**Dataset**

* Dataset contains 575k Meme Images and Caption from 100 most popular memes found on ImgFlip.com JSON format
* Obtained by web scraping by Scrapy (Extraction Date: ~April 2020)
* 100 Most Popular Memes: <https://imgflip.com/popular_meme_ids>
* Dataset Link: <https://github.com/schesa/ImgFlip575K_Dataset>

**Dataset Preprocessing**

* Take note of Ideal Boxes for each meme
* Filter out memes with same length ideal boxes
* Filter out memes with unprintable characters (non-english memes, emojis, etc.) and lowercase
* Limit to memes with 200 characters or less
* Result: ~310k memes
* Subsample dataset 1000 data points per meme (Result: ~88k memes)
* Encode Characters to indexes + Encode Images to indexes
* Create Dataset (Model1):
  + Example: <start>apple should make a big screen tv <sep> and call it the big mac<end>
  + Dataset Created:

|  |  |  |
| --- | --- | --- |
| **Input** | **Label** | **Meme Image** |
| “<start>” | a | 10-Guy (0 in img2idx) |
| “<start>a” | p | 10-Guy (0 in img2idx) |
| “<start>ap” | p | 10-Guy (0 in img2idx) |
| “<start>app” | l | 10-Guy (0 in img2idx) |
| “<start>appl” | e | 10-Guy (0 in img2idx) |
| “<start>apple” | space | 10-Guy (0 in img2idx) |
| “<start>apple “ | s | 10-Guy (0 in img2idx) |
| “<start>apple s” | h | 10-Guy (0 in img2idx) |

And continue

* + Need to pre-pad the input to have same length (Chose: 128)
    - Example: [<pad>, <pad>, <pad>, ….. , <start>] for the first data input
  + 95% data for Training 5% for Validation/Test
  + ~5M training examples, 260k validation examples
* Create Dataset (Model2):
  + Example: <start>apple should make a big screen tv <sep> and call it the big mac<end>
  + Dataset Created:

|  |  |  |
| --- | --- | --- |
| **Input** | **Label** | **Meme Image** |
| “<start>apple should make a big screen …..big mac” | “apple should make a big screen ….. big mac<end> | 10-Guy (0 in img2idx) |

And continue

(<start> -> a), (a -> p), (p -> p), …..

* + Input is one character (text) -> LSTM will generate one character at a time
  + Input array and Label array should have same length!!!
  + <END> token is not included in input
  + One Meme -> One training example row (NOT LIKE PREVIOUS ONE)
  + All characters are post-padded until they have length of 199
  + 95% data for Training 5% for validation/test
  + ~83k training examples, ~4.5k validation examples
* Save into NPZ file

**Modelling (Model1 CNN)**

* Why Character Embeddings?
  + Meme has lots of non-standard words and spelling mistakes
  + Model is significantly smaller compared to word embeddings (using GLOVE + others)
* BATCH\_SIZE = 256
* 72 characters (including <pad>, <start>, <end>, <sep> tokens), 99 Meme Images
* Model
  + **Input:** img\_num, input\_sequence
    - Img\_num = Image # from img2idx dictionary mapping
      * size = (batch\_size, 1)
    - Input\_sequence = Input sequence for next character prediction
      * size = (batch\_size, 128) -> 128 is predefined seqlen in prev. section (prepadded)
  + **Output:** Logits of the characters for next prediction -> Apply softmax to get probabilities
    - Size = (batch\_size, 72) -> 72 is number of characters/classes possible
* Loss
  + Cross Entropy Loss = Log Softmax + NLL loss
  + Cross Entropy (yhat, y)
* Metrics = Accuracy (% of correct predictions)

**Modelling (Model2 LSTM)**

* Why Character Embeddings?
  + Meme has lots of non-standard words and spelling mistakes
  + Model is significantly smaller compared to word embeddings (using GLOVE + others)
* BATCH\_SIZE = 32
* 72 characters (including <pad>, <start>, <end>, <sep> tokens), 99 Meme Images
* Model
  + **Input:** img\_num, input\_sequence, labels, prev\_hidden\_state\_h, prev\_hidden\_state\_c
    - Img\_num = Image # from img2idx dictionary mapping
      * size = (batch\_size, 1)
    - Input\_sequence = Input sequence for next character prediction
      * size = (batch\_size, 199) -> 199 is predefined seqlen in prev. section (postpadded)
    - labels = Label sequence for the current position
      * size = (batch\_size, 199) -> 199 is predefined seqlen in prev. section (post padded)
      * last element is <end> token
    - prev\_hidden\_state\_h, prev\_hidden\_state\_c = previous hidden states of LSTM, initialize to 0.
  + **Output:** Logits of the characters for next prediction -> Apply softmax to get probabilities
    - Size = (batch\_size, 199, 72) -> 199 is the seqlen, 72 is # of characters
* **Notes:**
  + In Training + Validation feed actual labels in each time step
  + For generating predictions, use previous predicted character as input to next timestep
* Loss
  + Cross Entropy Loss = Log Softmax + NLL loss
  + Cross Entropy (yhat, y)
* Metrics = Accuracy (% of correct predictions)

**Model1 Architecture CNN (Pytorch Implementation)**

**Input:** (batch\_size)

**Input:** (batch\_size, 128)

Convert to (batch\_size, 128) use repeat

**Image Embedding Dim:** 8

**Sequence Embedding Dim:** 16

Sequence Embedding

Image Embedding

(batch\_size, 128, 24)

(batch\_size, 128, 8)

(batch\_size, 128, 16)

Concatenate on dim=2

# channels = 16, seqlen = 128

**Projects from 24 -> 16 channels**

Project down (FC layer)

(batch\_size, 1024, 128)

**Kernel:** 5, **Padding:** 2, **Stride:** 1

(batch\_size, 128, 16) -> Swap to (batch\_size, 16, 128)

1D Conv1 + ReLU1

(batch\_size, 1024, 128)

BatchNorm1

Block1

(batch\_size, 1024, 64)

**Kernel:** 2, **Stride:** 2

MaxPool1

(batch\_size, 1024, 64)

Dropout1 (p=0.25)

(batch\_size, 1024, 64) -> 32, 16

1D Conv(2,3,4) + ReLU(2,3,4)

(batch\_size, 1024, 64) -> 32, 16

BatchNorm(2,3,4)

(batch\_size, 1024, 32) -> 16, 8

MaxPool(2,3,4)

Block 2,3,4

Dropout(2,3,4) p=0.25

(batch\_size, 1024, 8)

Dropout5 (p=0.25)

(batch\_size, 1024, 8)

**Max from 8 -> 1 channel**

Global Max Pooling

**Projects from 1024 -> 1024 channels**

(batch\_size, 1024, 1) -> (batch\_size, 1024)

(batch\_size, 1024)

FC layer + ReLU

**Output:** (batch\_size, # Classes)

(batch\_size, 1024)

BatchNorm6

(batch\_size, 1024)

FC Layer + Softmax

Dropout6 (p=0.25)

**Convolution 1D (Conv1 block)**

\*

L = 5, H = 16

L = 128, H = 16, W = Batch

L = 128, H = 1024, W = Batch

# Filters = 1024

* 128 = sequence length, 16 = # channels
* Kernel size = 5, # Filters = 1024

**Model2 Architecture LSTM (Pytorch Implementation)**

Unsqueeze to get (batch\_size, 1)

**Input:** (batch\_size)

**Input:** (batch\_size, 1)

**Sequence Embedding Dim:** 128

Sequence Embedding

**Image Embedding Dim:** 32

Image Embedding

(batch\_size, 1, 128)

(batch\_size, 1, 32)

Concatenate on dim=2

(1, batch\_size, lstm\_hidden\_size)

(1, batch\_size, lstm\_hidden\_size)

(1, batch\_size, lstm\_hidden\_size)

(1, batch\_size, lstm\_hidden\_size)

**Next\_hidden\_state\_c**

**Next\_hidden\_state\_h**

**Prev\_hidden\_state\_c**

**Prev\_hidden\_state\_h**

LSTM

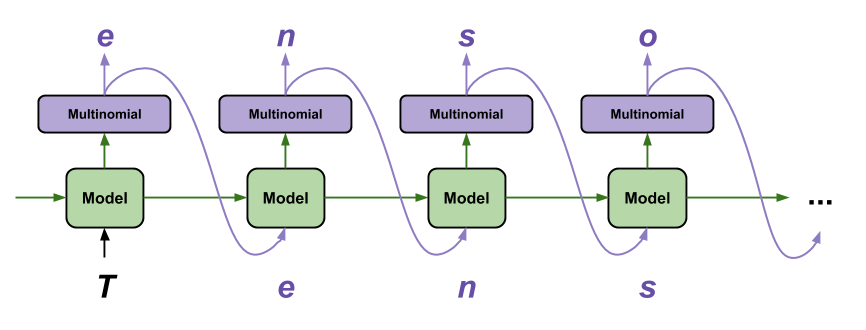
(Hidden\_size = 1024)

(batch\_size, 1, 128)

**Projects from 1024 -> 72**

FC (Dense)

**Output:** (batch\_size, # Classes)

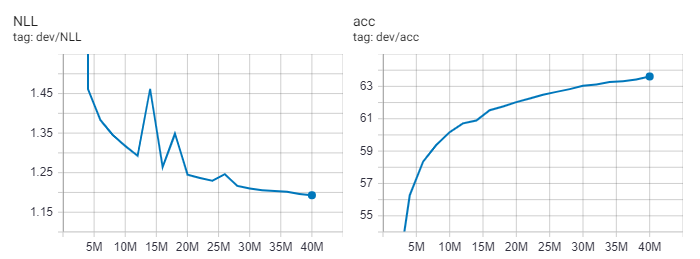
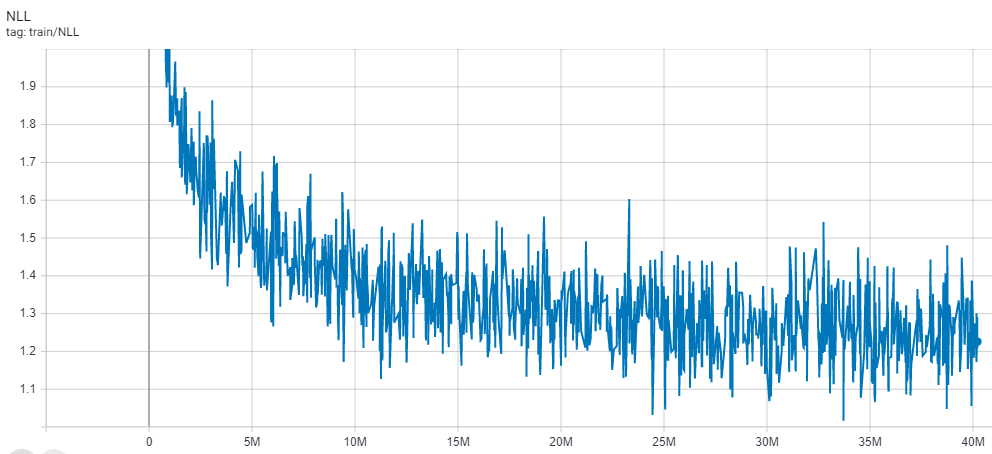


* Training + Validation -> Use Actual Labels from the dataset
* Generating Predictions -> Use predicted labels (as depicted in picture)
* Prev\_hidden\_state\_h and prev\_hidden\_state\_c is initialized to zeros for the initial state.

**Prediction Algorithm**

* Use <start> token first + any other characters supplied
* Feed into Network and obtain output (one next char) -> Feed into network again until <end> is generated
* Methodology:
  + Greedy
    - At each time step pick output with highest probability
  + Sampling
    - At each time step sample randomly based on the probability
  + Beam Search (k = beam width in this case = 3)
    - At each time step take k best output
    - Algorithm (modified slightly):
      * First generate k predictions with scores
      * Take each prediction and generate k more predictions at each time step from the previous output until we reach k end candidates. Score will be current score P(Y|X) \* prev score P(X) = P(X,Y)
      * Iterate until we reach k end candidates
      * Take output (with end token) that has highest score.

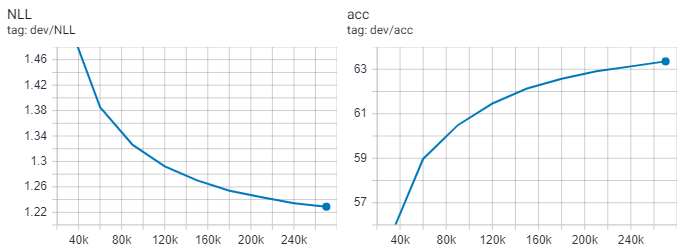
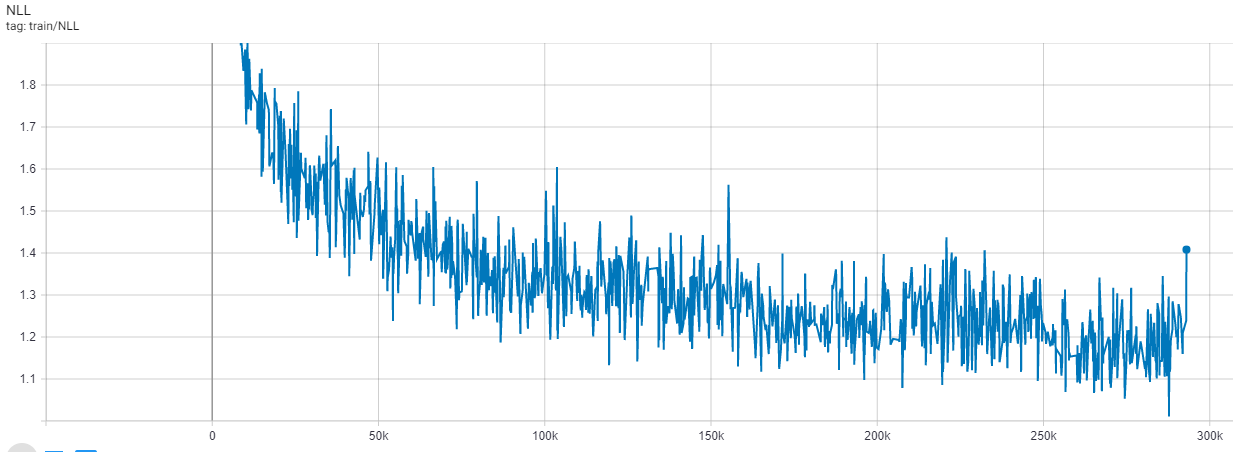
**Results Model1 CNN**

* Train for 8 hours (P100 GPU) -> 8 epochs -> ~5M examples per epoch
* Eval Dataset = ~265k examples (evaluate every 2M examples)
* Metrics = Loss + Accuracy
* Optimal Parameters:
  + BATCH\_SIZE = 256
  + Seq\_len = 128 (So that it can be nicely divisible by 2^n)
  + Learning Rate = 0.001
* Validation Loss + Validation Accuracy
* Train Loss
* Output Table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Step** | **Val Loss** | **Val Acc** |  | **Step** | **Val Loss** | **Val Acc** |  | **Step** | **Val Loss** | **Val Acc** |
| 2M | Large | 50.65% |  | 16M | 1.26 | 61.53% |  | 30M | 1.21 | 63.03% |
| 4M | 1.46 | 56.27% |  | 18M | 1.35 | 61.75% |  | 32M | 1.21 | 63.12% |
| 6M | 1.38 | 58.34% |  | 20M | 1.24 | 62.02% |  | 34M | 1.20 | 63.26% |
| 8M | 1.35 | 59.38% |  | 22M | 1.24 | 62.26% |  | 36M | 1.20 | 63.31% |
| 10M | 1.32 | 60.16% |  | 24M | 1.23 | 62.48% |  | 38M | 1.20 | 63.42% |
| 12M | 1.29 | 60.71% |  | 26M | 1.25 | 62.66% |  | 40M | 1.19 | 63.61% |
| 14M | 1.46 | 60.89% |  | 28M | 1.22 | 62.83% |  |  |  |  |

* **Best Output:** 40M Step -> 63.61% Val Accuracy

**Result Model2 LSTM**

* Train for 8 hours (T4 GPU) -> 4 epochs -> ~85k examples per epoch
* Eval Dataset = ~4.55k examples (evaluate every 30k examples)
* Metrics = Loss + Accuracy
* Optimal Parameters:
  + BATCH\_SIZE = 32
  + Seq\_len = 199
  + LSTM Layer = 1
  + LSTM Bidirectional = False (Only use Unidirectional LSTM)
  + LSTM Hidden Size = 1024
  + Learning Rate = 0.001
* Validation Loss + Validation Accuracy
* Train Loss
* Output Table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Step** | **Val Loss** | **Val Acc** |  | **Step** | **Val Loss** | **Val Acc** |  | **Step** | **Val Loss** | **Val Acc** |
| 30k | 1.52 | 55.28% |  | 120k | 1.29 | 61.46% |  | 210k | 1.24 | 62.91% |
| 60k | 1.39 | 58.97% |  | 150k | 1.27 | 62.14% |  | 240k | 1.23 | 63.13% |
| 90k | 1.33 | 60.49% |  | 180k | 1.25 | 62.57% |  | 270k | 1.20 | 63.35% |

* **Best Output:** 270k steps -> 63.35% validation accuracy

**Prediction Examples Model1 (See Tensorboard for more)**

* Example1: (probably predicting execute)
  + **Source:** “<start>exec”
  + **Label:** u
  + **Prediction:** u
  + **Img Name:** Trump-Bill-Signing
* Example2: (wrong prediction probably correct word is season)
  + **Source:** “<start>what if the bus is going to a place called<sep>"not in se”
  + **Label:** r
  + **Prediction:** a
  + **Img Name:** Conspiracy-Keanu
* Example3: (probably predicting really)
  + **Source:** “<start>when you commanded the ensign to scrub the poop deck<sep>not r”
  + **Label:** e
  + **Prediction:** e
  + **Img Name:** Captain-Picard-Facepalm

**Prediction Examples Model2 (See Tensorboard for more)**

* Example1:
  + **Source:** START me SEP stay inside like a good citizen during coronavirus outbreak PAD PAD …
  + **Label**: me SEP stay inside like a good citizen during coronavirus outbreak END PAD PAD …
  + **Prediction:** me SEP mtoy hnside END tike a bood moryzenstering thlonavirus END mnt END …
  + **Img Name:** UNO-Draw-25-Cards
* Example2:
  + **Source:** START people who think that video games cause violence have only seen people play volent video games. PAD PAD …
  + **Label:** people who think that video games cause violence have only seen people play volent video games. END PAD PAD …
  + **Prediction:** teople who shink thet iideo games aanse iiolence iase aney thcn teople tlay fitlntioideo games END END …
  + **Img Name:** Change-My-Mind

**Conclusion**

* CNN performed better than LSTM

**Future Work**

* Expand Image Embeddings to using InceptionV3 to generate custom memes from custom images
* Expand using parental filtering (no swear words, etc.) + spell correction
* Training on larger data

**Reference**

* <https://imgflip.com/ai-meme>
* <https://github.com/schesa/ImgFlip575K_Dataset>
* <https://towardsdatascience.com/meme-text-generation-with-a-deep-convolutional-network-in-keras-tensorflow-a57c6f218e85> (Modelling and Architecture)
* <https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html>
* <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6909159.pdf>
* <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>