Supervised Learning

\[(\text{genes}) \quad X_1 \quad X_2 \quad X_3 \quad (\text{health}) \quad Y\]
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

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

- **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.

* technically, still **operations**
Review: Gradient Descent

\[ J(w) = |\varepsilon | \]

Initial weight

Global cost minimum \( J_{\text{min}}(w) \)

Gradient

TensorFlow has built-in ability to derive gradients given a cost function. 
\[ \text{tf.gradients(cost, [params])} \]

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

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

\[ \hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \right\} \]
Weights Derived from Gradients

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

\[ \hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i} (y_i - \hat{y}_i)^2 \right\} \]

\[ \hat{y}_i = X_i \beta \]

Thus:

\[ \hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=0}^{N} (y_i - X_i \beta)^2 \right\} \]
Weights Derived from Gradients

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

\[ \hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \right\} \]

Thus:

\[ \hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=0}^{N} (y_i - X_i\beta)^2 \right\} \]

In standard linear equation:

\[ y = mx + b \]

let \( x' = x + [1, 1, ..., 1]_N^T \)

then, \( y = mx' \)

(if we add a column of 1s, \( mx + b \) is just matmul(m, x))
Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = \arg \min_\beta \left\{ \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \right\}$$

Thus:

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

Copyright 2014. Laerd Statistics.
Linear Regression: Trying to find “betas” that minimize:

\[ \hat{\beta} = \arg\min_\beta \left\{ \sum_{i}^{N} (y_i - \hat{y}_i)^2 \right\} \]

\[ \hat{y}_i = X_i \beta \]

Thus:

\[ \hat{\beta} = \arg\min_\beta \left\{ \sum_{i=0}^{N} (y_i - X_i \beta)^2 \right\} \]

How to update? \( \beta_{new} = \beta_{prev} - \alpha \times \text{grad} \)
Linear Regression: Trying to find “betas” that minimize:

\[ \hat{\beta} = \text{argmin}_\beta \left\{ \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \right\} \]

\[ \hat{y}_i = X_i \beta \]

Thus:

\[ \hat{\beta} = \text{argmin}_\beta \left\{ \sum_{i=0}^{N} (y_i - X_i \beta)^2 \right\} \]

How to update? \[ \beta_{\text{new}} = \beta_{\text{prev}} - \alpha \times \text{grad} \]

(for gradient descent) “learning rate”
Ridge Regression (L2 Penalized linear regression, $\lambda \|\beta\|_2^2$)

\[
\hat{\beta}_{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{m} x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2 \right\}
\]

1. Matrix Solution:

\[
\hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y
\]
Weights Derived from Gradients

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

1. Matrix Solution:

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

2. Gradient descent solution
(Mirrors many parameter optimization problems.)

1. Matrix Solution:

$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y$$
Ridge Regression (L2 Penalized linear regression, $\lambda ||\beta||^2_2$)

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

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

TensorFlow has built-in ability to derive gradients given a cost function.
Ridge Regression (L2 Penalized linear regression, $\lambda \| \beta \|^2_2$)

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])
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.

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

Pro: Parameters can be localized; Con: High communication for transferring intermediary data.
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.
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 intermediate data.
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 Intermediary data.

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

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

   Data Parallelism

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

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

   Model Parallelism

   Pro: Parameters can be localized;  Con: High communication for transferring Intermediary data.
Model Parallelism

Multiple devices on multiple machines

```python
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
Data Parallelism

\[
\beta = \text{tf.Variable}(...) \\
\text{pred} = \text{tf.matmul}(\beta, X)
\]

\[
\beta = \text{tf.Variable}(...) \\
\text{pred} = \text{tf.matmul}(\beta, X)
\]

\[
\beta = \text{tf.Variable}(...) \\
\text{pred} = \text{tf.matmul}(\beta, X)
\]

CPU:0

CPU:1

GPU:0
Data Parallelism

\[
\begin{align*}
\text{Beta} &= \text{tf.Variable}(\ldots) \\
\text{Pred} &= \text{tf.matmul} (\text{Beta}, X)
\end{align*}
\]

worker: 0

worker: 1

worker: 2
Distributing Data

\[ X \]

\[ y \]
# Distributing Data

<table>
<thead>
<tr>
<th>$X$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>batch_size-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>N-batch_size</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>
Distributing Data

\[ X \]

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

\[
\begin{align*}
X & \quad y \\
\text{batch\_size-1} & \quad \theta_{\text{batch0}} \\
\text{N-batch\_size} & \quad \theta_{\text{batch1}} \\
N & \quad \text{Combine parameters}
\end{align*}
\]
Distributing Data

\[ x_{\text{batch}_0} \rightarrow \theta_{\text{batch}_0} \]
\[ x_{\text{batch}_1} \rightarrow \theta_{\text{batch}_1} \]

Combine parameters

update params of each node and repeat
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)
Batch Gradient Descent

Stochastic Gradient Descent: One example at a time

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

(Geron, 2017)
Distributed TensorFlow

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

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

Program 1

CPU:0

CPU:1

GPU:0

Program 2
Multiple devices on single machine

```python
with tf.device("/cpu:1")
    beta=tf.Variable(...)  
with tf.device("/gpu:0")
    y_pred=tf.matmul(beta,X)
```
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)
Multiple devices on multiple machines

```python
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

\[
\begin{align*}
\text{beta} &= \text{tf.Variable}(...) \\
\text{pred} &= \text{tf.matmul}(	ext{beta}, X)
\end{align*}
\]
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

“ps”
- task 0
  - TF Server
    - Master
    - Worker
  - CPU:0

“worker”
- task 0
  - TF Server
    - Master
    - Worker
  - CPU:1
- task 1
  - TF Server
    - Master
    - Worker
  - CPU:0
  - GPU:0

Machine A

Machine B

(Geron, 2017: HOML: p.324)
Asynchronous Parameter Server

Parameter Server: Job is just to maintain values of variables being optimized.

Workers: do all the numerical “work” and send updates to the parameter server.
Workers do computation, send parameter updates to other workers, and store parameter updates from other workers. Requires low latency communication.
Workers do computation, send parameter updates to other workers, and store parameter updates from other workers. Requires low latency communication.
Review: Distributed ML

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 Parallelism

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

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

   Model Parallelism

   **Pro:** Parameters can be parallelized
   **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
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.
Two Examples

The WSJ Effect

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

distance from “standard” WSJ author demographics

model accuracy
Two Examples

The WSJ Effect

model accuracy

distance from “standard” WSJ author demographics

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

“Error Disparity”
Our data and models are (human) biased.

**“Outcome Disparity”**

Person-level:
- attribute = 1
- attribute = 2

**“Error Disparity”**
Predictive Bias Framework

The distribution of outcomes, given attribute $A$, is dissimilar than the ideal distribution:

$$Q(\hat{Y} \mid A) \neq P(Y \mid A)$$

The distribution of error ($\epsilon$) over at least two different values of an attribute ($A$) are unequal:

$$Q(\epsilon \mid A_i) \neq Q(\epsilon \mid A_j)$$

Predictive Bias Framework

Outcome Disparity
The distribution of outcomes, given attribute $A$, is dissimilar to the ideal distribution:
$$Q(Ŷ_t|A) \neq P(Y_t|A)$$

Error Disparity
The distribution of error ($ϵ$) over at least two different values of an attribute ($A$) are unequal:
$$Q(ϵ_t|A_i) \neq Q(ϵ_t|A_j)$$

Predictive Bias Framework

Potential origin

Consequence

Embedding Corpus

Features

$\theta_{embedding}$

(Pre-trained Side)

Source Population

Features

$X_{source}$

Outcomes

$Y_{source}$

(Model Side)

Target Population

Features

$X_{target}$

Biased outcomes

$\hat{Y}_{target}$

(Application Side)

Outcome disparity

The distribution of outcomes, given attribute $A$, is dissimilar than the ideal distribution:

$Q(\hat{Y}|A) \neq P(Y|A)$

Error disparity

The distribution of error ($\epsilon$) over at least two different values of an attribute ($A$) are unequal:

$Q(\epsilon|A_i) \neq Q(\epsilon|A_j)$

Label bias

Biased annotations, interaction, or latent bias from past classifications.

Predictive Bias Framework

**Embedding Corpus** (Pre-trained Side)
- features 
- $\theta_{embedding}$

**Source Population** (Model Side)
- features $X_{source}$
- outcomes $Y_{source}$
- **fit**

**Target Population** (Application Side)
- features $X_{target}$
- predict $\hat{Y}_{target}$

**potential origin**
- selection bias
  - The sample of observations themselves are not representative of the application population.

**consequence**
- label bias
  - Biased annotations, interaction, or latent bias from past classifications.

**outcome disparity**
- The distribution of outcomes, given attribute $A$, is dissimilar than the ideal distribution:
  \[ Q(\hat{Y}_t|A) \neq P(Y_t|A) \]

**error disparity**
- The distribution of error ($\epsilon$) over at least two different values of an attribute ($A$) are unequal:
  \[ Q(\epsilon_t|A_i) \neq Q(\epsilon_t|A_j) \]

Predictive Bias Framework

potential origin
- over-amplification: The model discriminates on a given human attribute beyond its source base-rate.
- label bias: Biased annotations, interaction, or latent bias from past classifications.

consequence
- selection bias: The sample of observations themselves are not representative of the application population.
- outcome disparity: The distribution of outcomes, given attribute $A$, is dissimilar than the ideal distribution:
  \[ Q(\hat{Y}_t|A) \neq P(Y_t|A) \]
- error disparity: The distribution of error ($\epsilon$) over at least two different values of an attribute ($A$) are unequal:
  \[ Q(\epsilon_t|A_i) \neq Q(\epsilon_t|A_j) \]

Embedding Corpus
- features $\theta_{\text{embedding}}$
- (Pre-trained Side)

Source Population
- features $X_{\text{source}}$
- outcomes $Y_{\text{source}}$
- (Model Side)

Target Population
- features $X_{\text{target}}$
- biased outcomes $\hat{Y}_{\text{target}}$
- (Application Side)

Predictive Bias Framework

- **potential origin**
  - over-amplification
    - The model discriminates on a given human attribute beyond its source base-rate.
  - label bias
    - Biased annotations, interaction, or latent bias from past classifications.

- **consequence**
  - outcome disparity
    - The distribution of outcomes, given attribute $A$, is dissimilar than the *ideal distribution*: $Q(\hat{Y}_t|A) \neq P(Y_t|A)$

- **Embedding Corpus**
  - features $\theta_{embedding}$
    - (Pre-trained Side)
  - semantic bias
    - Non-ideal associations between attributed lexeme (e.g. gendered pronouns) and non-attributed lexeme (e.g. occupation).

- **Source Population**
  - features $X_{source}$
    - (Model Side)
  - selection bias
    - The sample of observations themselves are not representative of the application population.

- **Target Population**
  - features $X_{target}$
    - (Application Side)
  - biased outcomes $\hat{Y}_{target}$
  - error disparity
    - The distribution of error ($\epsilon$) over at least two different values of an attribute ($A$) are unequal: $Q(\epsilon_t|A_i) \neq Q(\epsilon_t|A_j)$

---

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 ($\epsilon$) over at least two different values of an attribute ($A$) are unequal:
  $$Q(\epsilon|A_i) \neq Q(\epsilon|A_j)$$

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.
Privacy

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

(Histogram + KDE)

(Boxplot)

(Violin plot)

(Lewinson, 2019)
Useful Plots: Correlation

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

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

---

**Table:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Intelligence Quotient</th>
<th>Income</th>
<th>Sat W/ Life</th>
<th>Depression</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>0.04</td>
<td>0.12</td>
<td>0.02</td>
<td>-0.1</td>
</tr>
<tr>
<td>F2</td>
<td>-0.26</td>
<td>-0.19</td>
<td>-0.09</td>
<td>0.11</td>
</tr>
<tr>
<td>F3</td>
<td>-0.13</td>
<td>0.02</td>
<td>-0.02</td>
<td>-0.02</td>
</tr>
<tr>
<td>F4</td>
<td>0.27</td>
<td>-0.08</td>
<td>-0.12</td>
<td>0.11</td>
</tr>
<tr>
<td>F5</td>
<td>0.23</td>
<td>0.29</td>
<td>0.07</td>
<td>-0.21</td>
</tr>
</tbody>
</table>

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.

![Bar Plot Example](image)

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

![Line Plot Example](image)
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
Task: Determine a function, $f$ (or parameters to a function) such that $f(X) = Y$
Common Goal: Generalize to new data

Model

Does the model hold up?

Original Data

New Data?
Common Goal: Generalize to new data

Model
Does the model hold up?

Training Data

Testing Data
ML: GOAL

Training Data

Development Data

Testing Data

Model

Set training hyperparameters

Does the model hold up?
Goal: Decent estimate of model accuracy

N-Fold Cross Validation

Iter 1
- train
- dev
- test

Iter 2
- train
- dev
- test
- train

Iter 3
- test
- test
- test
- test
- test

...
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

   **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.
Combine parameters

update params of each node and repeat

$X$

$y$

$\theta_{\text{batch0}}$

$\theta_{\text{batch1}}$
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
Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$
Linear Regression: $y = wx$

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

From Linear Models to Neural Nets
\( z = wX \)

**Logistic:** \( \sigma(z) = \frac{1}{1 + e^{-z}} \)

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

**Rectified linear unit (ReLU):** \( \text{ReLU}(z) = \max(0, z) \)
Linear Regression: $y = wX$

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

(skymind, AI Wiki)
Linear Regression: $y = wX$

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

Batch Normalization

From Linear Models to Neural Nets
Batch Normalization

Input: Values of \( x \) over a mini-batch: \( B = \{x_1...m\} \); Parameters to be learned: \( \gamma, \beta \)

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

\[
\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}
\]

\[
\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{// mini-batch variance}
\]

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{// normalize}
\]

\[
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}
\]

(Ioffe and Szegedy, 2015)
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...x_m\}$; 
Parameters to be learned: $\gamma, \beta$

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

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

\[
\sigma^2_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \quad \text{// mini-batch variance}
\]

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \quad \text{// normalize}
\]

\[
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad \text{// scale and shift}
\]

(Ioffe and Szegedy, 2015)

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

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$;  
Parameters to be learned: $\gamma$, $\beta$  
**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

\[
\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean} \\
\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \quad \text{// mini-batch variance} \\
\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \text{// normalize} \\
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}
\]

*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

[Diagram of a feed-forward neural network with layers labeled as input, hidden layer 1, hidden layer 2, and output layer. The diagram includes nodes for inputs, weights, net input function, and activation function.]
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)):
    \[ h(i) = \text{tf.tanh}(\text{tf.matmul}(U, h(i-1)) + \text{tf.matmul}(W, x(i))) \] #update hidden state
    \[ y(i) = \text{tf.softmax}(\text{tf.matmul}(V, h(i))) \] #update output
...
\[ \text{cost} = \text{tf.reduce_mean}(-\text{tf.reduce_sum}(y*\text{tf.log}(y_{\text{pred}}))) \]
To find the gradient for the overall graph, we use \textbf{back propagation}, which \textit{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

\[ C(Y_{(2)}, Y_{(3)}, Y_{(4)}) \]

\[ W, b \]

\[ X_{(0)}, X_{(1)}, X_{(2)}, X_{(3)}, X_{(4)} \]

(Geron, 2017)
How to Addressing Vanishing Gradient?

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

RNN model

“unrolled” depiction

(Geron, 2017)
Gated Recurrent Unit

RNN: The GRU

(Geron, 2017)
Gated Recurrent Unit

relevance gate

update gate

\[ h(t) = (1 - z(t)) \odot h(t-1) + z(t) \odot \text{tanh}(W_{xh} \odot x(t) + W_{hh} \odot h(t-1)) \]

FC

GRU cell

Element-wise multiplication
Addition
logistic
tanh

(Geron, 2017)
RNN: The GRU

Gated Recurrent Unit

relevance gate

update gate

A candidate for updating $h$, sometimes called: $h_{\sim}$

(Geron, 2017)
The GRU

Gated Recurrent Unit

\[ z_{(t)} = \sigma(W_{xz}^T \cdot x_{(t)} + W_{hz}^T \cdot h_{(t-1)} + b_z) \]

\[ r_{(t)} = \sigma(W_{xr}^T \cdot x_{(t)} + W_{hr}^T \cdot h_{(t-1)} + b_r) \]

\[ g_{(t)} = \tanh(W_{xg}^T \cdot x_{(t)} + W_{hg}^T \cdot (r_{(t)} \otimes h_{(t-1)}) + b_g) \]

\[ h_{(t)} = z_{(t)} \otimes h_{(t-1)} + (1 - z_{(t)}) \otimes g_{(t)} \]

The cake, which contained candles, was eaten.
What about the gradient?

\[ z_{(t)} = \sigma(W_{xz}^T \cdot x_{(t)} + W_{hz}^T \cdot h_{(t-1)} + b_z) \]
\[ r_{(t)} = \sigma(W_{xr}^T \cdot x_{(t)} + W_{hr}^T \cdot h_{(t-1)} + b_r) \]
\[ g_{(t)} = \tanh (W_{xg}^T \cdot x_{(t)} + W_{hg}^T \cdot (r_{(t)} \otimes h_{(t-1)}) + b_g) \]
\[ h_{(t)} = z_{(t)} \otimes h_{(t-1)} + (1 - z_{(t)}) \otimes g_{(t)} \]

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?

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

(Geron, 2017)
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

(Barter, 2018)
Convolution Layer

(Barter, 2018)
Breakthrough in image classification: Let the model automatically learn the filter weights!
Subsampling (Pooling)

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

2x2 pooling
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
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: \[ J^{(t)} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_{i,j}^{(t)} \log \hat{y}_{i,j}^{(t)} \quad \text{-- "cross entropy error"} \]
RNN_cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred)))

#where did this come from?

Logistic Regression Likelihood:

\[ L(\beta_0, \beta_1, \ldots, \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} \]

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:

\[ J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_{i,j} \log p(x_{i,j}) \quad \text{ (a “multiclass” log loss)} \]

Final Cost Function:

\[ J^{(t)} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{|V|} y_{i,j}^{(t)} \log \hat{y}_{i,j}^{(t)} \quad \text{-- ”cross entropy error”} \]
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**

\[ h(t) = g(h(t-1)U + x(t)V) \]

---

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

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

- Long distance dependency when translating:

Kayla kicked the ball.

The ball was kicked by kayla.

Kayla kicked the ball.
The Transformer: Attention-only Models

Challenge:

- Long distance dependency when translating:

Kayla kicked the ball.

The ball was kicked by kayla.

Kayla kicked the ball.
Attention

The Transformer: Attention-only Models
Attention

Score function:

\[ \psi_{\text{mult}}(h_i, s) = s^T W h_i \]

\[ \alpha_{h_i \rightarrow s} = \text{softmax}(\psi(h_i, s)) \]
Attention

Score function:
\[
\psi_{\text{mult}}(h_i, s) = s^T W h_i
\]
\[
\alpha_{h_i \rightarrow s} = \text{softmax}(\psi(h_i, s))
\]
\[
c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \rightarrow s_n} z_n
\]
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:

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

\[ \alpha_{h_i \rightarrow s} = \text{softmax}(\psi(h_i, s)) \]

\[ c_{h_i} = \sum_{n=1}^{\mid s \mid} \alpha_{h_i \rightarrow s_n} z_n \]
Attention

Score function:

\[ \psi_{\text{mult}}(h_i, s) = s^T W h_i \]

\[ \alpha_{h_i \rightarrow s} = \text{softmax}(\psi(h_i, s)) \]

\[ c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \rightarrow s_n} z_n \]
Attention as weighting a value based on a query and key:

\[
\psi_{\alpha} \quad \text{activation}
\]

\[
\begin{array}{c}
\Box \quad \text{Query} \\
\Box \quad \psi_{\alpha} \\
\Box \quad \text{Key} \\
\Box \quad \text{Value} \\
\Box \quad \text{Output}
\end{array}
\]

(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

\[ \alpha \]

\[ \psi \]

\[ b \]

\[ h_{i-1} \] \[ h_i \] \[ h_{i+1} \]

(Eisenstein, 2018)
The Transformer: Attention-only Models

(Eisenstein, 2018)
The Transformer: Attention-only Models

\[
\begin{align*}
\text{Output} & \quad \alpha \\
\psi & \quad v \\
b & \quad \begin{cases} h_{i-1} \\ h_i \\ h_{i+1} \\ h_{i+2} \end{cases}
\end{align*}
\]
The Transformer: Attention-only Models

The Transformer: Attention-only Models
The Transformer: “Attention-only” models

\[ y_{i-1} \quad y_i \quad y_{i+1} \quad y_{i+2} \]

Output

\[ \alpha \quad \psi \quad \beta \]

\[ b_{i-1} \quad b_i \quad b_{i+1} \quad b_{i+2} \]

\[ w_{i-1} \quad w_i \quad w_{i+1} \quad w_{i+2} \quad \ldots \]
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

\[ \psi_{dp}(k, q) = (k^T q) \sigma \]
The Transformer: “Attention-only” models

Output

\[ \begin{align*}
\psi_{dp}(k, q) &= (k^t q) \sigma \\
\end{align*} \]

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

Residualized Connections

Stage 1
- Positional Encoding
- Input Embedding

Stage 2
- $N \times$
- Add & Norm
- Multi-Head Attention

Stage 3
- Add & Norm
- Feed Forward

Embedding lookup

$Y^{(0)}$, $Y^{(1)}$, $Y^{(2)}$, $Y^{(3)}$
Transformer for Encoder-Decoder

Residualized Connections

Stage 1: Positional Encoding
- Input Embedding
- Inputs

Stage 2: Multi-Head Attention
- Add & Norm
- Feed Forward

Stage 3: Add & Norm

Embedding lookup

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

essentially, a language model

Add conditioning of the LM based on the encoder
Transformer for Encoder-Decoder
### Transformer (as of 2017)

“WMT-2014” Data Set. BLEU scores:

<table>
<thead>
<tr>
<th>Model</th>
<th>EN-DE</th>
<th>EN-FR</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNMT (orig)</td>
<td>24.6</td>
<td>39.9</td>
</tr>
<tr>
<td>ConvSeq2Seq</td>
<td>25.2</td>
<td>40.5</td>
</tr>
<tr>
<td>Transformer*</td>
<td>28.4</td>
<td>41.8</td>
</tr>
</tbody>
</table>
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-PAIekQ/W3RkOJCBkAI/AAAAAAAADOg/gNZXo_5K3fOdNQmJfsuvP2zRfNh3qDPowvMCQCLcB
GAs/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)
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 of Vanilla Transformers:

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

---

BERT

Bidirectional Encoder Representations from Transformers

Produces contextualized embeddings (or pre-trained contextualized encoder)
**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 of Vanilla Transformers:**

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

**BERT**

**Bidirectional Encoder Representations from Transformers**

Produces contextualized embeddings
(or pre-trained contextualized encoder)

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

Differences from previous state of the art:

- Bidirectional transformer (through masking)
- Directions jointly trained at once.
- Capture sentence-level relations
  
  (Devlin et al., 2019)

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

![BERT Diagram](image)
Bert: Attention by Layers

https://colab.research.google.com/drive/1vlOJ1hdujVjfH857hvYKlDKPTD9Kid8

(Vig, 2019)
BERT Performance: e.g. Question Answering

GLUE scores evolution over 2018-2019

- Single generic models
- 2018 Task-specific-SOTA
- Human performance

- BILSTM+ELMo: 71
- GPT: 79.6
- BERT: 81.2
- BERT Big: 82.2
- BigBird: 82.2

https://rajpurkar.github.io/SQuAD-explorer/
BERT: Pre-training; Fine-tuning

Transformer encoder
12 or 24 layers
BERT: Pre-training; Fine-tuning

Transformer encoder
12 or 24 layers
BERT: Pre-training; Fine-tuning

Novel classifier
(e.g. sentiment classifier; stance detector...etc..)

Transformer encoder
12 or 24 layers
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_predictors
          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:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_{i} - \sum_{j=1}^{m} x_{ij} \beta_{j})^{2} \right\}$$

Ridge regression:

$$\hat{\beta}_{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_{i} - \sum_{j=1}^{m} x_{ij} \beta_{j})^{2} + \lambda \sum_{j=1}^{m} \beta_{j}^{2} \right\}$$
Regularization (L2, Ridge Regression)

Idea: Impose a penalty on size of weights:

Ordinary least squares objective:

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

Ridge regression:

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

$$\lambda \sum_{j=1}^{m} \beta_j^2$$
Regularization (L2, Ridge Regression)

Idea: Impose a penalty on size of weights:

Ordinary least squares objective:

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

Ridge regression:

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

In Matrix Form:

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

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

$I: m \times m$ identity matrix
Regularization (L1, The “Lasso”)

Idea: Impose a penalty and zero-out some weights

The Lasso Objective:

\[ \hat{\beta}_{\text{lasso}} = \arg\min_{\beta} \left\{ \frac{1}{2} \sum_{i=1}^{N} (Y_i - \sum_{j=1}^{m} x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^{m} |\beta_j| \right\} \]
Regularization (L1, The “Lasso”)

Idea: Impose a penalty and zero-out some weights

The Lasso Objective:

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

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

Application: $p \approx n$ or $p >> n$  \hspace{1cm} (p: features; n: observations)
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)

Model Parallelism

Transfer Tensors