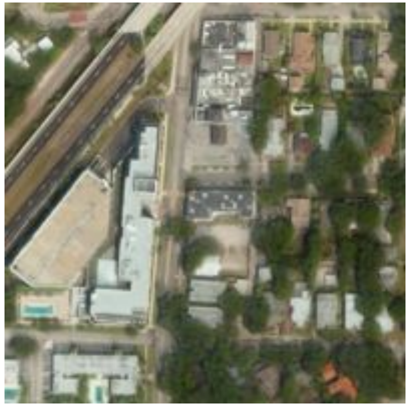
Active Learning for Pixel-wise Prediction

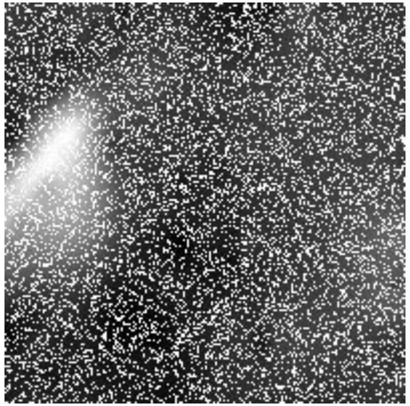
January 31st, 2020

Email: abdill@andrew.cmu.edu

Depth Completion Task

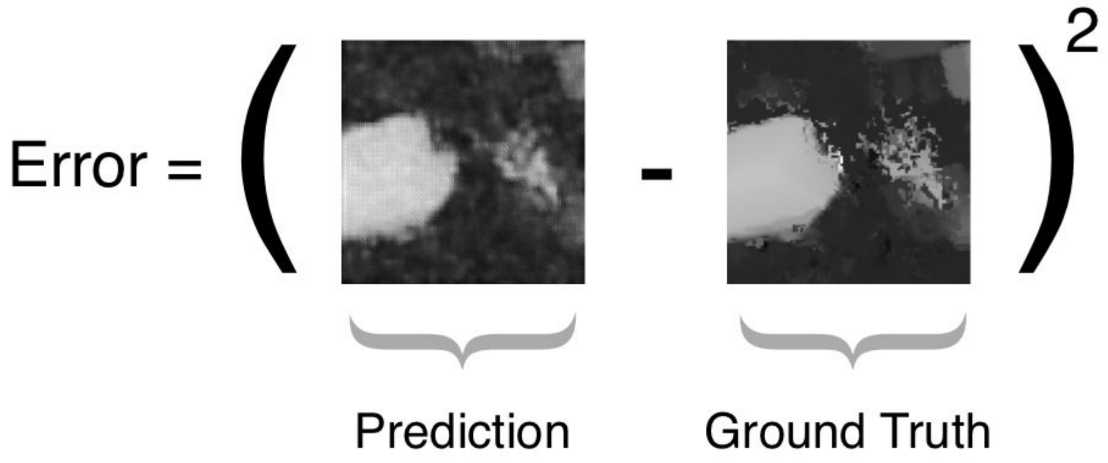
- Input: RGB aerial image and corresponding depth map with some percentage replaced with zeros.





Depth Completion Task

- Output: Predicted depth map

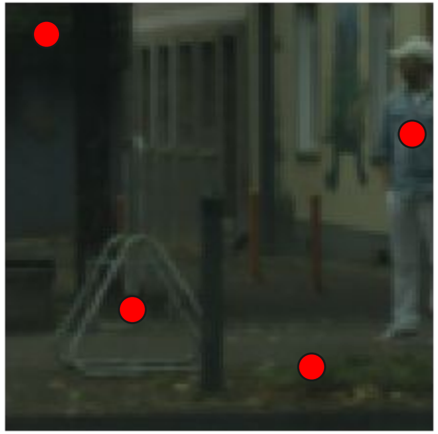


Depth Completion is a specific form of a more general problem.

Given an image and a prediction task, select individual pixels for annotation so that given this information along with the image, your learning task is easiest.

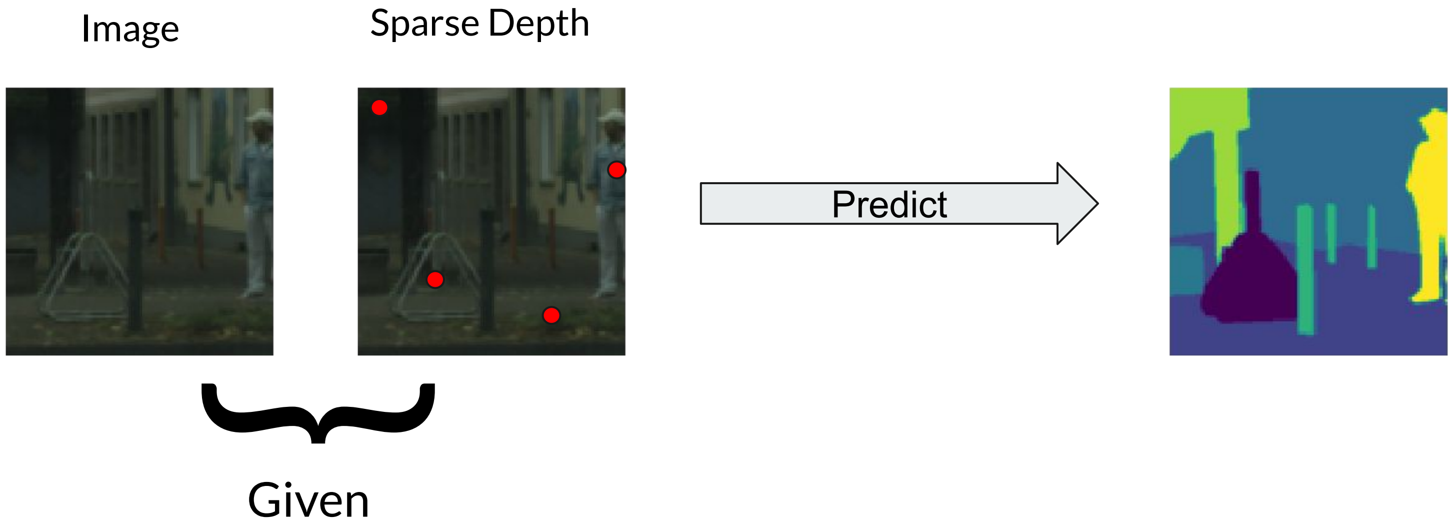
Semantic Segmentation

Input Image



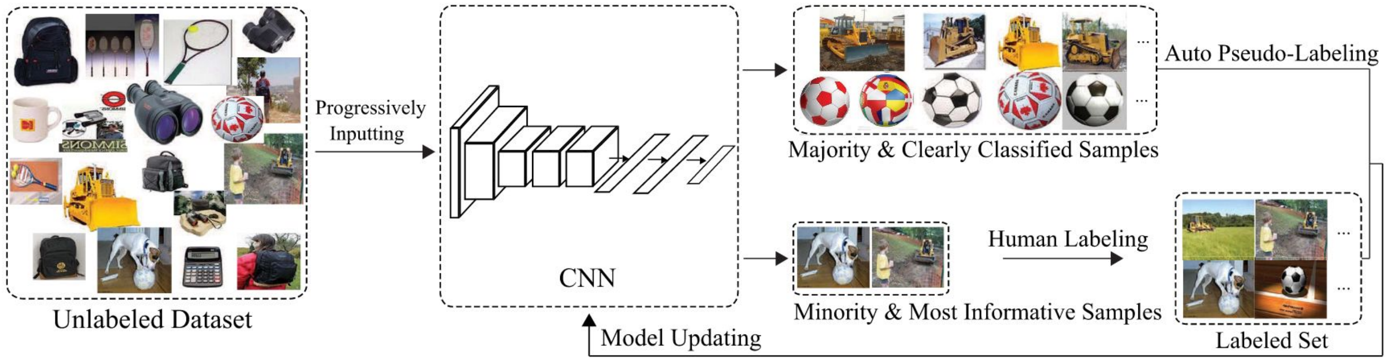
Suggested Annotation Locations

Semantic Segmentation



Active Learning and its Shortcomings

Active Learning techniques for image mostly focus on selecting a subset of *images* for annotation, not individual pixels for annotation.



Naive Model

Naive Masking

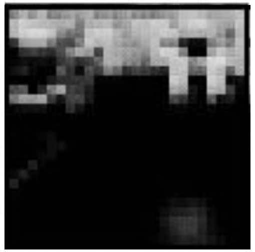
$$\tilde{S} = \underset{\uparrow}{M} \odot \underset{\uparrow}{S}$$

$$\underset{\text{Output from a neural net}}{} \qquad \underset{\text{Ground truth segmentation}}{}$$

$$\hat{S} = h_\theta(I, \tilde{S})$$

How do we encourage a sparse mask?

Error = (



-

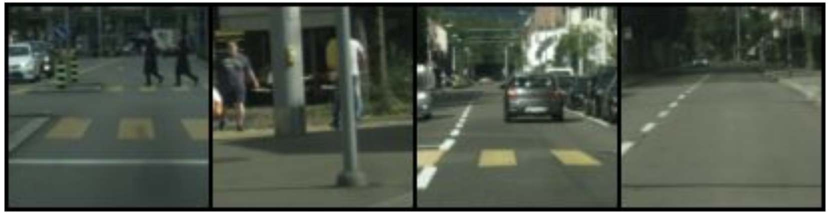
$)^2 + \lambda \|M\|_1$

Prediction

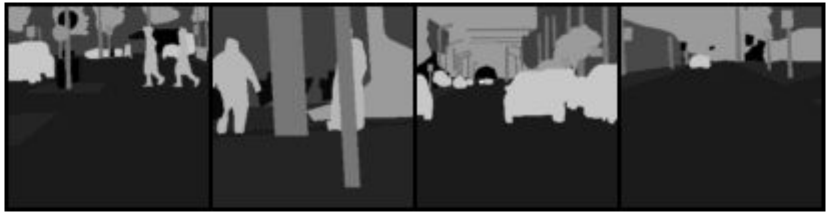
Ground Truth

Results

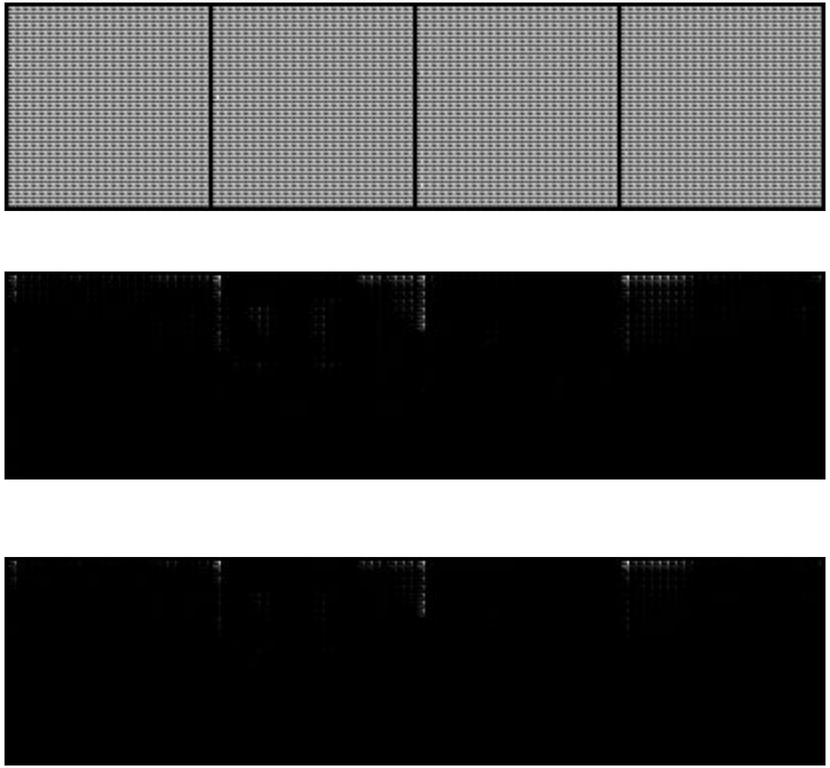
Input Images



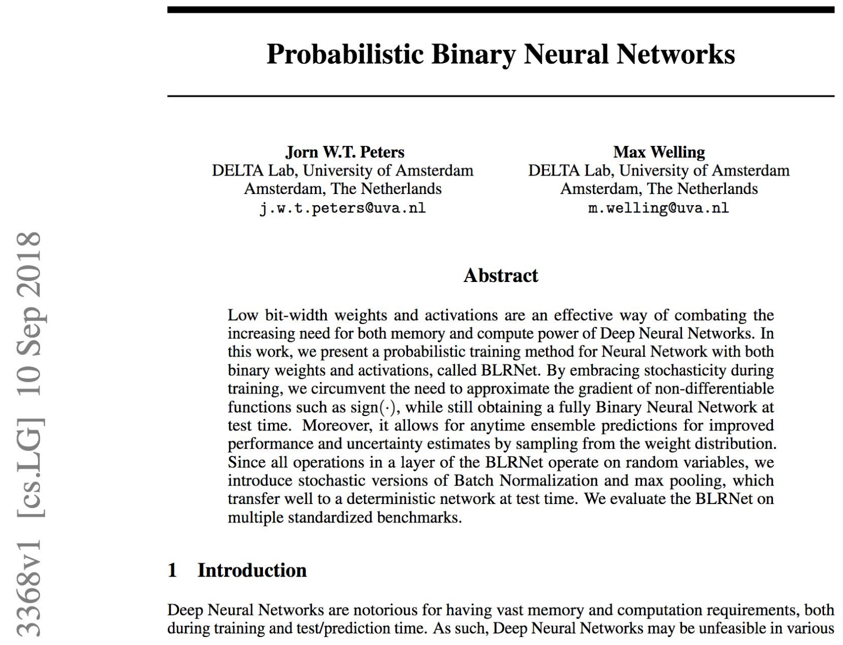
Ground Truth Segmentation



Learned Masks per Epoch



Possible Remedy



Probabilistic Binary Neural Networks

Jorn W.T. Peters
DELTA Lab, University of Amsterdam
Amsterdam, The Netherlands
j.w.t.peters@uva.nl

Max Welling
DELTA Lab, University of Amsterdam
Amsterdam, The Netherlands
m.welling@uva.nl

Abstract

Low bit-width weights and activations are an effective way of combating the increasing need for both memory and compute power of Deep Neural Networks. In this work, we present a probabilistic training method for Neural Network with both binary weights and activations, called BLRNet. By embracing stochasticity during training, we circumvent the need to approximate the gradient of non-differentiable functions such as sign(·), while still obtaining a fully Binary Neural Network at test time. Moreover, it allows for anytime ensemble predictions for improved performance and uncertainty estimates by sampling from the weight distribution. Since all operations in a layer of the BLRNet operate on random variables, we introduce stochastic versions of Batch Normalization and max pooling, which transfer well to a deterministic network at test time. We evaluate the BLRNet on multiple standardized benchmarks.

1 Introduction

Deep Neural Networks are notorious for having vast memory and computation requirements, both during training and test/prediction time. As such, Deep Neural Networks may be unfeasible in various

Probabilistic Mask Approach

$$S = M \odot D$$

$$M_{ij} \sim \mathbb{P}(M_{ij} = 1) = \sigma(W_{ij})$$

$$S_{ij} \sim \mathcal{N}(D_{ij}\mathbb{E}[M_{ij}], D_{ij}^2\mathbb{V}[M_{ij}])$$

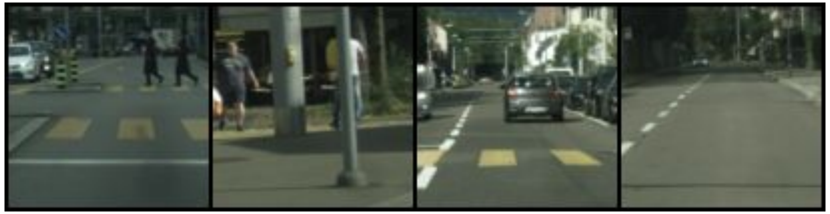
Reparameterization Trick

$$S_{ij} = D_{ij}\sigma(W_{ij}) + D_{ij}^2\sigma(W_{ij})(1 - \sigma(W_{ij}))\epsilon$$

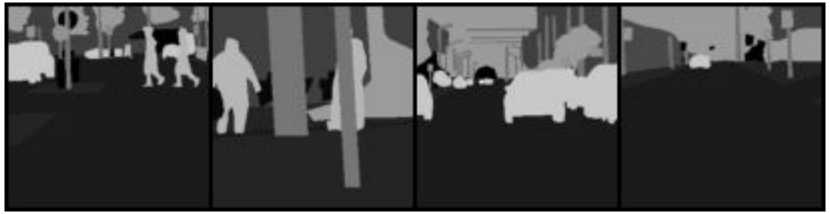
$$\epsilon \sim \mathcal{N}(0, 1)$$

Probabilistic Masking

Input Images

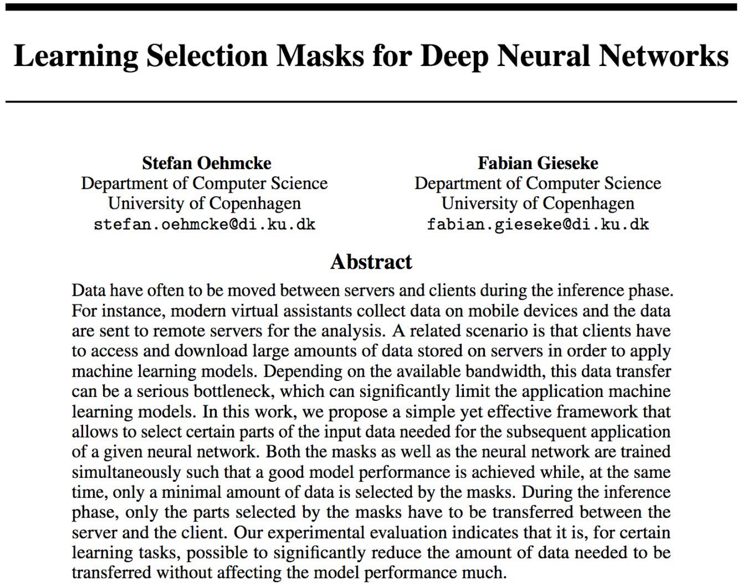


Ground Truth Segmentation



Clues From Related Work

Recent relevant work suggests part of the path forward.



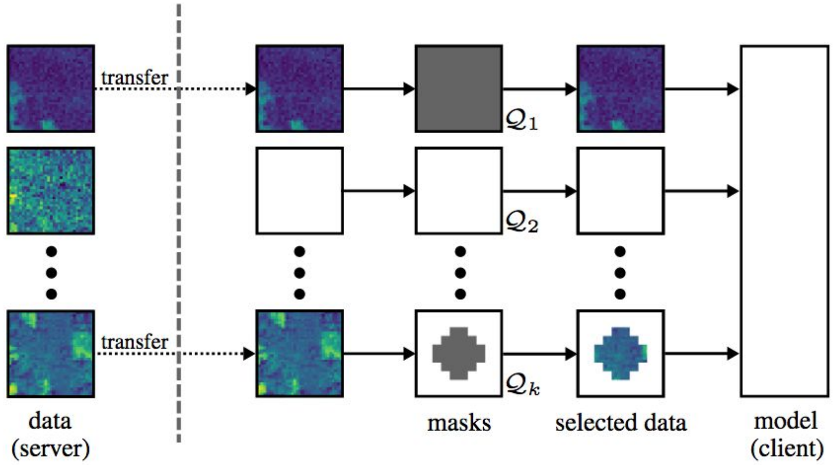
Learning Selection Masks for Deep Neural Networks

Stefan Oehmcke
Department of Computer Science
University of Copenhagen
stefan.oehmcke@di.ku.dk

Fabian Gieseke
Department of Computer Science
University of Copenhagen
fabian.gieseke@di.ku.dk

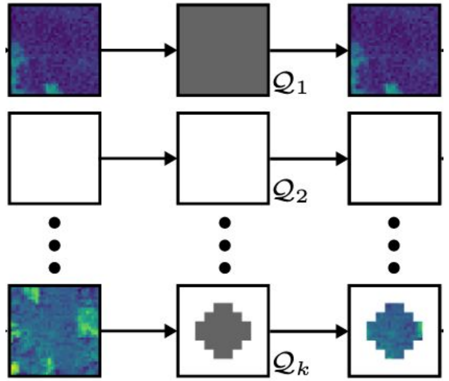
Abstract

Data have often to be moved between servers and clients during the inference phase. For instance, modern virtual assistants collect data on mobile devices and the data are sent to remote servers for the analysis. A related scenario is that clients have to access and download large amounts of data stored on servers in order to apply machine learning models. Depending on the available bandwidth, this data transfer can be a serious bottleneck, which can significantly limit the application machine learning models. In this work, we propose a simple yet effective framework that allows to select certain parts of the input data needed for the subsequent application of a given neural network. Both the masks as well as the neural network are trained simultaneously such that a good model performance is achieved while, at the same time, only a minimal amount of data is selected by the masks. During the inference phase, only the parts selected by the masks have to be transferred between the server and the client. Our experimental evaluation indicates that it is, for certain learning tasks, possible to significantly reduce the amount of data needed to be transferred without affecting the model performance much.



What is their task?

- Start with a set of arrays of the same size as their images initialized with random values between (0+ε, 1-ε).
- Want to update this arrays to learn “selection” masks to trade of the success of their ML task and the sparsity of their masks.



Rephrased with our Notation

$$\tilde{I} = \underset{\uparrow}{M} \odot \underset{\uparrow}{I}$$

Selection mask from our bank

Full query image

$$\hat{S} = h_\theta(\tilde{I})$$

How do they get binary masks?

- They use a trick for sampling from categorical distributions known as the “Gumbel Softmax Reparameterization”.
- Say π_i is the probability that pixel i of the mask is a 1. Naively, we would discretize this to get a mask by doing the following:

$$M_i = \text{argmax}_{\{0,1\}} \log \pi_i$$

How do they get binary masks?

- Unfortunately, this isn't differentiable. We can approximate it with a softmax distribution after "softening" the probabilities by adding noise in the form of IID draws from a Gumbel distribution.

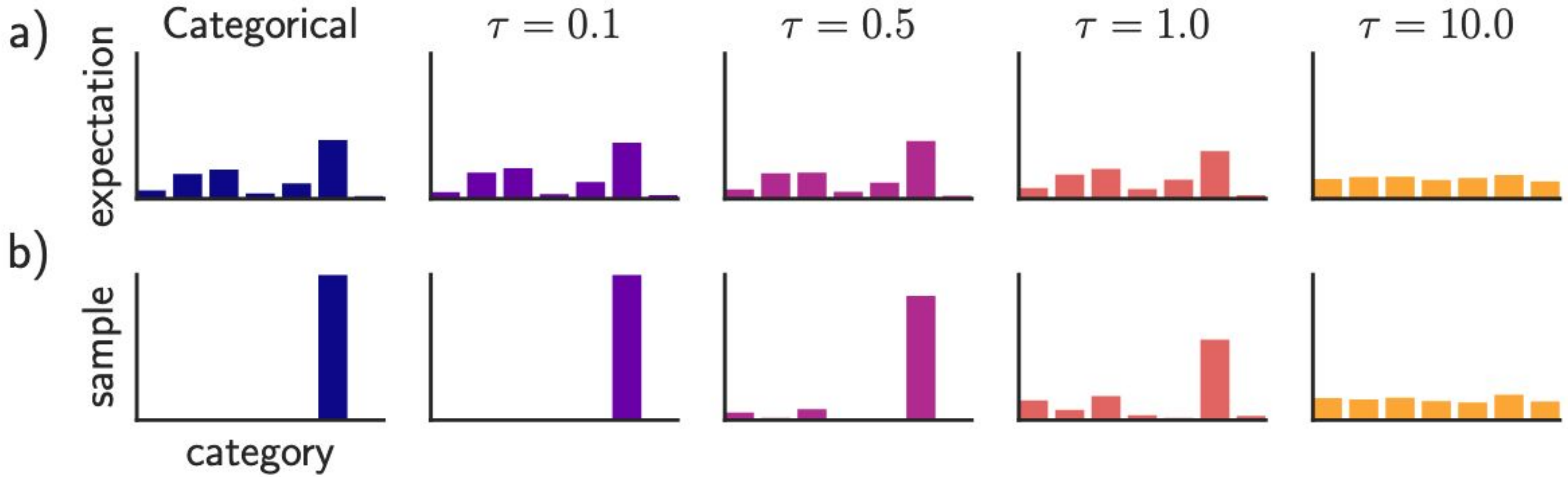
$$M_i = \text{softmax}\big(g_0 + \log \pi_i,\, g_1 + \log(1 - \pi_i)\big)$$

How do they get binary masks?

- Finally, we want to control how “peaked” our approximation is to ensure a stable learning process. We do this by introducing a temperature parameter τ.

$$M_i = \mathrm{softmax}\left(\frac{g_0 + \log \pi_i}{\tau}, \frac{g_1 + \log(1 - \pi_i)}{\tau}\right)$$

What is the effect of the temperature parameter?



Naive Masking

$$\tilde{S} = M \odot S$$

Output from a neural net parameterized Gumbel Softmax

Ground truth segmentation

$$\hat{S} = h_\theta(I, \tilde{S})$$

Gumbel Softmax Results

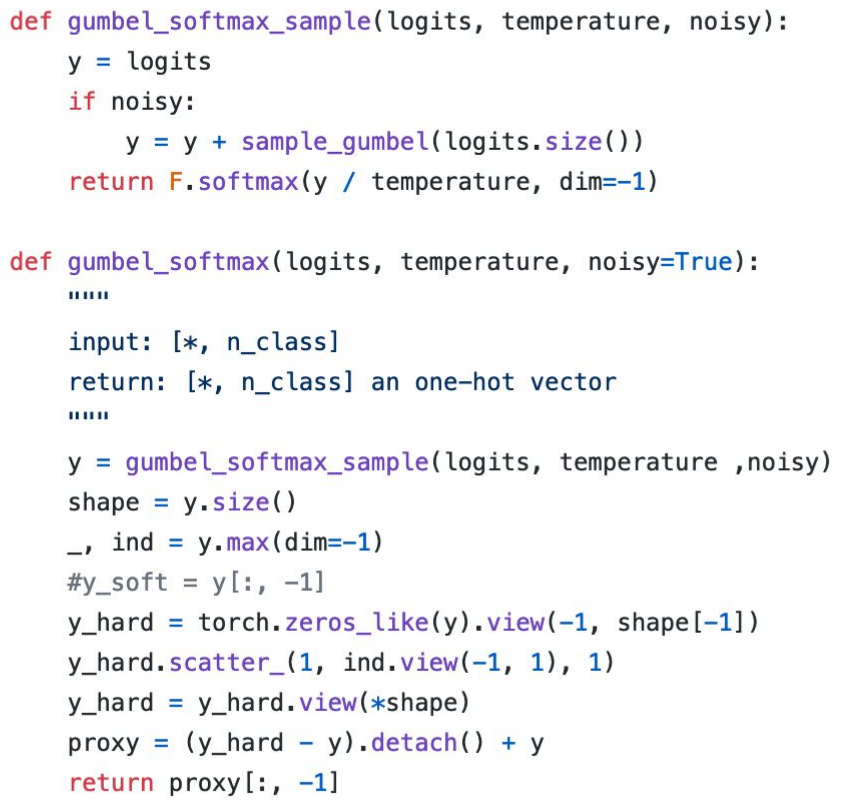
How do we get this idea to work in PyTorch?

```python
def sample_gumbel(shape, eps=1e-20):
    U = torch.rand(shape).cuda()
    return -Variable(torch.log(-torch.log(U + eps) + eps))

def gumbel_softmax_sample(logits, temperature):
    y = logits + sample_gumbel(logits.size())
    return F.softmax(y / temperature, dim=-1)

def gumbel_softmax(logits, temperature):
    """
    input: [*, n_class]
    return: [*, n_class] an one-hot vector
    """
    y = gumbel_softmax_sample(logits, temperature)
    shape = y.size()
    _, ind = y.max(dim=-1)
    y_hard = torch.zeros_like(y).view(-1, shape[-1])
    y_hard.scatter_(1, ind.view(-1, 1), 1)
    y_hard = y_hard.view(*shape)
    return (y_hard - y).detach() + y
```

How do we get this idea to work in PyTorch?

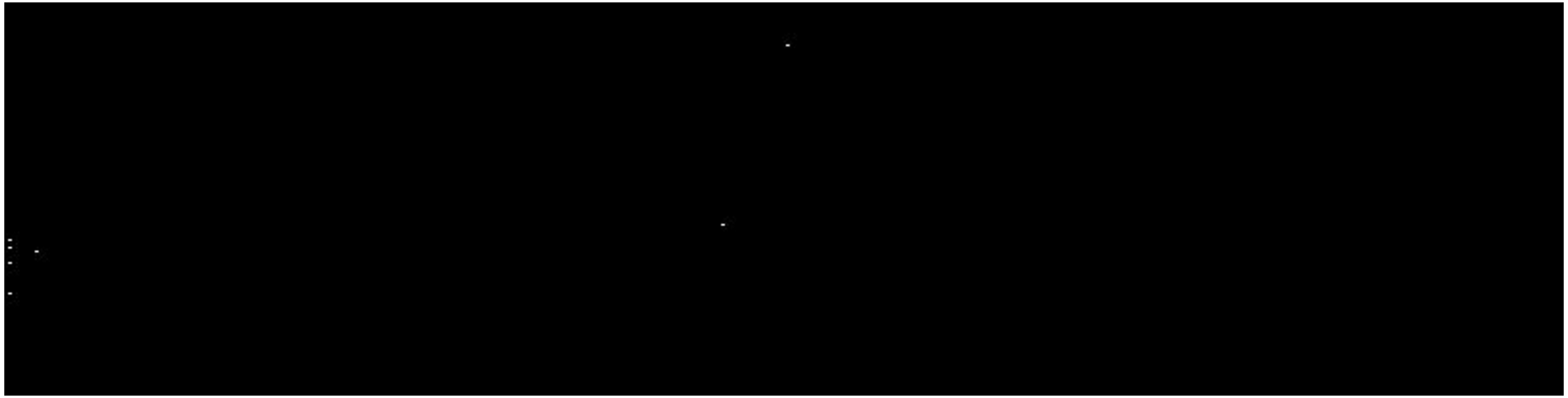


```python
def gumbel_softmax_sample(logits, temperature, noisy):
    y = logits
    if noisy:
        y = y + sample_gumbel(logits.size())
    return F.softmax(y / temperature, dim=-1)

def gumbel_softmax(logits, temperature, noisy=True):
    """
    input: [*, n_class]
    return: [*, n_class] an one-hot vector
    """
    y = gumbel_softmax_sample(logits, temperature ,noisy)
    shape = y.size()
    _, ind = y.max(dim=-1)
    #y_soft = y[:, -1]
    y_hard = torch.zeros_like(y).view(-1, shape[-1])
    y_hard.scatter_(1, ind.view(-1, 1), 1)
    y_hard = y_hard.view(*shape)
    proxy = (y_hard - y).detach() + y
    return proxy[:, -1]
```

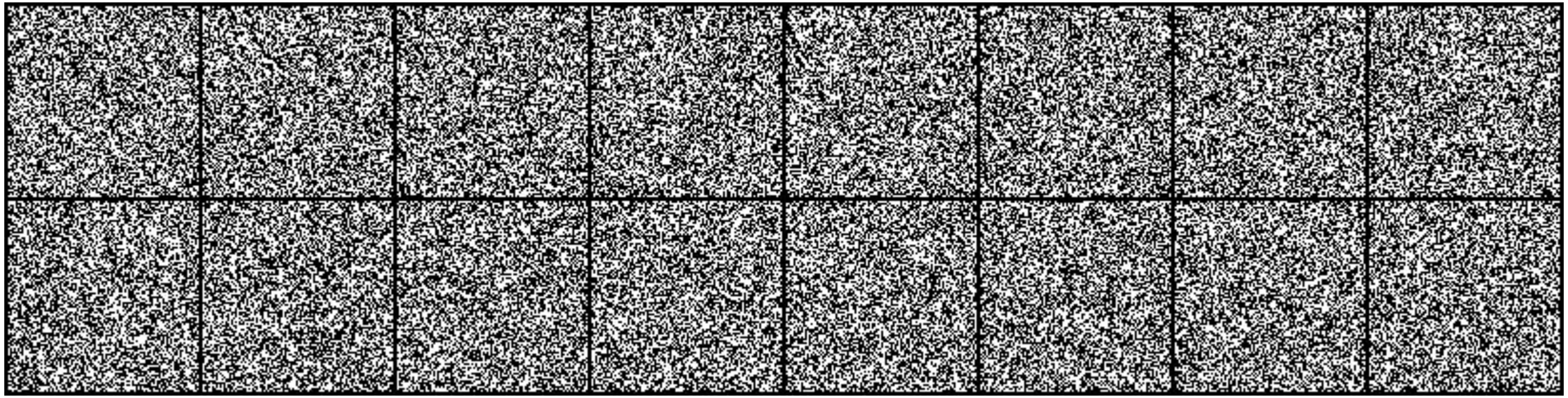
Noise Scheduling

- As previous work has discussed, varying the amount of Gumbel noise leads to an exploration-exploitation trade-off.
- Demonstrated schedule is to eliminate noise for every fifth example.



Noise Scheduling

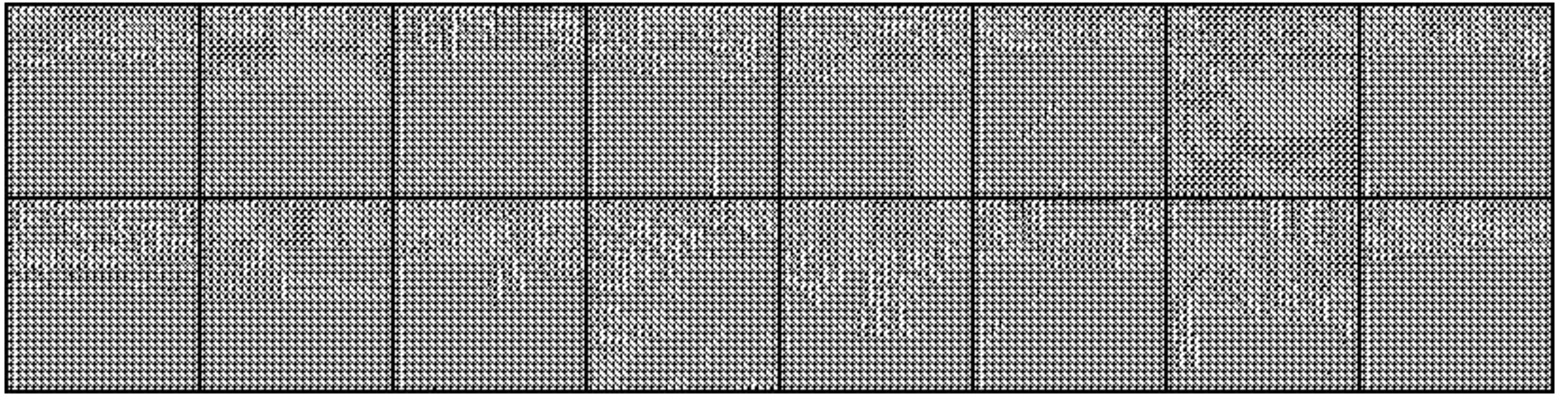
- As previous work has discussed, varying the amount of Gumbel noise leads to an exploration-exploitation trade-off.
- Demonstrated schedule is to eliminate noise for every fifth example.

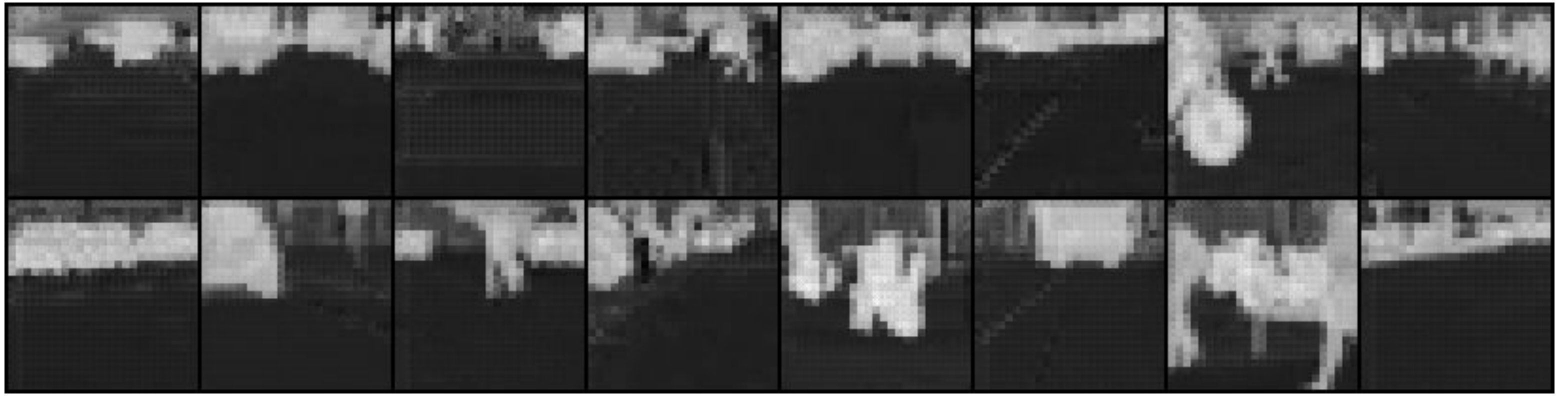


Temperature Scheduling

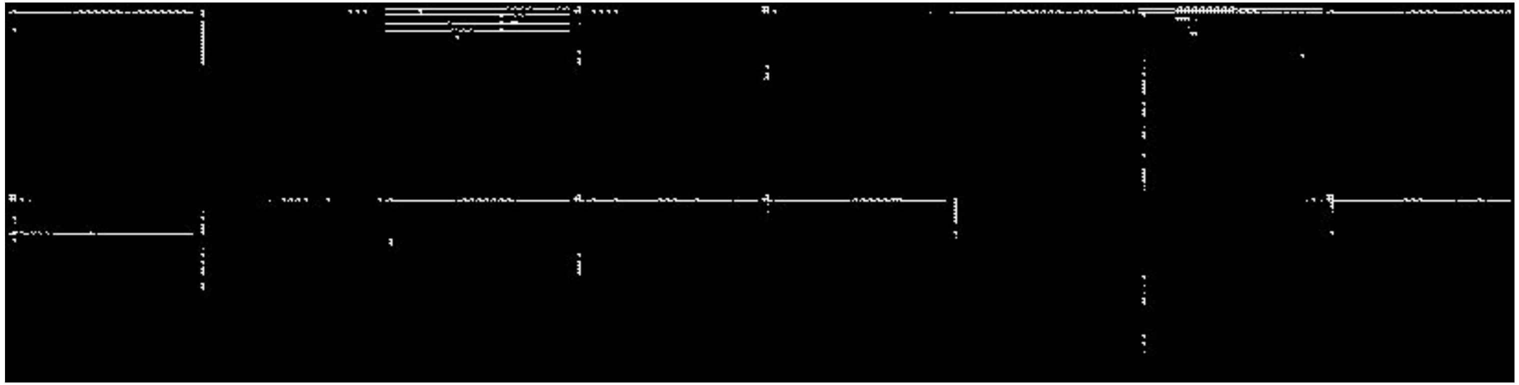
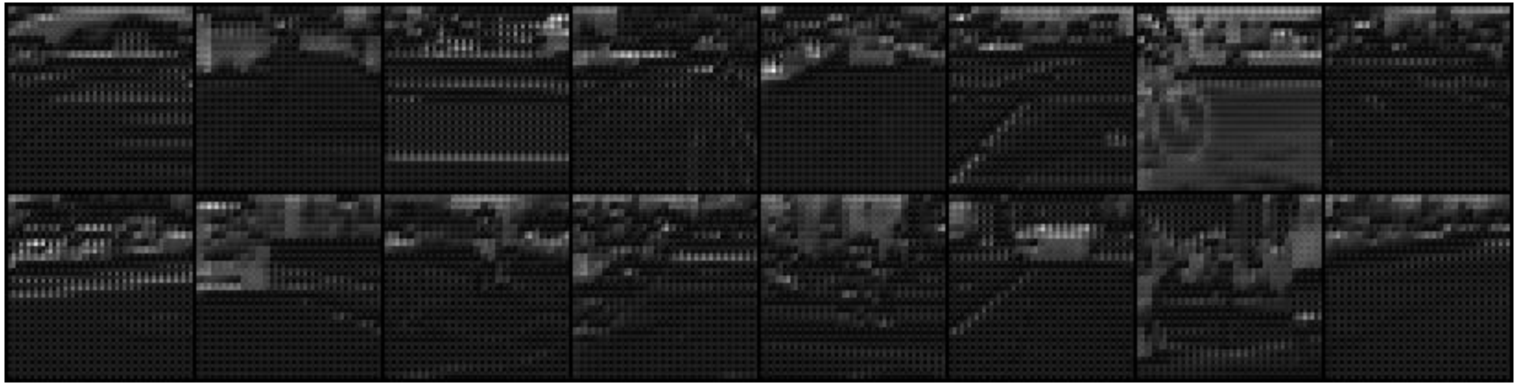
- Demonstrated schedule is to start with a temperature of 10 and multiply the temperature by 0.9 on each epoch.

Results without Regularization

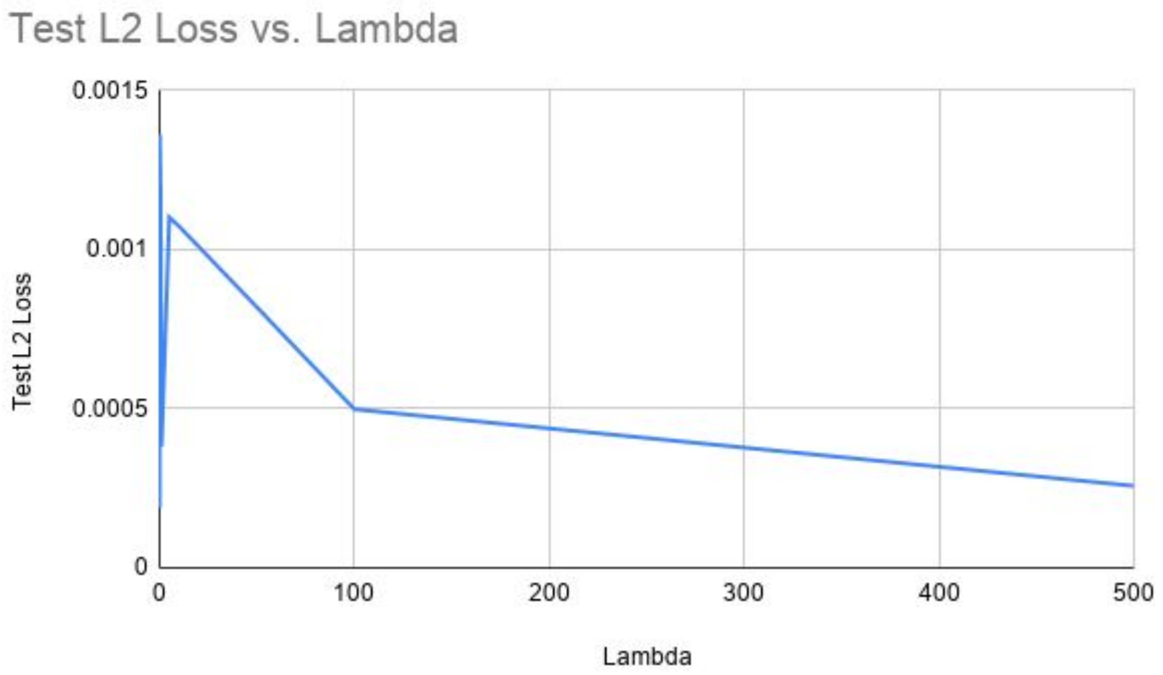




Results with Regularization

Comparison of Performance



Github Repositories

Public Github Links

- Sparse Aerial Depth Completion:

  https://github.com/austinbdill/sadc

- Active Learning for Pixelwise Prediction:

  https://github.com/austinbdill/pixelwise_al

References

ArXiv Links

- Categorical Reparameterization with Gumbel Softmax:

  https://arxiv.org/pdf/1611.01144.pdf

- Learning Selection Masks for Deep Neural Networks

  https://arxiv.org/pdf/1906.04673.pdf

Github References

- PyTorch Straight Through Gumbel Softmax Implementation:

  https://gist.github.com/yzh119/fd2146d2aeb329d067568a493b20172f