Deep Learning

Chapter 11 (+ 12…)
"Tricks of the trade"

Helge Langseth
helgel@ntnu.no

Disclaimer

- The content in this lecture is **mostly** based on the book, but also contains some "gut-feeling", "rules of thumb", and "urban legends".
- Some of what I say may therefore be "controversial":
  - You are allowed to disagree!
  - If you disagree, you may well be right!
- But then again, some slides are from the book, too, and those are obviously correct and just perfect!

RefGrp

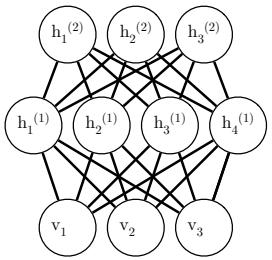
- We need at least 2 people…

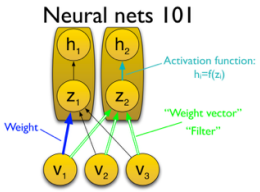
What drives success in ML?

Arcane knowledge of dozens of obscure algorithms?

Mountains of data?

Knowing how to apply 3-4 standard techniques?





Three Step Process

- Use needs to define metric-based goals
- Build an end-to-end system
- Data-driven refinement

Step 1: Identify needs

Choose Metrics… … or combinations thereof

- Accuracy? (% of examples correct)
- Coverage? (% of examples processed)
- Precision? (% of detections that are right)
- Recall? (% of objects detected)
- Amount of error? (For regression problems)

Metrics

- The system will try to “cheat” during learning:
  - If there is a short-cut to optimize the loss, the optimizer may well find it
  - Make sure a model with good results on your metric is a good model in production.
  - Ex.: Detection of a rare disease

What is required for this to be a success???

- High accuracy or low accuracy?
- Surgery robot: high accuracy
- Celebrity look-a-like app: low accuracy

Step 2: Build end-to-end system

End-to-end System

- Get up and running ASAP
- Build the simplest viable system first
- What baseline to start with though?
  - Copy state-of-the-art from related publication

Deep or Not?

- Lots of noise, little structure -> not deep
- Little noise, complex structure -> deep
- Good shallow baseline:
  - *Use what you know*
  - Logistic regression, SVM, boosted tree are all good

Entangled models

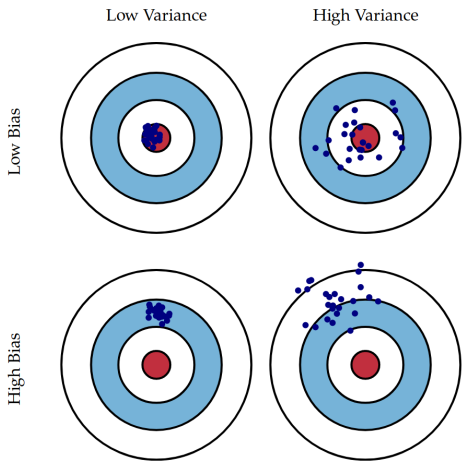
- If your model contains several “sub-models” (which is not always recommendable…):
  - Evaluate them separately if possible
  - Where does the system error come from?
  - Prioritize improvements based on cost/benefit
- If your model does **not** consist of sub-models, it can **still** be beneficial to pipe out intermediate results and do “QA” if at all reasonable.

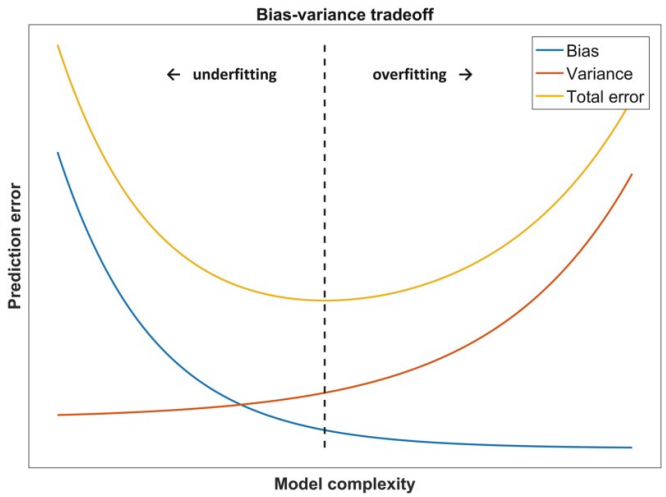
Step 3: Data driven refinement

Data Driven Refinement

- Understand your model and its strengths/weaknesses
  - Measure error on train and test sets
  - Overfitting vs underfitting
- Understand the learning process
  - Use Tensorboard and/or other visualizations
    - Know what to look for, and how to fix problems
- Choose a models with relevant learning bias
  - Don’t believe the hype
  - Feed Forward, CNN, RNN, more advanced tricks

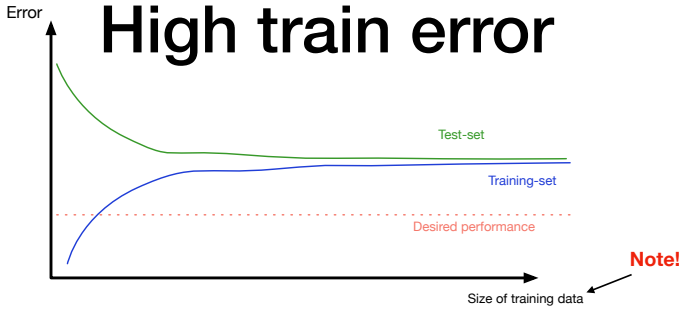
Variance - Bias tradeoff





17

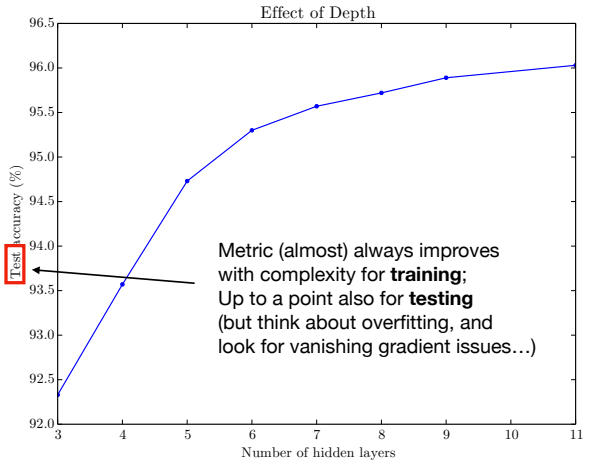
High train error



- This can be the high-bias case —> Under-fitting
  - Increase representational power (model depth, layer size, remove regularizers, …)
- … or a fundamental error somewhere:
  - Code issues (stick to what you know works — like TF…)
  - Data issues
  - Tuning of hyper-parameters

18

Increasing Depth



19

Checking Data for Defects

Can a human process it?

26624

Just as important, but more tedious to check:
Are there systematic errors? Are there trends in mistakes (e.g., always on brick wall or similar).

20

Hyper-parameters

Alternative 1: Think long and hard about the HPs, and their effect…

Hyperparameter	Increases capacity when…	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.

Table 11.1

Tuning the Learning Rate

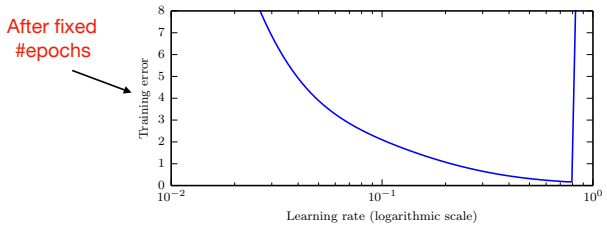


Figure 11.1

- **Too high lr:** Can go astray
- **Too low lr:**
  - Computationally costly (more epochs to get a decent move)
  - Potentially stuck in local minima of the loss landscape
- **Just right lr:** Works just right!!
- Either work very hard do tune the lr or use an lr scheduler (extremely simple with Keras: tf.keras.callbacks.LearningRateScheduler) … or both!!

Hyper-parameters

Alternative 2: Automated search

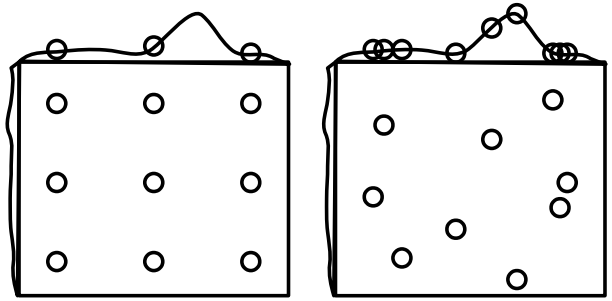


Figure 11.2

Practical Bayesian Optimization of Machine Learning Algorithms

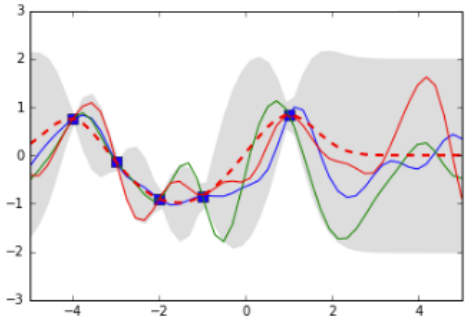
Jasper Snoek
Department of Computer Science
University of Toronto
jasper@cs.toronto.edu

Hugo Larochelle
Department of Computer Science
University of Sherbrooke
hugo.larochelle@usherbrooke.edu

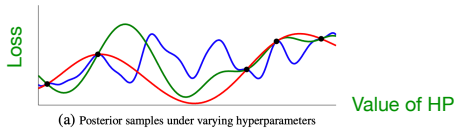
Ryan P. Adams
School of Engineering and Applied Sciences
Harvard University
rpa@seas.harvard.edu

Gaussian Processes

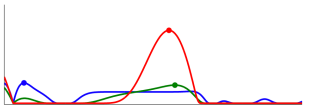
- A fancy way to make statistical distributions over **curves**.
- Assumes all possible points on the curve are jointly Gaussian, with a covariance between pairs depending on their distance in “input-space”.
- We can sample by (infinitely many times) generating “y-value” for some “x-value”, conditioned on previous samples



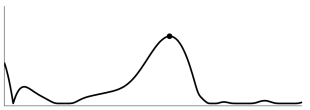
25



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters

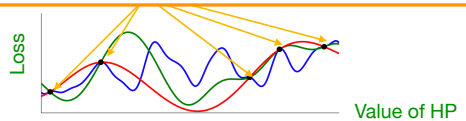


(c) Integrated expected improvement

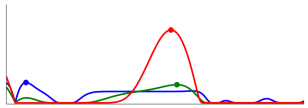
Figure 1: Illustration of integrated expected improvement. (a) Three posterior samples are shown, each with different length scales, after the same five observations. (b) Three expected improvement acquisition functions, with the same data and hyperparameters. The maximum of each is shown. (c) The integrated expected improvement, with its maximum shown.

26-1

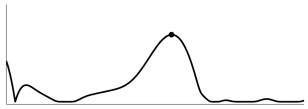
Already run experiments, with HP (x-axis) and Loss (y-axis) known



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters

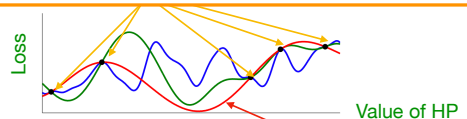


(c) Integrated expected improvement

Figure 1: Illustration of integrated expected improvement. (a) Three posterior samples are shown, each with different length scales, after the same five observations. (b) Three expected improvement acquisition functions, with the same data and hyperparameters. The maximum of each is shown. (c) The integrated expected improvement, with its maximum shown.

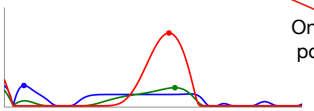
26-2

Already run experiments, with HP (x-axis) and Loss (y-axis) known

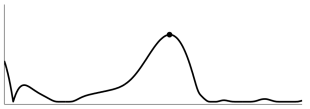


(a) Posterior samples under varying hyperparameters

One specific sample from posterior (i.e. given data)



(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

Figure 1: Illustration of integrated expected improvement. (a) Three posterior samples are shown, each with different length scales, after the same five observations. (b) Three expected improvement acquisition functions, with the same data and hyperparameters. The maximum of each is shown. (c) The integrated expected improvement, with its maximum shown.

26-3

Already run experiments, with HP (x-axis) and Loss (y-axis) known

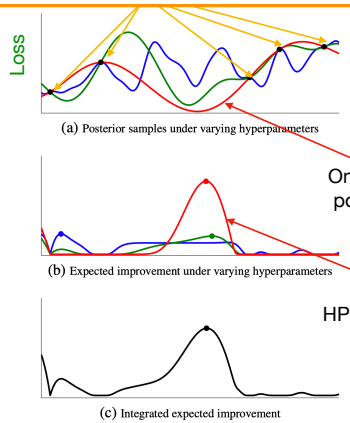


Figure 1: Illustration of integrated expected improvement. (a) Three posterior samples are shown, each with different length scales, after the same five observations. (b) Three expected improvement acquisition functions, with the same data and hyperparameters. The maximum of each is shown. (c) The integrated expected improvement, with its maximum shown.

Already run experiments, with HP (x-axis) and Loss (y-axis) known

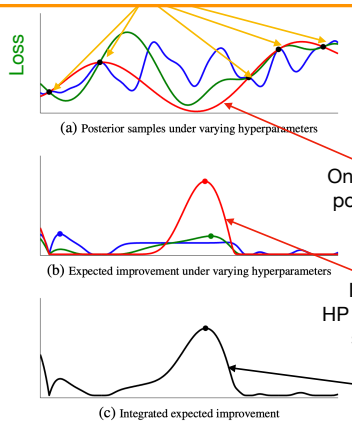


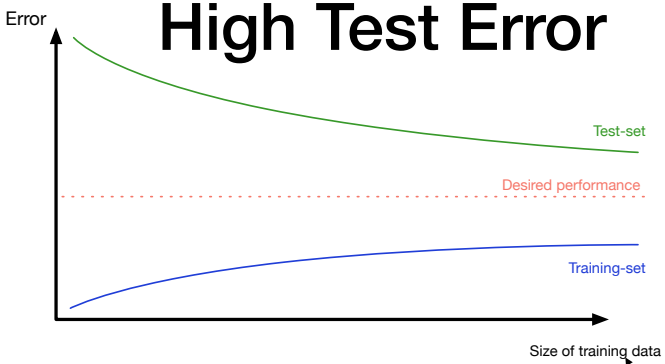
Figure 1: Illustration of integrated expected improvement. (a) Three posterior samples are shown, each with different length scales, after the same five observations. (b) Three expected improvement acquisition functions, with the same data and hyperparameters. The maximum of each is shown. (c) The integrated expected improvement, with its maximum shown.

Implementation in Keras tuner

```
model.add(
    Dropout(
        rate=hp.Float('dropout_1', min_value=0.0, max_value=0.5, default=0.25, step=0.05)
    )
)
model.add(
    Conv2D(
        filters=hp.Choice('num_filters', values=[32, 64], default=64),
        activation='relu',
        kernel_size=3
    )
)
bayesian_tuner = kerastuner.tuners.BayesianOptimization(
    hypermodel,
    objective='val_accuracy',
    seed=SEED,
    max_trials=MAX_TRIALS,
    executions_per_trial=EXECUTION_PER_TRIAL,
    directory=f'{directory}_bayesian',
    project_name=project_name
)
tuner.search(x_train, y_train,
             epochs=N_EPOCH_SEARCH,
             validation_split=0.1,
             verbose=0)

# Show a summary of the search
tuner.results_summary()

# Retrieve the best model.
best_model = tuner.get_best_models(num_models=1)[0]

# Evaluate the best model.
loss, accuracy = best_model.evaluate(x_test, y_test)
```

Show code?

High Test Error

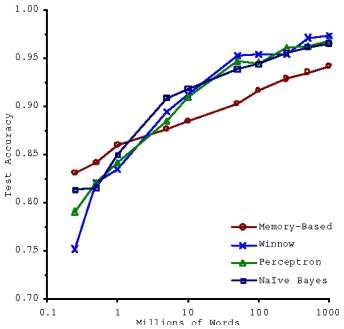


- This is the high-variance case —> Overfitting
- Regularization is good here (e.g., Dropout, L1, L2, …)
- Standard trick: **MORE DATA** (or data augmentations)

Note!

Data, data, data …

- Big dataset cannot be bad (if you have it, and time to use it):
  - Will be able to learn whatever is to be learned
  - Also: Less danger of overfitting



Results from Banko and Brill (2001)

Typical interpretations:
1. “It is not who has the best algorithm that wins. It’s who has the most data”
2. “Algorithms are overrated”

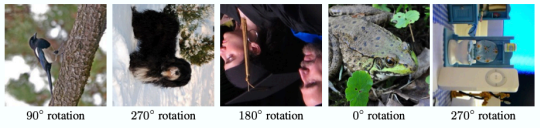
Self-supervision

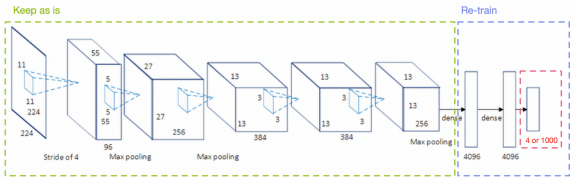
- **Self-supervision:** Supervised learning where the training data is automatically labelled.
- Typically done with the expectation that the internal representations learned will be beneficial for a different (sparsely labelled) problem. —> Transfer-learning into the original problem domain.
- **Standard example:** Filling in missing image patches.

UNSUPERVISED REPRESENTATION LEARNING BY PREDICTING IMAGE ROTATIONS

Spyros Gidaris, Praveer Singh, Nikos Komodakis

Learning rotations





Does it work?

(a) **Attention maps of supervised model** (b) **Attention maps of our self-supervised model**

Figure 3: Attention maps generated by an AlexNet model trained (**a**) to recognize objects (supervised), and (**b**) to recognize image rotations (self-supervised). In order to generate the attention map of a conv. layer we first compute the feature maps of this layer, then we raise each feature activation on the power p, and finally we sum the activations at each location of the feature map. For the conv. layers 1, 2, and 3 we used the powers $p = 1$, $p = 2$, and $p = 4$ respectively. For visualization of our self-supervised model's attention maps for all the rotated versions of the images see Figure 6 in appendix A.

Table 3: Evaluation of unsupervised feature learning methods on CIFAR-10. The *Supervised NIN* and the *(Ours) RotNet + conv* entries have exactly the same architecture but the first was trained fully supervised while on the second the first 2 conv. blocks were trained unsupervised with our rotation prediction task and the 3rd block only was trained in a supervised manner. In the *Random Init. + conv* entry a conv. classifier (similar to that of *(Ours) RotNet + conv*) is trained on top of two NIN conv. blocks that are randomly initialized and stay frozen. Note that each of the prior approaches has a different ConvNet architecture and thus the comparison with them is just indicative.

Method	Accuracy
Supervised NIN	92.80
Random Init. + conv	72.50
(Ours) RotNet + non-linear	89.06
(Ours) RotNet + conv	**91.16**
(Ours) RotNet + non-linear (fine-tuned)	91.73
(Ours) RotNet + conv (fine-tuned)	92.17
Roto-Scat + SVM Oyallon & Mallat (2015)	82.3
ExemplarCNN Dosovitskiy et al. (2014)	84.3
DCGAN Radford et al. (2015)	82.8
Scattering Oyallon et al. (2017)	84.7

Data Driven Refinement

- Understand your model
  - Measure error on train and test sets
  - Overfitting vs underfitting
- Understand the learning process
  - Use Tensorboard and/or other visualizations
    - Know what to look for, and how to fix problems
- Choose a models with relevant learning bias
  - Don't believe the hype
  - Feed Forward, CNN, RNN, more advanced tricks

Visualization

- Plot all the usual stuff, like loss, accuracy, other metrics (train and validation sets); both actual value and deltas between epochs can be useful. Log-scale sometimes beneficial when plotting
- Per layer:
  - Scalars (Mean, SD) + histograms of: Weights, biases, activations, gradients
  - Track sparsity of: Outputs, Gradients (tf.nn.zero_fraction)
- Spit out values of all "moving parts" (e.g., learning-rate), somewhere so it is easy to align with other per-epoch result

Example: Suboptimal learning in (too) deep FFNNs

- I found that layer activations from ReLUs were had:
  - Exhibiting problems with vanishing gradient
  - High fraction dead nodes
  - Some nodes potentially "overwhelming" the others in next layer
  - Behaviour changing considerably between epochs
- BatchNorm only partly solved it — still dead nodes
- LeakyReLU can also partly solve it, but vanishing gradients remains

Self-Normalizing Neural Networks

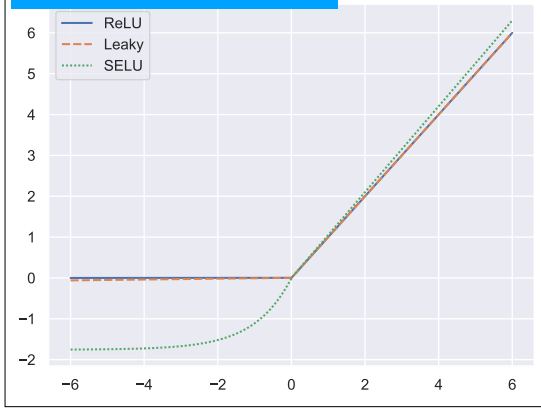
Günter Klambauer **Thomas Unterthiner** **Andreas Mayr**

Sepp Hochreiter
LIT AI Lab & Institute of Bioinformatics,
Johannes Kepler University Linz
A-4040 Linz, Austria
{klambauer,unterthiner,mayr,hochreit}@bioinf.jku.at

After lecture-comment:
I cut this a bit short. If you run the supplied selu.py, once with transfer RELU and once with SELU, you will see in Tensorboard that RELUs are very bad, have 40—50% neurons that do not give signal, and very small gradients in the initial layers. SELUs are self-regulating, and work well.

SELU

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$



$\alpha \sim 1.6732$
$\lambda \sim 1.0507$

Example with SELU

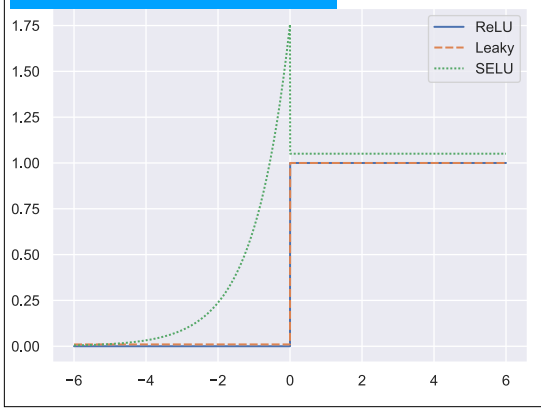
from Tensorboard?

37-1

After lecture-comment:
I cut this a bit short. If you run the supplied selu.py, once with transfer RELU and once with SELU, you will see in Tensorboard that RELUs are very bad, have 40—50% neurons that do not give signal, and very small gradients in the initial layers. SELUs are self-regulating, and work well.

SELU

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$



$\alpha \sim 1.6732$
$\lambda \sim 1.0507$

Example with SELU

from Tensorboard?

37-2

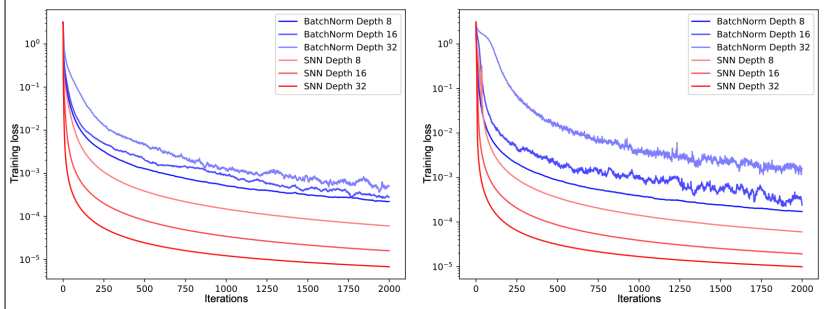


Figure 1: The left panel and the right panel show the training error (y-axis) for feed-forward neural networks (FNNs) with batch normalization (BatchNorm) and self-normalizing networks (SNN) across update steps (x-axis) on the MNIST dataset the CIFAR10 dataset, respectively. We tested networks with 8, 16, and 32 layers and learning rate 1e-5. FNNs with batch normalization exhibit high variance due to perturbations. In contrast, SNNs do not suffer from high variance as they are more robust to perturbations and learn faster.

38

Data Driven Refinement

- Understand your model
  - Measure error on train and test sets
  - Overfitting vs underfitting
- Understand the learning process
  - Use Tensorboard and/or other visualizations
    - Know what to look for, and how to fix problems
- Choose a models with relevant learning bias
  - Don't believe the hype
  - Feed Forward, CNN, RNN, more advanced tricks

39

CNN, RNN, FFNN, ???

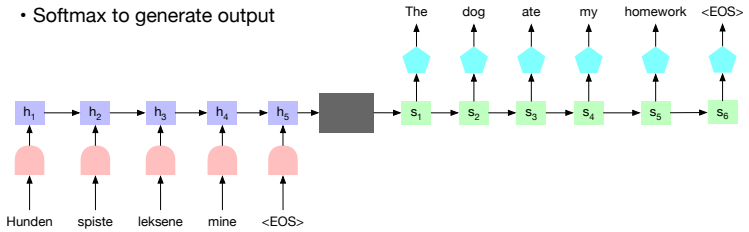
- CNNs are very good at data with localized correlation structure (e.g., neighbouring pixels in picture, or larger scale in hierarchy, i.e., [neighbours of …] * N pixels)
- RNN useful for sequences in particular when we can think of the domain as “Markov-model-like”. CNNs also much used in sequence data (… or combos of both)
- FFNNs can be useful if there is no known structure to relate to; rarely state-of-the-art (e.g., Kaggle) unless a bit deep (>5 layers).
- **USE THE DATA TO SELECT YOUR MODEL, AND CHECK IF LEARNING BIAS AND INFORMATION FLOW FITS!!!**

Sequence-to-Sequence

- Standard setup for, e.g., language translation
- For quite some time the “encode-decoder” setup with RNNs was the state-of-the-art.
- Now, moving towards attention-models…

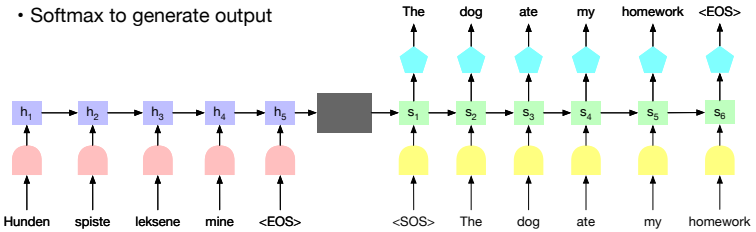
Encoder - Decoder

Building blocks:
- Word embeddings
- Recurrent model
- Softmax to generate output



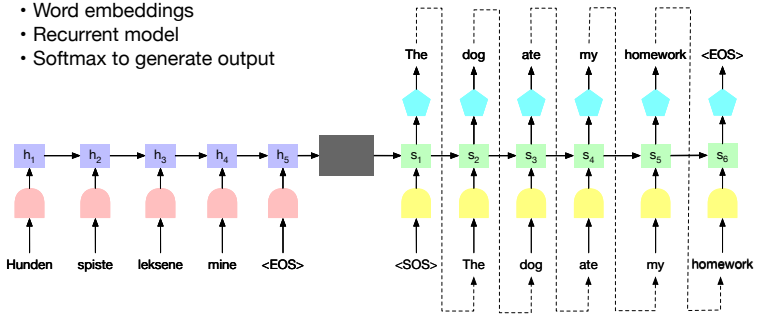
Encoder - Decoder

Building blocks:
- Word embeddings
- Recurrent model
- Softmax to generate output



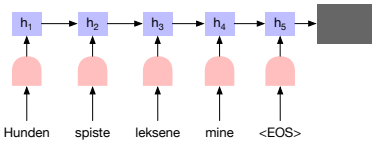
Encoder - Decoder

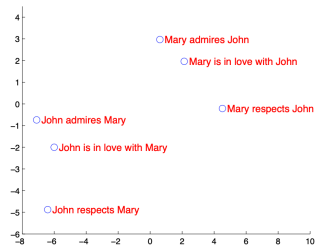
Building blocks:
- Word embeddings
- Recurrent model
- Softmax to generate output

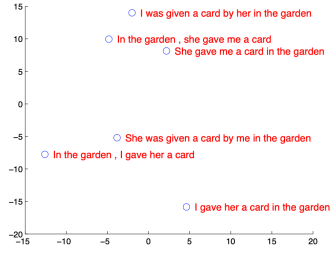


42-3

Does it work?

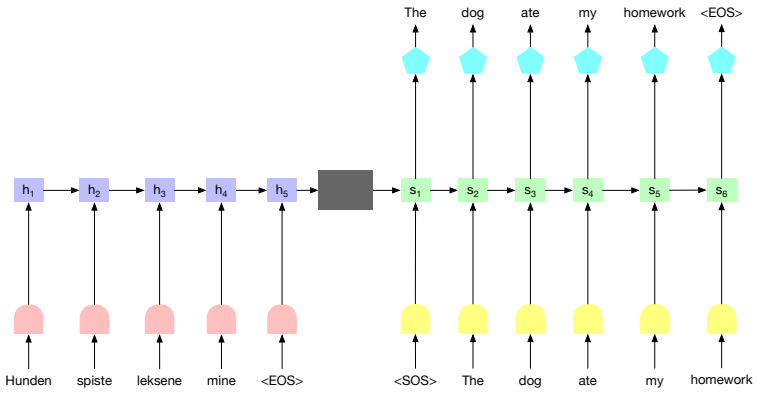






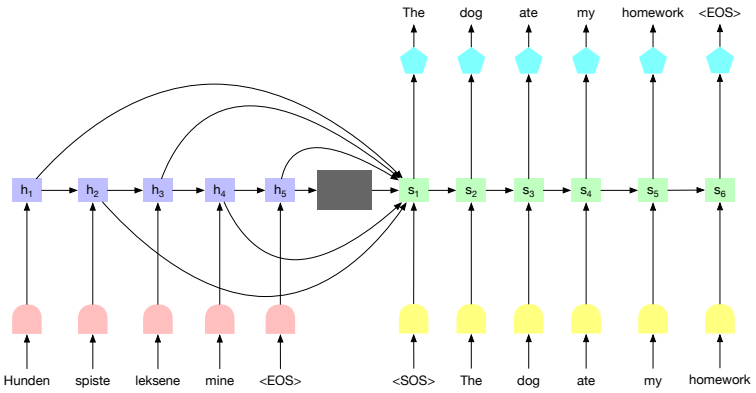
43

What's the problem



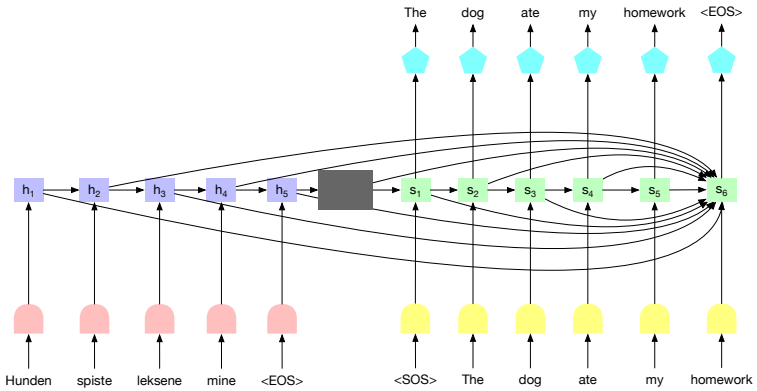
44-1

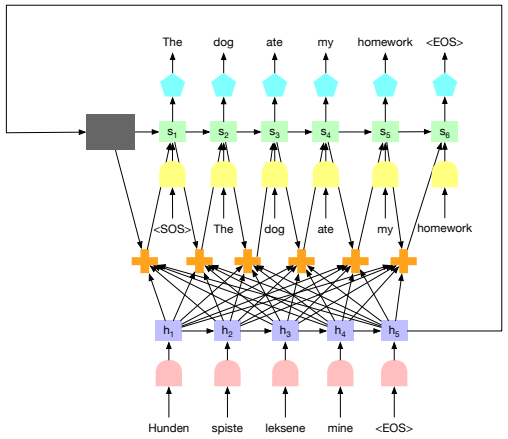
What's the problem



44-2

What's the problem





Plan:

- Input to $\boldsymbol{s}_t$ should be a weighted average of the values $\boldsymbol{h}_i$.
- The weight for h_i at position t should depend on previous state $\boldsymbol{s}_{t-1}$:

$$e_i(t) = g\left(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}\right)$$

for some function g.

- We will rescale $e_i(t)$ so that we have positive weights that sum to 1 using a softmax:

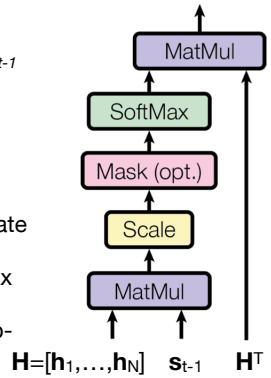
$$w_i(t) = \frac{\exp(e_i(t))}{\sum_i \exp(e_i(t))}$$

- Simple choice for g is

$$g\left(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}\right) = \boldsymbol{h}_i^\top \boldsymbol{s}_{t-1}$$

What goes on?

- Attention to vector $\boldsymbol{h}_i$ depends on vector $\boldsymbol{s}_{t-1}$ "pointing in similar direction" as $\boldsymbol{h}_i$.
  - Random vectors are approximately orthogonal in high-dim.
  - Typically only a few $e_i(t)$-values dominate
  - This is further strengthened by softmax
- Attention works like a differentiable lookup-table that averages over a (sparse) set.



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

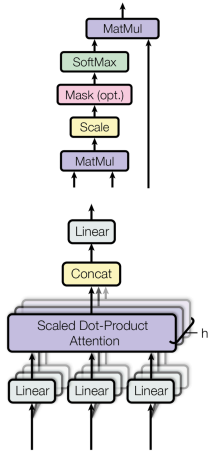
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

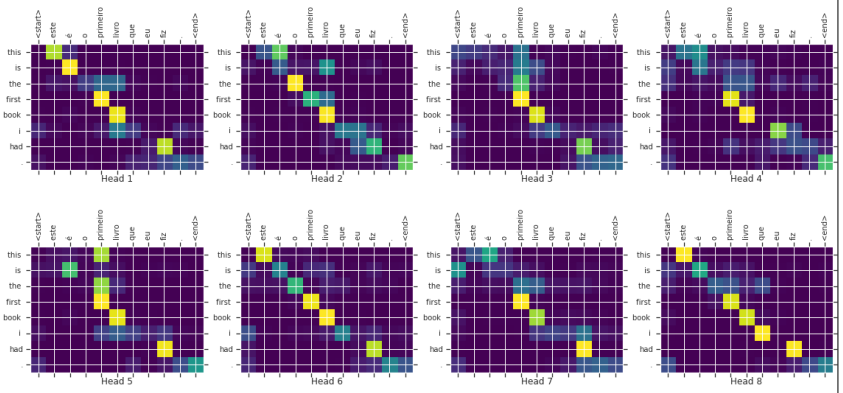
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Multi-head attention



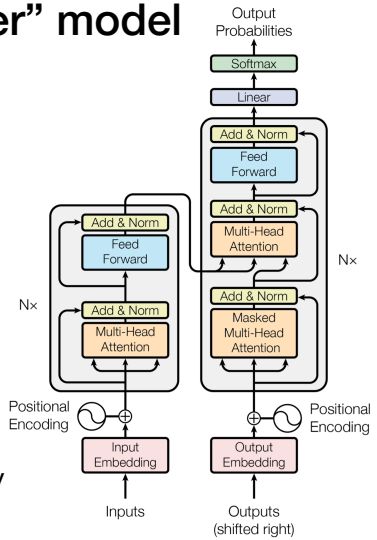
- The (single head) attention module typically pics out one or a few words as important
  - If more than one: Scaled average
- Sometimes several (sets of) words may be useful in "unrelated ways":
  - One (set) gives the word to translate
  - Another the associated grammar
  - … etc
- Solution: Several sets of attention, concatenated

Attention maps
Going from Portuguese to English



The full "Transformer" model

- No recurrence or convolutions
- Positional encoding + stacked substructures of attention and normalization
- Self-attention at encoder
- Masked self-attention and external attention at decoder
- Skip-connections to maintain gradients
- State of the art results in many (most) seq2seq problems



Summary

- Use needs to define metric-based goals
  - Be sure that you get what you need
- Build an end-to-end system
  - Start simple, check what to improve — cost/benefit
- Data-driven refinement
  - Understand your model and its strengths/weaknesses.
  - Optimize hyper-parameters
  - Understand the learning process
  - Choose a models with relevant learning bias