1 Supplementary material

This Supplementary material is organized as follows. Section 1.1 discusses the computational complexity of the proposed curve finding method. Section 1.2 describes how to apply batch normalization at test time to points on curves connecting pairs of local optima. Section 1.3 provides formulas for a polygonal chain and Bezier curve with n bends. Section 1.4 provides details and results of experiments on curve finding and contains a table summarizing all path finding experiments. Section 1.5 provides additional visualizations of the train loss and test accuracy surfaces. Section 1.6 contains details on curve ensembling experiments. Section 1.7 describes experiments on relation between mode connectivity and the number of parameters in the networks. Section 1.8 discusses a trivial construction of curves connecting two modes, where points on the curve represent reparameterization of the endpoints, unlike the curves in the main text. Section 1.9 provides details of experiments on FGE. Finally, Section 1.10 describes pathways traversed by FGE.

1.1 Computational complexity of curve finding

The forward pass of the proposed method consists of two steps: computing the point $\phi_\theta(t)$ and then passing a mini-batch of data through the DNN corresponding to this point. Similarly, the backward pass consists of first computing the gradient of the loss with respect to $\phi_\theta(t)$, and then multiplying the result by the Jacobian $\frac{\partial \phi_\theta}{\partial \theta}$. The second step of the forward pass and the first step of the backward pass are exactly the same as the forward and backward pass in the training of a single DNN model. The additional computational complexity of the procedure compared to single model training comes from the first step of the forward pass and the second step of the backward pass and in general depends on the parametrization $\phi_\theta(\cdot)$ of the curve.

In our experiments we use curve parametrizations of a specific form. The general formula for a curve with one bend is given by

$$\phi_\theta(t) = \hat{w}_1 \cdot c_1(t) + \theta \cdot c(t) + \hat{w}_2 \cdot c_2(t).$$

Here the parameters of the curve are given by $\theta \in \mathbb{R}^{|net|}$ and coefficients $c_1, c_2, c : [0, 1] \to \mathbb{R}$.

For this family of curves the computational complexity of the first step of the method is $\mathcal{O}(|net|)$, as we only need to compute a weighted sum of $\hat{w}_1$, $\hat{w}_2$ and $\theta \in \mathbb{R}^{|net|}$. The Jacobian matrix

$$\frac{\partial \phi_\theta(t)}{\partial \theta} = c(t) \cdot I,$$

thus the additional computational complexity of the backward pass is also $\mathcal{O}(|net|)$, as we only need to multiply the gradient with respect to $\phi_\theta(t)$ by a scalar. Thus, the total additional computational complexity is $\mathcal{O}(|net|)$. In practice we observe that the gap in time-complexity between one epoch of training a single model and one epoch of the proposed method with the same network architecture is usually below 50%.

1.2 Batch Normalization

Batch normalization (Ioffe and Szegedy [2015]) is essential to modern deep learning architectures. Batch normalization re-parametrizes the output of each layer as

$$\hat{x} = \gamma \frac{x - \mu(x)}{\sigma(x) + \epsilon} + \beta,$$

where $\mu(x)$ and $\sigma(x)$ are the mean and standard deviation of the output x, $\epsilon > 0$ is a constant for numerical stability and γ and β are free parameters. During training, $\mu(x)$ and $\sigma(x)$ are computed separately for each mini-batch and at test time statistics aggregated during training are used.

When connecting two DNNs that use batch normalization, along a curve $\phi(t)$, we compute $\mu(x)$ and $\sigma(x)$ for any given t over mini-batches during training, as usual. In order to apply batch-normalization to a network on the curve at the test stage we compute these statistics with one additional pass over the data, as running averages for these networks are not collected during training.

Table 1: The properties of loss and error values along the found curves for different architectures and tasks

Model		Length	Train Loss				Train Error (%)			Test Error (%)		
DNN	Curve	Ratio	Min	Int	Mean	Max	Min	Int	Max	Min	Int	Max
						MNIST						
FC	Single	—	0.018	—	—	0.018	0.01	—	0.01	1.46	—	1.5
FC	Segment	1	0.018	0.252	0.252	0.657	0.01	0.53	2.13	1.45	1.96	3.18
FC	Bezier	1.58	0.016	0.02	0.02	0.024	0.01	0.02	0.04	1.46	1.52	1.56
FC	Polychain	1.73	0.013	0.022	0.022	0.029	0	0.03	0.07	1.46	1.51	1.58
						CIFAR-10						
3conv3fc	Single	—	0.05	—	—	0.05	0.06	—	0.06	12.3	—	12.36
3conv3fc	Segment	1	0.05	1.124	1.124	2.416	0.06	35.69	88.24	12.28	43.3	88.27
3conv3fc	Bezier	1.30	0.034	0.038	0.037	0.05	0.05	0.1	0.2	12.06	12.7	13.66
3conv3fc	Polychain	1.67	0.04	0.044	0.044	0.05	0.06	0.15	0.31	12.17	12.68	13.31
VGG-16	Single	—	0.04	—	—	0.04	0	—	0.01	6.87	—	7.01
VGG-16	Segment	1	0.039	1.759	1.759	2.569	0	61.43	90	6.87	63.75	90
VGG-16	Bezier	1.55	0.028	0.03	0.03	0.04	0	0.01	0.02	6.59	6.77	7.01
VGG-16	Polychain	1.83	0.025	0.031	0.031	0.045	0	0.01	0.04	6.54	6.89	7.28
ResNet-158	Single	—	0.015	—	—	0.015	0.02	—	0.02	5.56	—	5.74
ResNet-158	Segment	1	0.013	0.551	0.551	2.613	0	16.37	81.41	5.57	20.79	80.00
ResNet-158	Bezier	2.13	0.013	0.017	0.018	0.022	0	0.02	0.07	5.48	5.82	6.24
ResNet-158	Polychain	3.48	0.013	0.017	0.017	0.047	0	0.05	0.139	5.48	5.88	7.35
WRN-10-28	Single	—	0.033	—	—	0.035	0	—	0	4.49	—	4.56
WRN-10-28	Segment	1	0.033	0.412	0.412	2.203	0	5.44	65.62	4.49	10.55	66.6
WRN-10-28	Bezier	1.83	0.03	0.033	0.038	0.038	0	0.01	0.04	4.4	4.62	4.83
WRN-10-28	Polychain	1.95	0.026	0.029	0.029	0.037	0	0	0	4.38	6.93	10.38
						CIFAR-100						
VGG-16	Single	—	0.14	—	—	0.141	0.05	—	0.06	29.44	—	29.94
VGG-16	Segment	1	0.137	3.606	3.606	4.941	0.04	73.25	99	29.44	80.59	99.01
VGG-16	Bezier	1.52	0.095	0.107	0.105	0.141	0.03	0.08	0.18	29.28	30.49	31.23
VGG-16	Polychain	1.64	0.118	0.139	0.139	0.2	0.04	0.19	0.39	29.33	30.13	30.92
ResNet-164	Single	—	0.079	—	—	0.08	0.06	—	0.09	24.41	—	24.4
ResNet-164	Segment	1	0.076	1.844	1.844	5.53	0.06	38.03	98.65	24.4	53.69	98.83
ResNet-164	Bezier	1.87	0.074	0.083	0.084	0.098	0.05	0.28	0.96	24.15	24.99	26.1
ResNet-164	Polychain	2.56	0.067	0.078	0.078	0.109	0.06	0.28	0.85	23.98	24.92	26.12

1.3 Formulas for curves with n bends

For n bends $\theta = \{w_1, w_2, \ldots, w_n\}$, the parametrization of a polygonal chain connecting points w_0, w_{n+1} is given by

$$\phi_\theta(t) = (n+1) \cdot \left(\left(t - \frac{i}{n+1} \right) \cdot w_{i+1} + \left(\frac{i+1}{n+1} - t \right) \cdot w_i \right),$$

for $\frac{i}{n+1} \leq t \leq \frac{i+1}{n+1}$ and $0 \leq i \leq n$.

For n bends $\theta = \{w_1, w_2, \ldots, w_n\}$, the parametrization of a Bezier curve connecting points w_0 and w_{n+1} is given by

$$\phi_\theta(t) = \sum_{i=0}^{n+1} w_i C_{n+1}^i t^i (1-t)^{n+1-i}$$

Table 2: The value of perplexity along the found curves for PTB dataset

Model		Train		Validation		Test	
DNN	Curve	Min	Max	Min	Max	Min	Max
PTB							
RNN	Single	37.5	39.2	82.7	83.1	78.7	78.9
RNN	Segment	37.5	596.3	82.7	682.1	78.7	615.7
RNN	Bezier	29.8	39.2	82.7	88.7	78.7	84.0

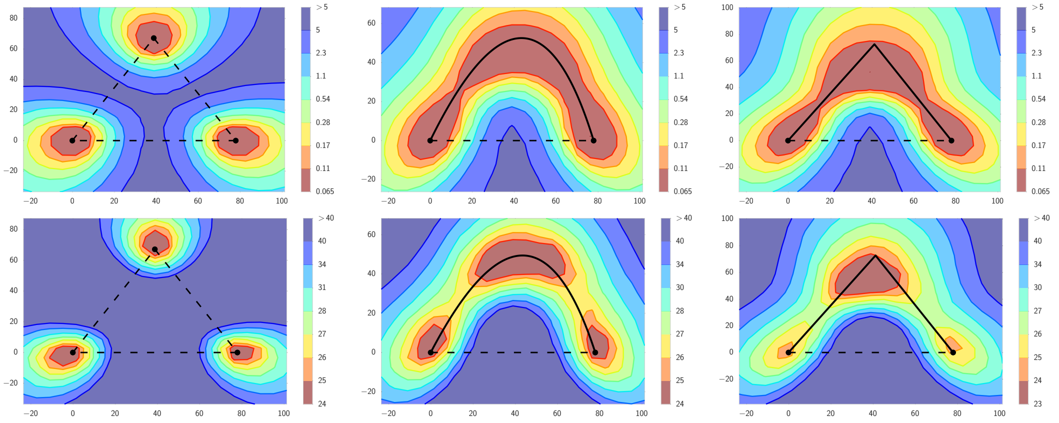

Figure 1: The ℓ_2-regularized cross-entropy train loss (**Top**) and test error (**Bottom**) surfaces of a deep residual network (ResNet-164) on CIFAR-100. **Left:** Three optima for independently trained networks. **Middle** and **Right**: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel, a direct linear path between each mode would incur high loss.

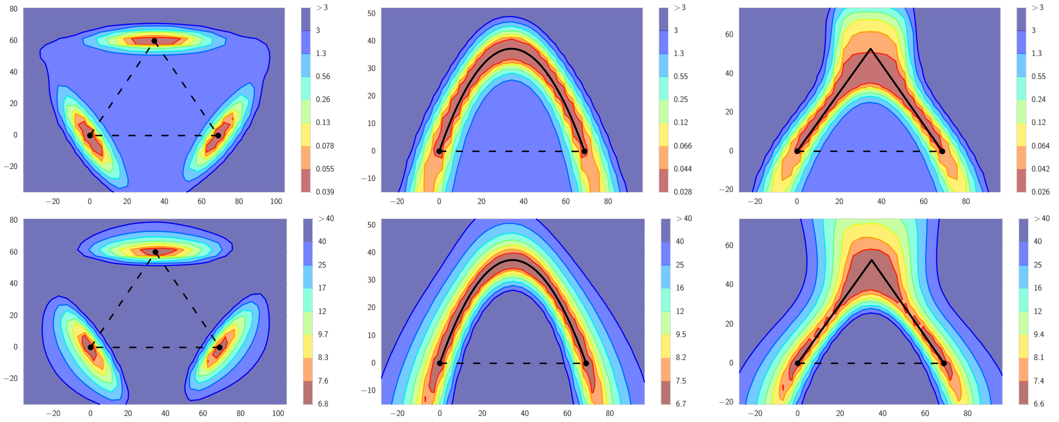

Figure 2: Same as Fig. 1 for VGG-16 on CIFAR-10.

1.4 Curve Finding Experiments

All experiments on curve finding were conducted with TensorFlow (Abadi et al. [2016]) and as baseline models we used the following implementations:

- ResNet-bottleneck-164 and Wide ResNet-28-10 (`https://github.com/tensorflow/models/tree/master/research/resnet`);
- ResNet-158 (`https://github.com/tensorflow/models/tree/master/official/resnet`);

- A reimplementation of VGG-16 without batch-normalization from (`https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py`);

Table 1 summarizes the results of the curve finding experiments with all datasets and architectures. For each of the models we report the properties of loss and the error on the train and test datasets. For each of these metrics we report 3 values: "Max" is the maximum values of the metric along the curve, "Int" is a numerical approximation of the integral $\int \text{<metric>}(\phi_\theta) d\phi_\theta / \int d\phi_\theta$, where <metric> represents the train loss or the error on the train or test dataset and "Min" is the minimum value of the error on the curve. "Int" represents a mean over a uniform distribution on the curve, and for the train loss it **coincides with the loss** (1) in the paper. We use an equally-spaced grid with 121 points on $[0, 1]$ to estimate the values of "Min", "Max", "Int". For "Int" we use the trapezoidal rule to estimate the integral. For each dataset and architecture we report the performance of single models used as the endpoints of the curve as "Single", the performance of a line segment connecting the two single networks as "Segment", the performance of a quadratic Bezier curve as "Bezier" and the performance of a polygonal chain with one bend as "Polychain". Finally, for each curve we report the ratio of its length to the length of a line segment connecting the two modes.

We also examined the quantity "Mean" defined as $\int < \text{metric} > (\phi_\theta(t))dt$, which **coincides with the loss** (2) from the paper, but in all our experiments it is nearly equal to "Int".

Besides convolutional and fully-connected architectures we also apply our approach to RNN architecture on next word prediction task, PTB dataset (Marcus et al. [1993]). As a base model we used the implementation available at `https://www.tensorflow.org/tutorials/recurrent`. As the main loss we consider perplexity. The results are presented in Table 2.

1.5 Train loss and test accuracy surfaces

In this section we provide additional visualizations. Fig. 1 and Fig. 2 show visualizations of the train loss and test accuracy for ResNet-164 on CIFAR-100 and VGG-16 on CIFAR-10.

1.6 Curve Ensembling

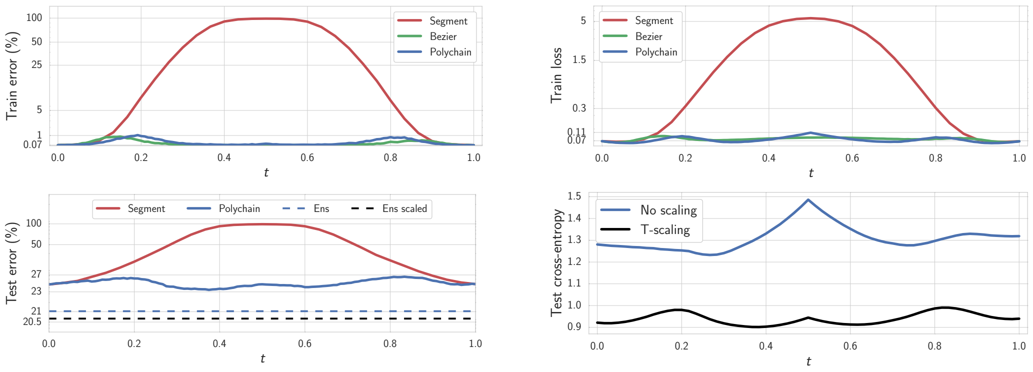

Figure 3: Error as a function of the point on the curves $\phi_\theta(t)$ found by the proposed method, using a ResNet-164 on CIFAR-100. **Top left:** train error. **Bottom left:** test error; dashed lines correspond to quality of ensemble constructed from curve points before and after logits rescaling. **Top right:** train loss (ℓ_2 regularized cross-entropy). **Bottom right:** cross-entropy before and after logits rescaling for the polygonal chain.

Here we explore ensembles constructed from points sampled from these high accuracy curves. In particular, we train a polygonal chain with one bend connecting two independently trained ResNet-164 networks on CIFAR-100 and construct an ensemble of networks corresponding to 50 points placed on an equally-spaced grid on the curve. The resulting ensemble had 21.03% error-rate on the test dataset. The error-rate of the ensemble constructed from the endpoints of the curve was 22.0%. An ensemble of three independently trained networks has an error rate of 21.01%. Thus, the ensemble of the networks on the curve outperformed an ensemble of its endpoints implying that the curves found by the proposed method are actually passing through diverse networks that produce predictions different from those produced by the endpoints of the curve. Moreover, the ensemble based on the polygonal chain has the same number of parameters as three independent networks, and comparable performance.

Furthermore, we can improve the ensemble on the chain without adding additional parameters or computational expense, by accounting for the pattern of increased training and test loss towards the centres of the linear paths shown in Figure 3. While the training and test accuracy are relatively constant, the pattern of loss, shared across train and test sets, indicates overconfidence away from the three points defining the curve: in this region, networks tend to output probabilities closer to 1, sometimes with the wrong answers. This overconfidence decreases the performance of ensembles constructed from the networks sampled on the curves. In order to correct for this overconfidence and improve the ensembling performance we use temperature scaling [Guo et al., 2017], which is inversely proportional to the loss. Figure 3, bottom right, illustrates the test loss of ResNet-164 on CIFAR-100 before and after temperature scaling. After rescaling the predictions of the networks, the test loss along the curve decreases and flattens. Further, the test error-rate of the ensemble constructed from the points on the curve went down from 21.03% to 20.7% after applying the temperature scaling, outperforming 3 independently trained networks.

However, directly ensembling on the curves requires manual intervention for temperature scaling, and an additional pass over the training data for each of the networks (50 in this case) at test time to perform batch normalization as described in section 1.2. Moreover, we also need to train at least two networks for the endpoints of the curve.

1.7 The Effects of Increasing Parametrization

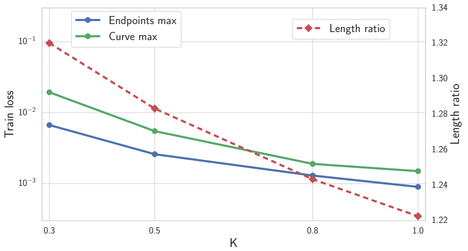

Figure 4: The worst train loss along the curve, maximum of the losses of the endpoints, and the ratio of the length of the curve and the line segment connecting the two modes, as a function of the scaling factor K of the sizes of fully-connected layers.

One possible factor that influences the connectedness of a local minima set is the overparameterization of neural networks. In this section, we investigate the relation between the observed connectedness of the local optima and the number of parameters (weights) in the neural network. We start with a network that has three convolutional layers followed by three fully-connected layers, where each layer has $1000K$ neurons. We vary $K \in \{0.3, 0.5, 0.8, 1\}$, and for each value of K we train two networks that we connect with a Bezier curve using the proposed procedure.

For each value of K, Figure 4 shows the worst training loss along the curve, maximum of losses of the endpoints, and the ratio of the length of the curve and the line segment connecting the two modes. Increasing the number of parameters we are able to reduce the difference between the worst value of the loss along the curve and the loss of single models used as the endpoints. The ratio of the length of the found curve and the length of the line segment connecting the two modes also decreases monotonically with K. This result is intuitive, since a greater parametrization allows for more flexibility in how we can navigate the loss surfaces.

1.8 Trivial connecting curves

For convolutional networks with ReLU activations and without batch normalization we can construct a path connecting two points in weight space such that the accuracy of each point on the curve (excluding the origin of the weight space) is at least as good as the minimum of the accuracies of the endpoints. Unlike the paths found by our procedure, these paths are trivial and merely exploit

Algorithm 1 Fast Geometric Ensembling

```
Require:
  weights $\hat{w}$, LR bounds $\alpha_1, \alpha_2$,
  cycle length $c$ (even), number of iterations $n$
Ensure: ensemble
  $w \leftarrow \hat{w}$ {Initialize weight with $\hat{w}$}
  ensemble $\leftarrow$ [ ]
  for $i \leftarrow 1, 2, \ldots, n$ do
    $\alpha \leftarrow \alpha(i)$ {Calculate LR for the iteration}
    $w \leftarrow w - \alpha \nabla \mathcal{L}_i(w)$ {Stochastic gradient update}
    if $\mathrm{mod}(i, c) = c/2$ then
      ensemble $\leftarrow$ ensemble $+ [w]$ {Collect weights}
    end if
  end for
```

redundancies in the parametrization. Also, the training loss goes up substantially along these curves. Below we give a construction of such paths.

Let $\hat{w}_1$ and $\hat{w}_2$ be two sets of weights. This path of interest consists of two parts. The first part connects the point $\hat{w}_1$ with 0 and the second one connects the point $\hat{w}_2$ with 0. We describe only the first part $\phi(t)$ of the path, such that $\phi(0) = 0, \phi(1) = \hat{w}_1$, as the second part is completely analogous. Let the weights of the network $\hat{w}_1$ be $\{W_i, b_i\}_{1 \le i \le n}$ where W_i, b_i are the weights and biases of the i-th layer, and n is the total number of layers. Throughout the derivation we consider the inputs of the network fixed. The output of the i-th layer $o_i = W_i \mathrm{ReLU}(o_{i-1}) + b_i$, $1 \le i \le n$, where $i = 0$ corresponds to the first layer and $i = n$ corresponds to logits (the outputs of the last layer). We construct $\phi(t) = \{W_i(t), b_i(t)\}_{1 \le i \le n}$ in the following way. We set $W_i(t) = W_i t$ and $b_i(t) = b_i t^i$. It is easy to see that logits of the network with weights $\phi(t)$ are equal to $o_n(t) = t^n o_n$ for all $t > 0$. Note that the predicted labels corresponding to the logits $o_n(t)$ and o_n are the same, so the accuracy of all networks corresponding to $t > 0$ is the same.

1.9 Fast geometric ensembling experiments

Alg. 1 provides an outline of the algorithm. As baseline models we used the following implementations:

- VGG-16 (`https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py`);
- Preactivation-ResNet-164 (`https://github.com/bearpaw/pytorch-classification/blob/master/models/cifar/preresnet.py`);
- ResNet-50 ImageNet (`https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py`);
- Wide ResNet-28-10 (`https://github.com/meliketoy/wide-resnet.pytorch/blob/master/networks/wide_resnet.py`);

For the FGE (Fast Geometric Ensembling) strategy on ResNet we run the FGE routine summarized in Alg. 1 after epoch 125 of the usual (same as Ind) training for 22 epochs. The total training time is thus $125 + 22 = 147$ epochs. For VGG and Wide ResNet models we run the pre-training procedure for 156 epochs to initialize FGE. Then we run FGE for 22 epochs starting from checkpoints corresponding to epochs 120 and 156 and ensemble all the gathered models. The total training time is thus $156 + 22 + 22 = 200$ epochs. For VGG we use cycle length $c = 2$ epochs, which means that the total number of models in the final ensemble is 22. For ResNet and Wide ResNet we use $c = 4$ epochs, and the total number of models in the final ensemble is 12 for Wide ResNets and 6 for ResNets.

1.10 Polygonal chain connecting FGE proposals

In order to better understand the trajectories followed by FGE we construct a polygonal chain connecting the points that FGE ensembles. Suppose we run FGE for n learning rate cycles obtaining

n points $w_1, w_2, \dots, w_n$ in the weight space that correspond to the lowest values of the learning rate. We then consider the polygonal chain consisting of the line segments connecting w_i to w_{i+1} for $i = 1, \dots, n-1$. We plot test accuracy and train error along this polygonal chain in Figure 5. We observe that along this curve both train loss and test error remain low, agreeing with our intuition that FGE follows the paths of low loss and error. Surprisingly, we find that the points on the line segments connecting the weights w_i, w_{i+1} have lower train loss and test error than w_i and w_{i+1}. See Izmailov et al. [2018] for a detailed discussion of this phenomenon.

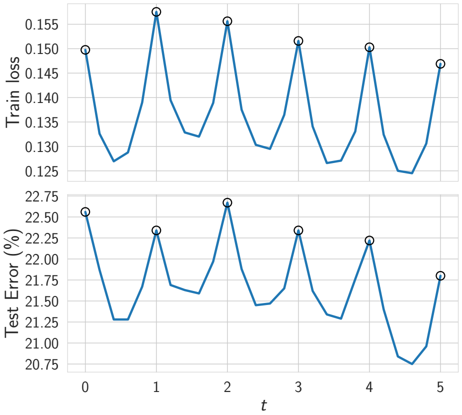

Figure 5: Train loss and test error along the polygonal chain connecting the sequence of points ensembled in FGE. The plot is generated using PreResNet-164 on CIFAR 100. Circles indicate the bends on the polygonal chain, i.e. the networks ensembled in FGE.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.