NATURAL LANGUAGE PROCESSING PROJECT

Group 4

Stefani Guevara

Mariko McDougall

Arathi Nair

CONTENTS

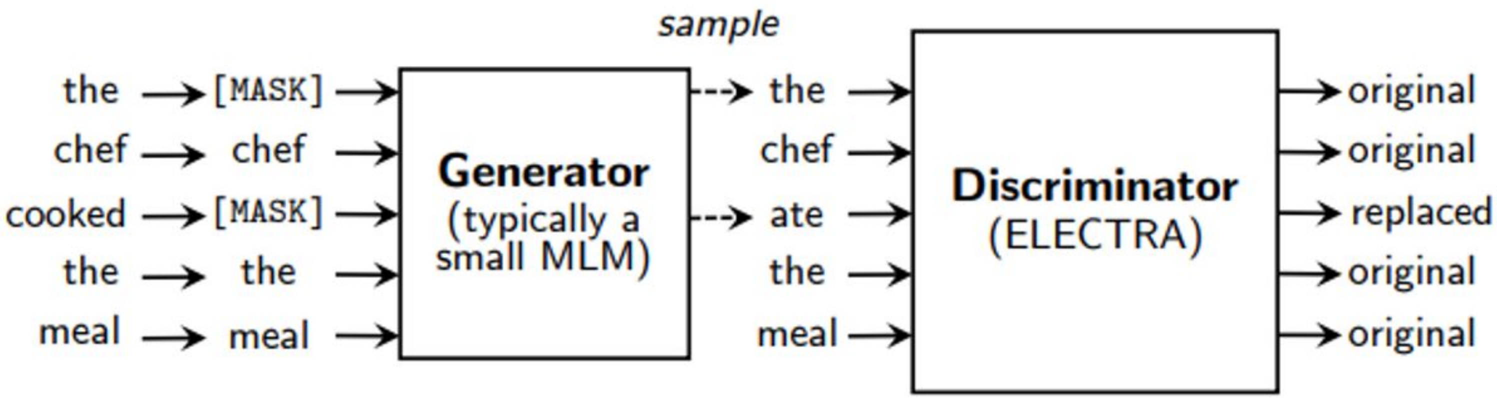
GLUE BENCHMARK

Single Sentence Tasks	***Cola -*** *Grammatical Correctness*
	SST-2 - *Sentiment Analysis*
Similarity & Paraphrase Tasks	*MRPC - Paraphrase detection of two sentences*
	QQP - Paraphrase detection of two questions
	STS-B - Sentence similarity
Inference Tasks	*MNLI - Sentences match/mismatch*
	QNLI - Question & Answer pairing
	RTE - Sentences match/mismatch
	WNLI - Sentences match/mismatch with pronoun substitution

ELECTRA

- Introduced a unique pre-training approach called *"replaced token detection"*
- Predicts all the input token unlike its predecessors that relied on MLM pre-training and predicts only 15% of the tokens
- Uses significantly less compute resources
- Results match or exceed downstream performance of a pre trained MLM
- More efficient pre-training compared to MLM
- Produces an improved comprehension of context

Model Architecture



1. MLM selects a random set of positions to mask out m = [m_1,...m_k]
2. Generator predicts original words of the [MASK] tokens
3. Discriminator distinguishes tokens replaced by the generator
4. Model is trained to distinguish *"real"* input tokens vs *"fake"* input tokens

GENERATOR

Output probability for a token x_t with softmax layer

$$p_G(x_t|\boldsymbol{x}) = \exp\left(e(x_t)^T h_G(\boldsymbol{x})_t\right) / \sum_{x'} \exp\left(e(x')^T h_G(\boldsymbol{x})_t\right)$$

x = sequence on input tokens

h(x) = contextualized vector representations

e = token embeddings

Loss function:

$$\mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) = \mathbb{E}\left(\sum_{i \in \boldsymbol{m}} -\log p_G(x_i|\boldsymbol{x}^{\text{masked}})\right)$$

DISCRIMINATOR

Predicts if token x_t is “real” , with a sigmoid output layer

$$D(\boldsymbol{x}, t) = \text{sigmoid}(w^T h_D(\boldsymbol{x})_t)$$

t = position of the token

h_D = hidden layers, embedding layers, attention heads

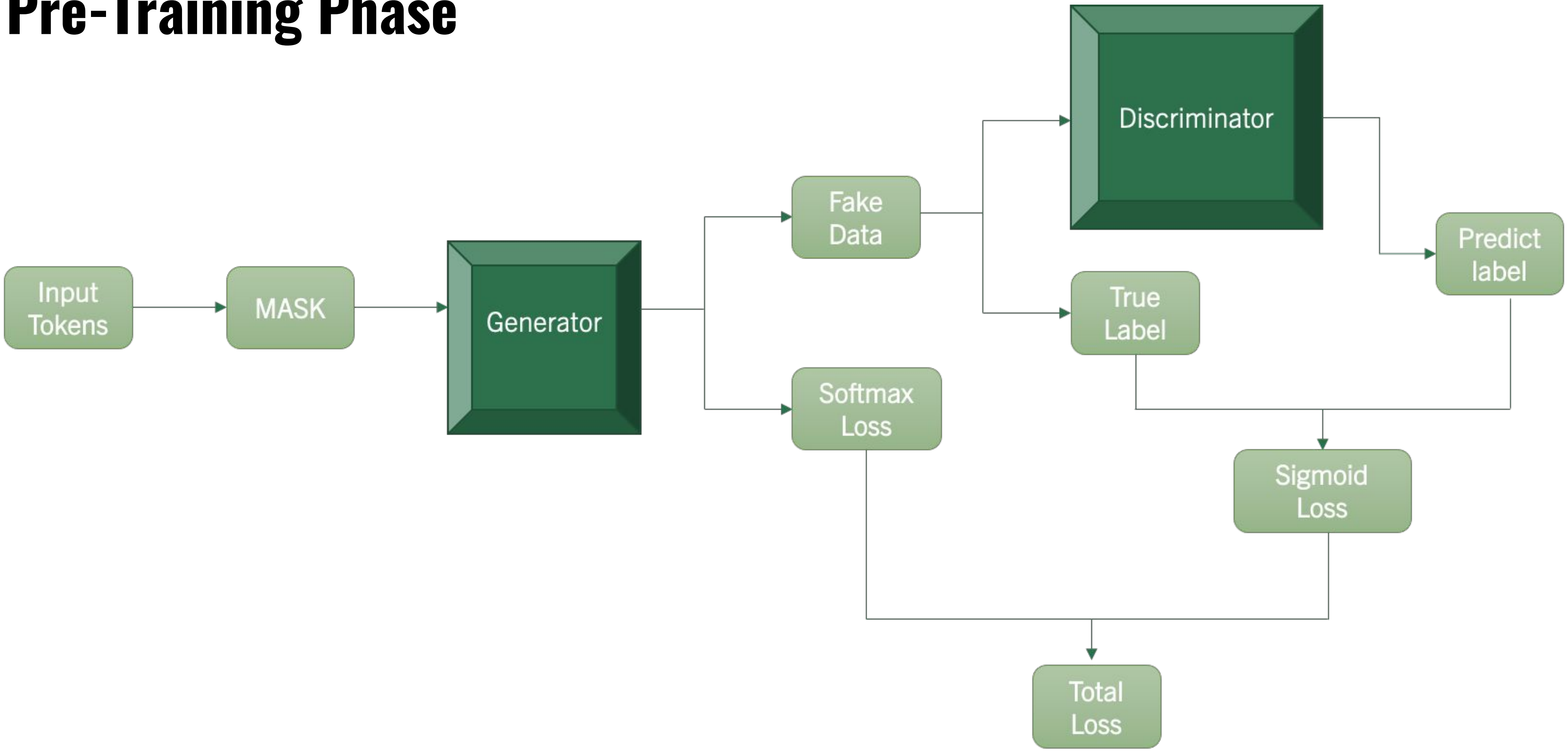
w^T = predicted output

Loss function:

$$\mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) = \mathbb{E}\left(\sum_{t=1}^{n} -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)$$

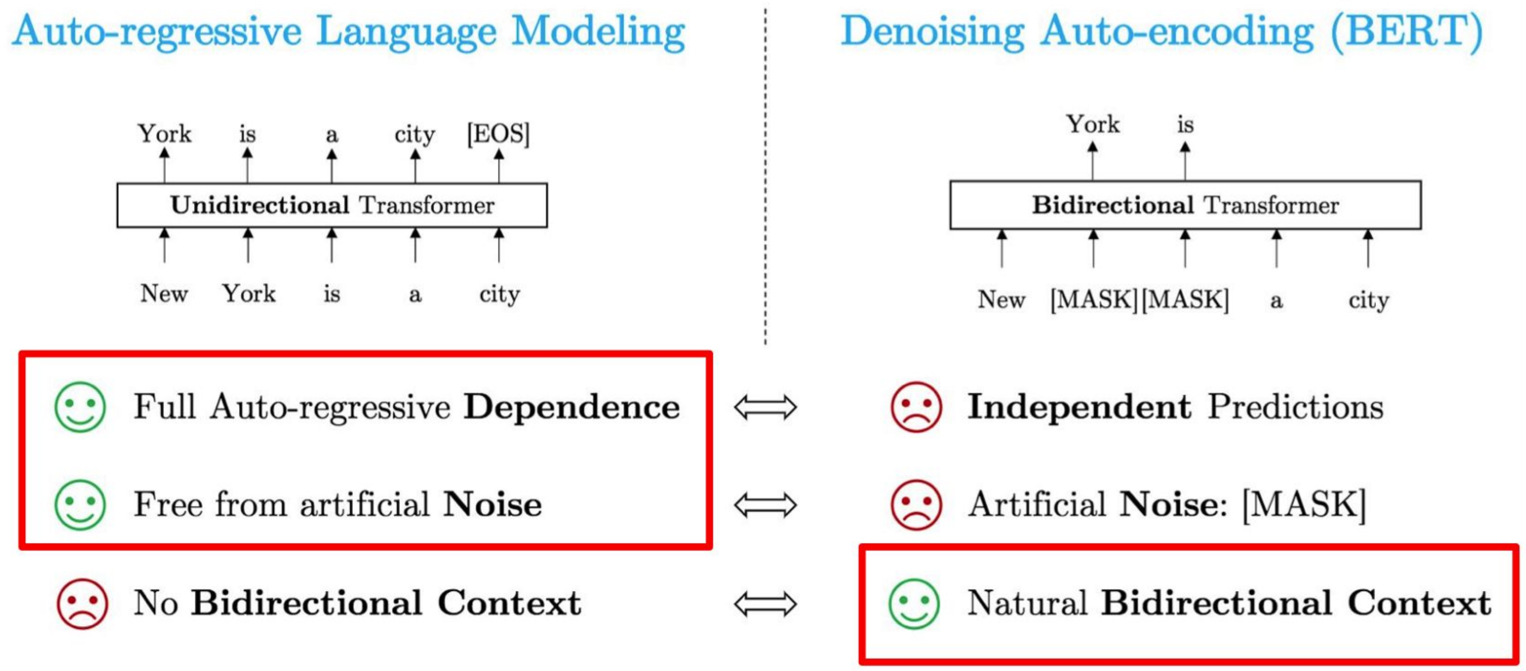
Combined Loss: $$\min_{\theta_G, \theta_D} \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) + \lambda \mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D)$$

Pre-Training Phase



Source : ELECTRA Code Interpretation Beyond BERT Model

XLNET



Source: XLNet Presentation, Princeton University, Andrew Or and Ksenia Sokolova

XLNET

General Autoregressive Model

- Bidirectional with permutation operation by maxing joint probability

Peter's ***cat*** *likes yarn*

Peter's cat likes yarn
Peter's cat yarn likes
Peter's likes cat yarn
yarn Peter's cat likes
yarn Peter's likes cat
yarn cat Peter's likes

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$

$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city}).$$

Source: XLNet Presentation, Princeton University, Andrew Or and Ksenia Sokolova

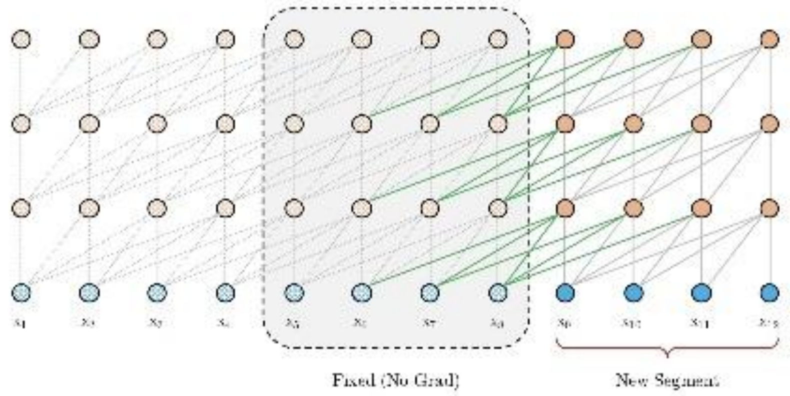
XLNET

General Autoregressive Model

- Bidirectional with permutation operation by maxing joint probability

Transformer-XL (SOTA LM) Integration

- Long-term dependencies via cache and reuse of previous hidden states



Google AI blog, Transformer-XL: Unleashing the Potential of Attention Models

XLNET

General Autoregressive Model

- Bidirectional with permutation operation by maxing joint probability

Transformer-XL (SOTA LM) Integration

- Long-term dependencies via cache and reuse of previous segment hidden states

Two-Stream Attention Mechanism

- Query: Keep positional encoding, blind to target
- Content: Gather context information (permutation)

$$g(z_t, \mathbf{x}_{\mathbf{z}_{<t}}) = \mathrm{Attn}_\theta\Big(\underbrace{\mathrm{Q} = \mathrm{Enc}(z_t)}_{\text{Stand at } z_t},\ \underbrace{\mathrm{KV} = \mathbf{h}(\mathbf{x}_{\mathbf{z}_{<t}})}_{\text{Gather info. from } \mathbf{x}_{\mathbf{z}_{<t}}}\Big)$$

DeBERTa

Decoding-**E**nhanced **BERT** with disentangled **A**ttention

DeBERTa

Decoding-**E**nhanced **BERT** with disentangled **A**ttention

- Disentangled Attention
  "I love **deep learning**." vs "We are **learning** about modeling techniques and **deep** neural nets."
  BERT: vector(position embedding + word embedding) -> Attention Mask
  DeBERTa: position vector, word vector-> Attention Mask
  Effect: Attention weights of words modulated by their relative positions

DeBERTa

Decoding-**E**nhanced **BERT** with disentangled **A**ttention

- Disentangled Attention

  "I love **deep learning**." vs "We are **learning** about modeling techniques and **deep** neural nets."
  BERT: vector(position embedding + word embedding) -> Attention Mask
  DeBERTa: position vector, word vector-> Attention Mask
  Effect: Attention weights of words modulated by their relative positions

- Enhanced Decoding Mask

  "A new **store** opened beside the new **mall**."
  BERT: -> Softmax layer to decode masked words
  DeBERTa: Absolute position embeddings -> Softmax layer
  Effect: Absolute position of words are taken into account

DeBERTa

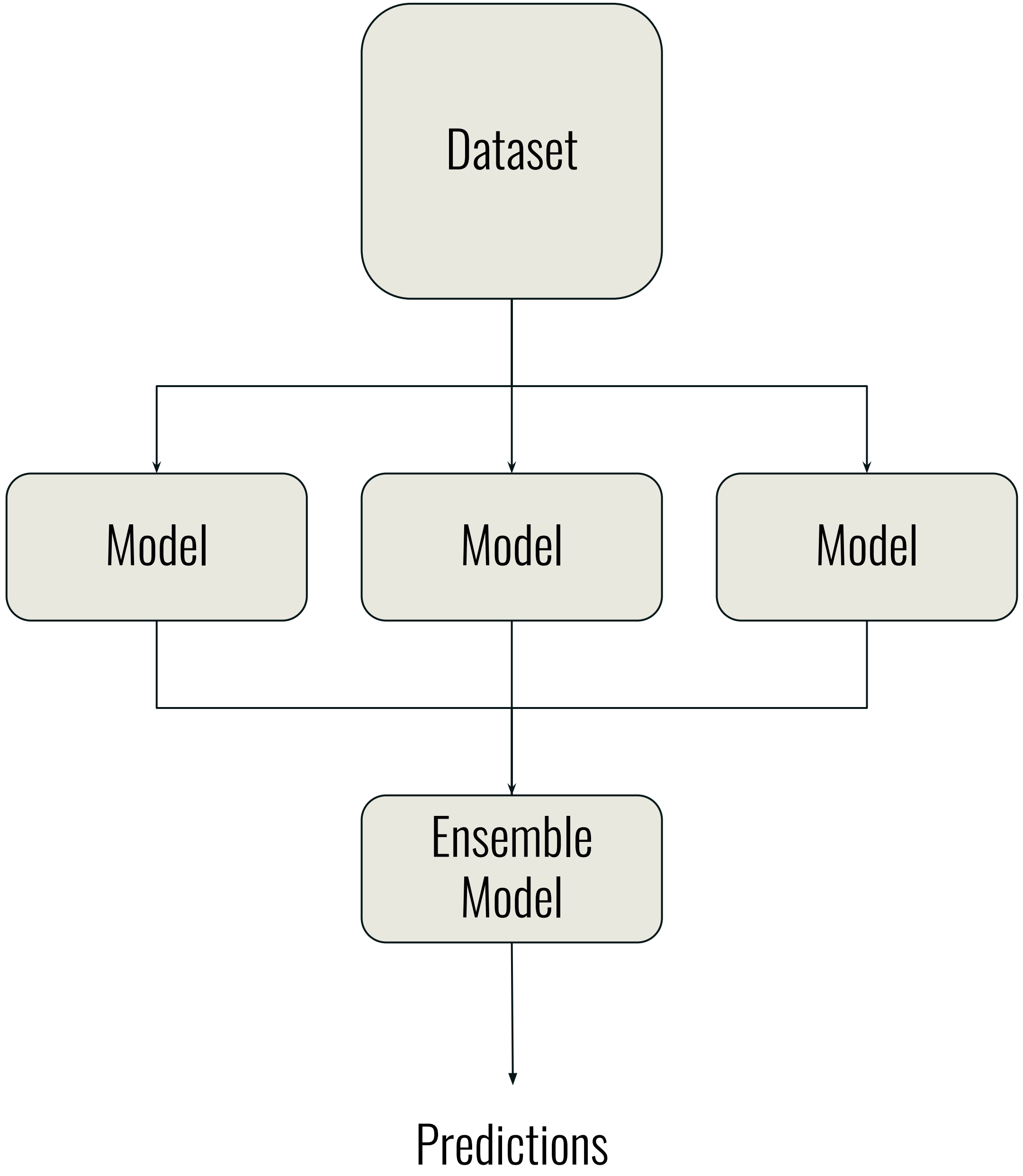
Decoding-**E**nhanced **BERT** with disentangled **A**ttention

- Full Size Deberta V3
  24 layers, hidden size of 1024
  304M backbone parameters
  Vocabulary of 128K tokens = 131M parameters

- Deberta V3-Small
  6 layers, hidden size of 768
  44M backbone parameters
  Vocabulary of 128K tokens = 98M parameters

Ensemble Model

- **Ensemble learning**
  - Meta approach to modeling
  - Leverages the power of multiple predictive models

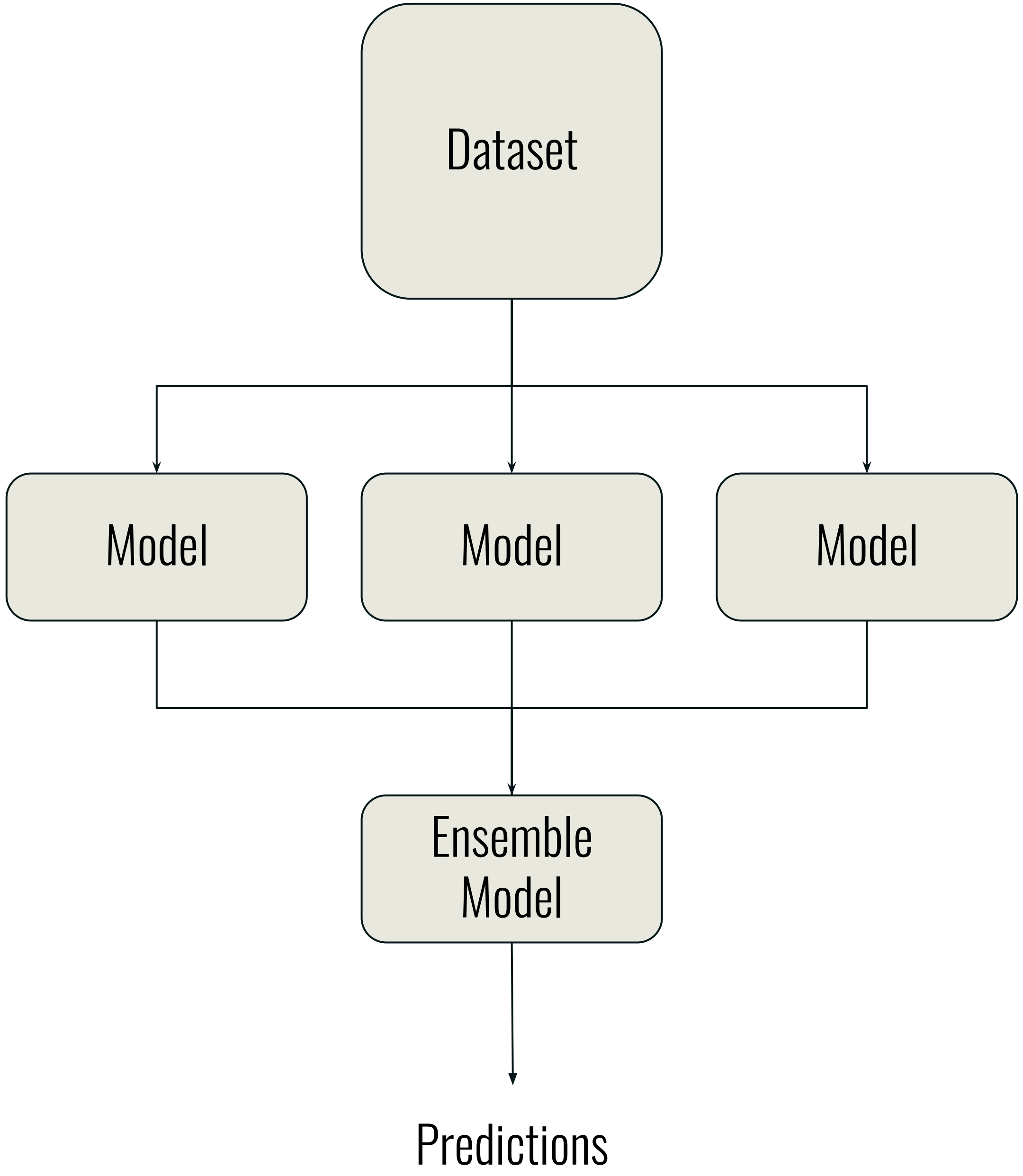


Ensemble Model

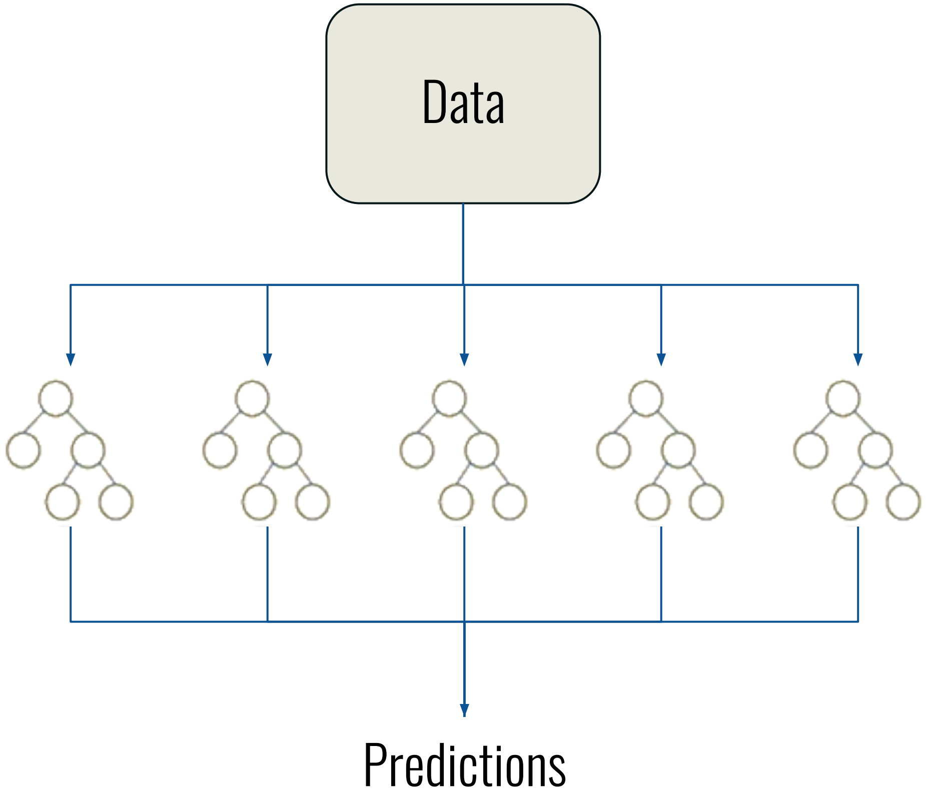
- **Ensemble learning**
  Meta approach to modeling
  Leverages the power of multiple predictive models

- **Random Forests**
  Build all trees simultaneously and independently

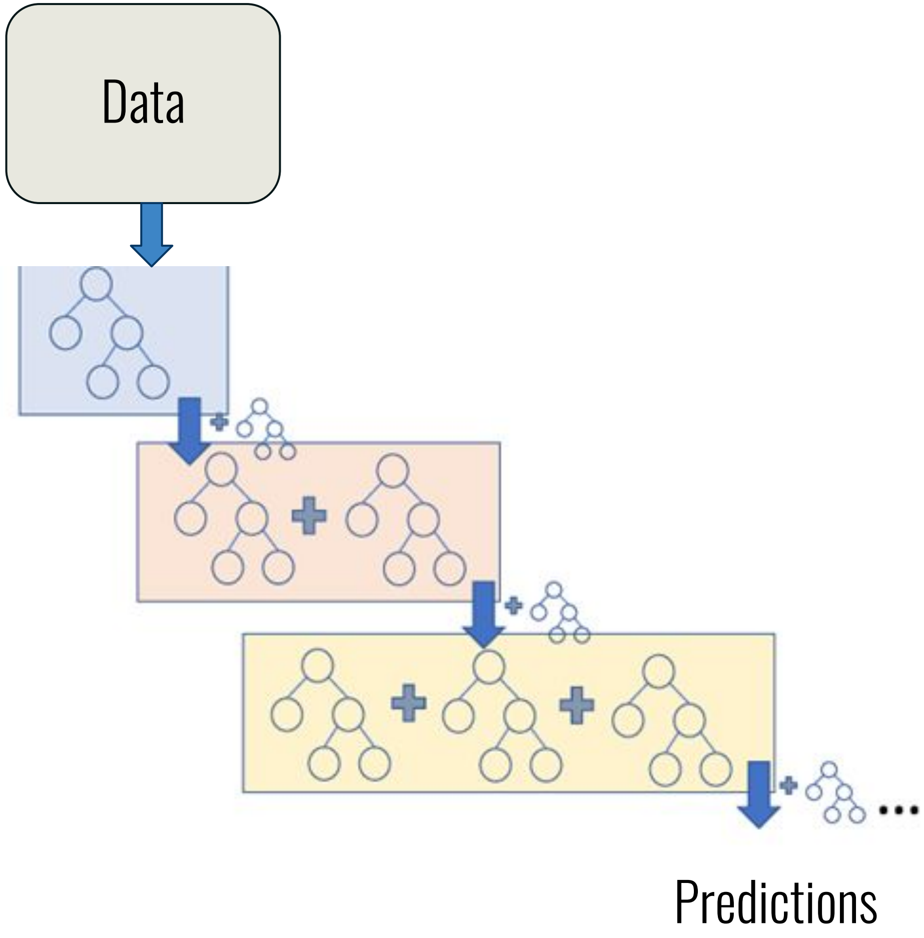
- **Gradient Boosting**
  Builds trees sequentially, with each tree being built in a chain.



Ensemble Model



Random Forest Model



Gradient Boosting Model

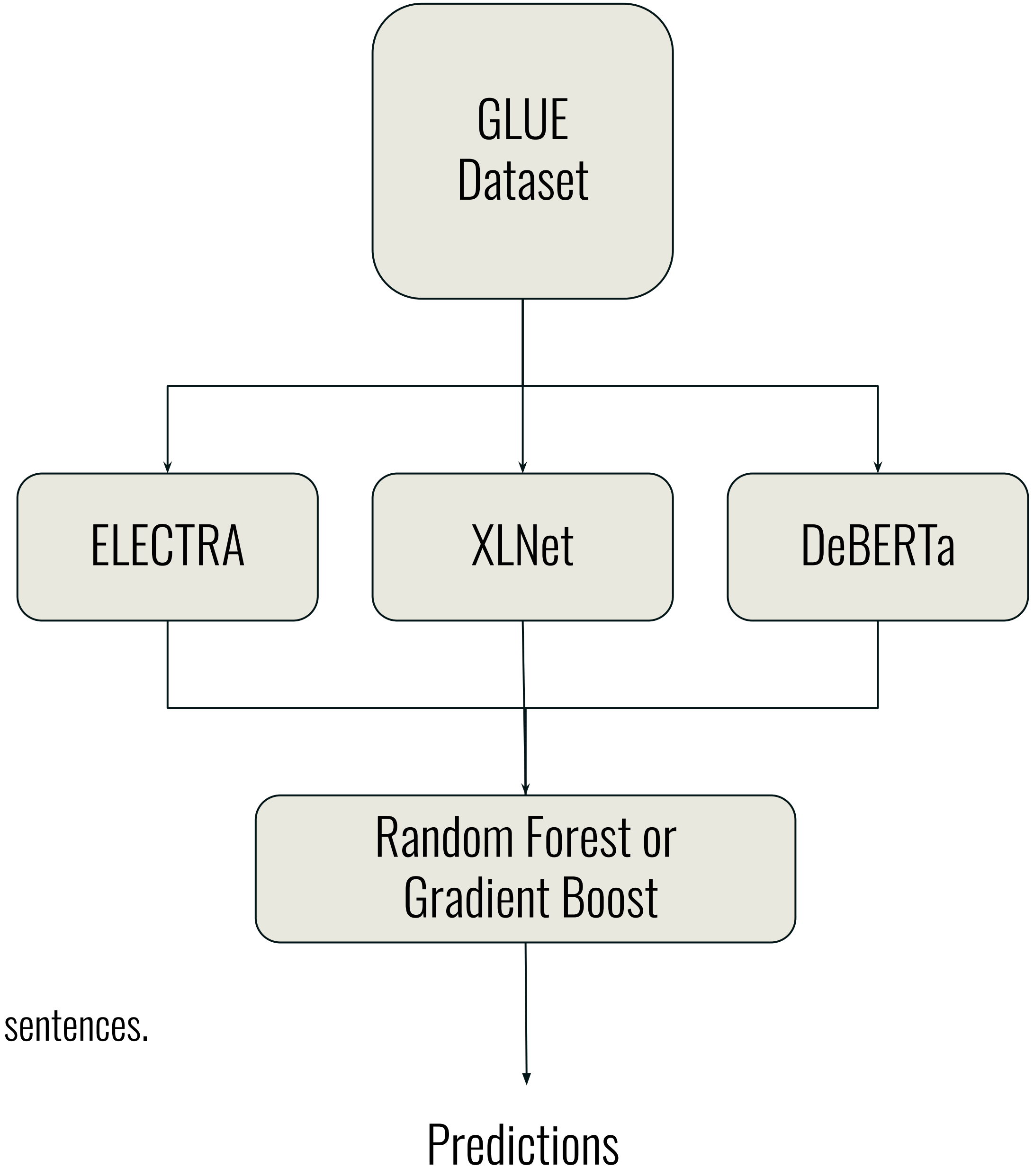
Ensemble Model

- **Ensemble learning**
  Meta approach to modeling
  Leverages the power of multiple predictive models

- **Random Forests**
  Build all trees simultaneously and independently (horizontally)

- **Gradient Boosting**
  Builds trees sequentially, with each tree being built in a chain.

- **Classifier and Regressor Models**
  STS-B has a continuous variable which measures the similarity of the two sentences.
  All other tasks are binary or class based



Experiment Results

- Ensembles were competitive against each other
- Gradient Boosting outperformed, except on MRPC and RTE
- XLNet's long-term dependencies strong suite may have helped

		Transformers			Ensemble Methods	
Corpus	**Metric**	**Electra**	**XLNet**	**Deberta**	**Random Forest**	**Grad Boost**
Single-Sentence Tasks						
CoLA	*Matthew's correlation*	0.607	0.400	0.621	**0.688**	0.657
SST-2	*Accuracy*	0.917	0.940	0.935	**0.958**	0.954
Similarity and Paraphrase Tasks						
MRPC	*Accuracy and F1*	0.882 f1:0.915	**0.892 f1:0.923**	0.865 f1:0.903	0.878 f1:0.912	0.854 f1:0.893
QQP	*Accuracy and F1*	0.900 f1:0.866	0.874 f1:0.833	0.906 f1:0.875	0.909 f1:0.878	**0.910 f1:0.879**
STS-B	*Pearson and Spearman*	P:0.873 S:0.872	P:0.893 S:0.889	P:0.868 S:0.868	P:0.894 S:0.889 *Note: Regressor	**P:0.897 S:0.892 *Note: Regressor**
Inference Tasks						
MNLI	*Accuracy*	0.817	0.857	0.875	**0.877**	**0.877**
MNLI-MM	*Accuracy*	0.821	0.858	0.872	0.875	**0.877**
QNLI	*Accuracy*	0.889	0.878	0.915	0.916	**0.924**
RTE	*Accuracy*	0.682	**0.740**	0.668	0.726	0.702
WNLI	*Accuracy*	0.465	0.563	0.437	0.500	**0.636**

Conclusions

Results + Analysis

- Ensembling works!
- Head-to-head: DeBERTa and XLNet
- Gradient-Booster Ensemble was our winner

Limitations & Future Work

- 12-Hour VPN limit
- Employing TPUs for handling computationally expensive models
- Add different models such as T5 or LUKE to Ensemble

References

GLUE Explained: Understanding BERT Through Benchmarks
Dataset description sheet
GLUE: A MULTI-TASK BENCHMARK AND ANALYSIS PLATFORM FOR NATURAL LANGUAGE UNDERSTANDING
DeBERTa Original Paper
Hugging Face Transformers Examples
ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.
Paper Reading #2: XLNet Explained
XLNet: Generalized Autoregressive Pretraining for Language Understanding
Guide to XLNet for Language Understanding
Data Science Central
XLNet Explained
XLNet: Generalized Autoregressive Pre-training for Language Understanding

ELECTRA SMALL

Built with 12 layers, 256 hidden size and 14M parameters

```
(electra): ElectraModel(
  (embeddings): ElectraEmbeddings(
    (word_embeddings): Embedding(30522, 128, padding_idx=0)
    (position_embeddings): Embedding(512, 128)
    (token_type_embeddings): Embedding(2, 128)
    (LayerNorm): LayerNorm((128,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (embeddings_project): Linear(in_features=128, out_features=256, bias=True)
  (encoder): ElectraEncoder(
    (layer): ModuleList(
      (0): ElectraLayer(
        (attention): ElectraAttention(
          (self): ElectraSelfAttention(
            (query): Linear(in_features=256, out_features=256, bias=True)
            (key): Linear(in_features=256, out_features=256, bias=True)
            (value): Linear(in_features=256, out_features=256, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): ElectraSelfOutput(
            (dense): Linear(in_features=256, out_features=256, bias=True)
            (LayerNorm): LayerNorm((256,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): ElectraIntermediate(
          (dense): Linear(in_features=256, out_features=1024, bias=True)
        )
        (output): ElectraOutput(
          (dense): Linear(in_features=1024, out_features=256, bias=True)
          (LayerNorm): LayerNorm((256,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
```

```
( vocab_size = 30522, embedding_size = 128, hidden_size = 256,
num_hidden_layers = 12, num_attention_heads = 4, intermediate_size =
1024, hidden_act = 'gelu', hidden_dropout_prob = 0.1,
attention_probs_dropout_prob = 0.1, max_position_embeddings = 512,
type_vocab_size = 2, initializer_range = 0.02, layer_norm_eps = 1e-
12, summary_type = 'first', summary_use_proj = True,
summary_activation = 'gelu', summary_last_dropout = 0.1, pad_token_id
= 0, position_embedding_type = 'absolute', use_cache = True,
classifier_dropout = None, **kwargs )
```

```
(classifier): ElectraClassificationHead(
  (dense): Linear(in_features=256, out_features=256, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (out_proj): Linear(in_features=256, out_features=2, bias=True)
)
```