Optymalizacja rozmiaru modeli językowych

mgr inż. Grzegorz Wojdyga

March 21, 2019

Cel seminarium

Celem tej prezentacji jest przedstawienie najciekawszych zdaniem autora metod optymalizacji rozmiaru sieci neuronowych które mogą mieć zastosowanie w tworzeniu modeli językowych oraz zaprezentowanie wyników implementacji jednej z nich.

O projekcie

Projekt był sponsorowany przez firmę Samsung i był prowadzony od września 2018 do lutego 2019.

Koordynatorem projektu był dr Łukasz Kobyliński.

Uwagi wstępne

- W tym raporcie będę zajmował się jedynie modelami językowymi opartymi na rekurencyjnych sieciach neuronowych.
- Coraz więcej dostawców frameworków dla developerów systemów deep learning dostarcza oprogramowanie do automatycznej optymalizacji sieci neuronowych (np. TensorFlow Lite). Nie były one tutaj badane.

Redukcja precyzji

- Polega na kwantyzowaniu danych zawartych w modelu
- Do najbardziej popularnych metod należą kwantyzacja liniowa, logarytmiczna oraz oparte na uczeniu maszynowym
- Można kwantyzować cały model jedną metodą albo różne składowe.

Kwantyzacja liniowa

- Polega na zamianie liczb typu 32-bit float na N-bitowe liczby.
- Zazwyczaj N mieści się w przedziale od 1 do 12 bitów.
- Wszystkie przedziały kwantyzacyjne są równe.

Binarne sieci neuronowe

Jednym z najciekawszych zastosowań redukcji precyzji jest stworzenie sieci binarnych to znaczy takich, w których wagi lub funkcje aktywacji zajmują jeden bit.
Istnieje kilka wariacji tworzenia tego typu sieci.

Binarne sieci neuronowe

Courbariaux [1] proponuje, żeby mapować wagi sieci neuronowych do dwóch wartości 1 i -1 podczas propagacji i propagacji wstecznej, natomiast dokonywać aktualizacji parametrów przy użyciu wartości zmiennoprzecinkowych.

Binarne sieci neuronowe

Stochastycznie:

$$\omega_b = \begin{cases} +1 & \text{z prawdopodobieństwem} \quad p = \sigma(\omega) \\ -1 & \text{z prawdopodobieństwem} \quad 1 - p \end{cases} \tag{1}$$

Deterministycznie:

$$\omega_b = \begin{cases} +1 & \text{jeśli} \quad \omega_b >= 0 \\ -1 & \text{jeśli} \quad \omega_b < 0 \end{cases} \tag{2}$$

Binarne sieci neuronowe

Inne wariancje binarnych wag sieci neuronowych proponują Soundry [2] i Cheng [3]. W ich przypadku sieć zwykłą propagację wstecz zastępuje „oczekiwana propagacja wstecz" opartej na zaproponowanej przez Minka metodzie oczekiwanej propagacji [4].

Ternaryzacja

Hwang [5] i Kim [6] proponują potrójne wagi. Ich metoda polega na wytrenowaniu tradycyjnej sieci neuronowej, a następnie mapowaniu wag do trzech możliwych wartości $-H, 0, H$ i wybraniu takiego H, dla którego wyniki będą najmniej odbiegały od wyników oryginalnej sieci. Opcjonalnie można na koniec po raz kolejny wytrenować sieć neuronową z trzema możliwymi wartościami wag przy propagacji i propagacji wstecz, a z zmiennoprzecinkowymi wagami przy aktualizacji parametrów.

Ternaryzacja

Inną metodę, z ustalonymi wartościami -1, 0 i 1 zaproponował Li [7], potem rozwijaną przez Zhu [8].

Konwolucyjne sieci neuronowyme

Najwięcej badań zostało poświęcone konwolucyjnym sieciom neuronowym – redukcja precyzji filtrów w wielkich sieciach neuronowych typu AlexNet przyniosła niewielkie różnice w wynikach (np. Rastegari [9])

Redukcja precyzji w LSTM

Zheng [10] zaproponował w swoim artykule, aby w binaryzować wagi w macierzy wejściowej i przejściowej. Autorzy zaproponowali binaryzację macierzy W_f oraz U_f w równaniu 3. Można również próbować binaryzować inne składowe. Ta metoda może być bardzo użyteczna przy modelach językowych.

Redukcja precyzji w LSTM

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
\tag{3}
$$

Destylacja wiedzy

Bardzo ważną metodą w ostatnich latach była przedstawiona przez Hintona w artykule „Distilling the Knowledge in a Neural Network" [11] „destylacja wiedzy". Mając dużą i efektywną sieć neuronową – "sieć nauczyciela", tworzy się mniejszą – „sieć ucznia". Następnie podczas treningu sieci ucznia korzysta się z funkcji straty sieci nauczyciela w warstwie przed softmaxem. Schemat uczenia przedstawiono na rysunku 4. Sieć ucznia podczas treningu korzysta z funkcji straty przedstawionej na równaniu 4.

Destylacja wiedzy

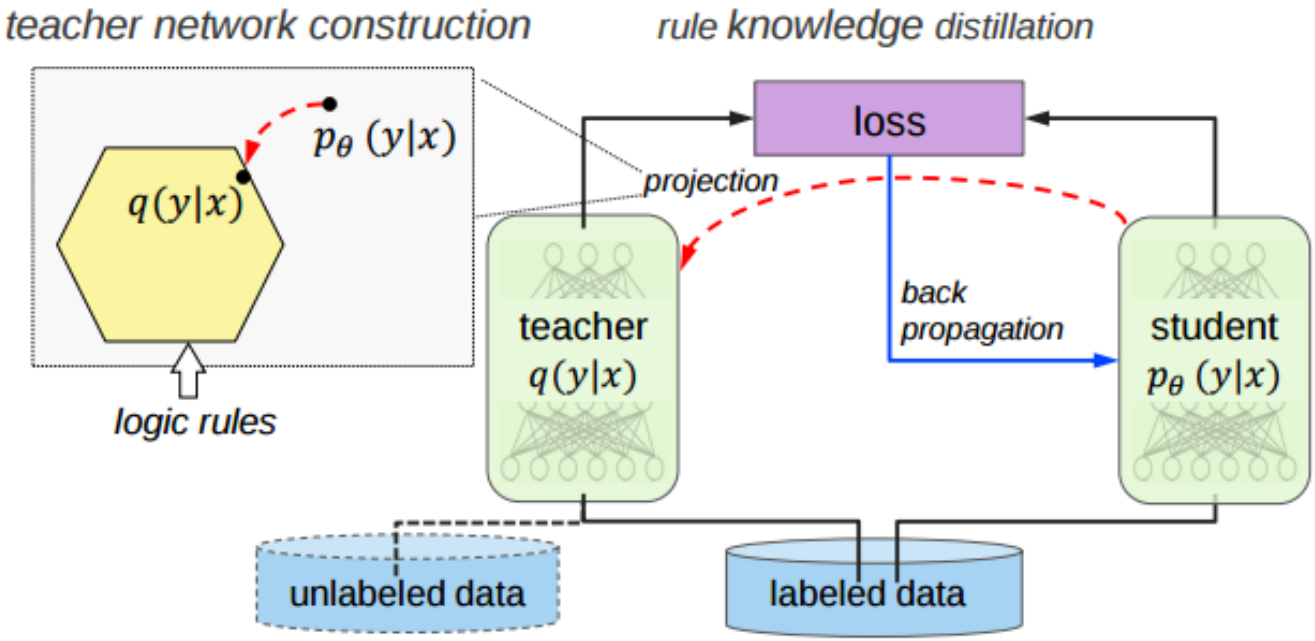


Figure: Trening sieci ucznia zaproponowany w artykule Hintona [11]

Destylacja wiedzy

$$L_{KD}(W_s) = H(y_{true}, P_S) + \lambda H(P_T^\tau, P_S^\tau) \tag{4}$$

gdzie

- L_{KD} – funkcja straty sieci ucznia
- W_s – parametry sieci ucznia
- H – entropia krzyżowa
- P_S – prawdopodobieństwo na wyjściu sieci ucznia
- y_true – label na wyjściu sieci ucznia
- P_S^τ, P_P^τ – prawdopodobieństwo na wyjściu sieci ucznia lub nauczyciela przez czynnik relaksacji τ

Destylacja wiedzy

Tego typu trening pozwala na znaczne zmniejszenie rozmiarów sieci studenta przy jednoczesnym zachowaniu wysokiej dokładności. Metoda ta została potem udoskonalona przez Romero [12], która zaproponowała „wskazówki” dawane przez sieć nauczyciela dla sieci ucznia.

Destylacja wiedzy

Bardzo ciekawą metodę, powiązaną z powyższą, zaproponował Inan [13] specjalnie dla modeli językowych. Tym razem jednak nie mamy dwóch oddzielnych sieci „nauczyciela” i „ucznia”, ale w ramach jednego modelu językowego wiążemy warstwę wejściową – embeddingów i wyjściową – projekcji, co znacząco zmniejsza warstwę parametrów w podobny sposób co Hinton [11] wiązał sieć nauczyciela i ucznia.

Destylacja wiedzy

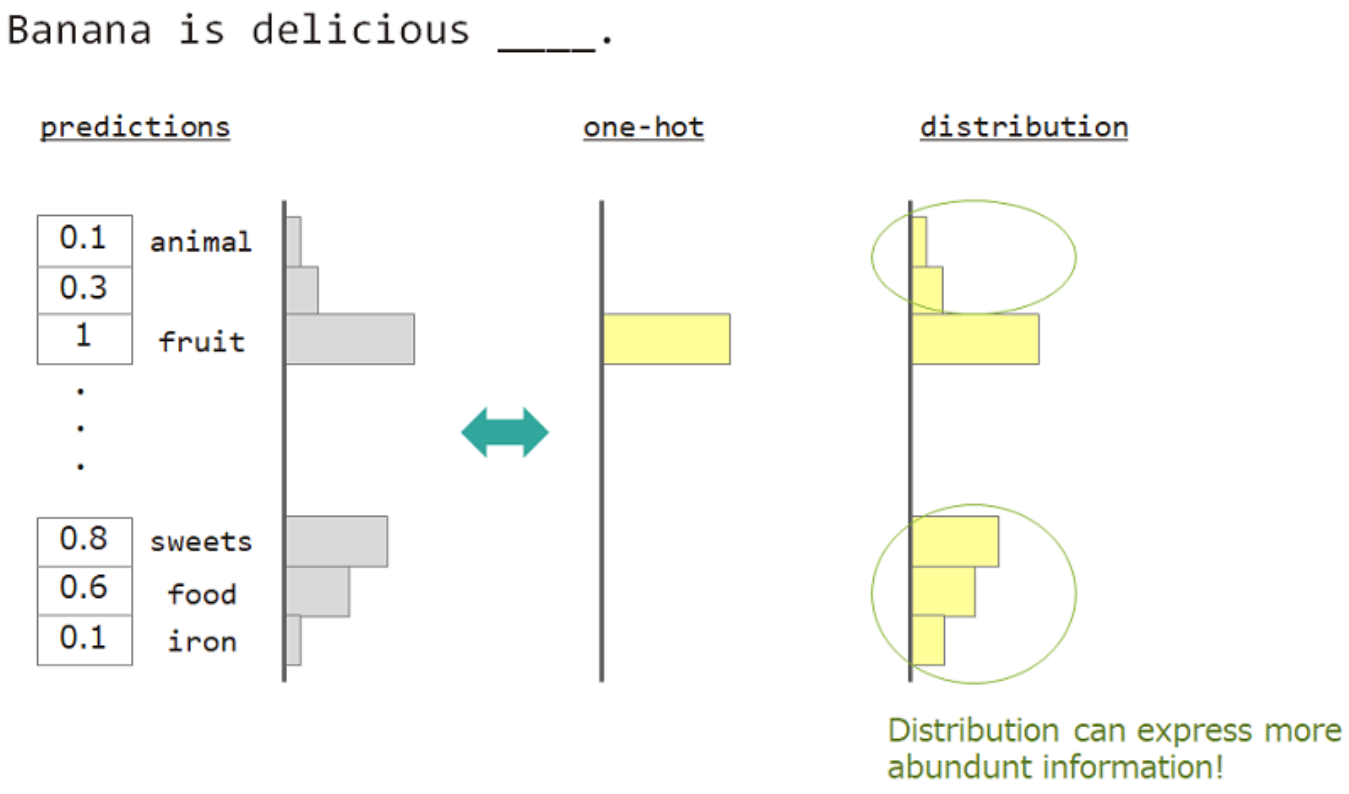


Figure: Dystrybucja wyników w warstwie wyjściowej [13]

Destylacja wiedzy

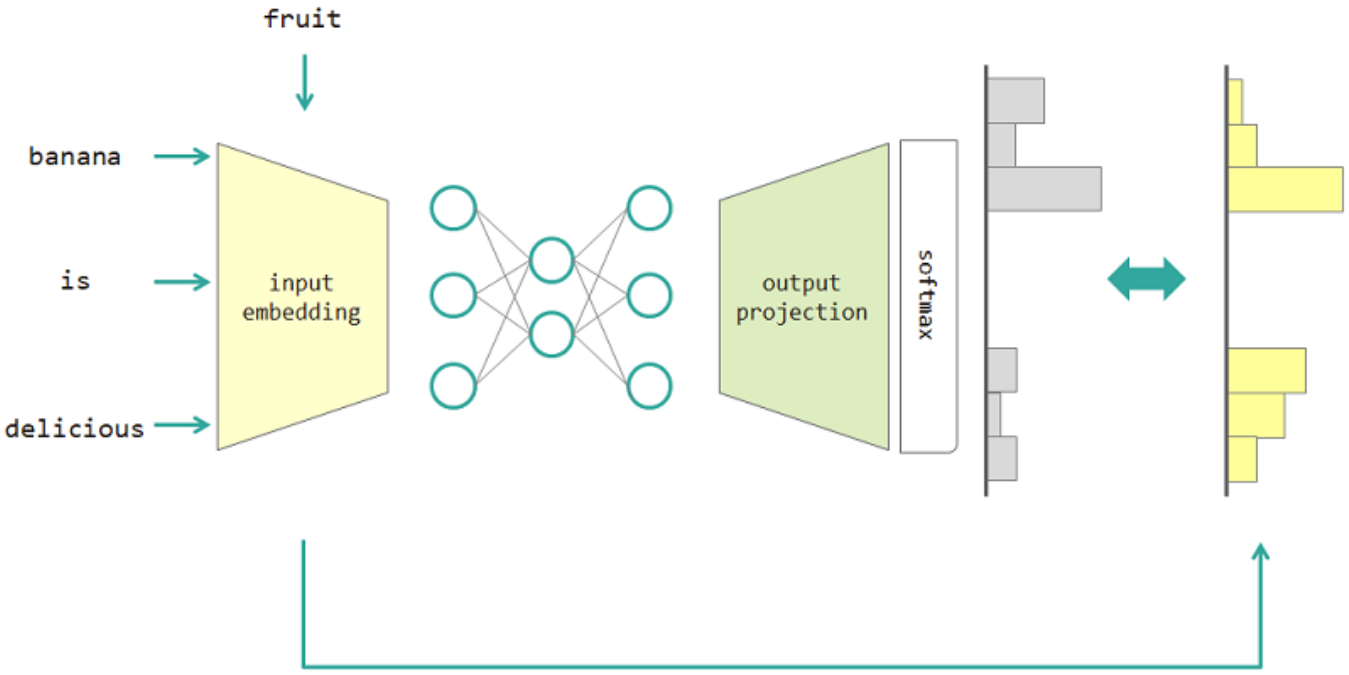


Figure: Dystrybucja wyników w warstwie wyjściowej [13]

Destylacja wiedzy

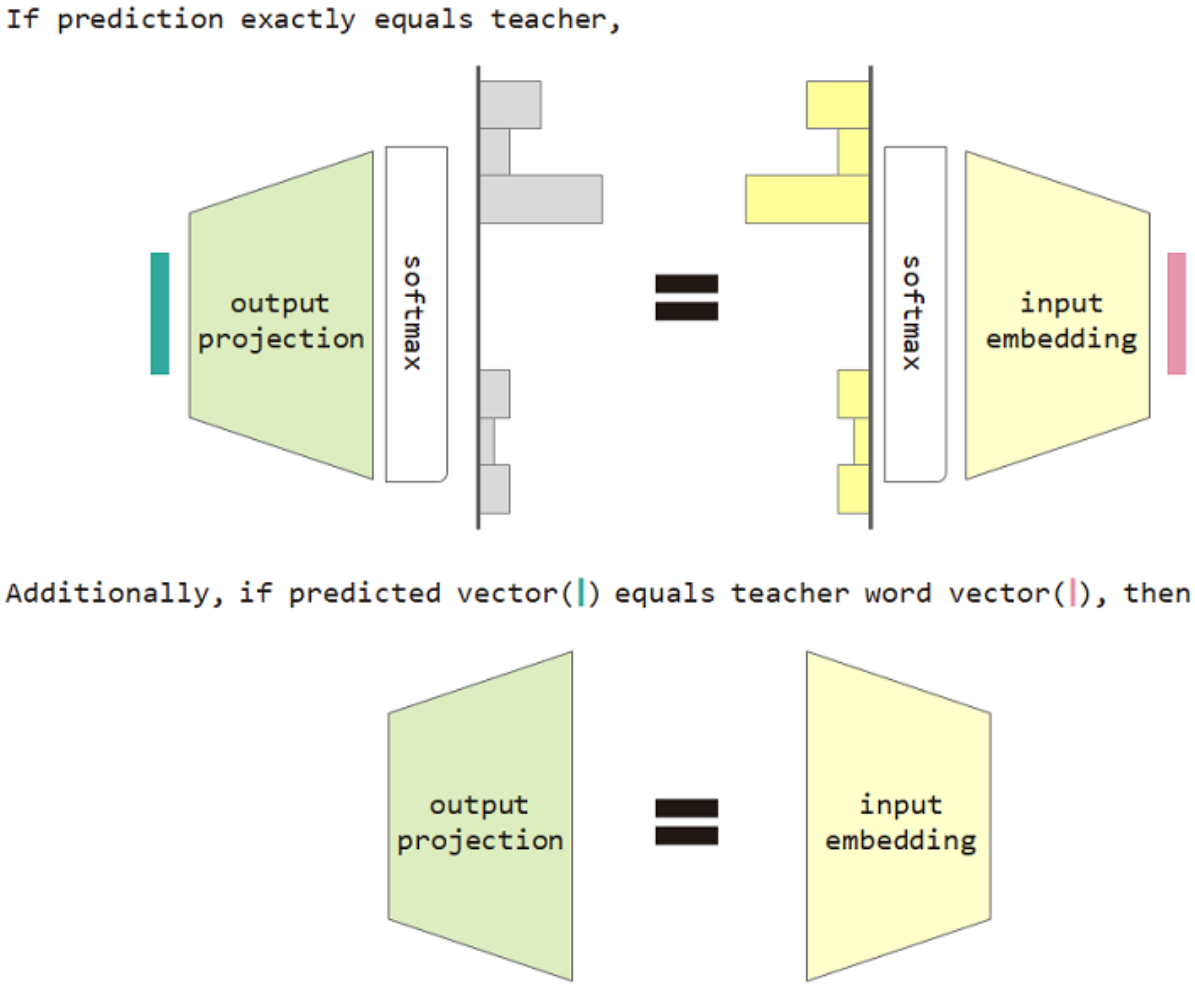


Figure: Dystrybucja wyników w warstwie wyjściowej [13]

Destylacja wiedzy

Nieco podobną metodę zaproponował w swoim artykule Ravi [14]. Podczas treningu sieci korzysta z fukcji straty zaproponowanej przez Hintona [11]. Jednak architektura sieci jest inna – przedstawia ją rysunek 5. W tej metodzie sieć „ucznia" została nazwana siecią projekcji. Środkową warstwę stanowi projekcja wektora wejścia. Projekcja być na przykład dokonana przy użyciu word2vec, jednak autorzy sięgnęli po metodę LSH – „locality sensitive hashing" [15]), która pozwala na znaczącą redukcję wymiarów mapując bliskoznaczne wyrażenia do jednej wartości.

Destylacja wiedzy

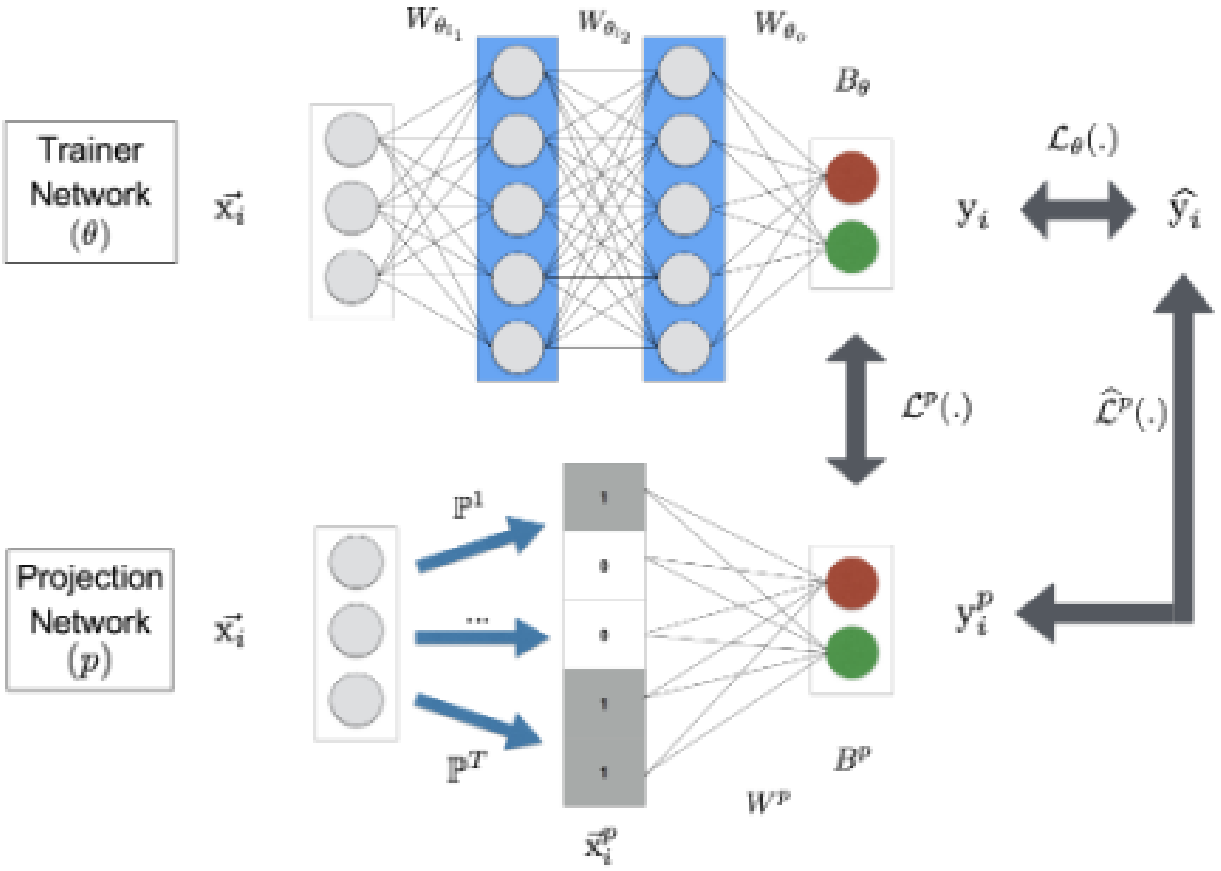


Figure: Architektura sieci zaproponowanej przez Ravi'ego [14]

Kompresja sieci

Połączenie różnych technik optymalizacyjnych zostało wykorzystane w jednym z najczęściej cytowanych artykułów w tej dziedzinie – Song Hana i innych „Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding" [16].

Kompresja sieci

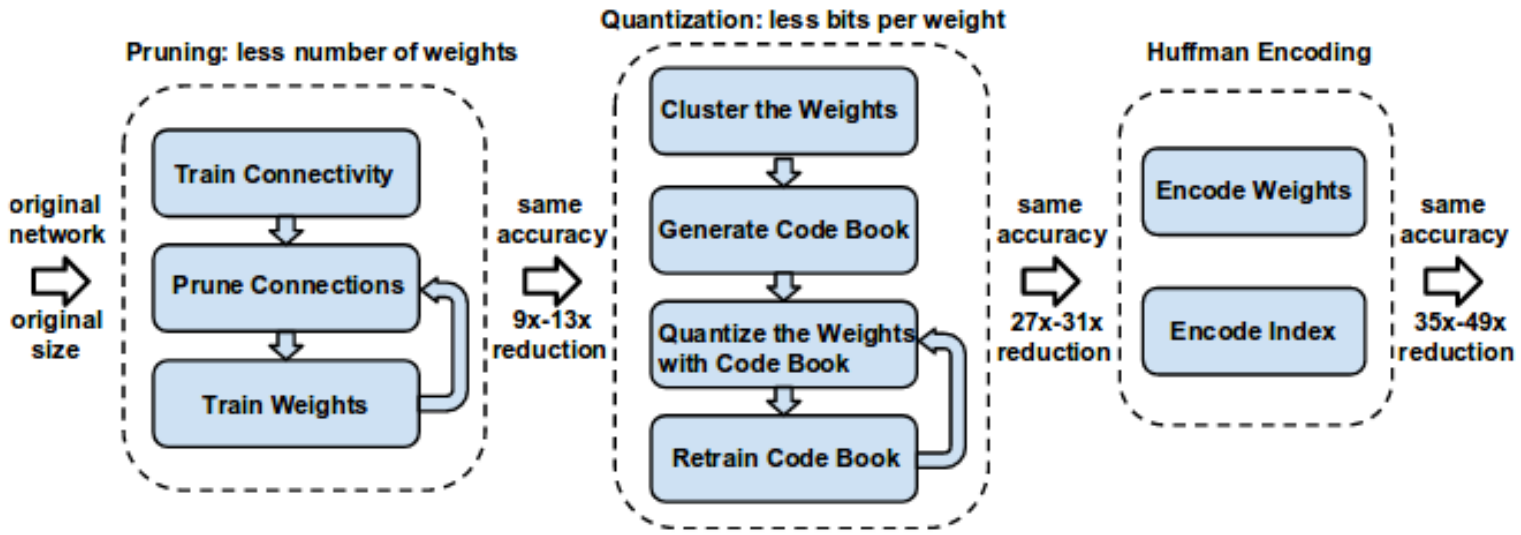


Figure: Metody optymalizacji w artykule Hana i innych [16]

Kompresja sieci

Mając w pełni wytrenowaną sieć neuronową

- redukcja połączeń - usunięcie wszystkich połączeń, których wartości są mniejsze niż wybrana wartość progowa
- specjalne struktury danych do efektywnego przechowywania pozostałych połączeń
- ponownie trenowanie sieci na pozostałych połączeniach

Kompresja sieci

Kolejnym krokiem jest współdzielenie wag – pozwala na zmniejszeniu pamięci potrzebnej do przechowania wag.

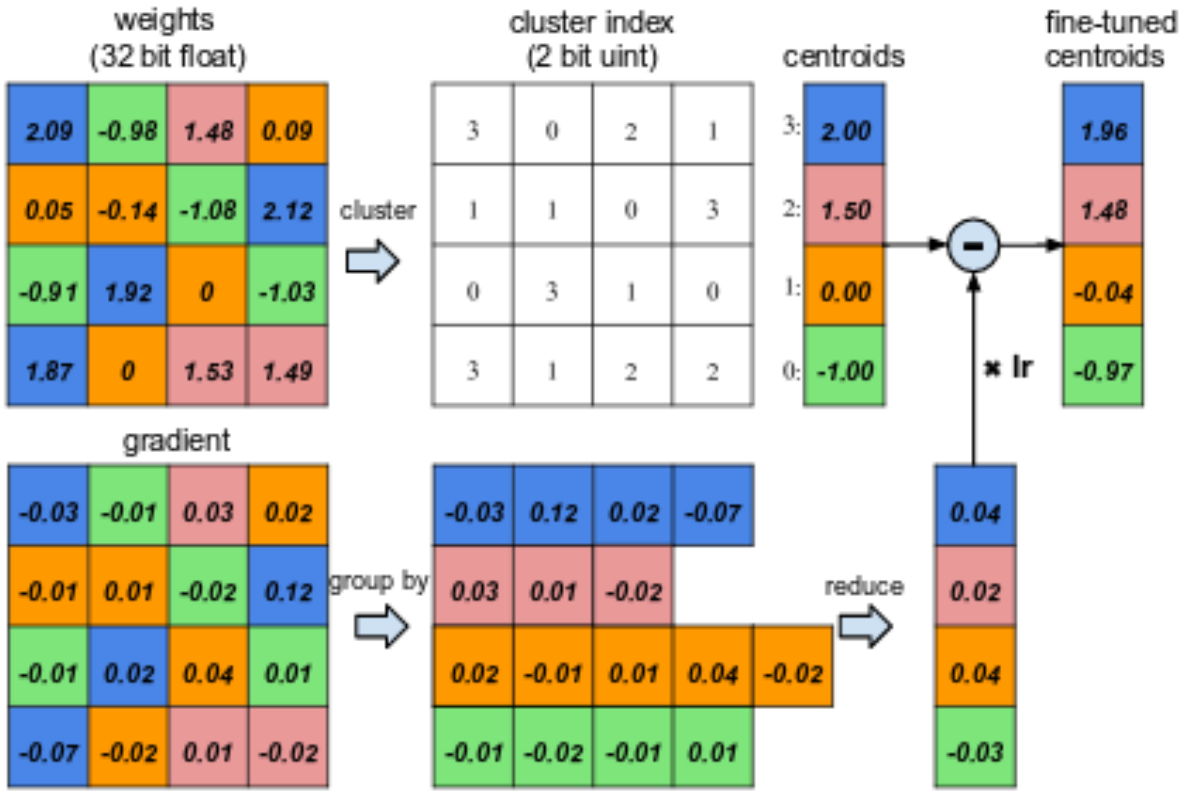


Figure: Współdzielenie wag sieci neuronowej w artykule Hana i innych [16]

Kompresja sieci

Jak widać na rysunku występuje podział na cztery klastry ze zbliżonymi centroidami. Gradienty wag w danym przedziale są redukowane w ramach jednego klastra do jednej wartości i z tej wartości korzysta się podczas aktualizacji parametrów. Nieco podobną metodę, opartą nie na współdzieleniu, ale na haszowaniu zaproponował Chen [17].

Kompresja sieci

Ostatnim etapem jest kodowanie Huffmana. Łącznie autorzy podają, że udało im się pomniejszyć rozmiar modelu AlexNet 35 do 49 razy.

Kompresja sieci

Podobną metodę zaproponował Guo [18]. U niego występuje tylko pierwsze stadium - to znaczy niwelowanie niepotrzebnych połączeń, jednak w przeciwieństwie do Hana, operacja ta dzieje się dynamicznie i połączenia nie są usuwane. Jeśli w kolejnych iteracjach okaże się, że połączenie jednak jest potrzebne, wówczas połączenie to może być ponownie wykorzystane. Schemat został zaprezentowany na rysunku 8.

Kompresja sieci

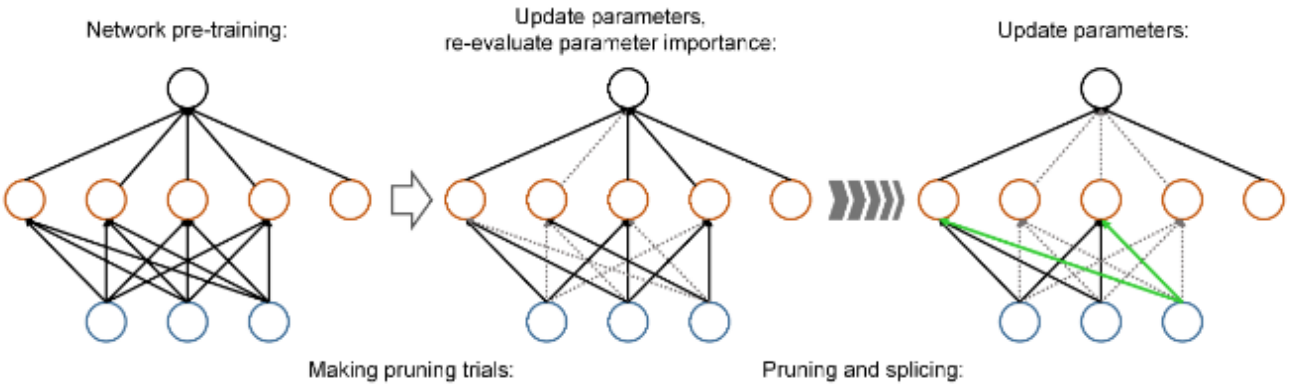


Figure: Niwelowanie niepotrzebnych połączeń w artykule Guo [18]

Binaryzacja

Stochastycznie:

$$\omega_b = \begin{cases} +1 & \text{z prawdopodobieństwem} \quad p = \sigma(\omega) \\ -1 & \text{z prawdopodobieństwem} \quad 1 - p \end{cases} \tag{5}$$

Deterministycznie:

$$\omega_b = \begin{cases} +1 & \text{jeśli} \quad \omega_b >= 0 \\ -1 & \text{jeśli} \quad \omega_b < 0 \end{cases} \tag{6}$$

Binaryzacja

Z powodów praktycznych korzystałem (za wszystkimi autorami zajmującymi się tym tematem) wersją deterministyczną.

Binaryzacja

Algorytm:

- Podczas propagacji do przodu - korzystamy z wag po binaryzacji
- Podczas propagacji wstecz wyliczamy gradienty (niebinarne)
- Podczas aktualizacji paramentów aktualizujemy niebinarne wagi.

Środowisko

Wszystkie testy robiłem na modelu językowym Embeddings + LSTM

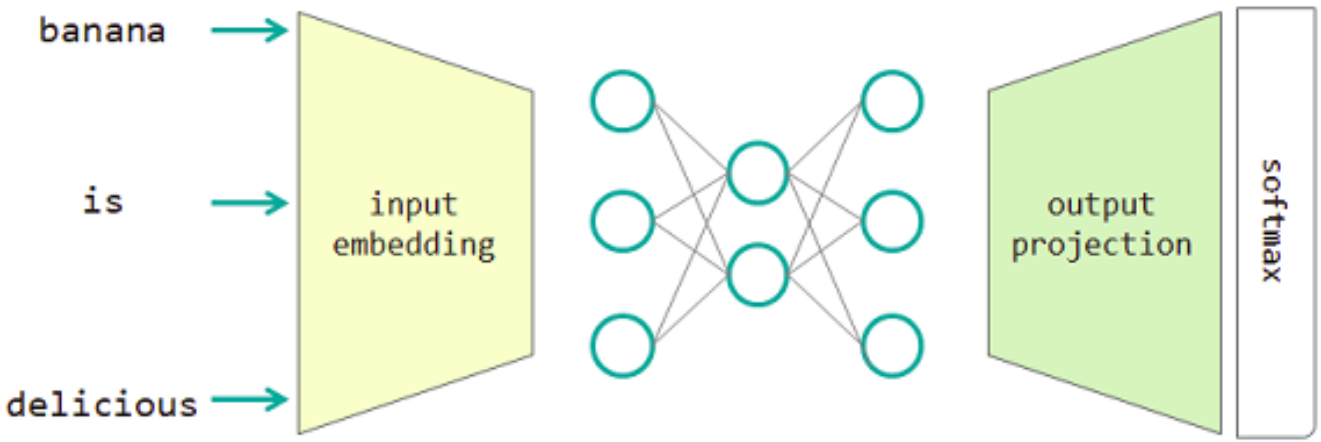


Figure: Przykładowa architektura

Środowisko

Do wstępnych testów korzystałem z korpusu porównywalnego z PTB.

Środowisko

Wszystkie testy zostały puszczone na wirtualnej maszynie na Google Cloudzie posiadającej kartę graficzną Tesla i z działającą CUDĄ.

Wyniki binaryzacji

Wyniki były mierzone na korpusie porównywalnym z PTB.

Model	**Perplexity**
Standardowy	120,45
Zbinaryzowany najlepszy	410,13

Table: Wyniki PPL

Próby w rozmaitych konfiguracjach warstw, ilości jednostek nie przynosiły poprawy. Mail do Stanfordu też nie...

Dla embeddingów:
Przymierzałem się, ale doczytałem, że to nie ma najmniejszego sensu.

Ternaryzacja

$$\omega_b = \begin{cases} +1 & \text{jeśli} \quad \omega_b >= T \\ 0 & \text{jeśli} \quad -T < \omega_b <= T \\ -1 & \text{jeśli} \quad \omega_b < -T \end{cases} \tag{7}$$

T może być dowolnie wybrane – ja przeprowadziłem pierwsze testy dla T=0.5

Problemy z Ternaryzacją:

- musi być zużyte 2 bity
- nie ma zoptymalizowanych architektur pod nią
- nie da się korzystać z boolean

Wyniki Ternaryzacji

Wyniki były mierzone na korpusie porównywalnym z PTB.

Model	**Perplexity**
Standardowy	120,45
Zternaryzowany najlepszy	114,63

Table: Wyniki PPL

Embeddingsy

Ternaryzacja - sama w sobie wydaje się nie ma sensu, ale w trakcie są prace dotyczące kwantyzacji - zarówno liniowej jak i logarytmicznej

Wyniki prac

- Zaimplementowane custom layery LSTM do binaryzacji i ternaryzacji w Kerasie.
- W Pytorchu jest zaimplementowana custom layer do ternaryzacji, ale wymaga jeszcze optymalizacji.

Pomysł dla Embeddingsów

„Online Embedding Compression for Text Classification using Low Rank Matrix Factorization" by Anish Acharya et al. from Amazon

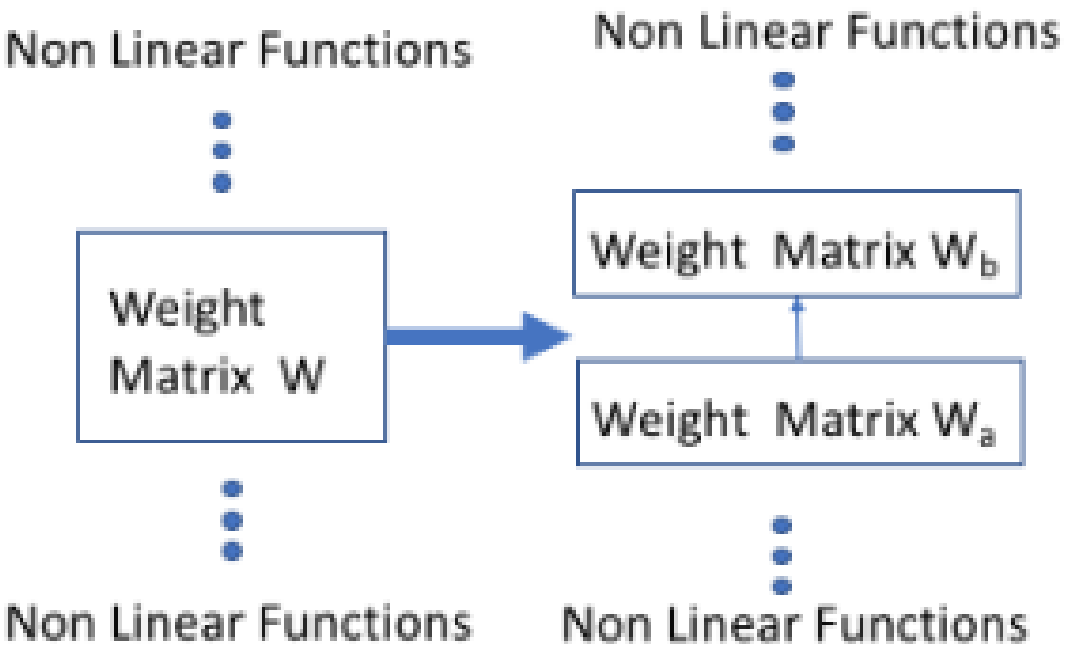


Figure: Przykładowa architektura

Testy

W tej sekcji zostaną omówione szczegółowe wyniki testów.

Clicking ratio

Poniższy algorym został zaimplementowawy i sprawdzany co 5 epok.

```
sorted_results = sorted(prob_dict.items(), key=itemgetter(1), reverse=True)

click_queue = deque(["", "", ""])
begin = ""
for letter in out_word[:-1]:
    begin += letter
    for letter in range(3):
        click_queue.append(begin)

for words in sorted_results:
    if words[0].startswith(click_queue[0]):
        if out_word == words[0]:
            break
        else:
            click_queue.popleft()
            if len(click_queue) == 0:
                break

if len(click_queue) == 0:
    hits = max_hits
else:
    hits = max_hits - (len(click_queue) + 2) // 3

return hits, max_hits
```

Figure: Click algorithm

Modele

Zostały użyte 4 modele sieci:

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 15, 200)	18600
dropout_1 (Dropout)	(None, 15, 200)	0
lstm_1 (LSTM)	(None, 15, 200)	320800
dropout_2 (Dropout)	(None, 15, 200)	0
lstm_2 (LSTM)	(None, 200)	320800
dropout_3 (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 93)	18693

Figure: Model 1

Models

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 15, 500)	46500
lstm_1 (LSTM)	(None, 15, 50)	110200
flatten_1 (Flatten)	(None, 750)	0
dropout_1 (Dropout)	(None, 750)	0
dense_1 (Dense)	(None, 64)	48064
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 93)	6045

Figure: Model 2

Models

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 15, 300)	27900
dropout_1 (Dropout)	(None, 15, 300)	0
lstm_1 (LSTM)	(None, 15, 200)	400800
flatten_1 (Flatten)	(None, 3000)	0
dropout_2 (Dropout)	(None, 3000)	0
dense_1 (Dense)	(None, 93)	279093

Figure: Model 3

Models

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 15, 400)	37200
dropout_1 (Dropout)	(None, 15, 400)	0
lstm_1 (LSTM)	(None, 15, 1150)	7134600
dropout_2 (Dropout)	(None, 15, 1150)	0
lstm_2 (LSTM)	(None, 15, 1150)	10584600
dropout_3 (Dropout)	(None, 15, 1150)	0
lstm_3 (LSTM)	(None, 1150)	10584600
dropout_4 (Dropout)	(None, 1150)	0
dense_1 (Dense)	(None, 93)	107043

Figure: Model 4

Vocab info

Testy zostały przeprowadzone dla korpusów o słownikach w wielkości 10k, 20k, 50k, 100k z Polevalu 2018.

10k Model 1

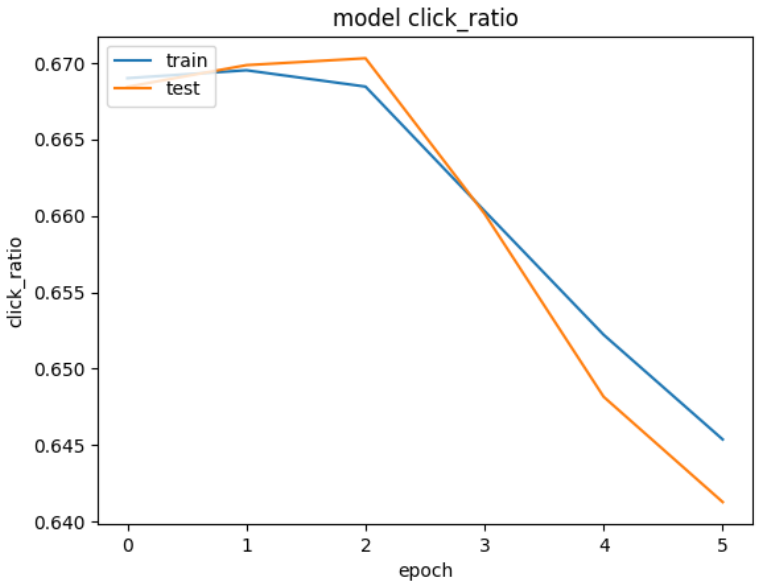


Figure: Click ratio with normal LSTM

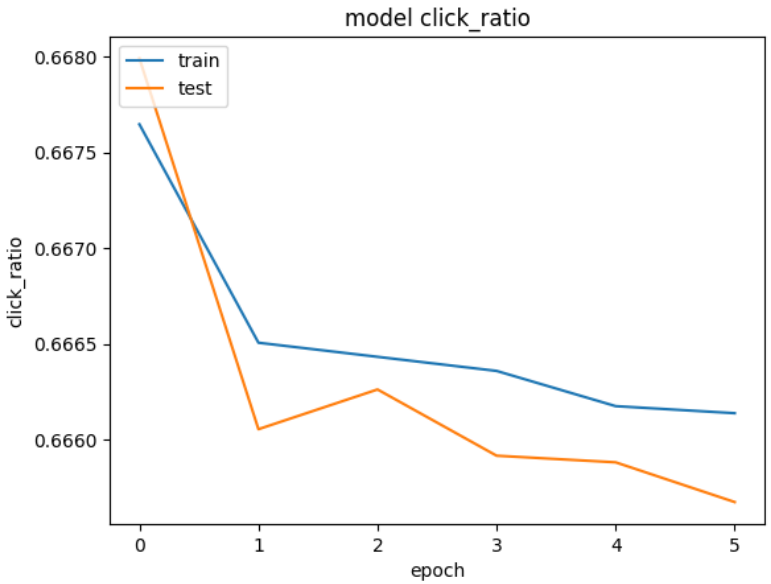


Figure: Click ratio with ternary LSTM

20k Model 1

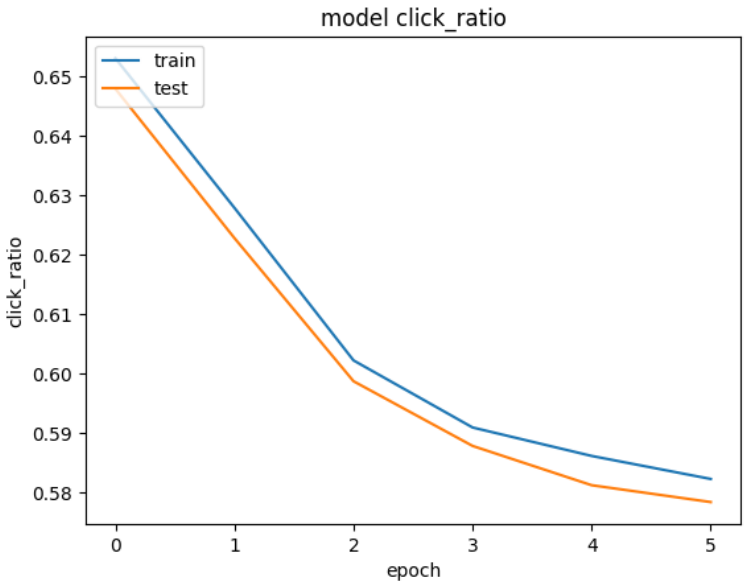


Figure: Click ratio with normal LSTM

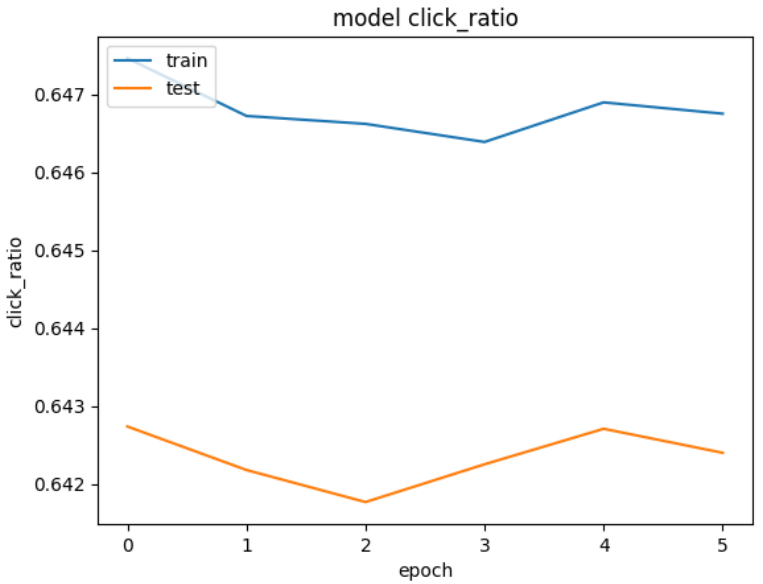


Figure: Click ratio with ternary LSTM

50k Model 1

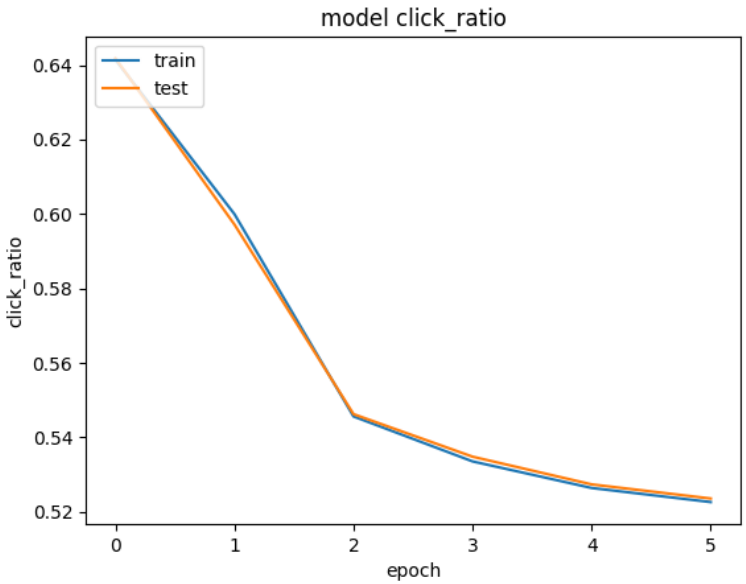


Figure: Click ratio with normal LSTM

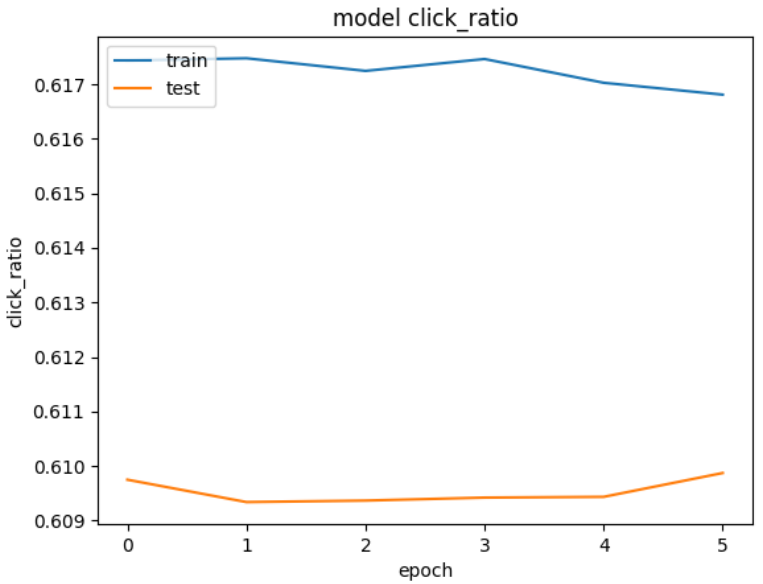


Figure: Click ratio with ternary LSTM

100k Model 1

10 epoch

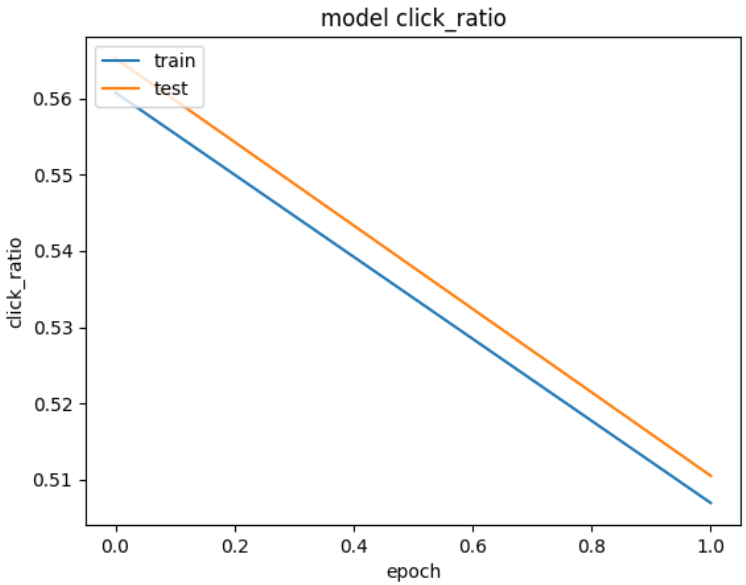


Figure: Click ratio with normal LSTM

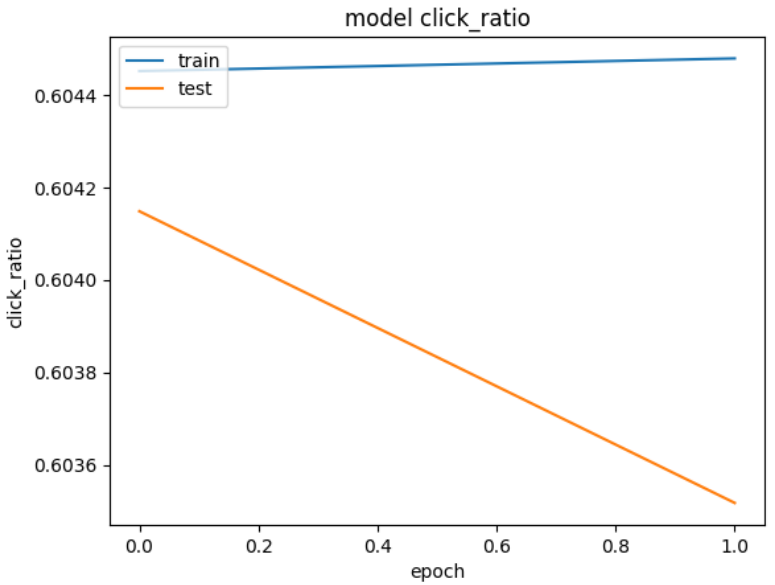


Figure: Click ratio with ternary LSTM

10k Model 2

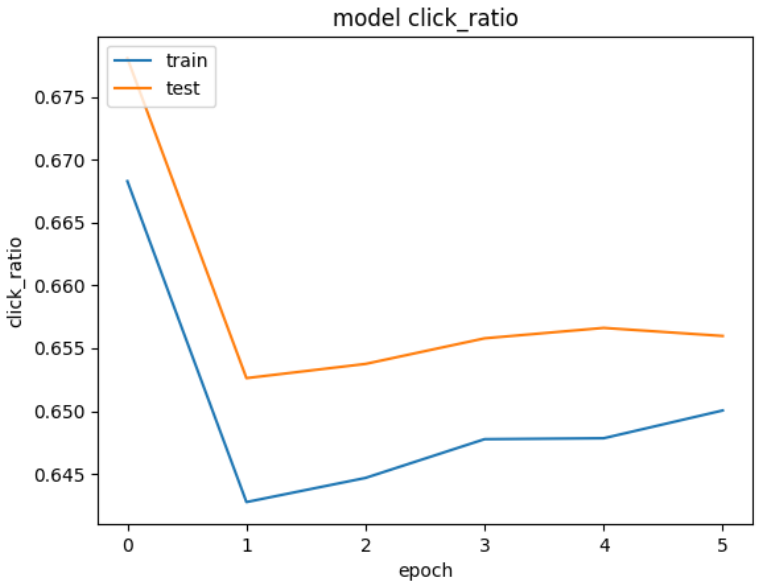


Figure: Click ratio with normal LSTM

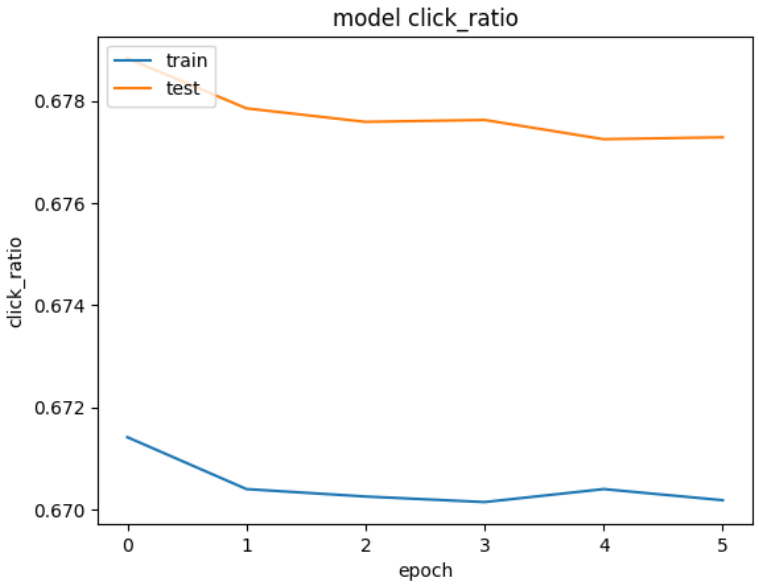


Figure: Click ratio with ternary LSTM

20k Model 2

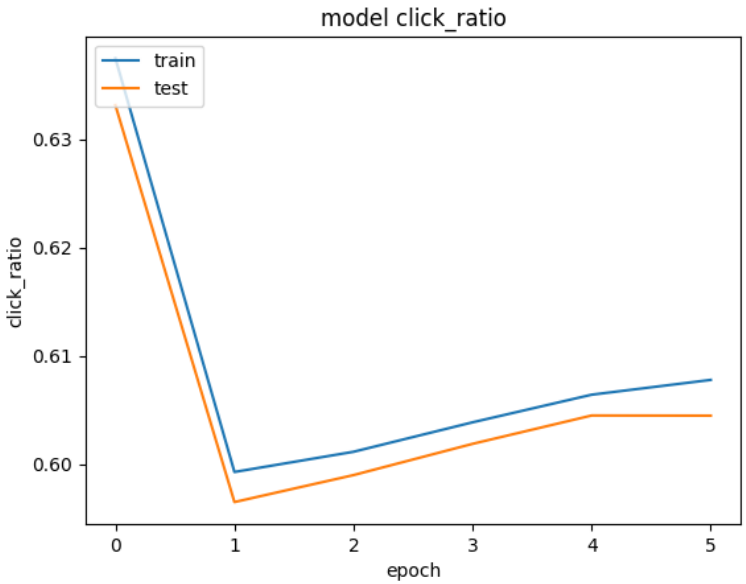


Figure: Click ratio with normal LSTM

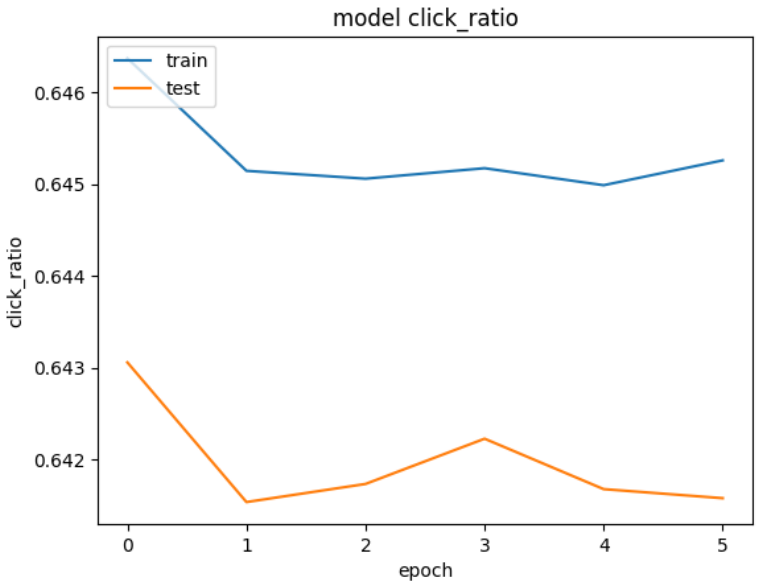


Figure: Click ratio with ternary LSTM

50k Model 2

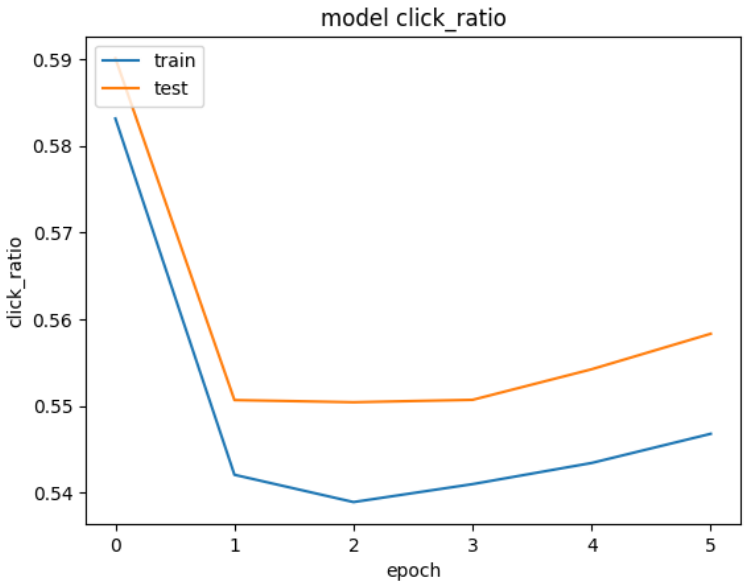


Figure: Click ratio with normal LSTM

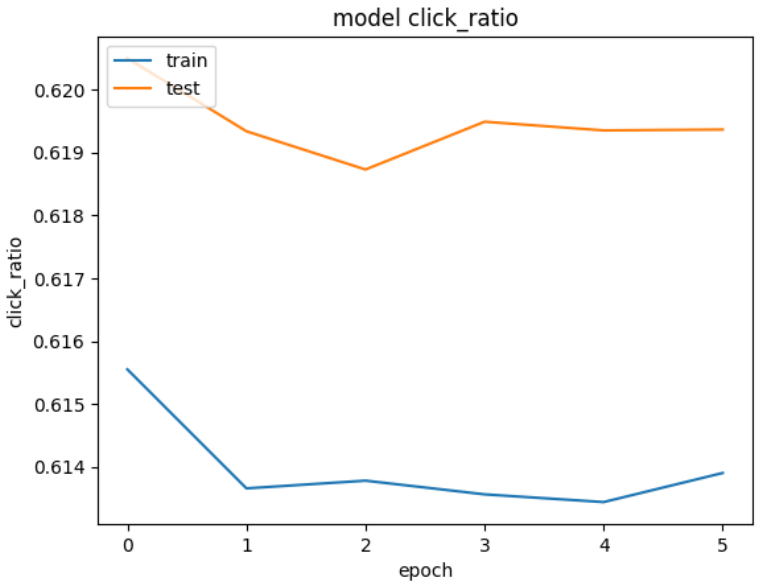
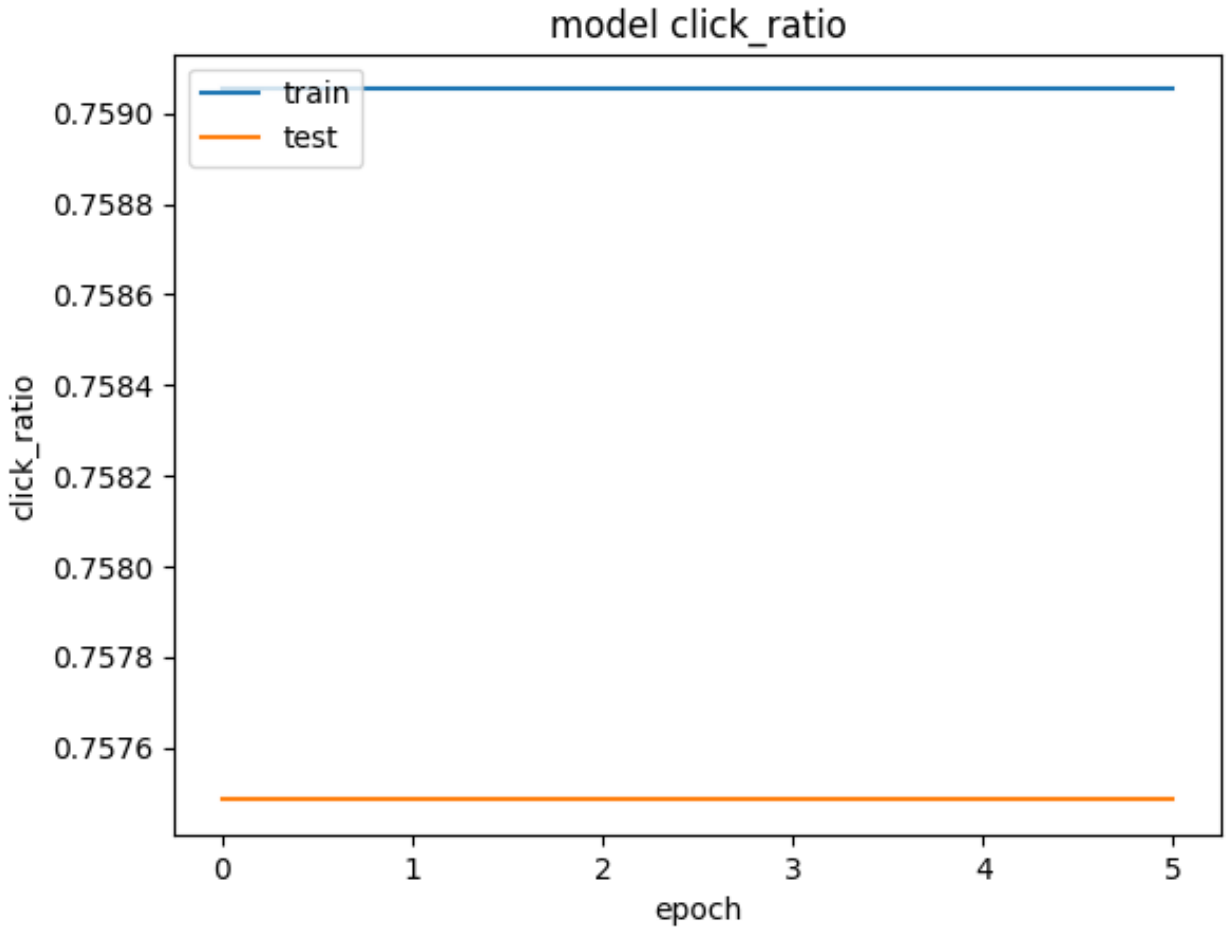


Figure: Click ratio with ternary LSTM

Binarization

The results were significantly worse.



New idea

Evangelos Stromatias et al. *Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspiredhardware platforms*

```
[[ 0.22093558  0.7366972   0.81512976 -0.57212734 -0.20211267]
 [ 0.04812169 -0.8214588   0.24493837 -0.81740904 -0.8414831 ]
 [-0.50879765  0.6940167  -0.25690055  0.05504251 -0.3099959 ]
 [-0.10914421 -0.8009751  -0.4856975  -0.24098349  0.00294948]
 [ 0.59754944  0.7821708   0.96892786 -0.51667476 -0.4494908 ]], shape=(5, 5), dtype=float32)
tf.Tensor(
[[ 0.   0.5  1.  -0.5  0. ]
 [ 0.  -1.   0.  -1.  -1. ]
 [-0.5  0.5 -0.5  0.  -0.5]
 [ 0.  -1.  -0.5  0.   0. ]
 [ 0.5  1.   1.  -0.5 -0.5]], shape=(5, 5), dtype=float32)
```

Figure: Example of 3-bit ternarization

Wnioski

- Przy małych korpusach różnice w klikalności są bardzo niewielkie
- W większych korpusach (>50k) różnice się zwiększają i sięgają 10 punktów procentowych
- Alorytm Stromatias może być rozwiązaniem problemu ternaryzacji przy większych korpusach

Wnioski naukowe

- Click ratio nie jest ściśle skorelowany z perplexity – tego typu zależności trzeba badać osobno dla każdego korpusu
- Algorytm Stromatiasa da się zaimplementować w wersji automatycznej – można wybrać metrykę, która nas interesuje (np. click ratio) i określić o ile procent może się pogorszyć, a program sam wybierze minimalną ilość bitów do zakodowanie LSTMu.
- Warto przeprowadzić podobne testy dla GRU.

Dziękuję za uwagę!

References I

[1] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[2] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Advances in Neural Information Processing Systems*, pp. 963–971, 2014.

[3] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan, "Training binary multilayer neural networks for image classification using expectation backpropagation," *arXiv preprint arXiv:1503.03562*, 2015.

References II

[4] T. P. Minka, "Expectation propagation for approximate bayesian inference," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pp. 362–369, Morgan Kaufmann Publishers Inc., 2001.

[5] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, IEEE, 2014.

[6] J. Kim, K. Hwang, and W. Sung, "X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 7510–7514, IEEE, 2014.

References III

[7] F. Li and B. Liu, "Ternary weight networks," *CoRR*, vol. abs/1605.04711, 2016.

[8] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[10] W. Zheng and Y. Tang, "Binarized neural networks for language modeling," tech. rep., Technical Report cs224d, Stanford University, 2016.

References IV

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[12] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *CoRR*, vol. abs/1412.6550, 2014.

[13] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," *CoRR*, vol. abs/1611.01462, 2016.

[14] S. Ravi, "Projectionnet: Learning efficient on-device deep networks using neural projections," *CoRR*, vol. abs/1708.00630, 2017.

References V

[15] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388, ACM, 2002.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[17] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, pp. 2285–2294, 2015.

References VI

[18] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.