**Foundational Research Notes**

Contents

[Machine Learning Basics 3](#_Toc187776070)

[Natural Language Processing 9](#_Toc187776071)

[BERT-Based Transfer-Learning Approach for Nested Named-Entity Recognition Using Joint Labeling 9](#_Toc187776072)

[Few Shot Learning for Sumerian Named Entity Recognition 11](#_Toc187776073)

[A Neural Pipeline for POS-tagging and Lemmatizing Cuneiform Languages 13](#_Toc187776074)

[Unsupervised Sumerian Personal Name Recognition 15](#_Toc187776075)

[Prompt-Based Engineering 17](#_Toc187776076)

[Plain Template Insertion: Korean-Prompt-Based Engineering for Few-Shot Learners 19](#_Toc187776077)

**Steps to Project**

**Planning – Begin Dec 23rd**

* Exploratory experiments and research within the field
* High level plan of how the system will look
* Establishment of roles within the team
* Establishment of platform we are using
  + **Just Lemmatization**
* Resource analysis:
  + Time requirements
  + Work Breakdown Structure

**Modeling - Begin**

* Low-level creation of the system’s model

**Execution - Begin**

* Construction of the model, evaluation, repeat. Done on training data.

**Testing – Begin Feb 15th**

* Testing of the model on test data

**Closing**

* Writing of technical report
* Writing of presentable report

# Machine Learning Basics

What is a Neural Network? A series of nodes that are set to mimic the human brain. In a high-level concept, a neuron is a single node within the brain. These neurons ingest information and produce information. They are linked to one another along different layers and each layer has a specific number of nodes. Information is passed into a node and that information is predicted by that node to indicate what that information is. Then, that information is passed to the next node(s). After many passes, the final nodes make a prediction as to what the given information is in the form of a percentage in confidence. The highest percentage is the highest confidence in that result. After the prediction, if the prediction was right, then the nodes that indicate confidence in the information being that prediction are adjusted to in the future, when receiving similar information, indicate strongly towards that prediction. The opposite happens if the prediction is wrong.

**Neuron**: A place that holds information.

* **Input Node**: The nodes responsible for holding a fraction of the information incoming from a source to be analyzed by a neural network.
* **Output Node**: A node that holds a prediction from the neural network. This prediction is in the form of a percentage and indicates how confident the system is that the information from the Input Nodes is a specific meaning. That meaning is tied to the output node. For instance, an Output Node associated with the letter “A” would the contain the confidence that information presented is the letter “A” if say coming from an image.
* **Hidden Node**: Holds a number and theoretically represents features of given information.

**Hidden Layers:**

Layers of hidden nodes between the output and input node layers. The intention with the hidden layers is to perform feature extraction. Typically, when humans analyze a subject, they break it down into components. The letter A is the letter A because of its shape and it’s shape isn’t the letter B because it contains different lines and angles. Hidden layers serve to allow relevant features to be extracted from incoming information and then decide what it may be. In theory, it goes something like “I’m looking at an image, I see a line downward and moving left connected to one going downward and right. I see a bridge in the middle. These features indicate that the image is the letter A” and the feature recognition phase is handled by the hidden layers.

**Weights**:

Each node holds a value. Each edge between nodes holds a weight. To compute the value of the next hidden layer’s node, multiply together a node’s value and the weight of the edge leading to that next hidden layer’s node. Then, repeat for all nodes leading to the next hidden layer’s node from the previous layer. The resulting value is the number for that node. Weights are adjusted such that when information is coming from a node, the weight pointing to a specific node can be adjusted to indicate that the information does not suggest lighting up that node and predicting its value and vice versa.

**Node Values**:

Node values are typically between 0 and 1. This indicates information that is there or a prediction. Values exceeding 1 or below 0 during calculation are then restrained to 0 and 1 through a sigmoid function or other common transformation function.

**Calculating the next layers of Neurons**

From the node value calculation, if you know the values of each nodes and the weights of each edge, then you can create 2 matrices and multiply them together. Given that there are X neurons and Y weights towards the next layer and the next layer’s size is Y, we want to create matrices that multiply together to create a matrix of size 1 x Y.

* Create a matrix ‘A’ of 1 x X containing the values of the previous layer’s nodes.
* Create a matrix ‘B’ of X x Y containing in each row the weights from the edges of the X nodes towards a node within Y. Then create a row for each output layer node.
* Multiple B x A to create the output layer of size 1 x Z

A screenshot of a computer

Description automatically generated

**Weights Continued**:

Based on the output nodes from the system, a cost can be calculated from the results. For the correct output, 1 is subtracted from the percentage and then squared. Similarly, the result in the other nodes has 0 subtracted from the value and then squared. The resulting number is the cost of the output and shows roughly how successful the model was. The lower the cost value, the better the performance (0 is best) and any other value greater than 0 signifies how poorly the system performed.

The cost function takes in the inputs of weights and biases from the system and outputs a single number. The goal then is to adjust the weights to minimize cost across all testing scenarios.

Based on the input weights, the goal is to calculate the gradient of those weights as a vector, or the direction to best adjust each weight to minimize cost, and then apply that to each weight. This is done through back propagation.

During this process, the cost function based on weights will adjust rapidly before settling with a gradient of 0 for the vectors (roughly). This is known as a local minimum, or a point where cost is minimized based on where the gradient started and moved towards. This is a minimization of cost and known as gradient decent, but it isn’t guaranteed to be the global minimum, or the minimum of all local minimums across all weights. Calculating the global minimum is difficult, but a local minimum is not.

**CRF Models:**

Conditional Random Fields. A class of statistical modeling methods applied to pattern recognition and machine learning used for structure prediction. In a Neural Network, a classifier predicts a label for information or a chunk of information without considering context. CRF takes into account context and is done so through a graphical model. The graphical model is as follows:

Consider X to be observations, classifiers, etc.

Consider Y to be random variables

Let G = (V, E) be a graph s.t. Y = (Y\_v) where v exists within V so that Y is indexed by the vertices of G.

Then (X, Y) is a conditional random field where each random variable Y\_v, conditioned on X, obeys the *Markov property* with respect to the graph; that is, its probability is dependent only on its neighbours in G.

P(Y\_v : X, {Y\_w : w != v} ) = P(Y\_v | X, {Y\_w : w ~ v}) where w~v means that w and v are neighbors in G. (The probability of Y\_v given X and given w != x is the probability of Y\_v given X where w and v are neighbors, connected by an edge).

**Markov Property:**

Refers to the memoryless property of a stochastic process (random process). This means that future evolution is independent of its history. The future state of the system depends only on the current state of the system, not past systems or past versions of that system.

**Cross Entropy**:

Based on the output prediction of the neural network as a percentage, the cross entropy for a given prediction is f(P) = -log(P)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A green and black text

Description automatically generated

A math equation with numbers and arrows

Description automatically generated with medium confidence

A graph of a function

Description automatically generated with medium confidence

A graph of a graph

Description automatically generated with medium confidence

Cross Entropy is a cost function in comparison to residuals. It is more lenient on good predictions, but the value of cost is much larger and explodes the worse the prediction becomes. Therefore, it is used as a cost function to quickly learn at first by making dramatic corrections while later making smaller adjustments.

# Natural Language Processing

## BERT-Based Transfer-Learning Approach for Nested Named-Entity Recognition Using Joint Labeling

* Ankit Agrawal , Sarsij Tripathi , Manu Vardhan, Vikas Sihag , Gaurav Choudhary and Nicola Dragoni

**Definitions:**

**Semantic Class**: Words of a similar type. Nouns, adjectives, verbs, etc.

**Named-entity Recognition**: The process of classifying words into their semantic class(es) based on the current context, usually in an effort to identify important words.

**Flat Entity Recognition**: Classifying a word only by 1 entity instead of multiple.

**Transfer Learning**: Training data for a model is unlabeled for the first part where the model then learns what each label it categorizes means, then named labels are introduced later.

**LSTM**: Stands for “Long Short-Term Memory” and refers to a type of recurrent neural network. It introduces memory cells and gating mechanisms to selectively retain and forget information over time. LSTMs have an internal memory state and can store information for long durations and capture patterns seen in frequently fed data.

**Bi-Directional LSTM:** Traditional RNNs process input sequences in only one direction. Bidirectional processing is the idea of passing data in both directions through an RNN. In this model, two LSTM layers are used, one processing information forward and the other backwards and both having their own hidden states and memory cells.

**Forward Pass:** Information is fed into the forward LSTM layer from first to last step. At each step, the LSTM computes its hidden state and updates its memory cell-based o the current input and the previous hidden state and memory cell.

**Backward Pass:** The input sequence is also fed backwards into the backward LSTM layer with its own hidden states and memory cells being updated.

**Combining Forward and Backward:** Once each model has completed its run of the input, the memory cells and hidden states are transformed and combined.

**Motivation:** The forward pass captures context before different the forward pass, but also maintains that context during the backward pass. The idea is that the resulting output over time will not diminish the context of the input information.

**Pros:** Best for natural language processing where context is always needed, better performance, high customization in transformation.

**Cons:** High computational cost (same time complexity, but essentially doubled for any input), requires large amounts of data compared to single LSTM.

**Research Objective**:

* Analyze effectiveness of BERT transfer-learning models in Natural Language Processing (application is DNA, RNA, protein recognition) in comparison to existing complex-architecture named-entity recognition problems.

Three common ways of solving nested named-entity recognition:

* **Layering**: Parts of a sentence will be classified under different named-entity components. Layering involves stacking the classification where they are seen. For example,
  + **“**John went to his house to eat breakfast”

Contains the phrase “his house” which “his” is an adjective, “house” a noun, “his house” being a noun.

* **Cascading**: Models dependencies sequentially.
  + “John went to his house to eat breakfast”

Used alongside CRF models, involves associating words with the context of the words next to it. Then, the tags are cascaded together based on prediction. This type of tag comes after this tag and so on. Cascading can be applied to a higher power by considering more tags afterwards or strings of tags, though this is computationally expensive.

## Few Shot Learning for Sumerian Named Entity Recognition

https://aclanthology.org/2022.deeplo-1.15/

**Research Objective**:

* Explore how to perform named entity recognition focusing on few-shot learning within a low resource language setting (few rules or knowledge on the language).
* Specifically, analyze Sumerian
* Named entity recognition systems are challenged when resources are low. Many large language models are trained on high resource languages and perform well in this area, but not as well when resources are low.
* The solution created explores prompt-based learning in large-language models to teach a model Sumerian in order to perform named entity recognition (NER)

**Contributions**:

* Construction of two few-shot learning systems, ProtoBERT and NNShot, and apply them on Sumerian NER task. Explores few-shot methodology as the field grows.
* Demonstrate performance of ProtoBERT in comparison to supervised BERT-based models and that the performance is greater in general.

**Challenges in the Field**

* With lack of annotated and labeled data, it is difficult to train models on low resource languages. One solution has been training a model on a high resource language before transferring knowledge learned to a low resource language application. Similar to Transfer Learning. However, the accuracy has only been observed at its highest during this method at 59.41% and only decreases after that.

**Method**:

* 3 methods were used and applied to the Sumerian NER task, BERT+LC, ProtoBERT, and NNShot.
* BERT+LC: A standard linear classifier built on top of the BERT model. Used as control group. Trained to minimize cross-entropy loss.
* ProtoBERT: Based on few shot learning models. This model aims to build an embedded space through training, so inputs are centered around the embedded space. Training is done in episodes where each episode is a few-shot test scenario with N classes and K samples. In this model, for each class C, its prototype P\_C is calculated by averaging the embeddings of examples that belong to class c in the support set S.
  + A black and white math symbols

    Description automatically generated with medium confidence

Where S\_C denotes the set of all elements in S that belong to the class C and the function F denotes the BERT architecture augmented with a linear classifier. F is updated after each episode in the training process by minimizing the cross-entropy loss between the probability calculated through softmax and the one-hot ground-truth label of x.

* NNShot: A few-shot learning method based on token-level nearest neighbor classification. ProtoBERT uses clustered training classes where NNShot does inference on a query example directly based on the nearest neighbor metric. It computes the distance score between example “X” in the query set and all examples in the support set, then assigns x the label of the example in the support set that is closest to x.

**Questions Remaining**:

* What does “Embedding” look like for few shot methods?
* What does an “Episode” look like?
* What is a prototype P\_C? I’m assuming this is implying that clusters are formed and P\_C represents the prototype of that cluster.

## A Neural Pipeline for POS-tagging and Lemmatizing Cuneiform Languages

https://aclanthology.org/2023.alp-1.23.pdf

**Research Objectives**

* The objective is to construct a model with the ability to lemmatize words as well as identify with Part-of-speech (POS) by tagging them with what their POS is. Traditional methods of lemmatizing often lack the POS capabilities as they are built on rule-based systems that cannot consider how a word appears in a sentence. Other models exist that can do this but are often inefficient as they can’t deal with spelling variations. Thus, the objective is to do both at the same time.

**Contributions**

* An OpenNMT-based (Open Neural Machine Translation Toolkit) neural lemmatizer and POS-tagger is presented as a solution for the objective and is tested on Akkadian. It learns many-to-many relationships between spellings of word forms and their lemma in context while also using the mappings to predict annotations for previously unseen word forms and unseen spellings.

**Notes on Akkadian**

* Akkadian’s morphology has linear prefixation and suffixation
  + Linear implies a direct attachment of the prefix or suffix to the front or end of a word.
* Has nonlinear root-patterns and infixation
  + A component will appear in the middle of a word to imply a different meaning, or roots will change to show the same word under a different context.
* Akkadian appears in 2400bc – 100ad.
  + The Assyrians had their own dialect, split into:
    - Old Assyrian (1950–1500bc)
    - Middle Assyrian (1500–1000bc)
    - Neo-Assyrian (1000 – 612bc)
  + The Babylonians had their own dialect, split into:
    - Old Babylonian (2000-1500bc)
    - Middle Babylonian (1500-1000bc)
    - Neo-Babylonian (1000-626bc)
    - Late Babylonian (626bc-100ad)
    - Standard Babylonian (n/a, but based on Old Babylonian with residue from contemporary spoken Babylonian and Assyrian dialects depending on when it was written)

**Thoughts**

* The system works on a BiLSTM with OpenNMT. It appears that using such a system (which maintains context throughout analysis of information ingested) yields successful results and results that can be applied broadly to other languages as well. Latin and Greek were used as examples of languages which were treated as low-resource but still successful in lemmatizing and tagging.

**Questions Remaining**

* What is OpenNMT?
* Is PyTorch the dominant environment?

## Unsupervised Sumerian Personal Name Recognition

<https://cdn.aaai.org/ocs/10406/10406-46104-1-PB.pdf>

**Research Objectives**

* In the Sumerian Empire, many financial records remain that dictate its history. Such transactions include exchange of goods, animals, and more. Records also include different names of individuals and theoretically, tablets from the same era could indicate not only who existed, but what social network they inhabited. The objective of this project is then to create a model that can extract personal names from tablets or recognize names from Sumerian texts. This is done through an unsupervised AI model.

**Challenges in the Field**

* Sumerian is low resource
* Tablets containing Sumerian are often damaged
* Not enough scholars who know Sumerian to process and understand the plethora of tablets (90k published, 10k+ unpublished, more being uncovered)

**Notes within the Field**

* Data of tablets is often in UTF-8 or similar encoding, making processing convenient and without having to be done with image processing necessarily
* Names are often single words, no last names

**Contributions**

* The NER system developed has 3 components, pre-processing, the decision list, and post-processing.
  + **Pre-processing**: Involves removing characters that are damaged or modified. “Best Guesses,” damaged, or corrected characters are removed as they may represent noise in the system. 13 tags are then assigned to different words under {GN, FN, TN, WN, MN, n, TO-TLE, UNIT, GOODS, OCCUPATION, YEAR, MONTH, DAY} representing Geographical Names, Field Names, Temple Names, Watercourse Names, Month Names, Numbers, Title Names, Unit Names, Trade Good Names, Occupation Names, and indicators for Year, Month, and Day.
  + **Decision list**: Built over the DL-CoTrain model and uses context and spelling rules to create a decision list. Context is analyzed as the left and right of a word to give the word its context. Spelling rules specify the spelling of a named-entity.
    - Certain rules were seeded into the system to strongly indicate named-entity. These include giri3 which indicates the person was an intermediary, kiszib3 which indicates the person sealed the tablets, and mu-DU which indicates that a delivery was made by the person.

The decision list is applied to the labels from pre-processing along with the certain rules used to gain new spelling rules and contextual rules. Strength of a contextual rule is derived from its total appearance and its appearance alongside certain words.

* + **Post-Processing**: Two additional rules are added, a word that starts with a number isn’t a name and a word following the word iti, the month indicator, is not a name.
* The main idea is a combination of implicit and learned rules from training with some correction explicit rules

**Questions Remaining**

* I understand the high-level concept. I would love to see the code that makes this work.

# Prompt-Based Engineering

**Prompt Tuning and the Embedding Layer**

https://www.youtube.com/watch?v=wgfSDrqYMJ4

https://www.youtube.com/watch?v=yu27PWzJI\_Y

* **Word Embedding**: Words are represented as a map where words are placed on a map in relation to each other. Words have a Vector associated with them indicating their classification and words that have similar classifications naturally form clusters near each other. This process is known as word embedding.
  + Used in text classification, NER (Named Entity Recognition) (Entity implying proper noun), Word similarity between each other, Question and Answering systems
  + Involves pre-processing the text, removing noise and making small corrections. Then involves learning relationships based on how they appear (processing). Finally, post-processing as needed.
  + Can be done by Frequency-Based embedding (based on how frequently a word appears and where it appears. Associates words with similar words).
  + Can be done by Prediction-Based embeddings (captures semantic and context of a word to embed a word. Associates words with what words come near it.)
* **Prompt-Engineering/Prompt Based Learning**: The process of creating a prompt to train an LLM to perform a certain task.
  + Ex. Translate English to French Process:
    - Tell the model the objective
    - Give the model some translation examples.
    - Test
  + Works well for common tasks, but the complexity of the task can lead to difficulty in prompt engineering.
  + **Soft Prompts**: Prompts created by AI for prompt engineering a specific task. Difficult to interpret.
  + **Hard Prompts**: Prompts created by humans for prompt engineering a specific task.

**Foundational Models**:

A type of neural network that is generalizable, pretrained, adaptable, large, and self-supervised. They are trained on general tasks or in a general field but can then be trained to a specific application as needed. Large Language Models are a subset of Foundation Models.

A diagram of a foundation model

Description automatically generated

**Prompt Engineering Tools:**

* Prompt Runner: <https://marketplace.visualstudio.com/items?itemName=JinShang.prompt-runner&ssr=false#overview>
  + Install into VSCode
  + Allows prompts to be built and then ran against an AI of choice. Includes ChatGPT

**Prompt Engineering Learning Course:**

<https://github.com/microsoft/generative-ai-for-beginners/tree/main/01-introduction-to-genai>

<https://learn.microsoft.com/en-us/shows/generative-ai-for-beginners/understanding-prompt-engineering-fundamentals-generative-ai-for-beginners?WT.mc_id=academic-105485-koreyst>

* LLMs are Stochastic, they give different responses based on the same input. This is an issue when thinking about how to use a model in application where we want consistency. Prompt-Engineering is a solution to this as it helps remove stochasticity.
* Typically, LLMs have both a space for prompts as well as a system personification. The system personification is a set of rules that are given to an LLM before prompts are given and affects how the model responds to prompts.
* Prompts can also create rules by giving the model continuous context. For example, “What is Gallium” will cause an LLM to explain the element Gallium. Another prompt after “What comes after it” will cause the LLM to explain the element after Gallium, Germanium, even though “it” was used in the prompt instead of “Gallium.” Context is preserved through continuous prompts, a prompt history.

## Plain Template Insertion: Korean-Prompt-Based Engineering for Few-Shot Learners

https://ieeexplore.ieee.org/document/9913979

**Research Objectives:**

* Prompt-based learning is a new approach to learning in AI models. Few-shot learners achieve better performance because they alleviate catastrophic forgetting in models.
* Few-shot learning contributes towards solving the data scarcity problem, the idea that models need large amounts of data for training, but such data isn’t always available or is difficult to produce.

**Contribution to knowledge:**

* Plain template insertion (PTI) is method of prohibiting changing the number of few shot training samples and places predefined templates containing prompts and a [MASK] token into a specific position without refining the input sample. It places the template considering minimal Korean contextual information for the given question and the connection between sentences. It can be integrated with hard or soft prompt tuning.
  + Template content: The template consists of a [MASK] token and prompts combined to determine the content. An example of a template that identifies the relationship between two input sentences could be something like “The two sentences are [MASK] related.”
  + Template position: The template is inserted at a specific position given an input sequence. The position is either before or after the input sequence. For example, if the input sequence is <X> and the template is <T\_d>, the order in which they appear could be <X><T\_d> or <T\_d><X>.
  + Mapping label: Part of the template and represents the answer word to be predicted in the task. Prompt-based learning infers the answer in the same way as masked language modeling pretrained with self-supervised learning, which prevents catastrophic forgetting.
* It is argued that manipulation and exploitation of high-quality data for training obscures the main purpose of few-shot learning. It is necessary to reconsider whether a scheme adequately alleviates the data scarcity problem.

**Preliminary**:

* **Catastrophic Forgetting**: The idea that models performance decreases over time. Catastrophic forgetting specifically defines a model forgetting previously learned information as the model is trained on new data or fine-tuned for specificity. This can be chalked up to prioritizing recent data over earlier data.
* **Hard-Prompt**: A hard prompt is human interpretable and serves as training for a large language model. Template is written in natural language in order to be human interpretable. We call this template T\_d, a template in discrete space and it is fed into the model along with input sequence X.
  + T\_d = d\_0:m, d\_m, d\_m+1:n

Where each d\_i represents natural language tokens, including the [MASK] token. d\_m specifically represents the max token. There are “n” tokens total with d\_0:m representing tokens before m and d\_m+1:n representing tokens after m. The model is trained to restore the mth position of T\_d to the mapped label word W(y) based on linguistic knowledge acquired during pretraining.

The hard prompt method can then be described as follows when calculating for W(y):

For each W(y) that could be placed in y\_mask, the probability is that y\_mask = W(y) given input sequence X and template T\_d. Then, the model chooses the highest probability W(y).

A black text with a white background

Description automatically generated

* **Soft-Prompt**: Consists of human-uninterpretable template T\_c = c\_0:n, c\_m. That is composed of trainable embedding vectors in a continuous latent space and a mask token c\_m = [MASK]. For implementation, prompt template token c\_0:n and distinct prompt embedding e’() are initialized. This enables the adaptation of the template to fit the optimal template, which may not be grasped by discrete natural language tokens. e() represents the original token embeddings of the pretrained language model. This leads to the equation of:

For each W(y) that could be placed in y\_mask, the probability that y\_mask = W(y) given the embedding of input sequence X, the distinct embedding of tokens c\_0:n, and the embedding of c\_m. Then the model chooses the highest probability W(y).A black text with black letters

Description automatically generated with medium confidence