Bayesian Deep Learning with 10 % of the weights

Practical approach to Bayesian deep learning

Rob Romijnders

robromijnders.github.io

PyData Amsterdam, 2018

Outline

Outline

Problems with neural networks

Neural networks have three problems:

1. Neural networks give no **uncertainty** in predictions
   $\rightarrow$ easily fooled by **adversarial examples**
2. Neural networks have **millions of parameters**

Motivation

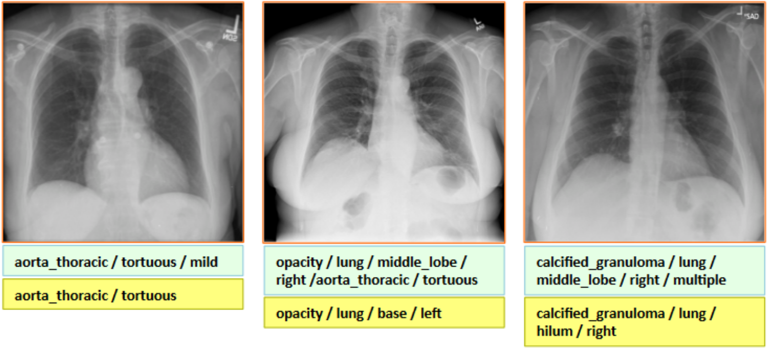


Figure: Uncertainty is important when making diagnoses

Motivation

Figure: Uncertainty is important when making a critical decision

Motivation

Figure: Uncertainty is important when prediction bitcoin

Adversarial attack

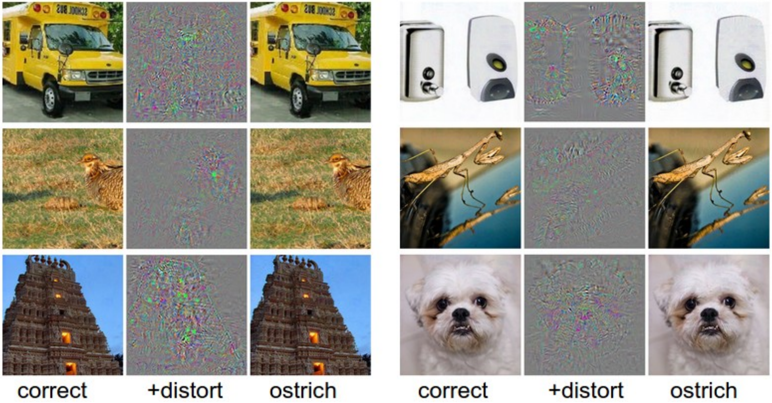


Figure: Uncertainty is necessary to find adversarial examples

Embedded applications

Figure: Pruning reduces the memory and computation usage (Pruning = dropping parameters)

Real time inference

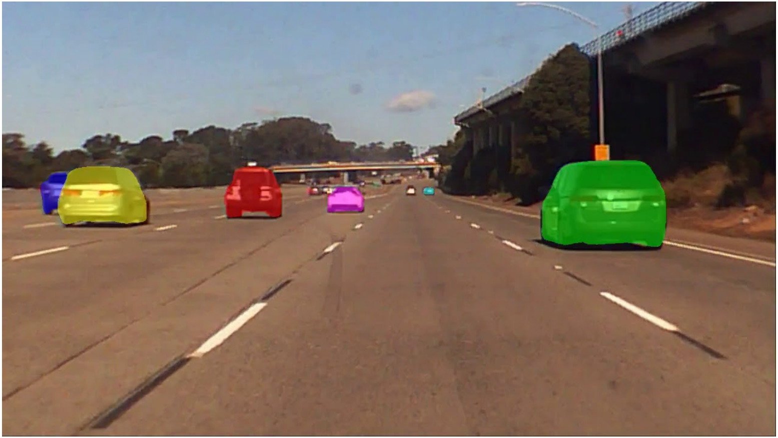


Figure: Pruning reduces the computation requirements

Outline

Pseudo code

In summary, this talk covers the following pseudo code

```
model = Model()

model.train(data)
model.prune()

# Actually, the next line is all we care about:
prediction, uncertainty = model.predict(input)
```

How to make a prediction?

Outline

Historical perspective

This content is not new: it has been around for decennia/centuries

Being Bayesian about neural networks

- Bayes lived in 18th century
- Variational inference for neural networks: Hinton and van Camp (1993)
- Bayesian Neural networks: Neal (1995)

Uncertainties for a model

Shannon published information theory in 1948

Outline

Bayes rule

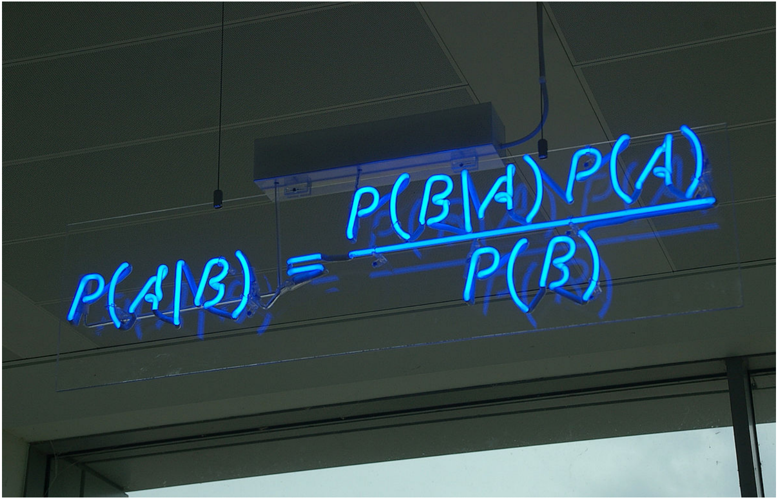


Figure: Every presentation on Bayesian machine learning has this image. So this presentation too

Bayes rule

$$posterior \propto likelihood \times prior$$

$$p(w|data) \propto p(data|w)p(w)$$

$$logp(w|data) = logp(data|w) + logp(w) + constant$$

Bayes rule

We have been using Bayes' rule all the time

Weight decay .. L2 regularisation

$$
\begin{aligned}
-\mathit{log}\ \mathit{posterior} &= -\mathit{log}\ \mathit{likelihood} \quad -\mathit{log}\ \mathit{prior} \quad +\mathit{constant} \\
\mathit{loss} &= \mathit{classification}\ \mathit{loss} \quad +\lambda \sum_i w_i^2 \quad +\mathit{constant}
\end{aligned}
$$

Stochastic gradient descent

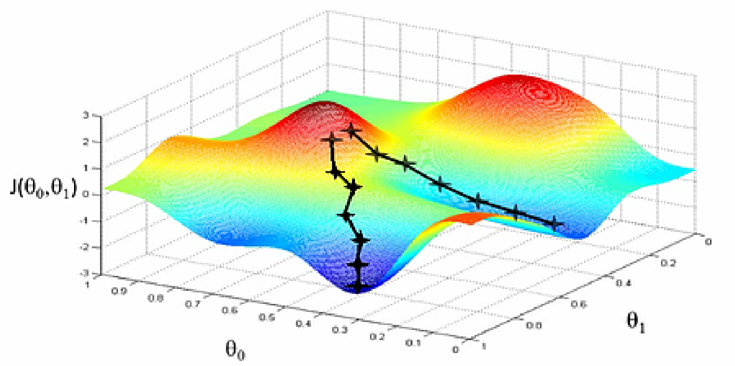


Figure: But we inferred only one parameter vector

From this...

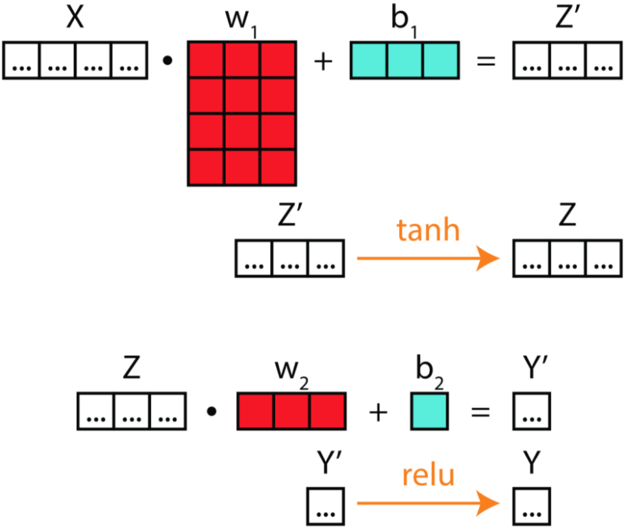


Figure: Used with kind permission of Eric Ma

...to this

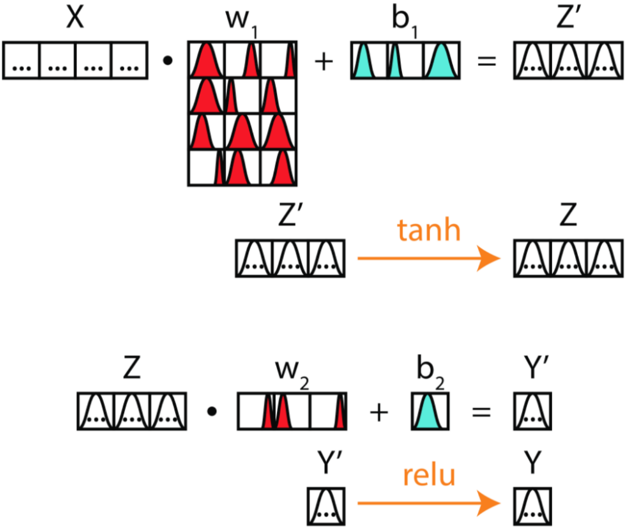


Figure: Used with kind permission of Eric Ma

Parameters of a Gaussian

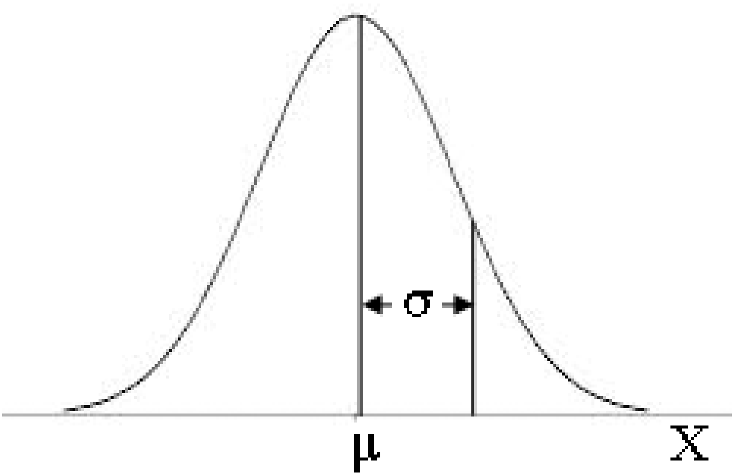


Figure: For a Gaussian, we need parameters μ and σ

Outline

Posterior probability

The parameter posterior will:

- Enable more samples for prediction $\rightarrow$ uncertainty over prediction
- Tell us which parameters have high zero-probability $\rightarrow$ pruning

Loss functions

Loss functions

old loss

$$= \textit{classification loss} + \sum_i \underbrace{\lambda w_i^2}_{\text{L2 penalty}} + \textit{constant}$$

new loss

$$= \textit{classification loss} + \sum_i \underbrace{\frac{1}{2}\lambda \mu_i^2}_{\text{L2 penalty}} \underbrace{- \log \sigma_i + \frac{1}{2}\lambda \sigma_i^2}_{\text{penalty on } \sigma} + \textit{constant}$$

Interpretation

- L2 penalty on the parameter remains

Intuition

$$\mathit{loss} = \underbrace{\mathit{classification\ loss} + \sum_i \frac{1}{2}\lambda\mu_i^2}_{\text{loss on location of weights}} \underbrace{-\log\sigma_i + \frac{1}{2}\lambda\sigma_i^2}_{\text{loss on }\sigma} + \mathit{constant}$$

Summary

- **What do we care about?**
  Uncertainties and pruning
- **How we do it?**
  Find many parameter vectors and average
- **How we do that?**
  Bayesian inference
- **How we do that?**
  Approximate the parameter posterior
- **What do we do in the end?**
  Minimize the loss function on the previous slide

Outline

Use entropy as uncertainty metric

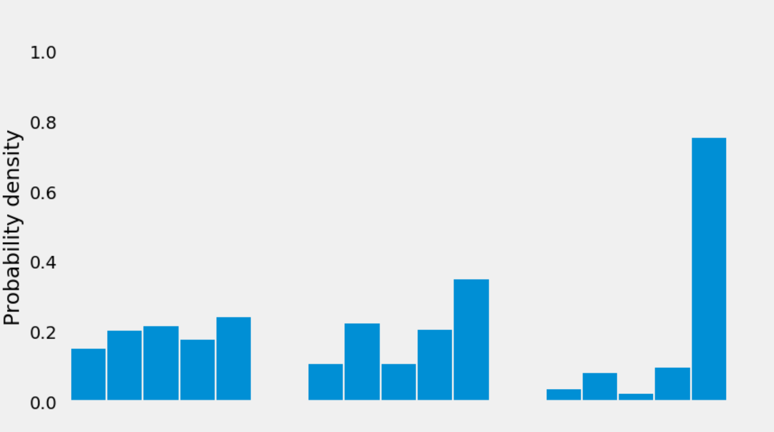


Figure: Which prediction has least uncertainty?

Use entropy as uncertainty metric

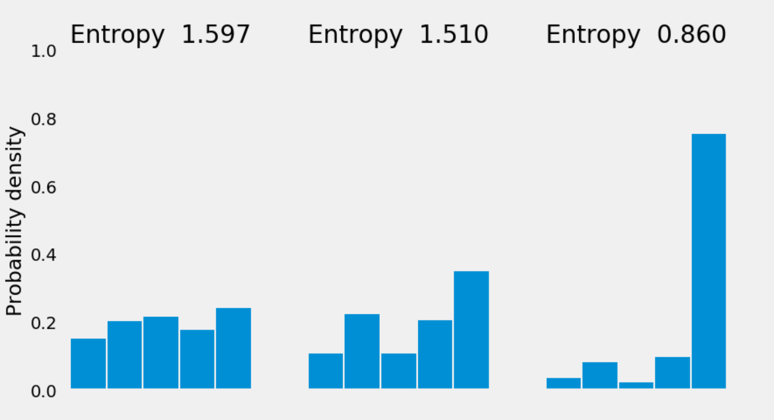


Figure: Which prediction has least uncertainty?

Outline

From this...

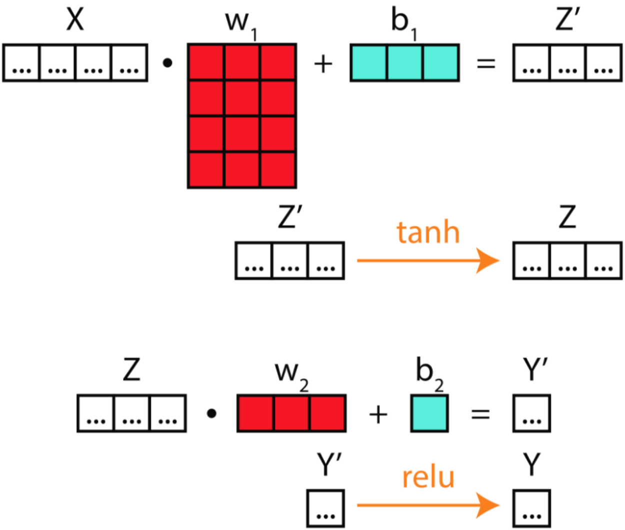


Figure: Used with kind permission of Eric Ma

From this...

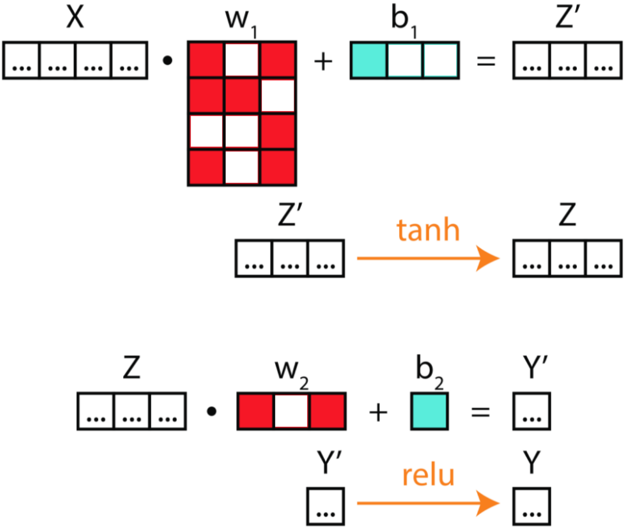


Figure: Used with kind permission of Eric Ma

Pruning according to posterior

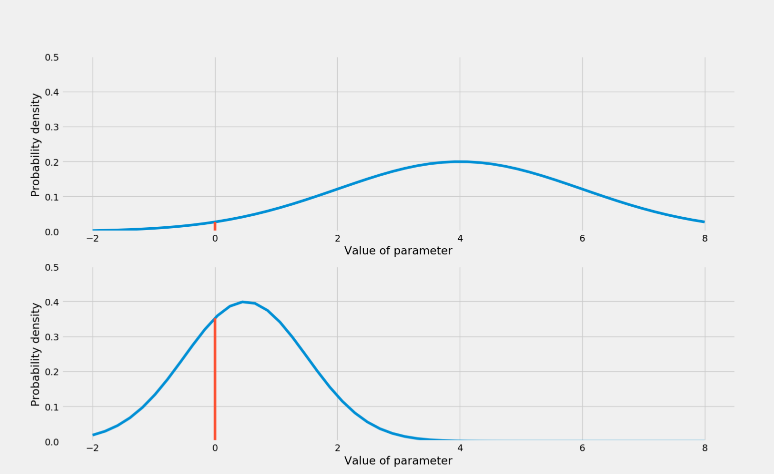


Figure: Which parameter would you rather prune?

Data sets

Fun

No deep learning project is complete without **MNIST**

Serious

Two most common applications of deep learning:

- Image recognition: **CIFAR10** data set
- Time series classification: **UCR - ECG's**
  - Train set only 500 time series → Bayesian's don't overfit

MNIST examples

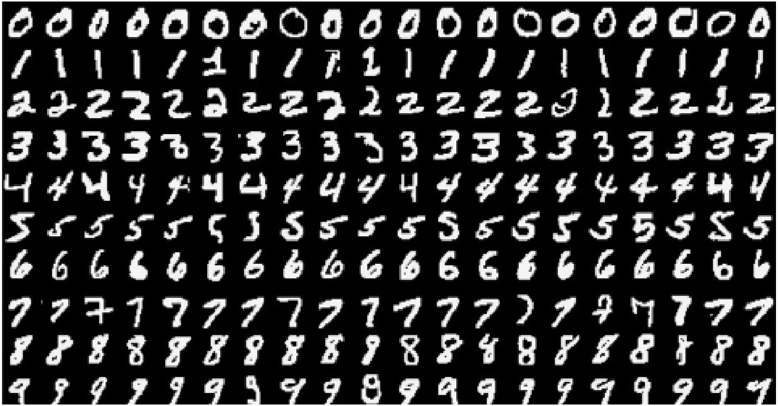


Figure: Examples of MNIST. Train set: 50k samples. Test set: 10k samples

CIFAR examples

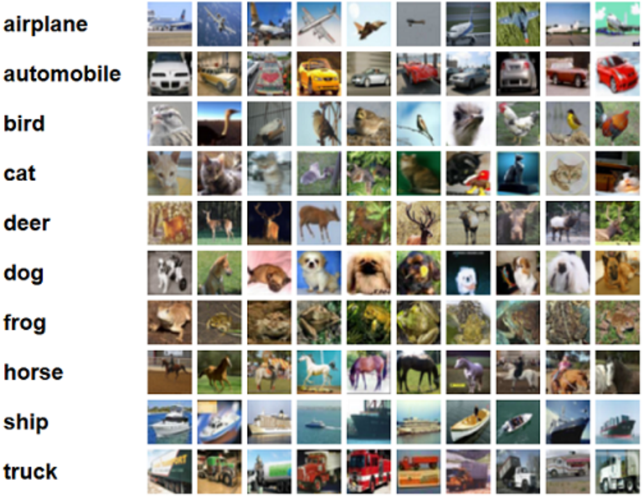


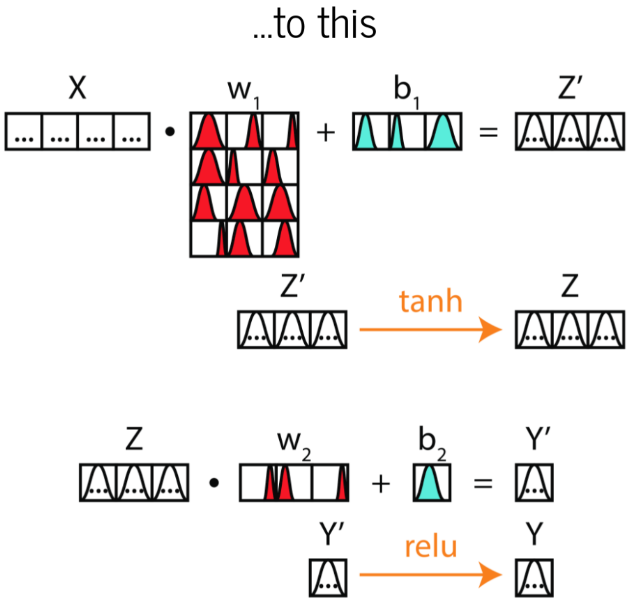
Figure: Examples of CIFAR. Train set: 50k samples. Test set: 10k samples

ECG examples

??? MAKE ECG EXAMPLES HERE

Outline

Remember the model



Pruning MNIST

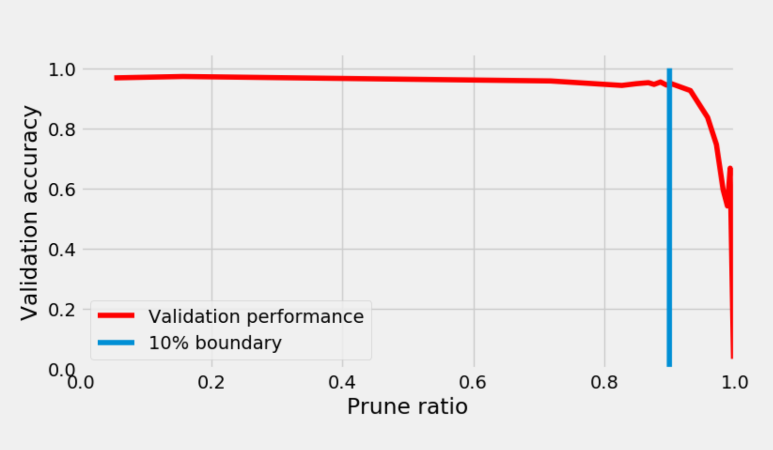


Figure: Pruning curve for MNIST

Pruning CIFAR

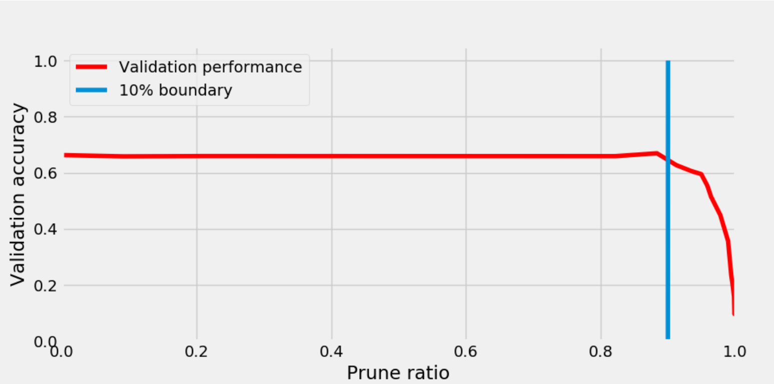


Figure: Pruning curve for CIFAR

Pruning ECG

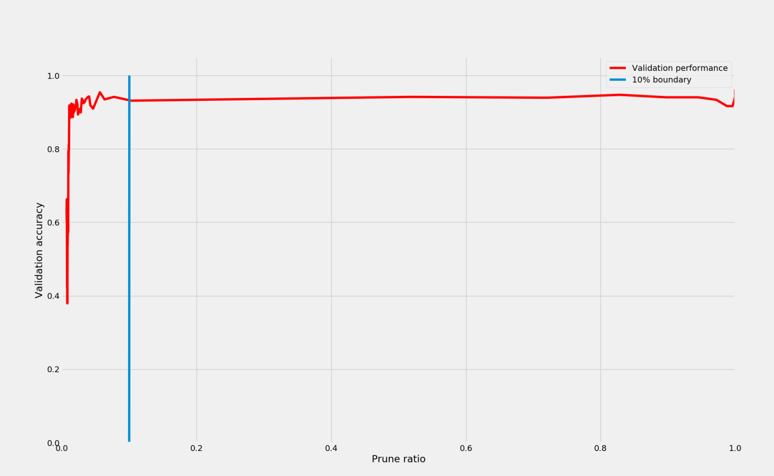


Figure: Pruning curve for ECG

Outline

Experiment uncertainty

How to mutilate images to raise uncertainty?

- Add noise
- Warping

Here will be slides with experiments to show uncertainty on CIFAR10 when we add noise or rotate or warp

Take aways

- Get uncertainty for critical predictions
- Robust against adversarial attacks
- Prune networks for small memory and small compute

Questions?

robromijnders.github.io

Material

`github.com/RobRomijnders/weight_uncertainty`

- All code
- Further reading
- More explanation

Additional slides

Learning the sigma's

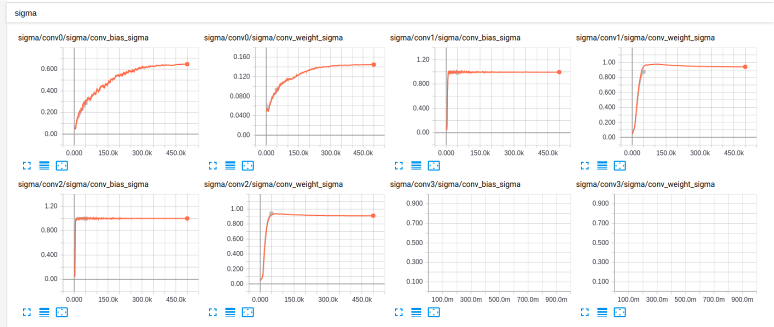
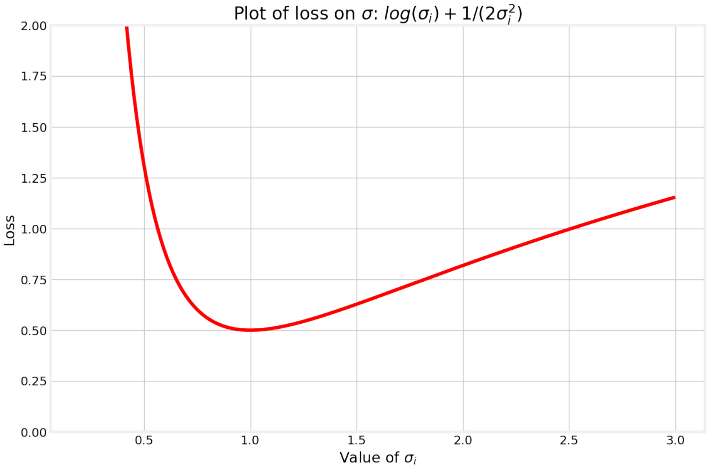


Figure: The VI objective increases the sigma's by itself!!

Loss on σ

What does the loss for σ look like?



Make predictions

Sampling

Make multiple predictions with sampled parameters. One can think of this sampling as an ensemble method

```
def make_prediction(input):
    for param_vec in param_vecs:
        yield model.get_output(input, param_vec)
prediction = np.mean(make_prediction(input))
```

Pseudo code

Pseudo code for training our neural network

```
# OLD CODE
while not converged:
  # Get the loss
  x, y = sample_batch()
  loss = loss_function(x, y, w)

  #Update the parameters
  w_grad = gradient(loss, w)
  w = update(w, w_grad)

##########################################################
# NEW CODE
while not converged:
  # Get the loss
  x, y = sample_batch()
  w = approximation.sample()
  loss = loss_function(x, y, w)

  # Update the approximation
  w_grad = gradient(loss, w)
  approximation = update(approximation, w_grad)
```

```
while not converged:
    # Get the loss
    x, y = sample_batch()
    w = approximation.sample()
    loss = loss_function(x, y, w)

    # Update the approximation
    w_grad = gradient(loss, w)
    approximation = update(approximation, w_grad)
```

Research

Pruning: speed

Bayesian compression for deep learning, Louizos @ NIPS2017

Uncertainty: adversarial attack

Adversarial phenomenon in Bayesian deep learning, Rawat, 2017

Gaussian approximation

Approximate with a normal distribution

- Captures local structure of the posterior, which indicates the uncertainty
- Simple for parameter pruning

Anything is better than point estimation !!!