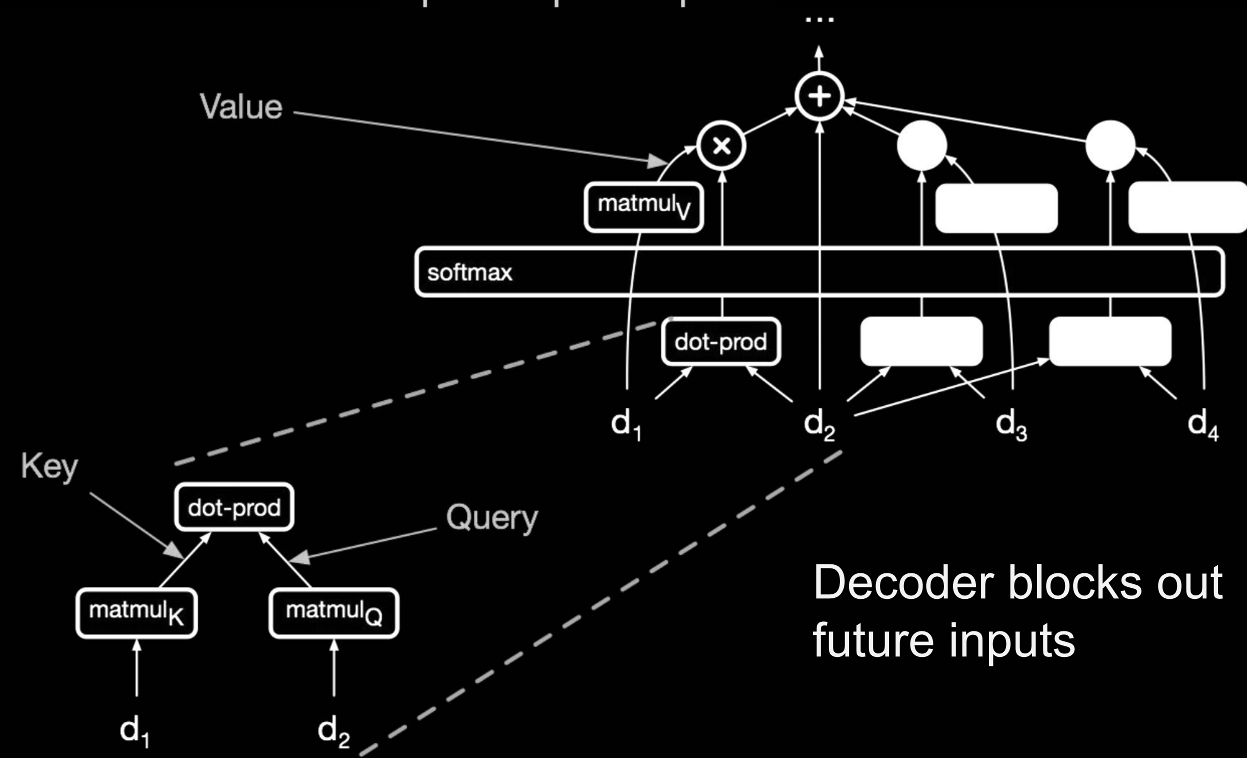
essentially, a language model



Transformer for Encoder-Decoder



essentially, a language model

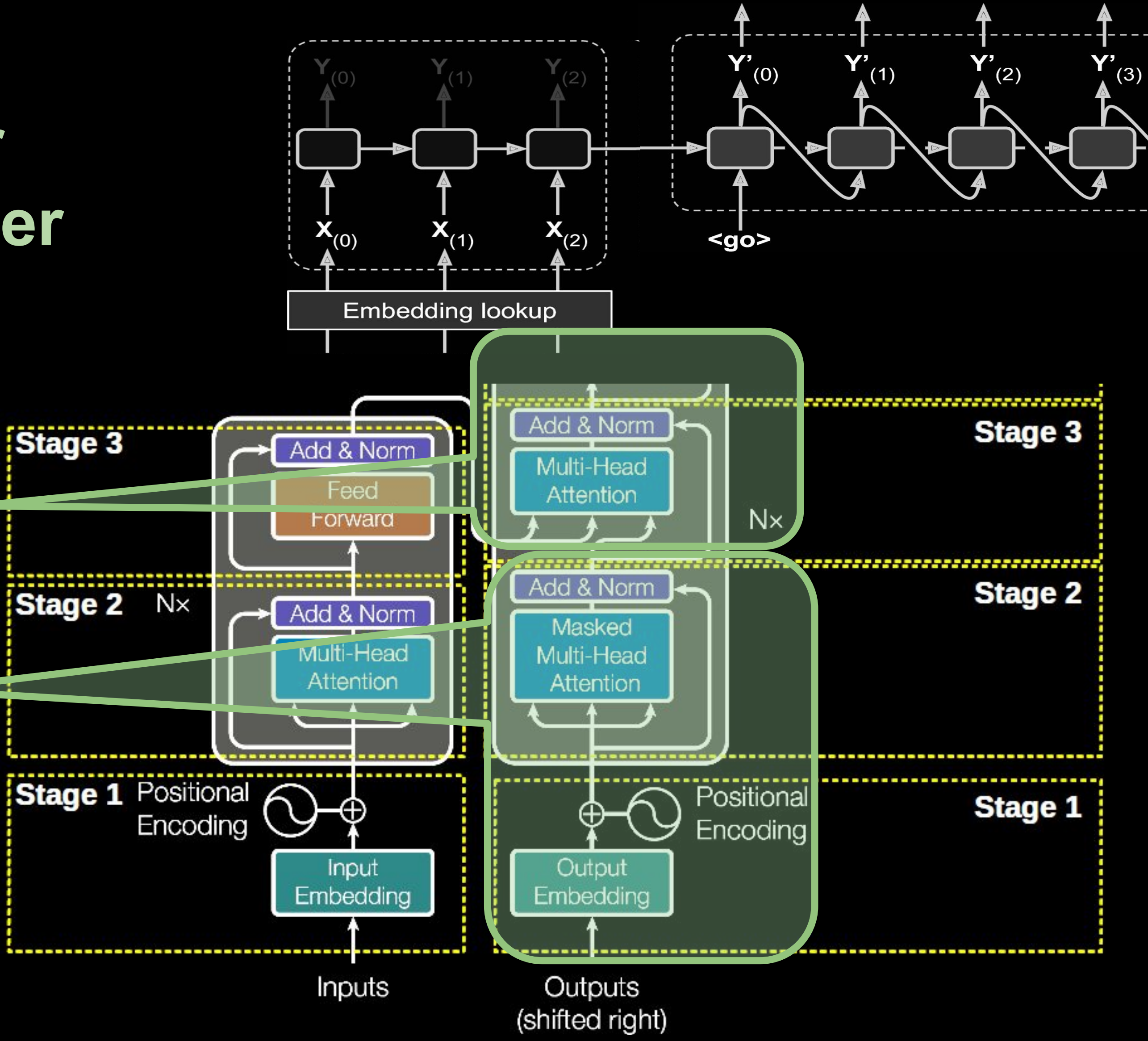


Decoder blocks out future inputs

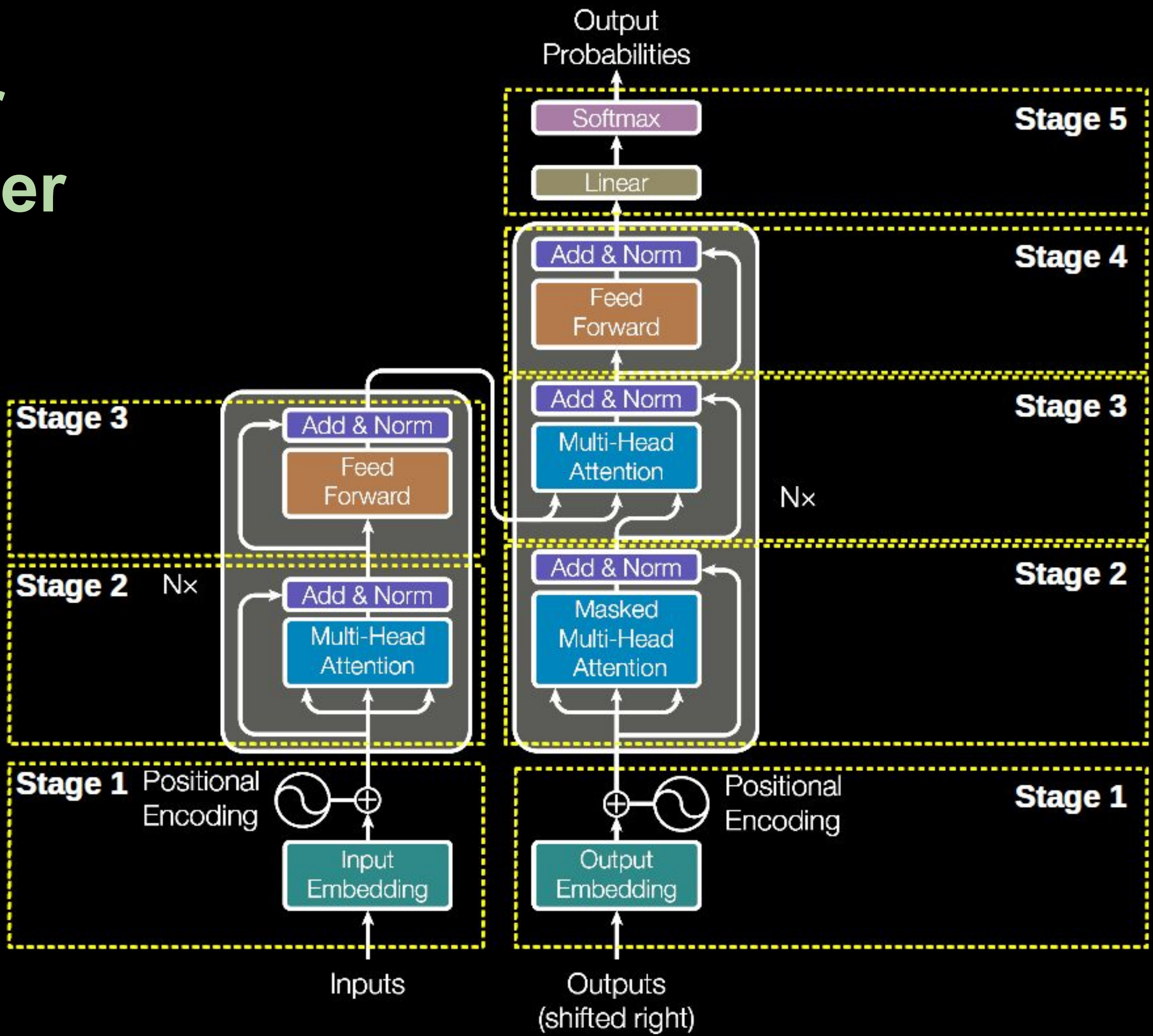
Transformer for Encoder-Decoder

Add conditioning of the LM based on the encoder

essentially, a language model



Transformer for Encoder-Decoder



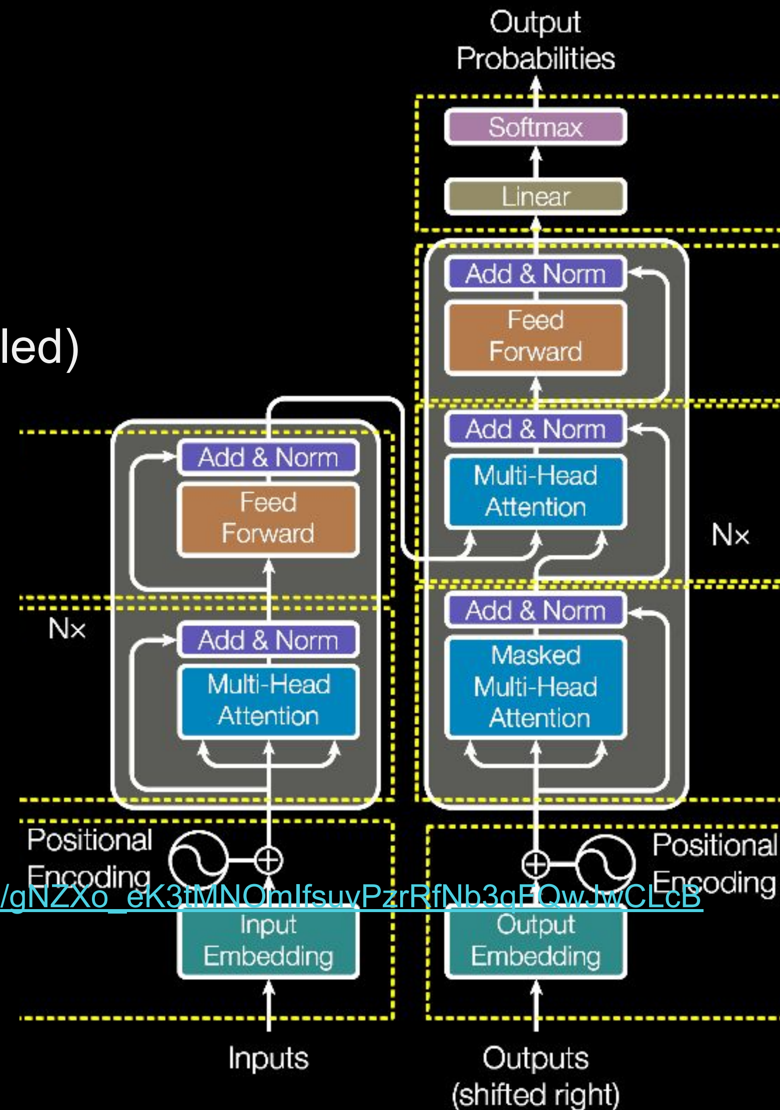
Transformer (as of 2017)

"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**

Transformer

- Utilize Self-Attention
- Simple att scoring function (dot product, scaled)
- Added linear layers for Q, K, and V
- Multi-head attention
- Added positional encoding
- Added residual connection
- Simulate decoding by masking



https://4.bp.blogspot.com/-OlrV-PAtEkQ/W3RkOJCBkaI/AAAAAAAADOg/gNZXo_eK3tMNOmIfsuvPzrRfNb3qFQwJwCLcBGAs/s640/image1.gif

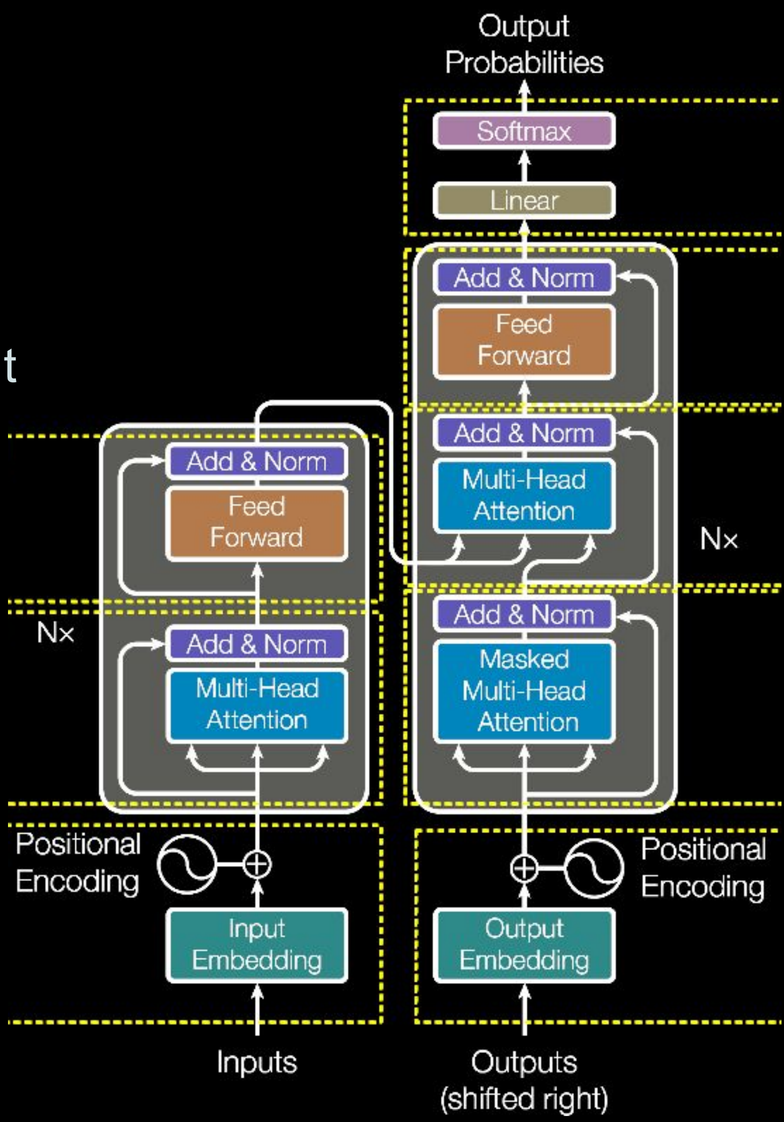
Transformer

Why?

- Don't need complexity of LSTM/GRU cells
- Constant num edges between words (or input steps)
- Enables "interactions" (i.e. adaptations) between words
- Easy to parallelize -- don't need sequential processing.

Drawbacks:

- Only unidirectional by default
- Only a "single-hop" relationship per layer (multiple layers to capture multiple)



BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

Produces contextualized embeddings
(or pre-trained contextualized encoder)

Drawbacks of Vanilla Transformers:

- Only unidirectional by default
- Only a “single-hop” relationship per layer
  (multiple layers to capture multiple)

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

Produces contextualized embeddings
(or pre-trained contextualized encoder)

- Bidirectional context by “masking” in the middle
- A lot of layers, hidden states, attention heads.

Drawbacks of Vanilla Transformers:

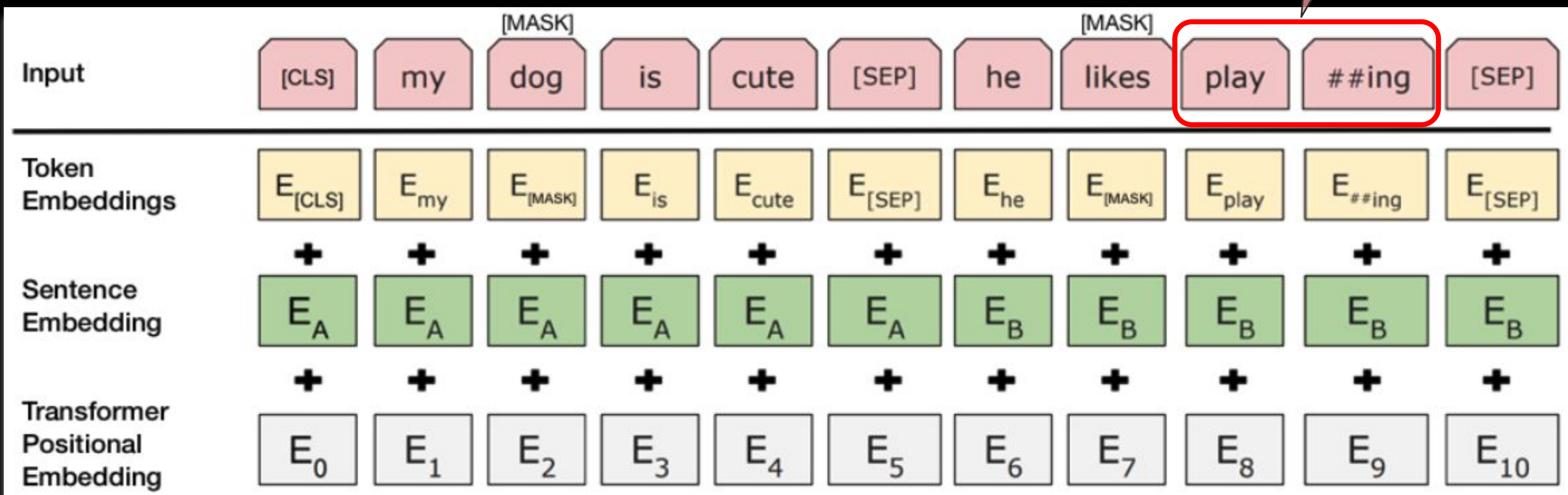
- Only unidirectional by default
- Only a “single-hop” relationship per layer
  (multiple layers to capture multiple)

BERT

tokenize into “word pieces”

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

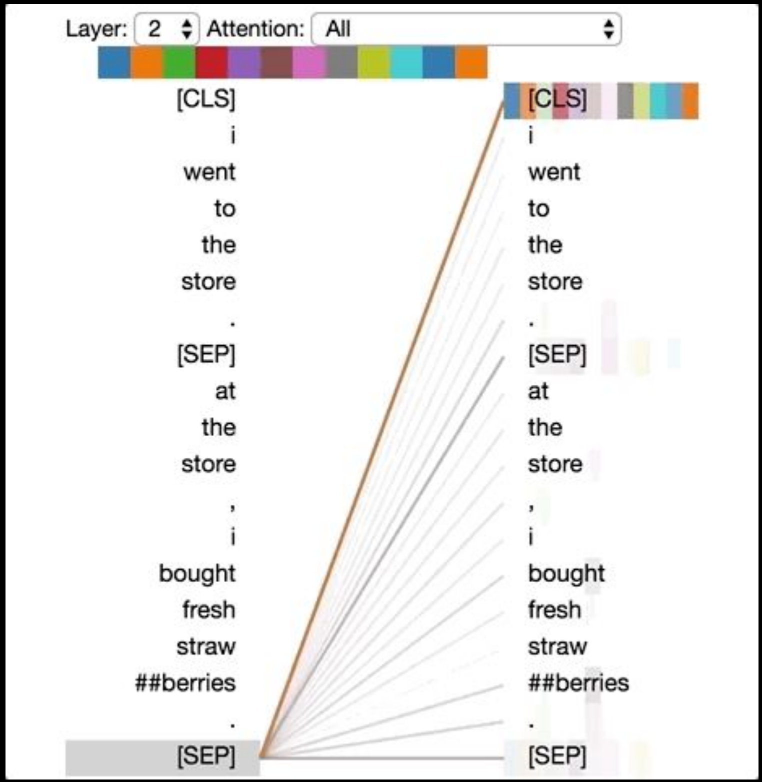
Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence



(Devlin et al., 2019)

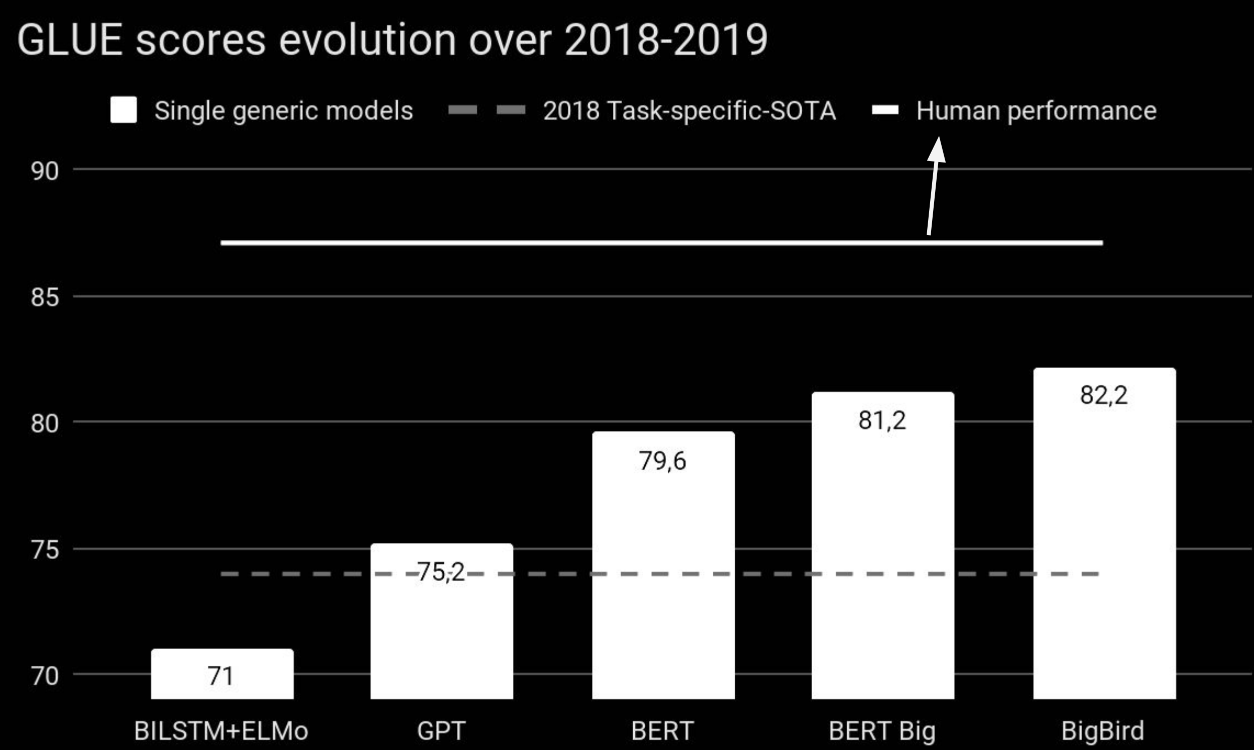
Bert: Attention by Layers

https://colab.research.google.com/drive/1vlOJ1lhdujVjfH857hvYKIdKPTD9Kid8



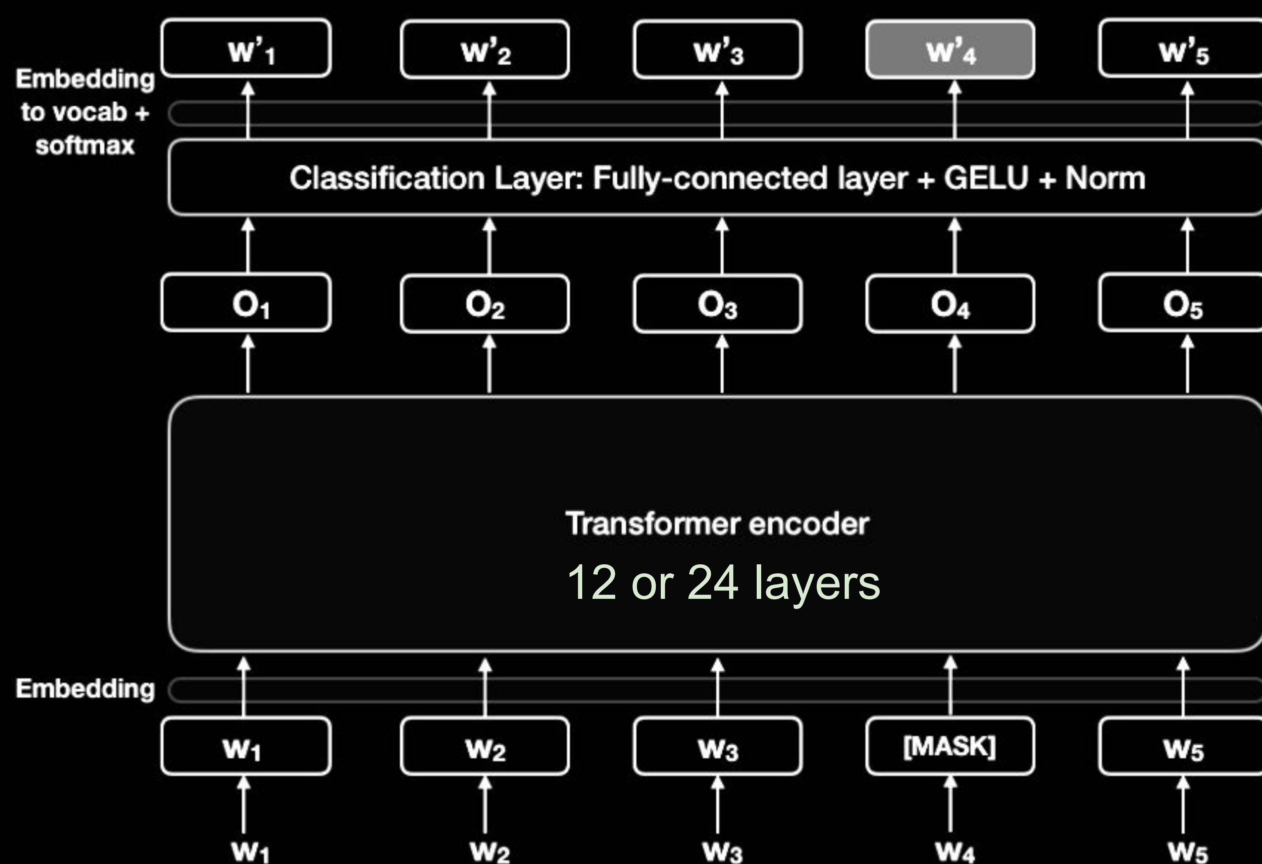
(Vig, 2019)

BERT Performance: e.g. Question Answering

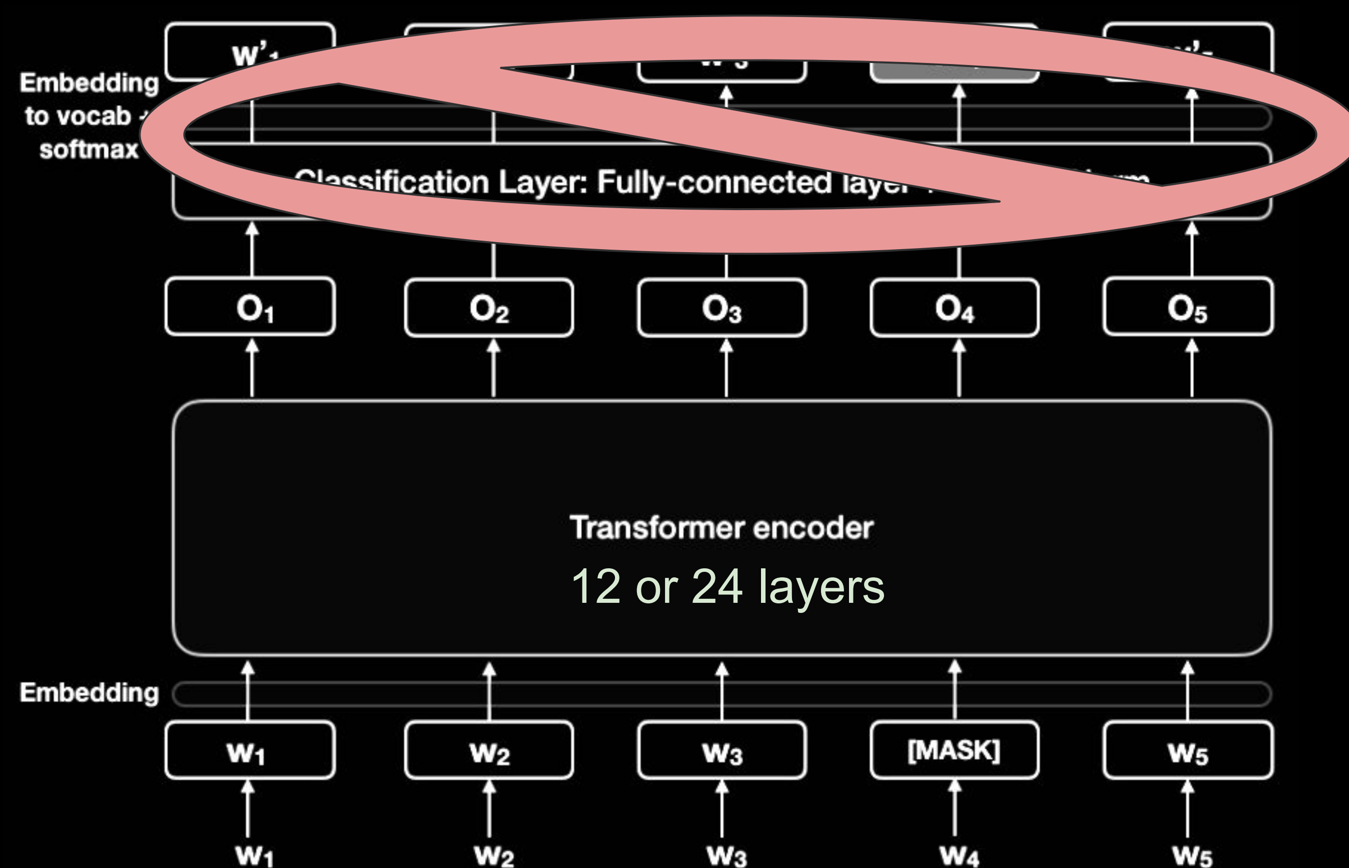


https://rajpurkar.github.io/SQuAD-explorer/

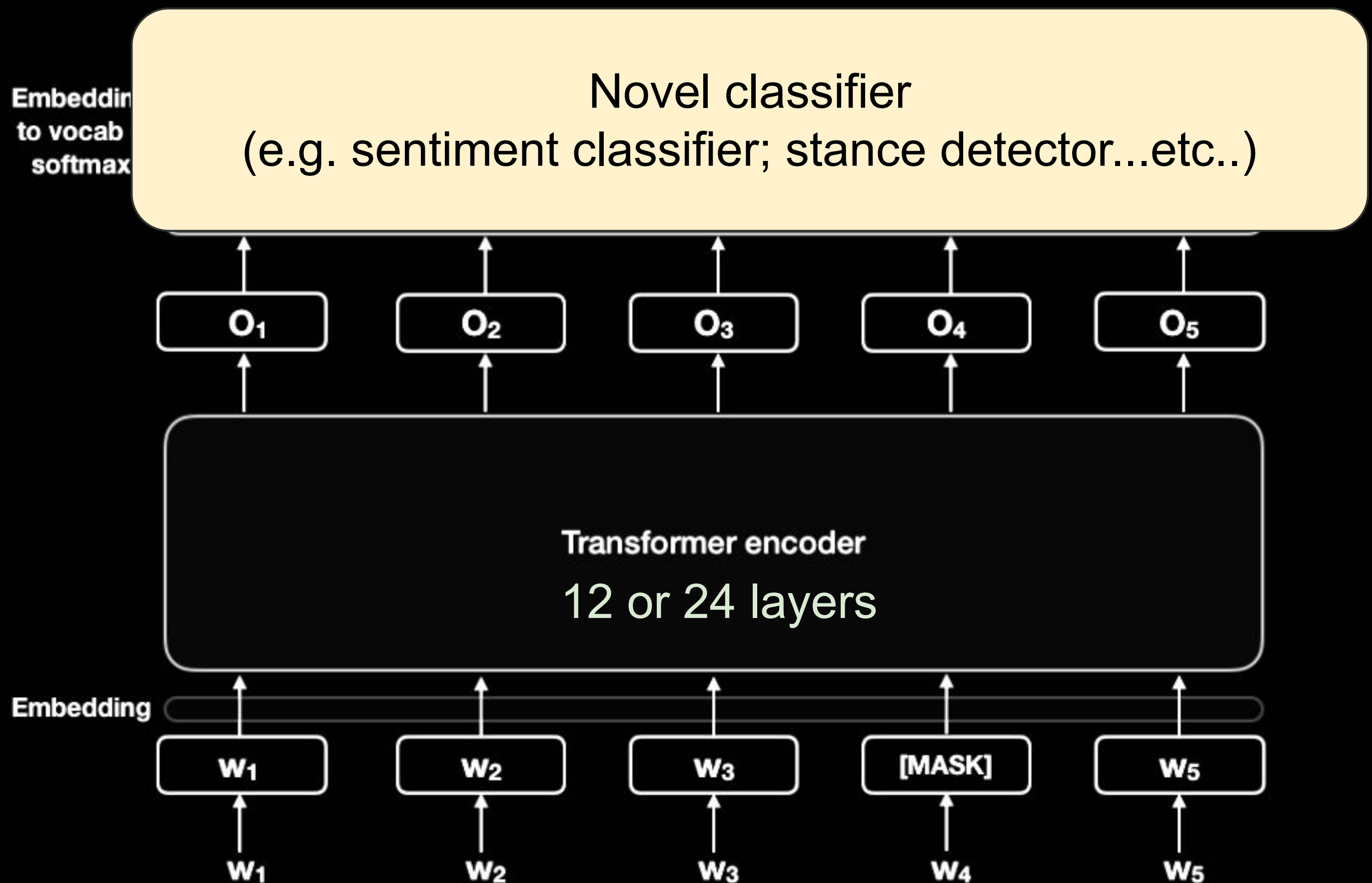
BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



Neural Network Summary

- Goal is accurate prediction of y (outcome) given features (x)
- Use L1 or L2 penalization (as a regularization) to avoid overfit
- Reason for Train, Dev, Test split
- Components of a neural network
- Batch Normalization
- Distribution options: why is data parallelism preferred?
- Recurrent Neural Network
- Convolution Operation with Filters

Feature Selection / Subset Selection

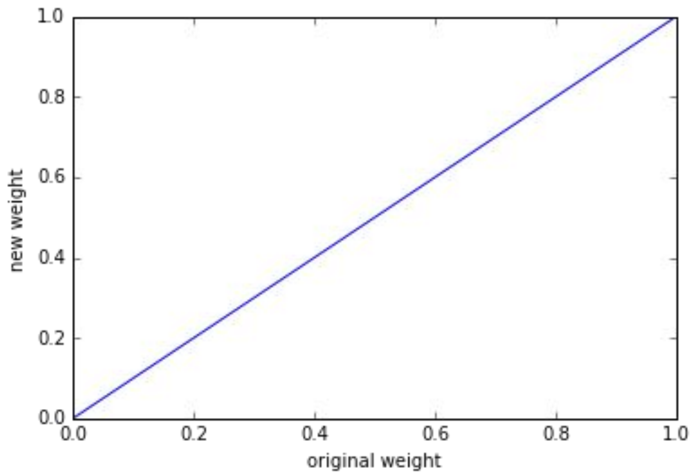
(bad) solution to overfit problem

Use less features based on Forward Stepwise Selection:

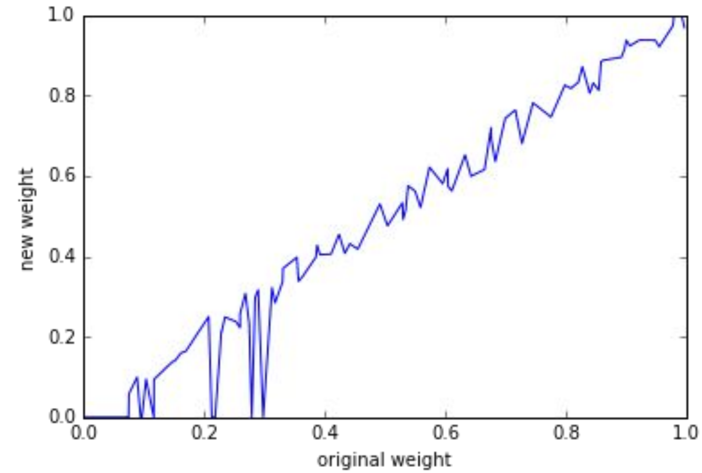
- start with current_model just has the intercept (mean)
  remaining_predictors = all_predictors

```
 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 Regression)

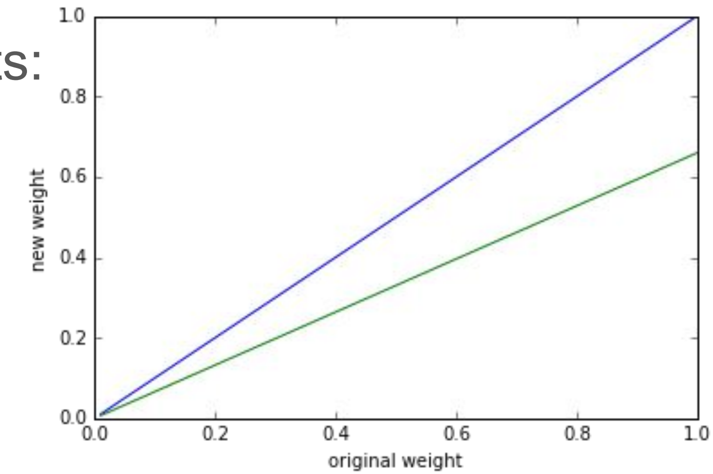
Idea: Impose a penalty on size of weights:

Ordinary least squares objective:

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

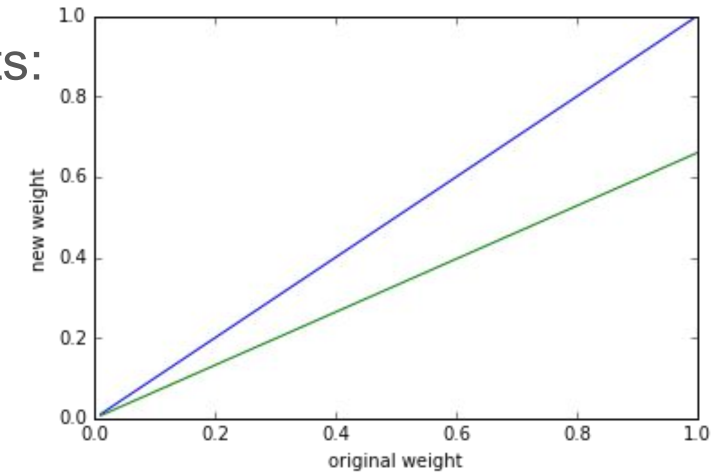
Ridge regression:

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



Regularization (L2, Ridge Regression)

Idea: Impose a penalty on size of weights:



Ordinary least squares objective:

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

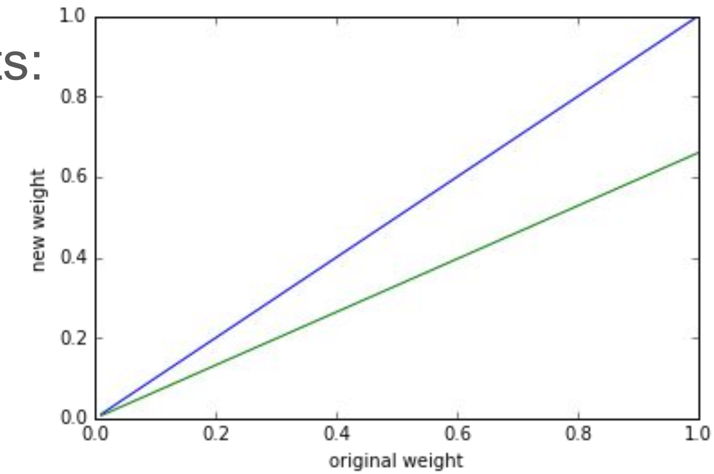
Ridge regression:

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

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

Regularization (L2, Ridge Regression)

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:

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

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

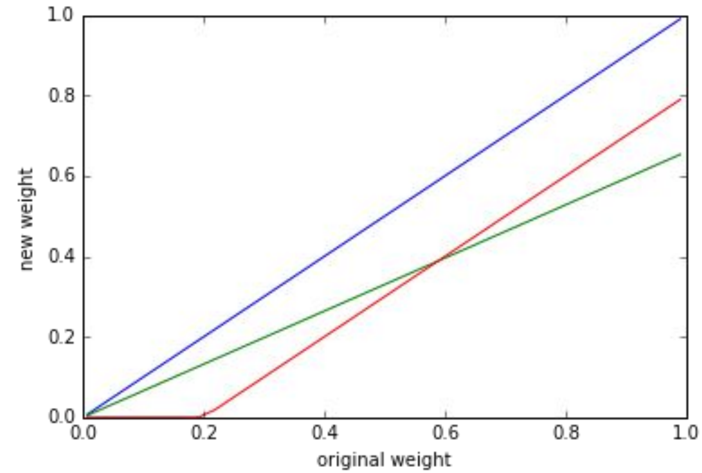
I: m x m identity matrix

Regularization (L1, The "Lasso")

Idea: Impose a penalty and zero-out some weights

The Lasso Objective:

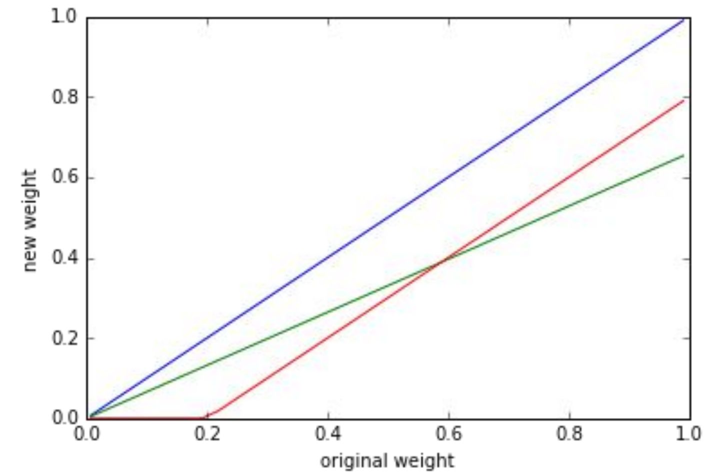
$$\hat{\beta}^{lasso} = argmin_{\beta}\{\frac{1}{2}\sum_{i=1}^{N}(Y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{m} |\beta_j|\}$$



$$\lambda||\beta||_1$$

Regularization (L1, The "Lasso")

Idea: Impose a penalty and zero-out some weights



The Lasso Objective:

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

$$\lambda||\beta||_1$$

No closed form matrix solution, but often solved with coordinate descent.

Application: p ≅ n or p >> n (p: features; n: observations)

Cluster Distribution

Model Parallelism

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
   beta=tf.Variable(...)
```

```
with tf.device("/gpu:0")
   y_pred=tf.matmul(beta,X)
```