CSC2535: Advanced Machine Learning

Lecture 11a
Priors and Prejudice

Geoffrey Hinton

There are many ways to use prior information for learning

- If we know the form of the process that actually generated the data, we can try to learn the parameters of the process.
  - When using generative models, we typically pretend we know the form of the process even though we don't.
- If we have some idea about the distributions of the parameters in the generative process, we can be more Bayesian about the learning.
  - This is only important when we have too little data.
  - People often pretend that they have Gaussian or conjugate prior beliefs about the parameters in order to make the Bayesian computation easy.

Another kind of prior

- Sometimes we have a black-box, conditional generative model that converts codes into data.
  - A graphics model that converts a high-level code into an image can be treated as a black-box. Can it help us learn to convert images into high-level codes?
- We can try to learn an inverse model that converts images into codes.
  - Where do we get the training data if we have the images but not the codes?

More ways to use priors

- If we know some invariances that the data should satisfy, we can constrain the model to satisfy the same invariances.
  - Convolutional nets are sometimes claimed to do this. More on this later.
- Alternatively, we can use the known invariances to turn each training case into many different training cases.
  - This is a computationally inefficient way to incorporate prior information.

The advantage of using priors to create more training data

- It has the huge advantage that we do not need to know anything about the form of the model.
  - This way of using priors does not prejudice the form of the model.
- If the model involves a complicated neural net, for example, we probably have very little idea of how to make good use of the representational possibilities that the model allows.
  - So putting in priors by saying that the activities should be sparse or the weights should be small is really a prejudice that may rule out the best parts of the parameter space.

Learning to produce a handwritten character

- Give an image of a hand written character, how can a system with a very simple body learn to draw a very similar image?
- If we can solve this problem, we should be able to make much better recognizers for handwritten characters.
  - In difficult cases we “see” how they were written.
  - So if we can model how they were written we should get a much better recognizer.

The “motor” theory of perception (“analysis-by-synthesis”)

- The best way to recognize something is by figuring out how its was generated.
- Speech and handwriting are generated by motor programs
- So the best way to recognize speech and handwriting is by inferring the motor programs from the sensory data.
  - In the space of motor programs, letters or phonemes of the same class are very similar, even if their images are very different.
  - So a motor system implicitly contains a very good prior for generalization.

What happens in pixel space if we add noise to a motor program?

Infer the motor program from this image

add noise

Digit classes are far more natural in the space of motor programs

Two different ways to use neural networks for handwritten digit recognition

- Standard method: Train a neural network with 10 output units to map from pixel intensities to class labels.
  - This works well (with a few tricks) but it does not make use of a lot of prior knowledge that we have about how the images were generated.

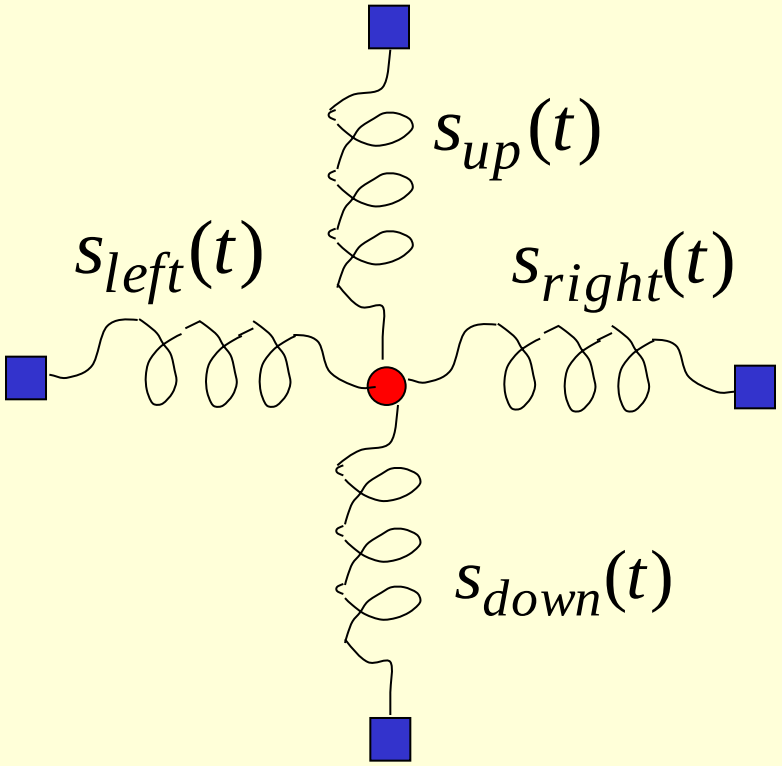
- The generative approach: First write a graphics program that converts a motor program (a sequence of muscle commands) into an image.
  - Then learn a neural network that maps pixel intensities to motor programs.

A variation

- It is very difficult to train a single net to recognize very different motor programs. So train a separate network for each digit class.
  - When given a test image, get each of the 10 networks to extract a motor program.
  - Then see which of the 10 motor programs is best at reconstructing the test image.
    - What metric should we use for comparing images and their reconstructions?
    - We should also take into account the probability of the extracted motor program for each class.

A simple generative model

- We can generate digit images by simulating the physics of drawing.
- A pen is controlled by four springs.
  - The trajectory of the pen is determined by the 4 spring stiffnesses at 17 time steps .
- The ink is produced from 60 evenly spaced points along the trajectory:
  - Use bilinear interpolation to distribute the ink at each point to the 4 closest pixels.
  - Use time-invariant parameters for the ink intensity and the width of a convolution kernel.
  - Then clip intensities above 1.



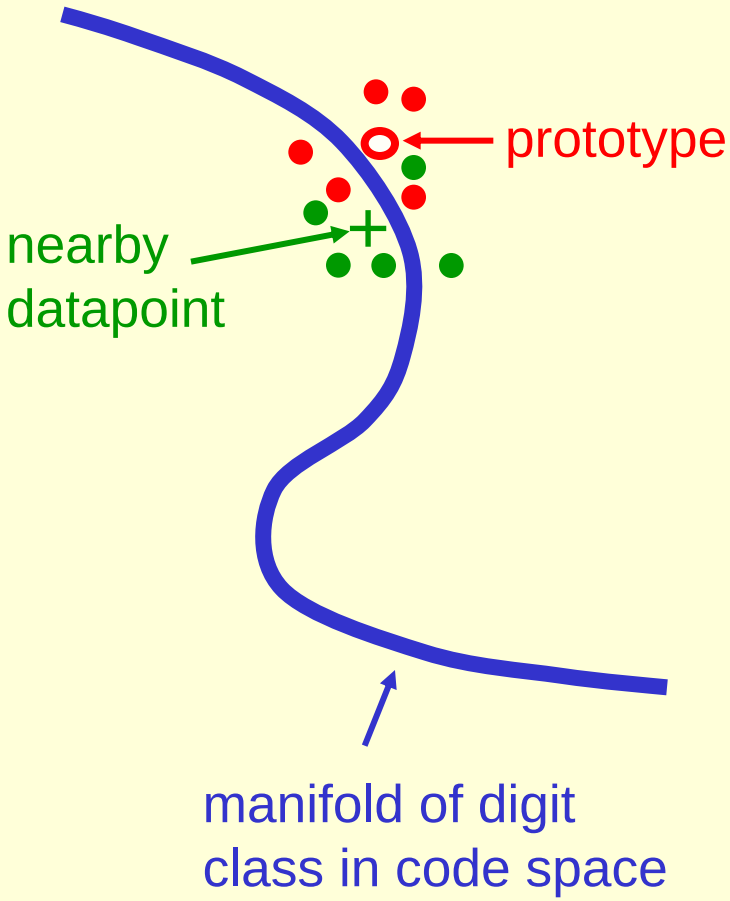
We could also learn to set the mass, viscosity, and positions of the four endpoints differently for each image.

Some ways to invert a generator

- Look inside the generator to see how it works and try to invert each step of the generative process.
  - Its hard to invert processes that lose information
    - The third dimension (not a problem for digits).
    - The correspondence between model-parts and image-parts.
- Define a prior distribution over codes and generate lots of (code, image) pairs. Then train a "recognition" neural network that does image → code.
  - But where do we get the prior over codes?
  - The distribution of codes is exactly what we want to learn from the data!
  - Is there any way to do without a the prior over codes?

A way to train a class-specific model from a single prototype

- Start with a single prototype code
- Learn to invert the generator in the vicinity of the prototype by adding noise to the code and generating (code, image) pairs for training a neural net.
- Then use the learned model to get codes for real images that are similar to the prototype. Add noise to these codes and generate more training data.
  - This extends the region that can be inverted along the data manifold.

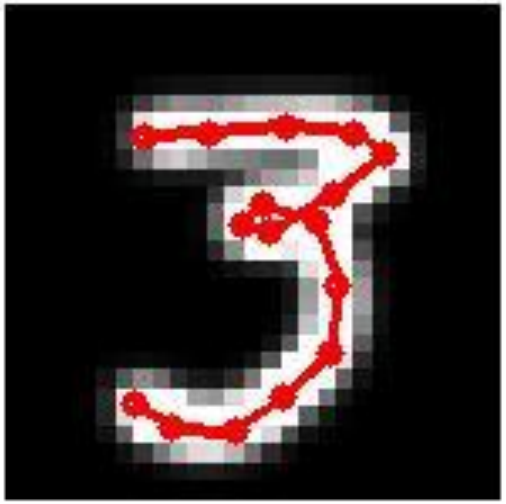


An example during the later stages of training

reconstruction from fitted code using true generator

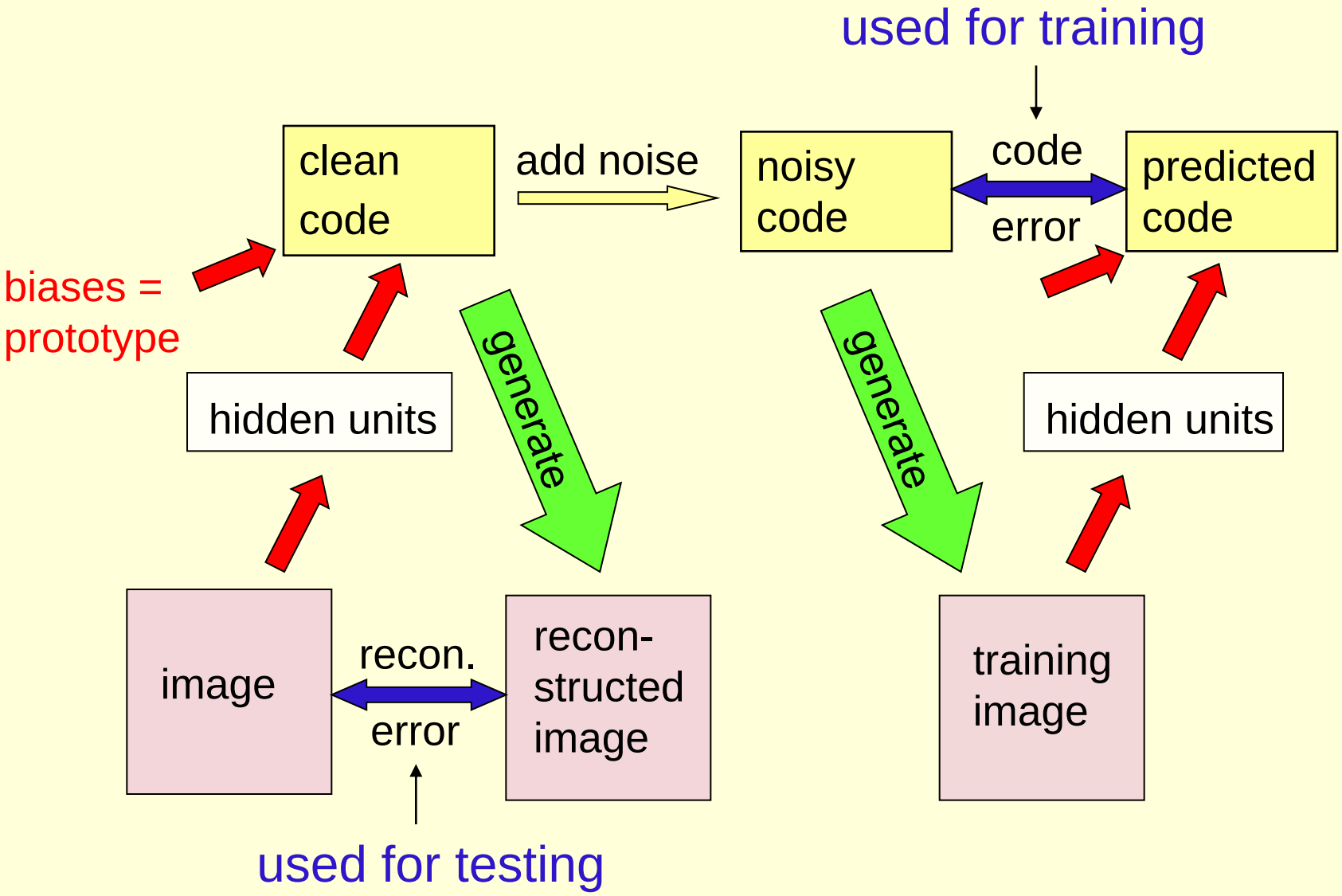
FIT TO DATA
squared pix err 12.8
thickness 0.40
intensity 0.79

ADD NOISE FOR TRAINING CASE
thickness 0.40
intensity 0.82

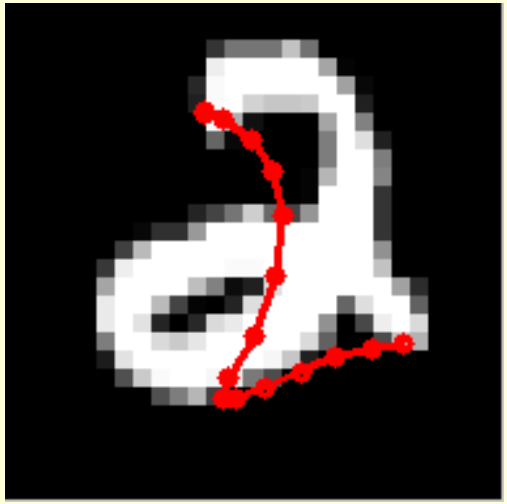
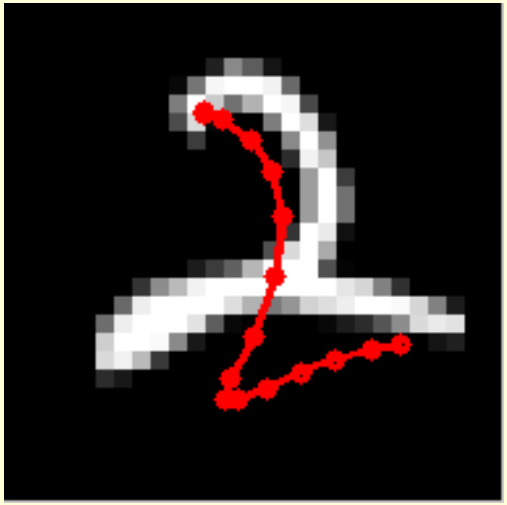
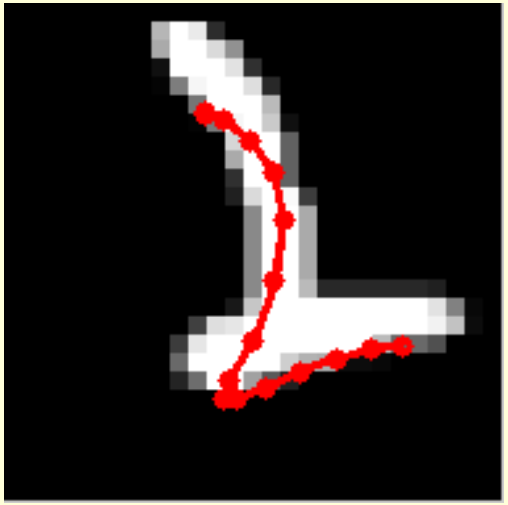
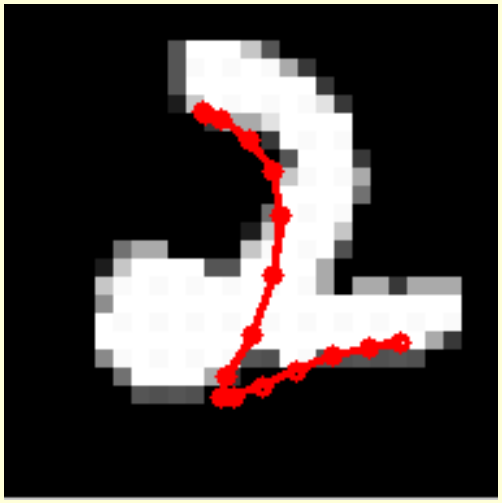
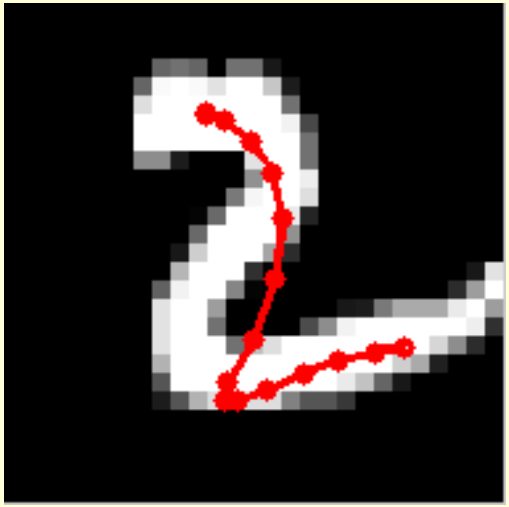
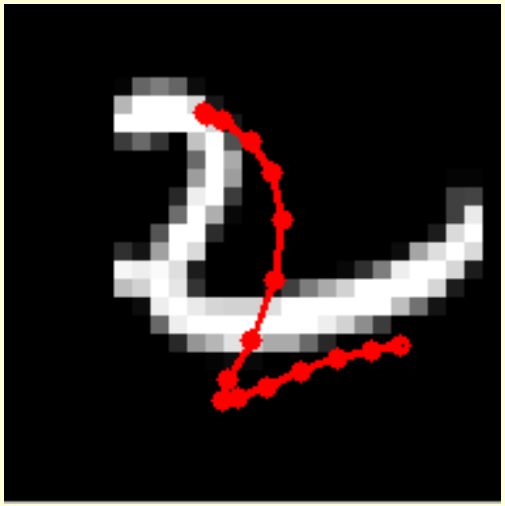


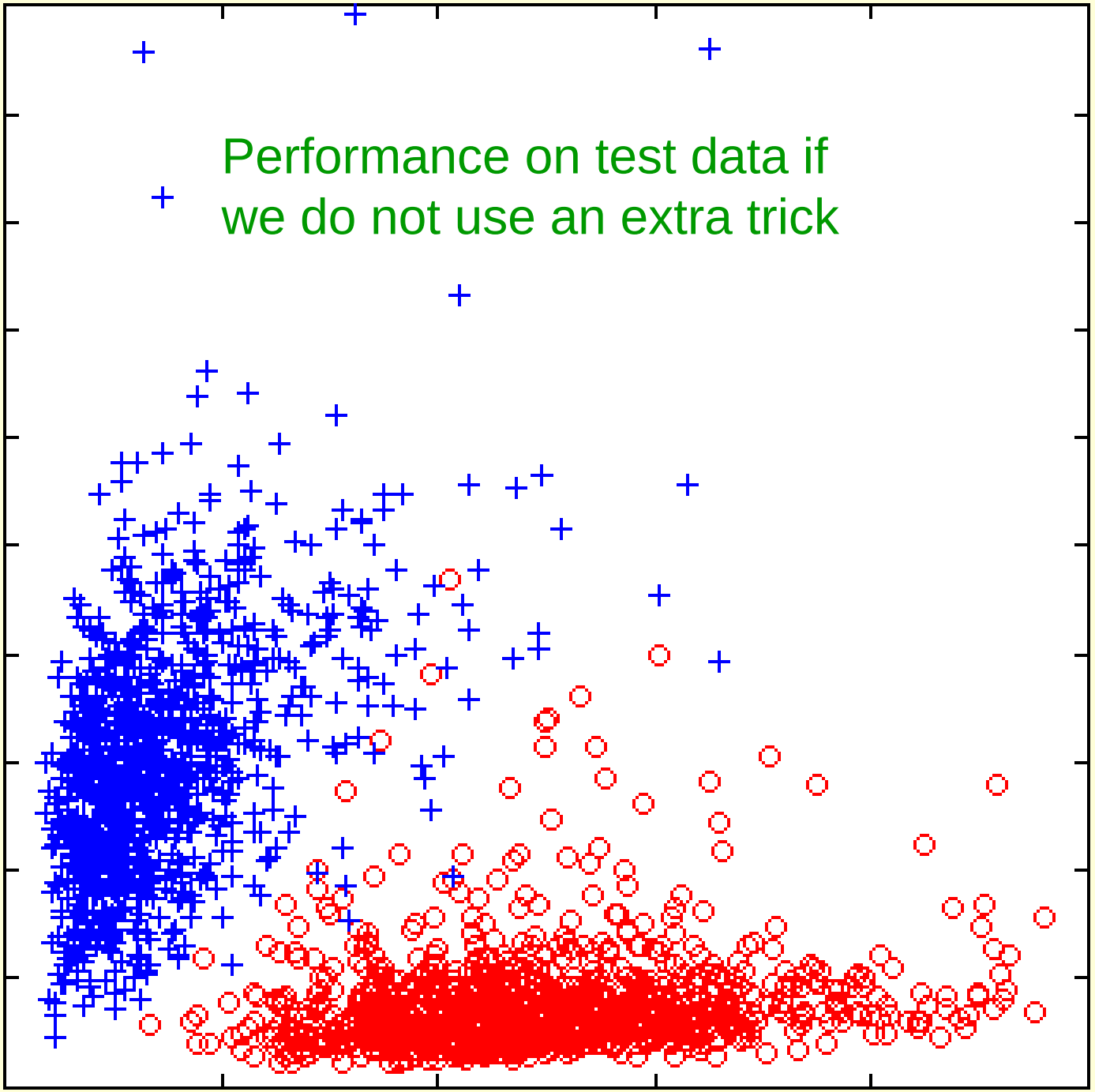
About 2 ms

How training examples are created

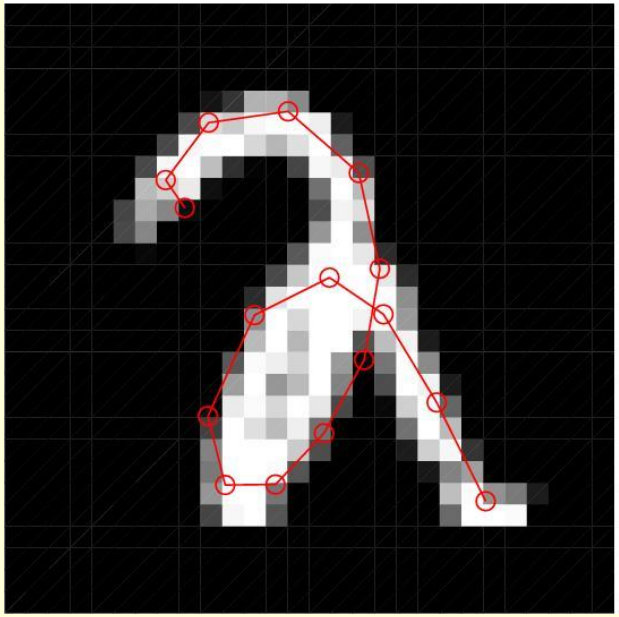


How the perceived trajectory changes at the early stages of learning

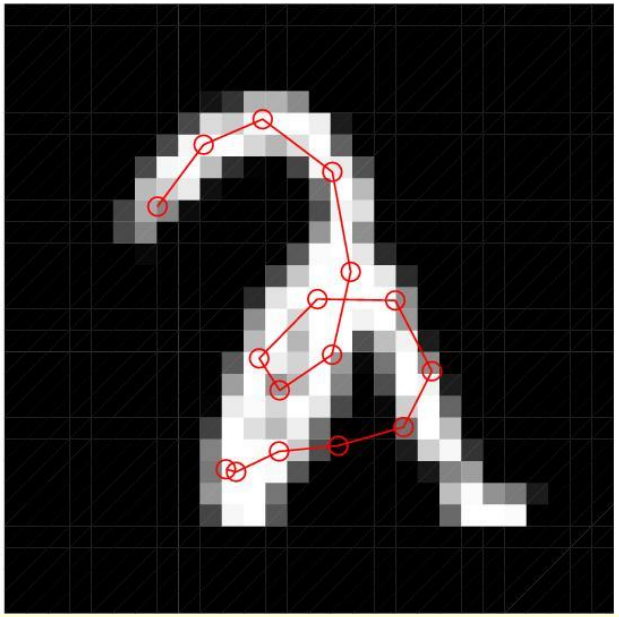








Performance on test data if
we do not use an extra trick

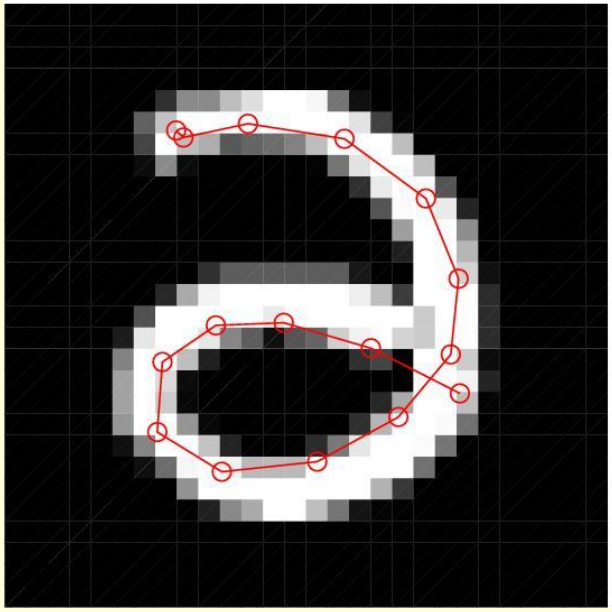
44.4464



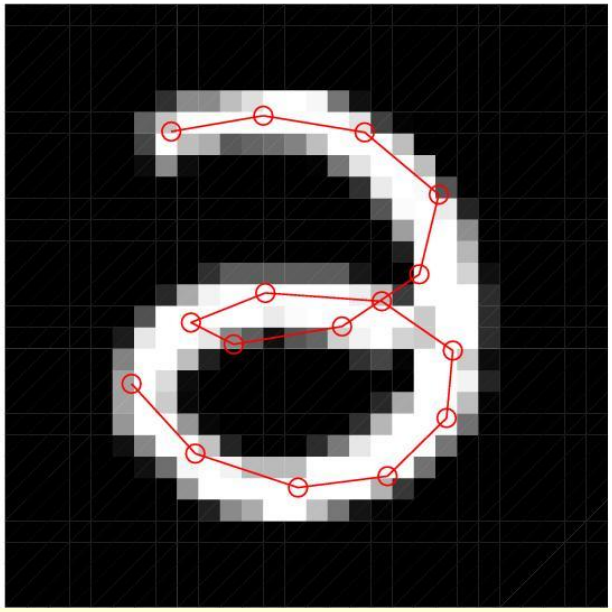
39.8321

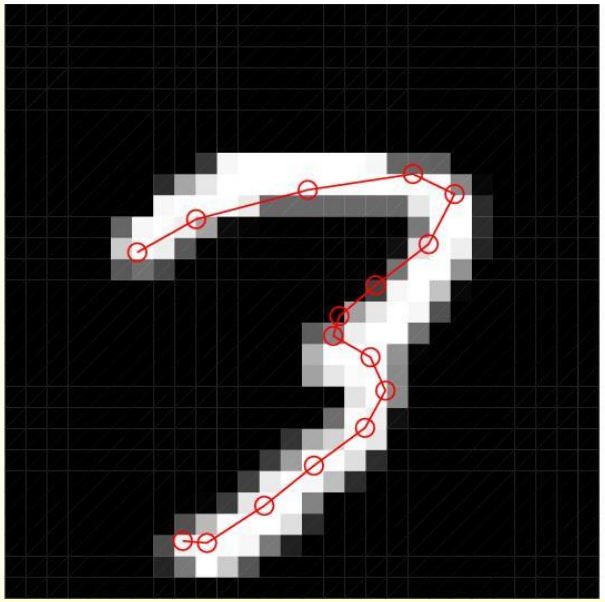


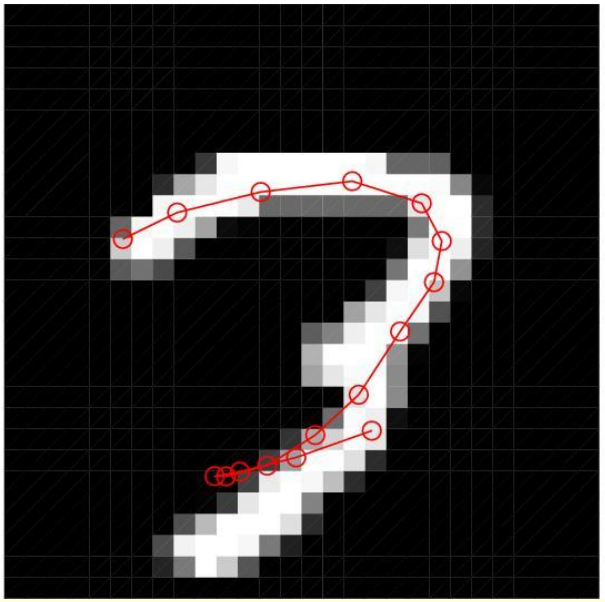
40.2784

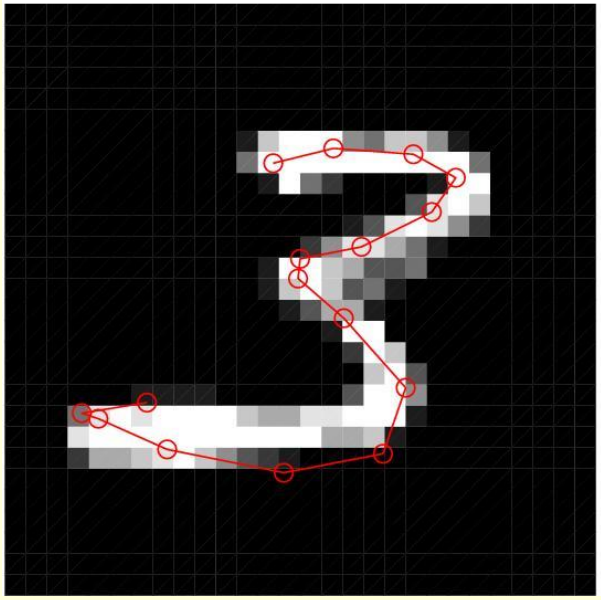


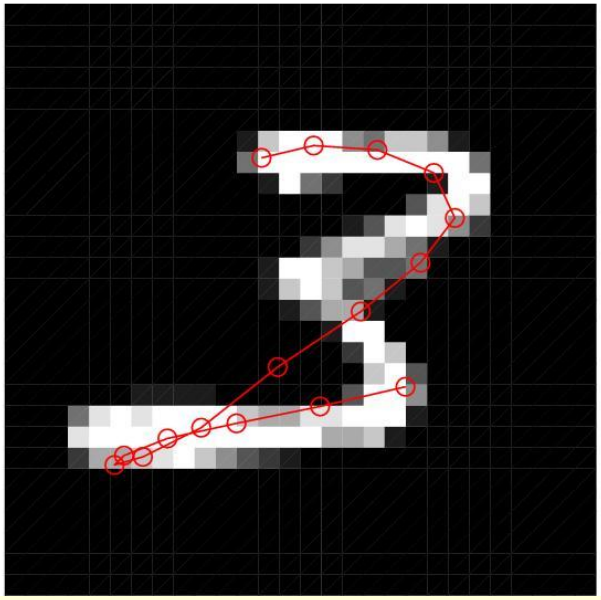
23.5612



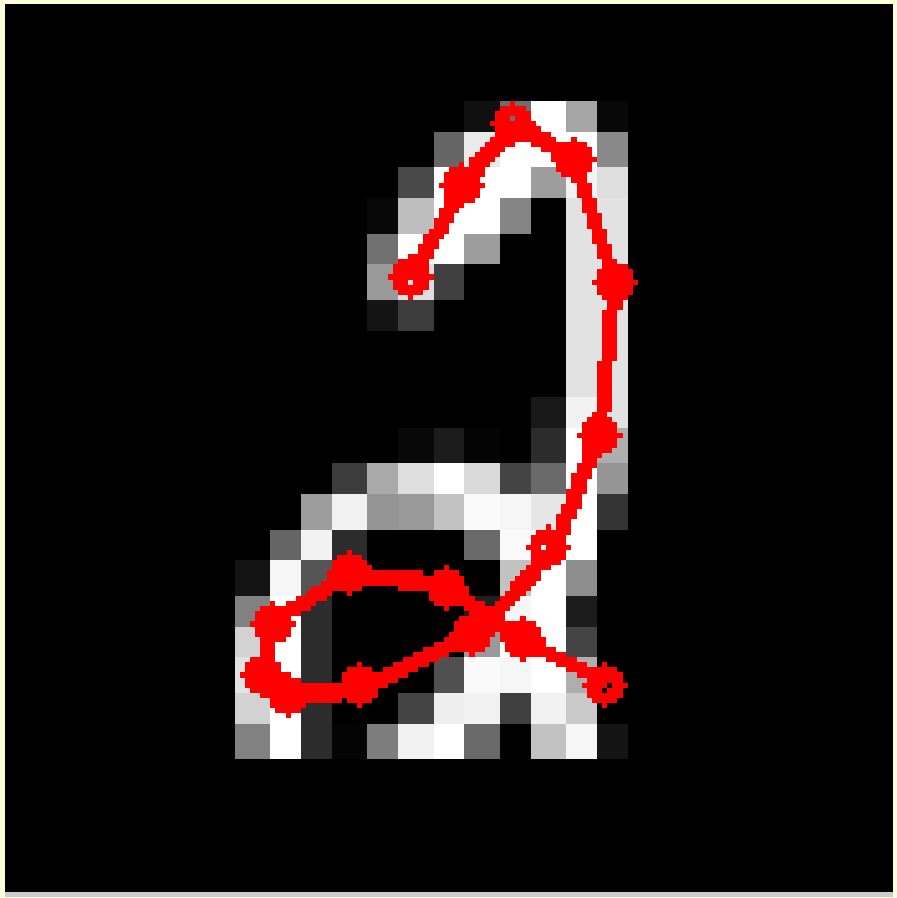

26.0019


30.4654


34.9806


39.6905

On test data, the model often gets the registration slightly wrong



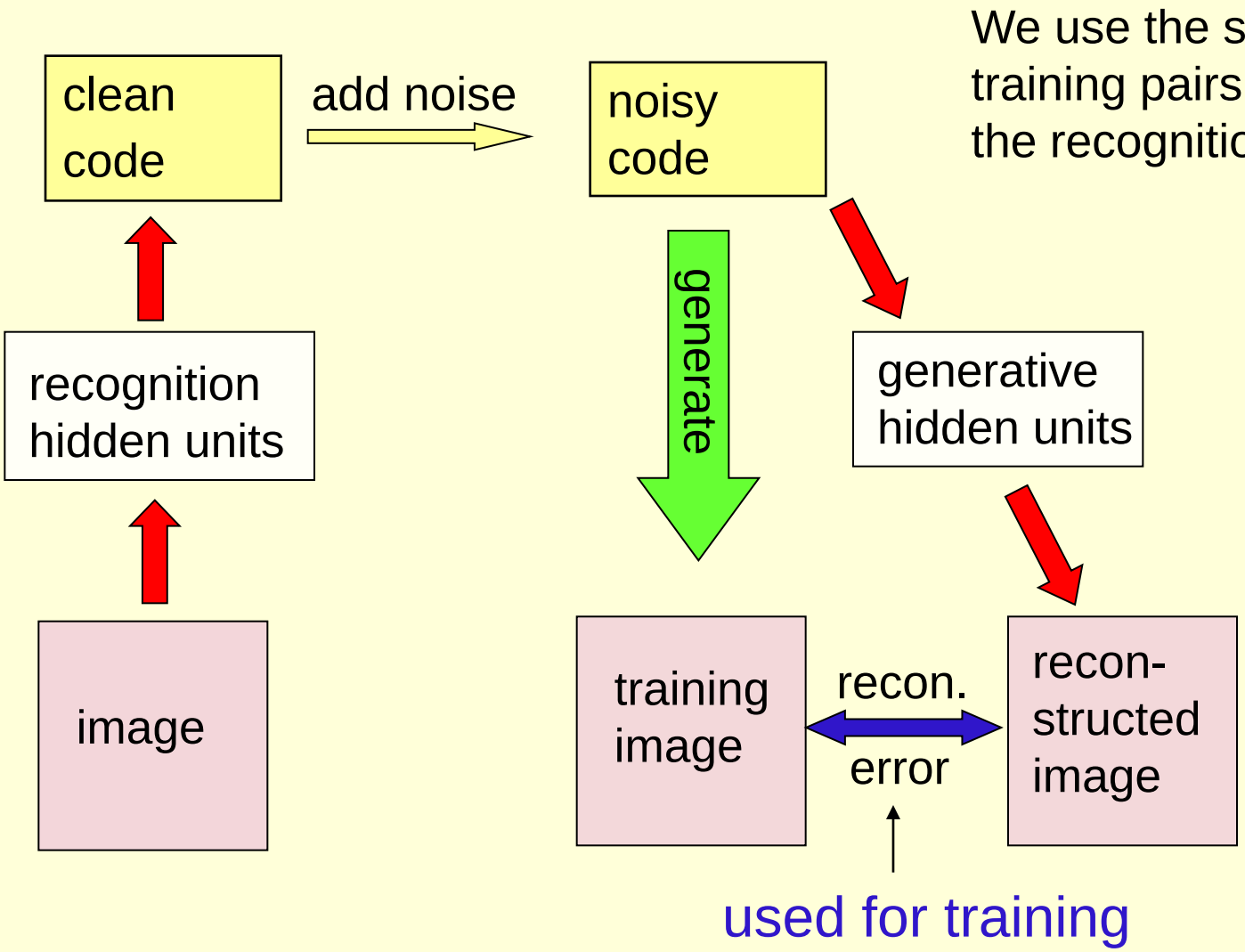
The neural net has solved the difficult global search problem, but it has got the fine details wrong.

So we need to perform a local search to fix up the details

Local search

- Use the trajectory produced by the neural network as an initial guess.
  - Then use the difference between the data and the reconstruction to adjust the guess.
- This means we need to convert residual errors in pixel space into gradients in motor program space.
  - But how to we backpropagate the pixel residuals through the generative graphics model?
- Make a neural network version of the generative model.

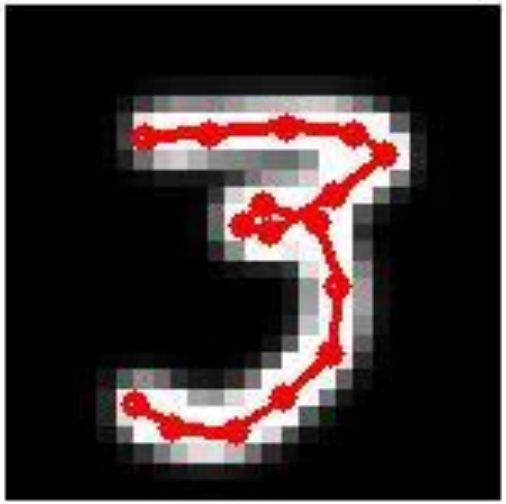
How the generative neural net is trained



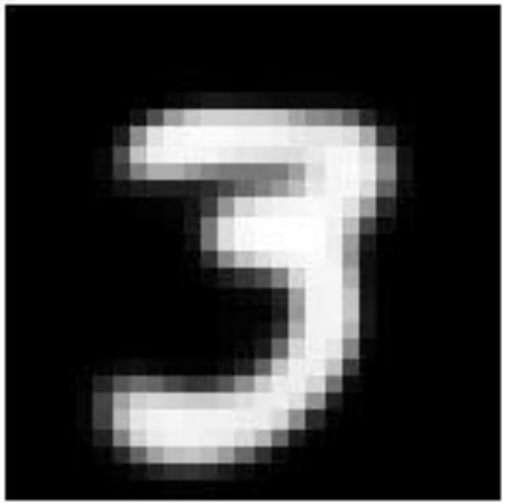
An example during the later stages of training the generative neural network

FIT TO DATA
squared pix err 12.8
thickness 0.40
intensity 0.79

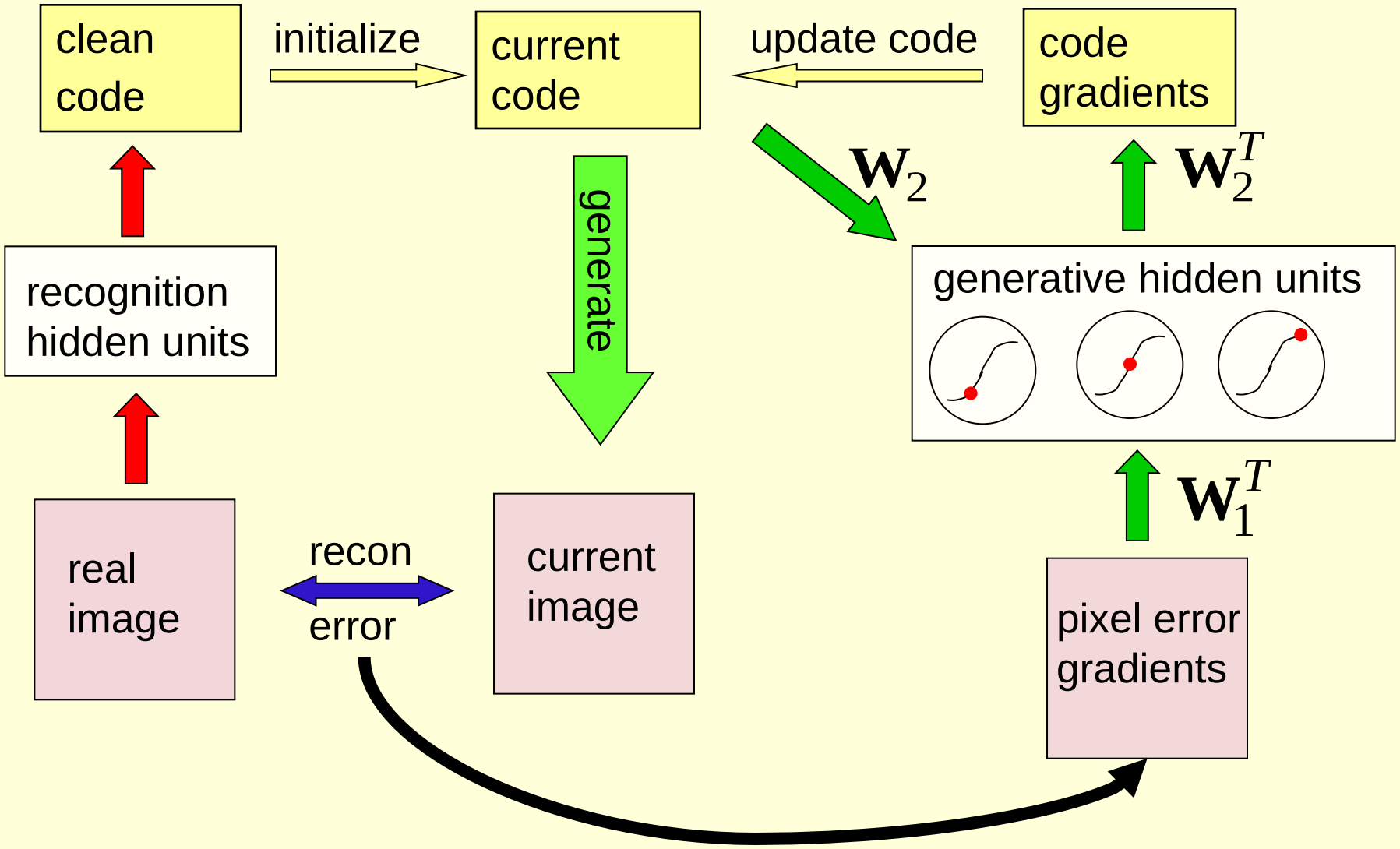
ADD NOISE FOR
TRAINING CASE
thickness 0.40
intensity 0.82



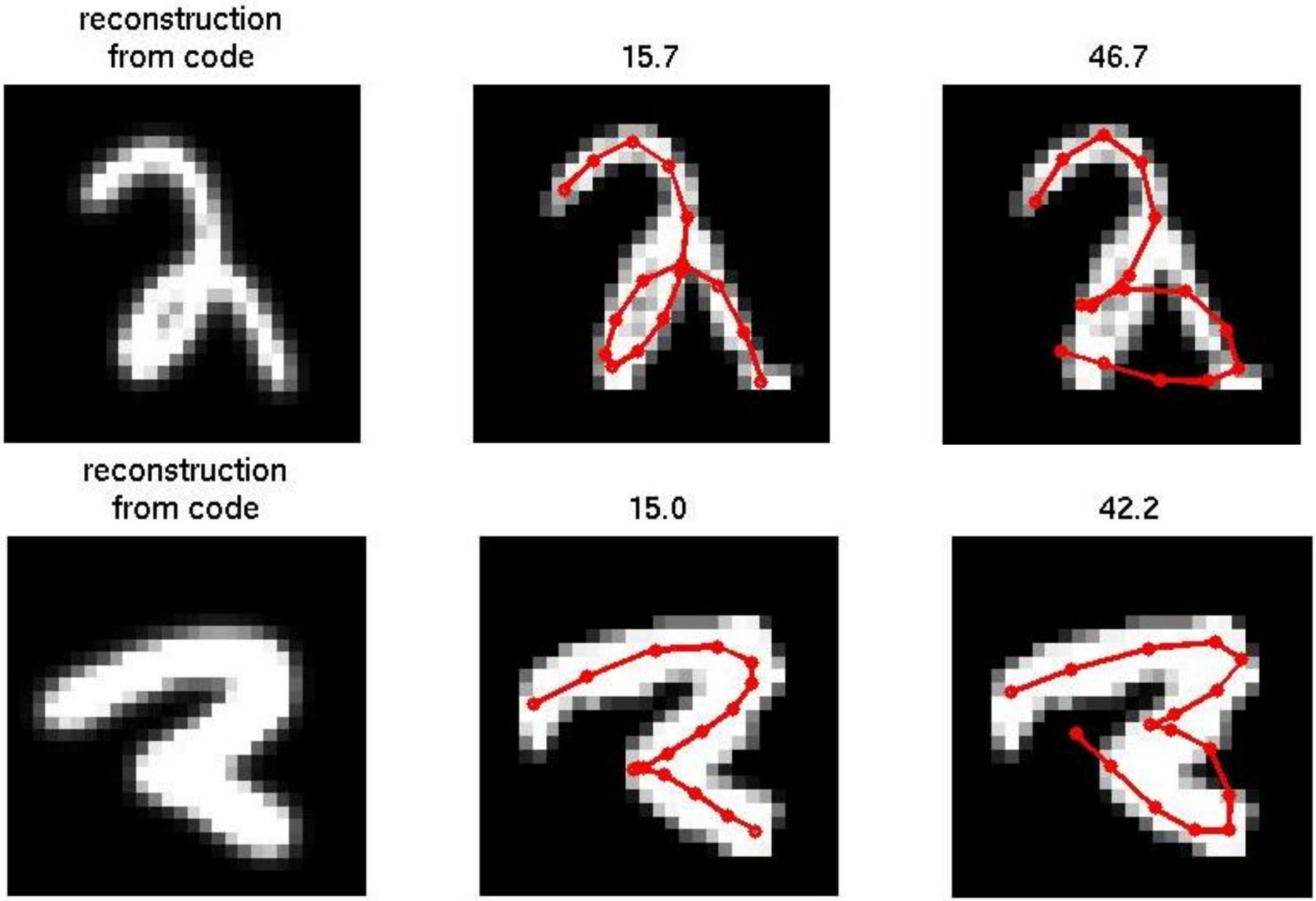
reconstruction from
training code
using neural net



How the generative neural net is used

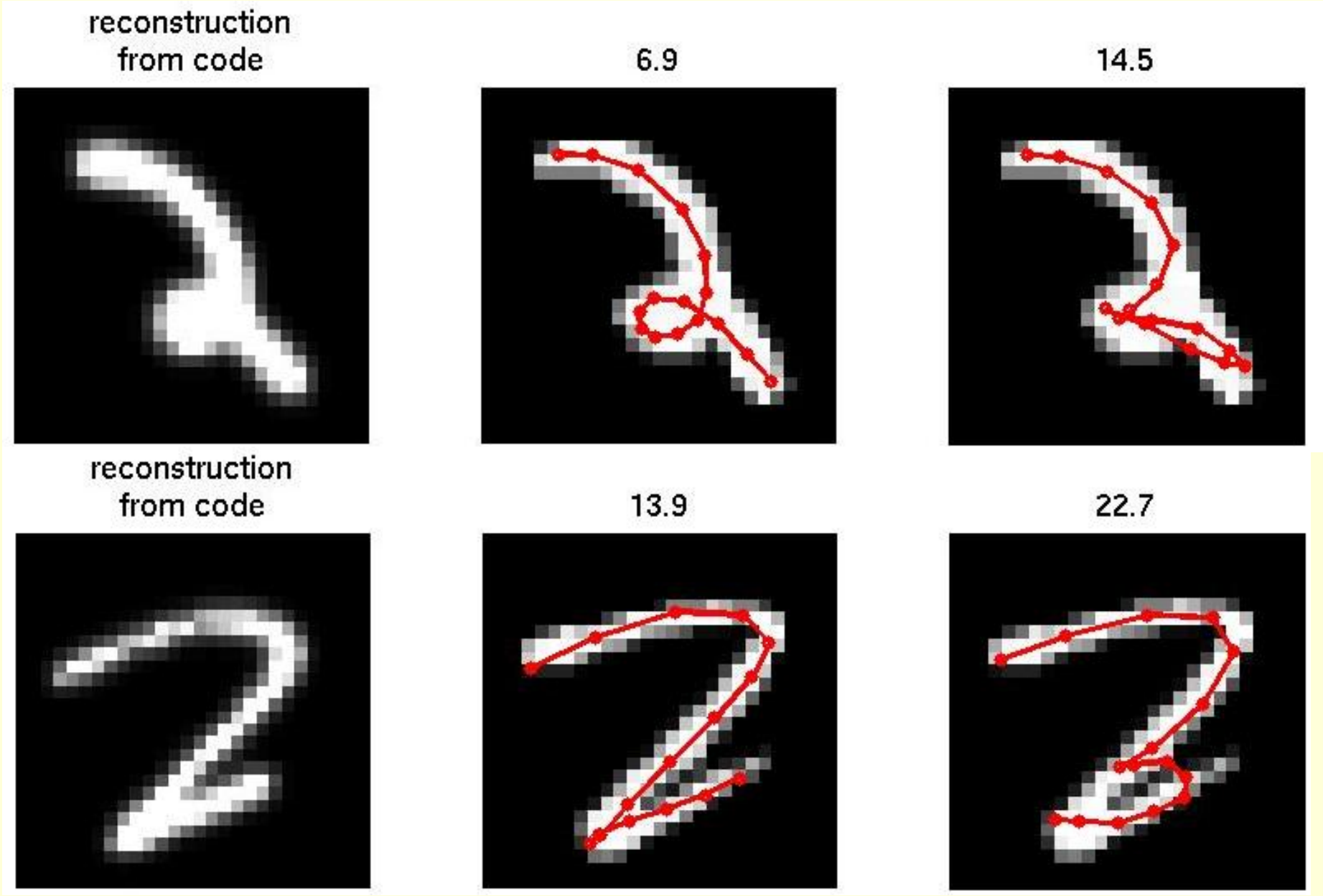


Typical fits to the test data at the end of training

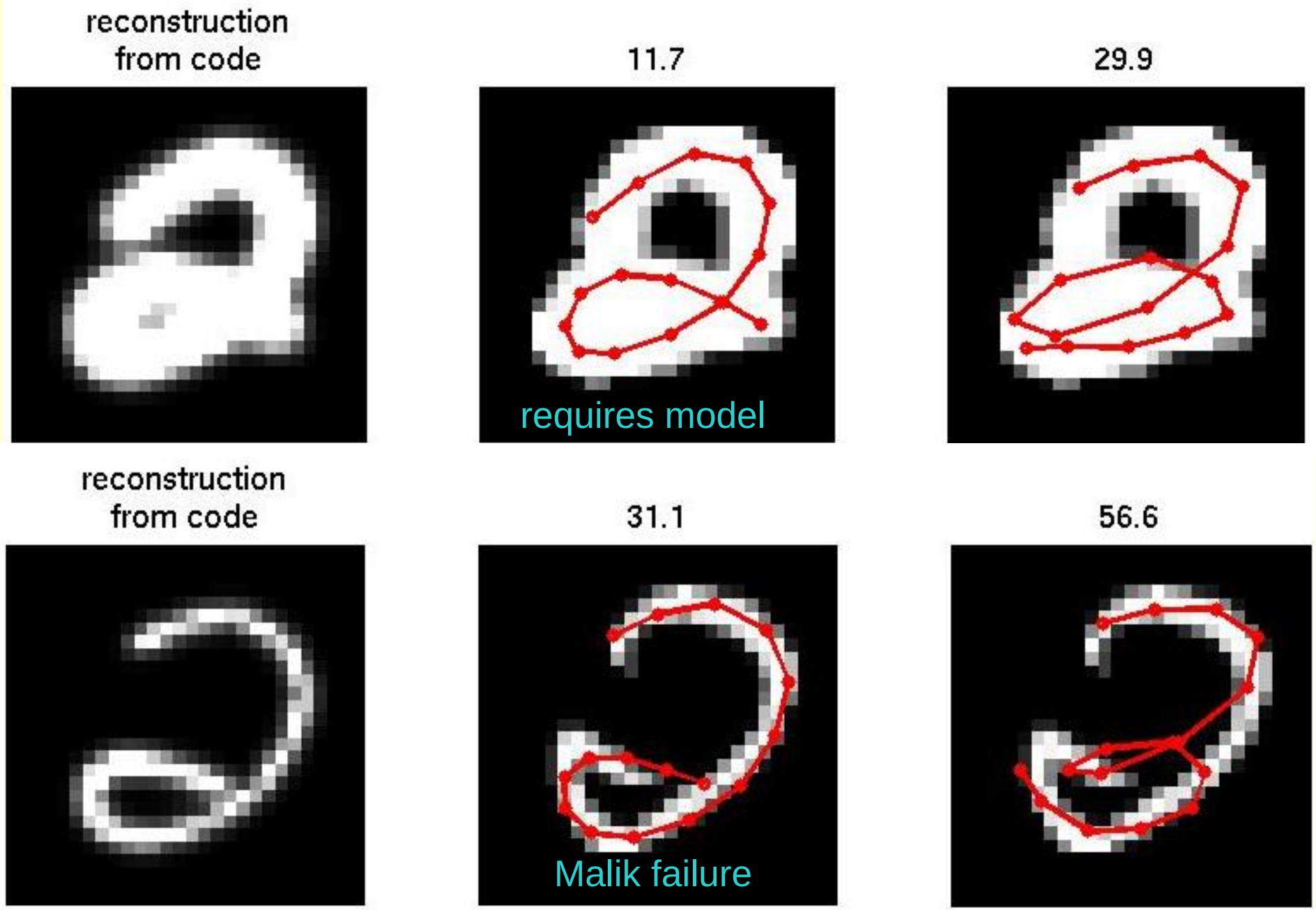


A Malik failure

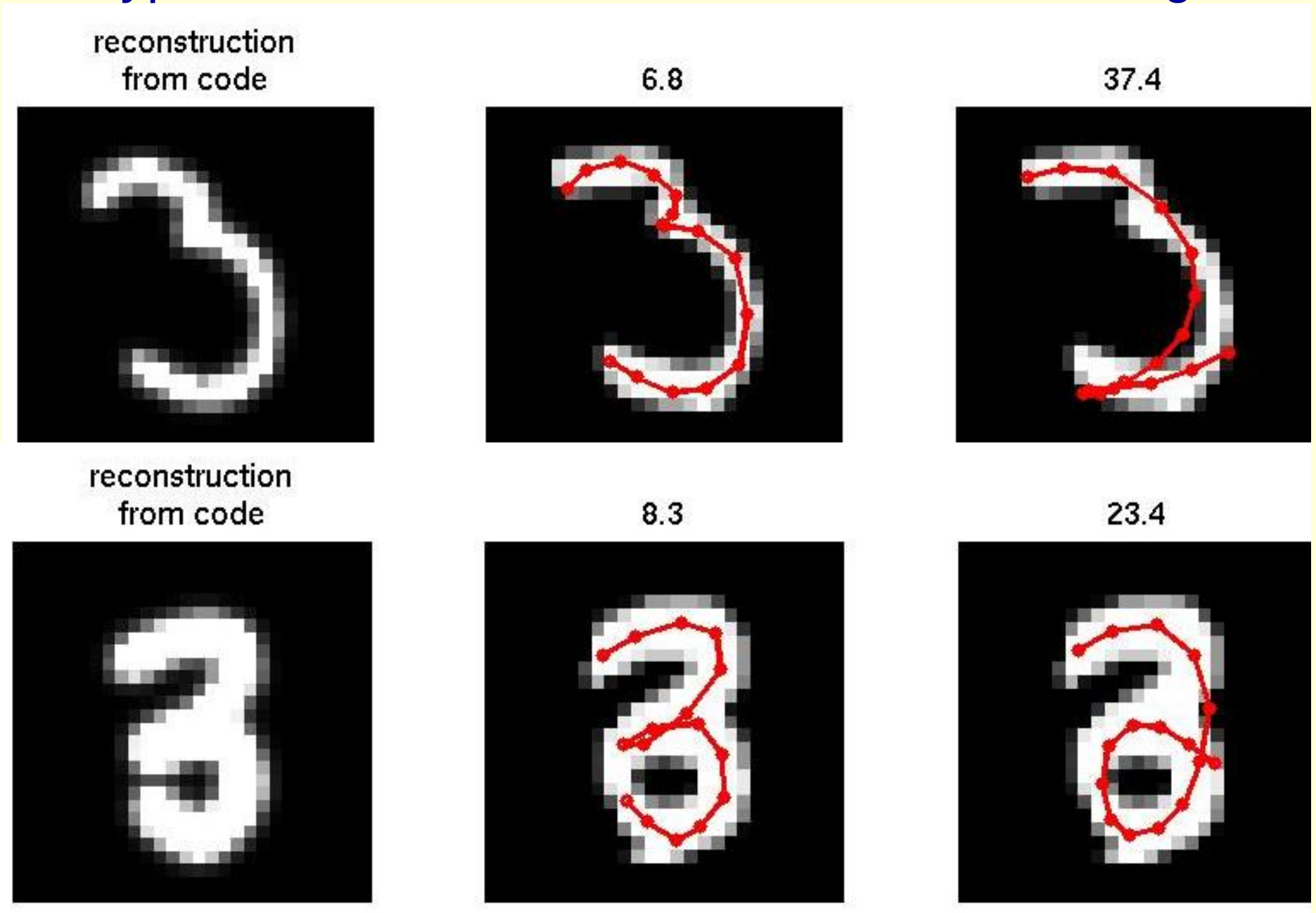
Typical fits to the test data at the end of training

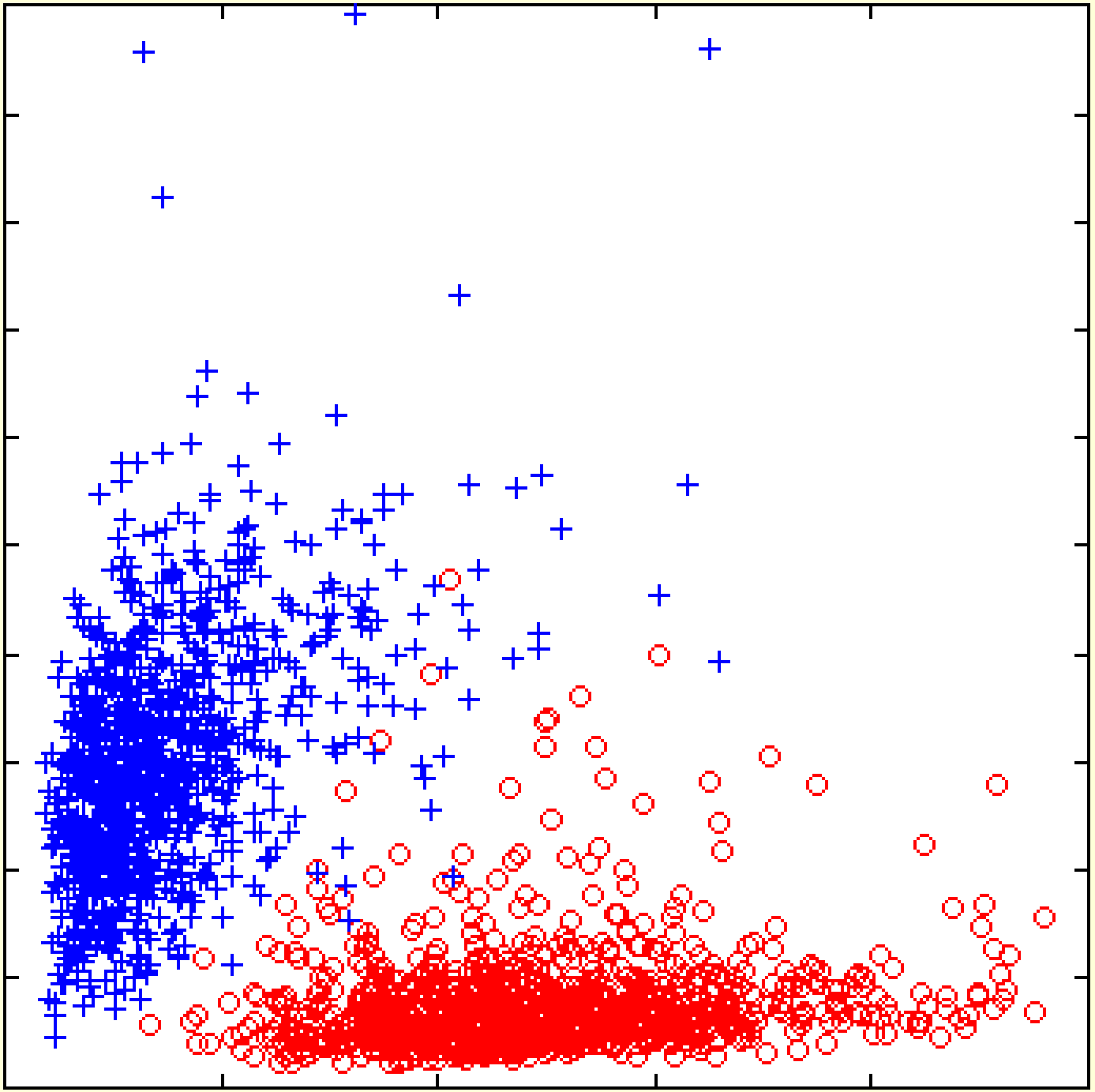


Typical fits to the test data at the end of training

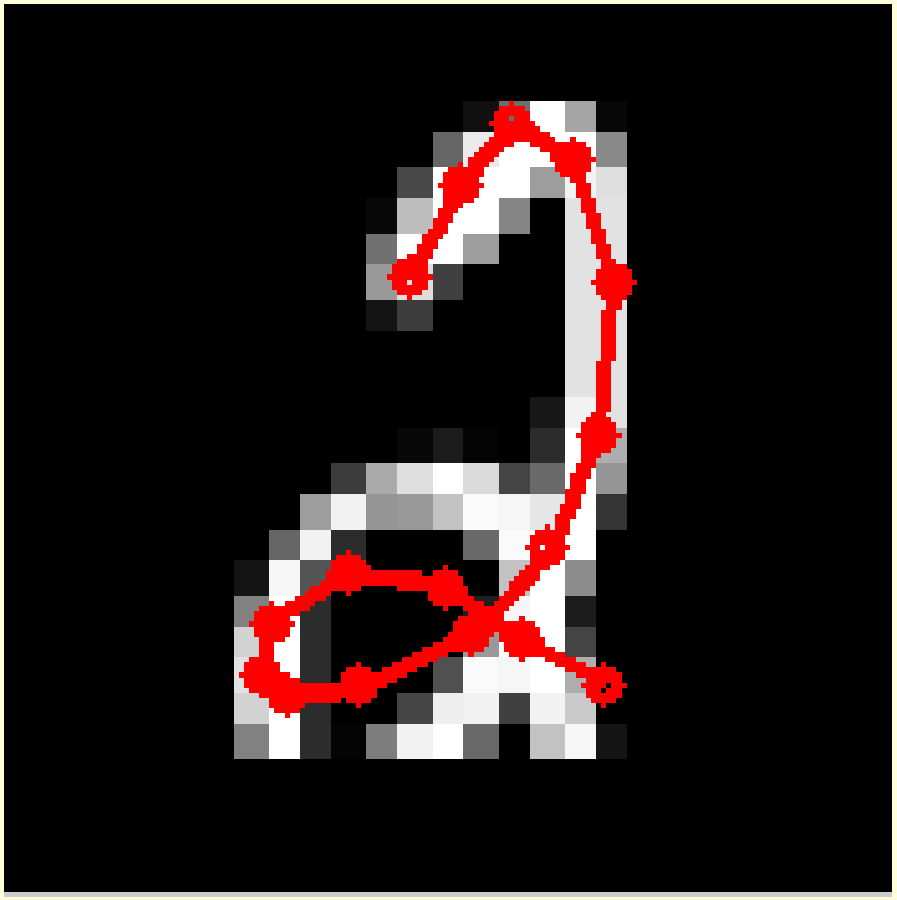


Typical fits to the test data at the end of training

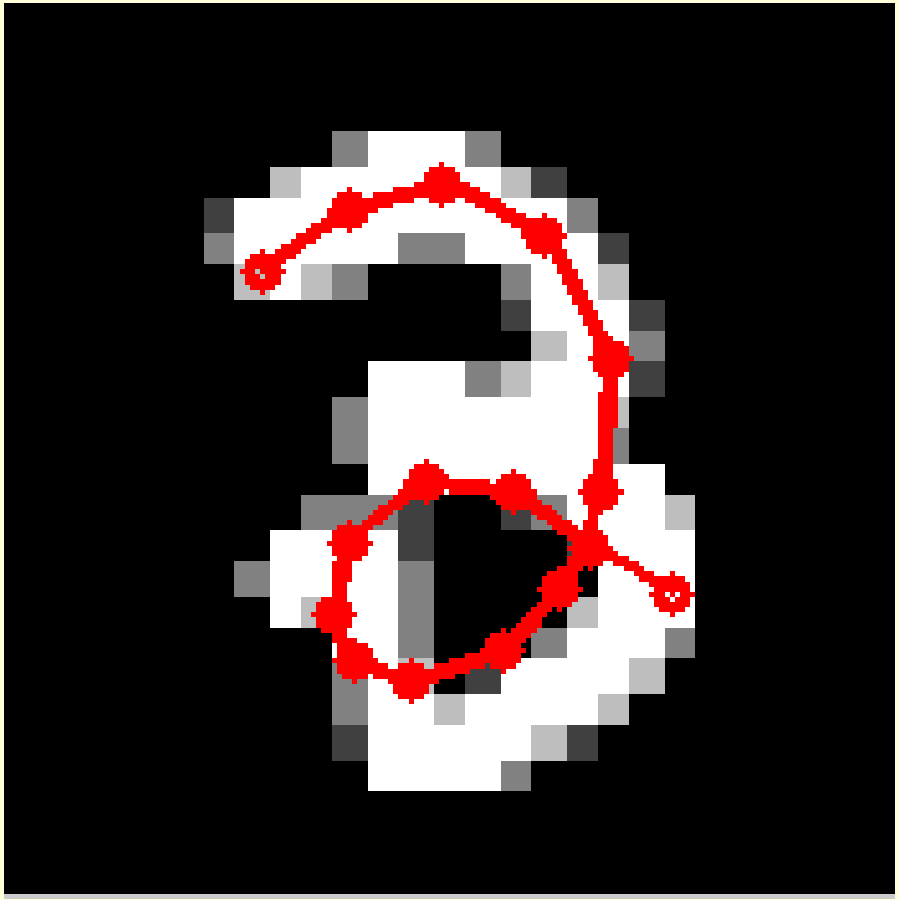


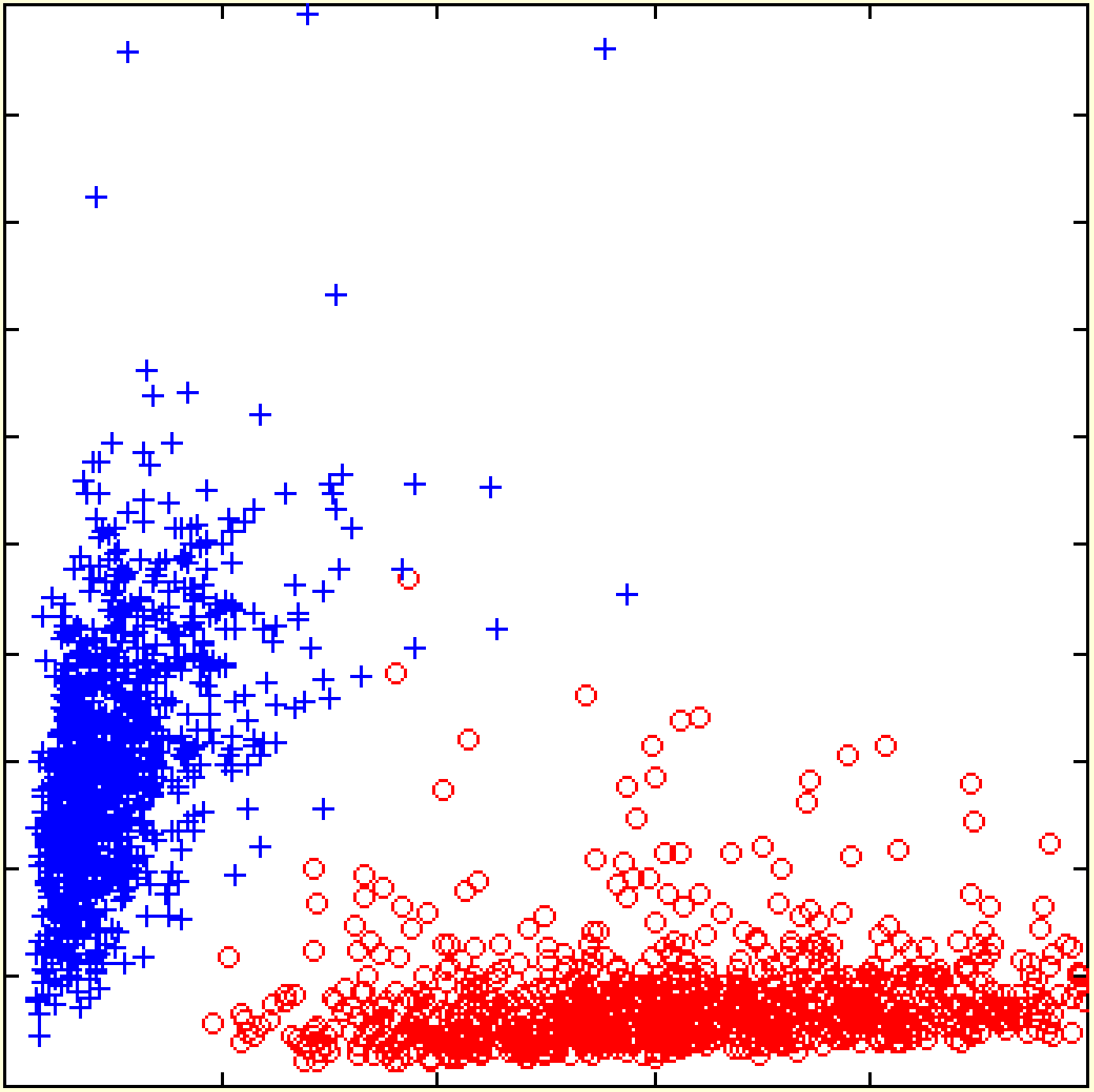


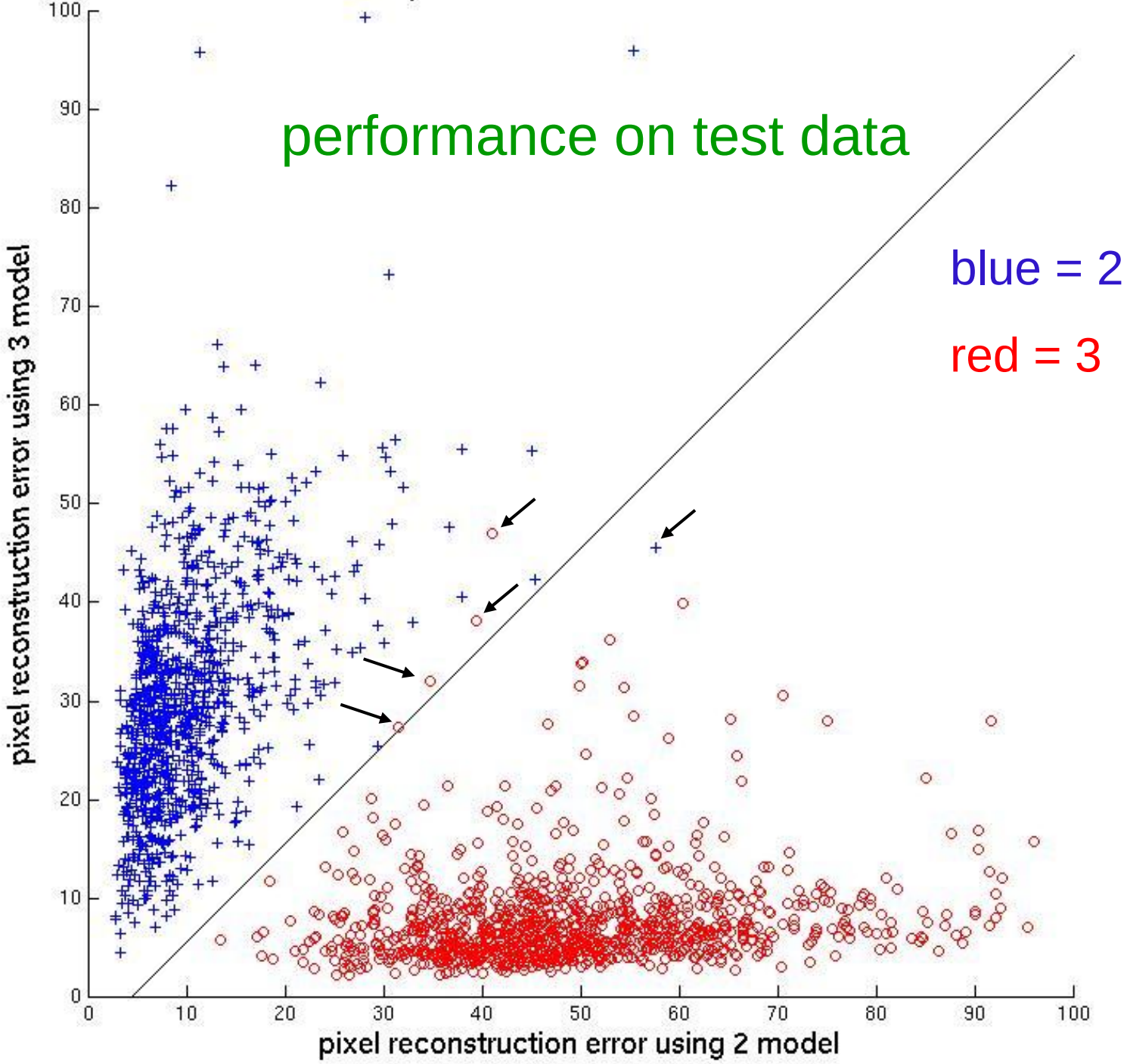
The 2 model gets better at fitting 2's



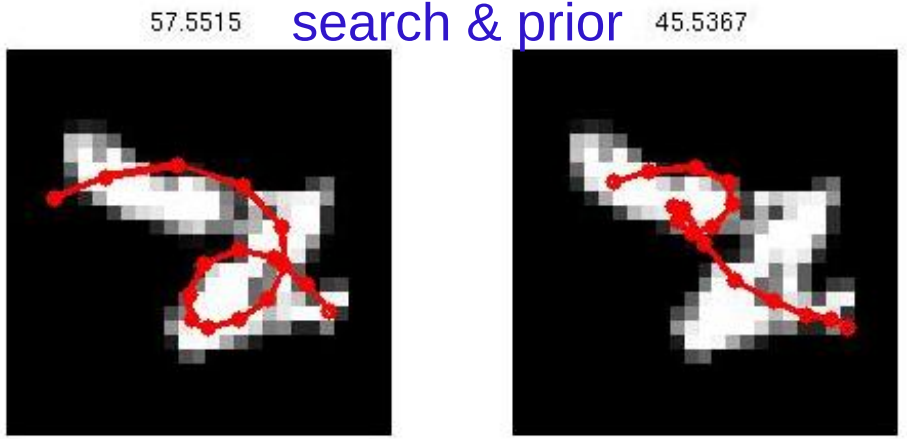
But it also gets better a fitting 3's

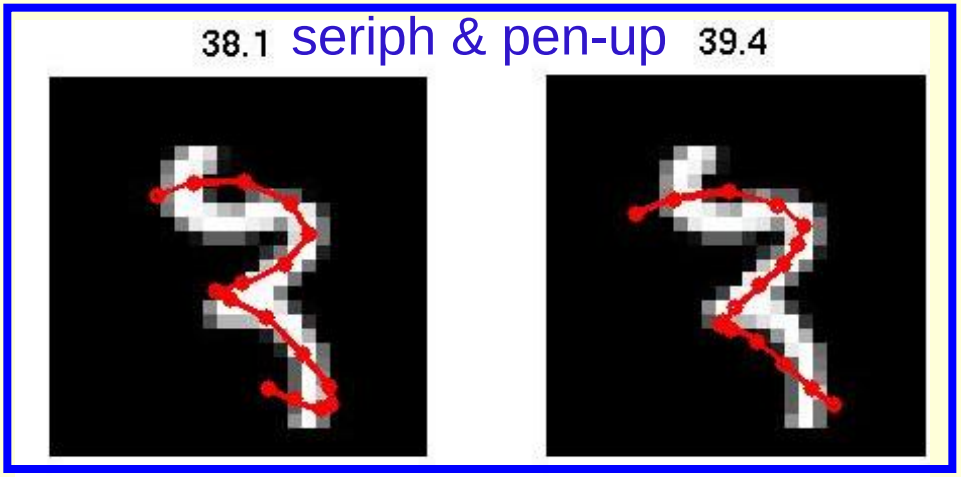


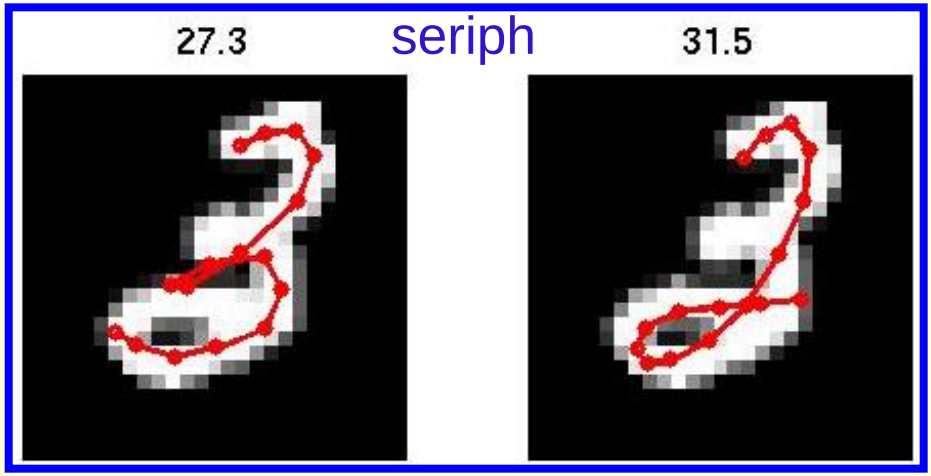


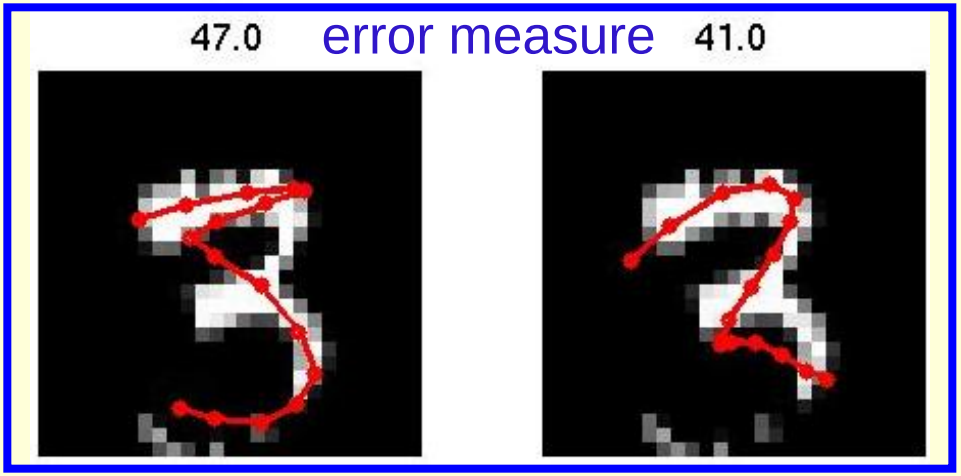

performance on test data
blue = 2
red = 3
pixel reconstruction error using 3 model
pixel reconstruction error using 2 model
0
10
20
30
40
50
60
70
80
90
100

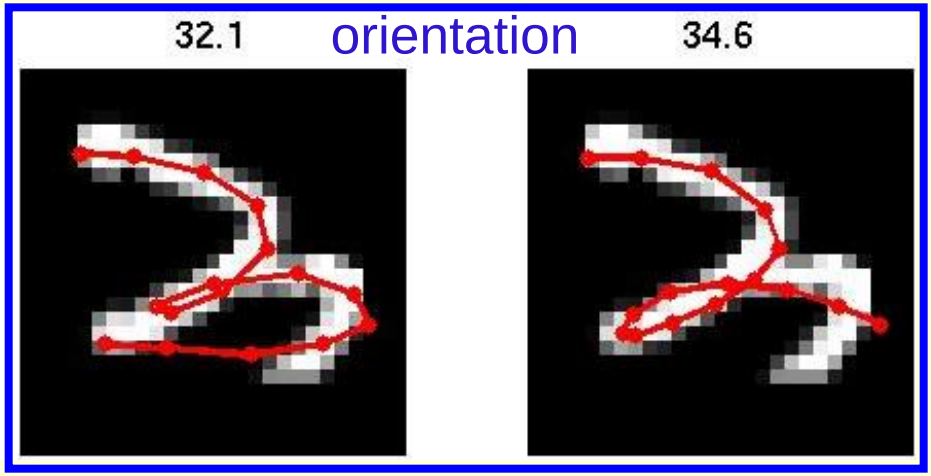
The 5 errors
(with excuses)









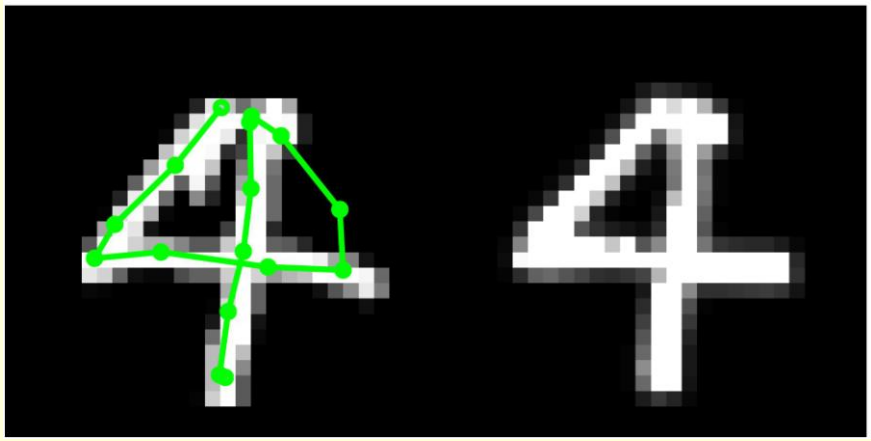


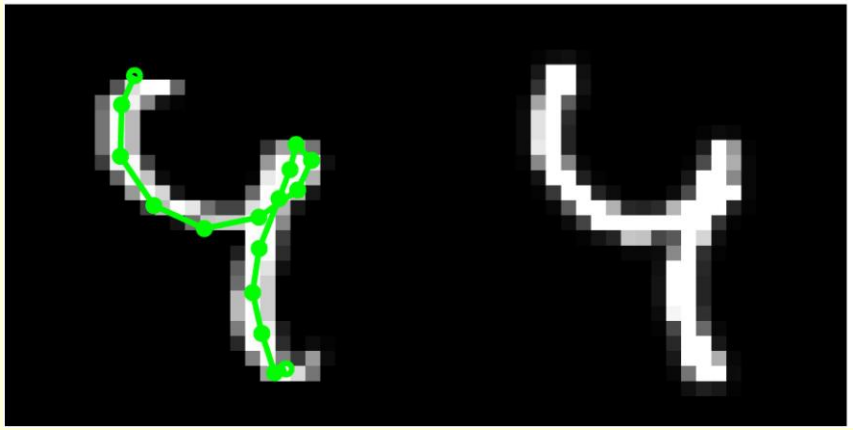
Improvement from local search

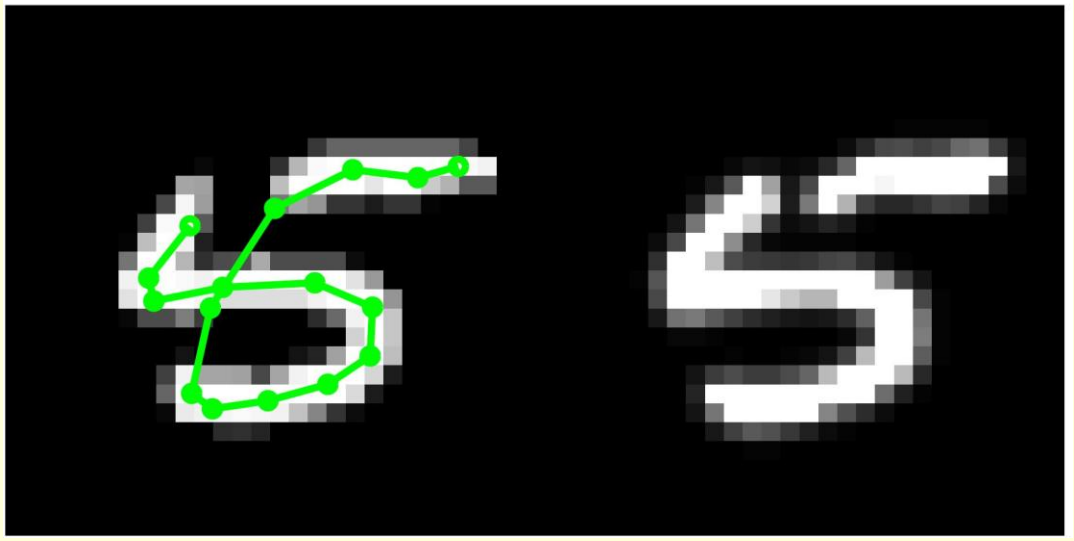
- Local search reduces the squared pixel error of the correct model by 20-50%
  - It depends on how well the networks are trained
- The squared pixel error of the wrong model is reduced by a much smaller fraction
  - It often increases because the pixel residuals are converted to motor program adjustments using a generative net that has not been trained in that part of the space.
- The classification error improves dramatically.

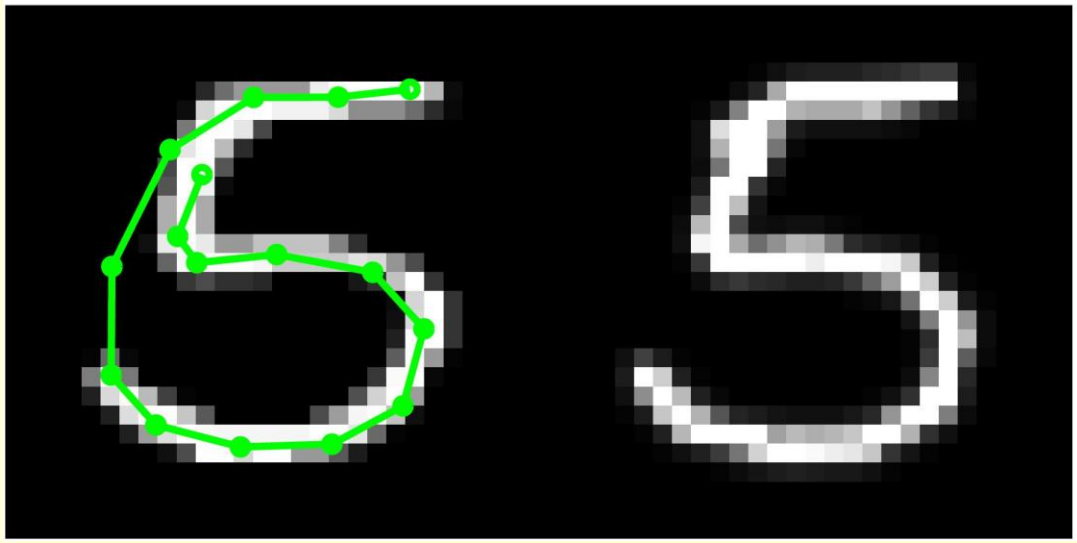
Digits that require multiple strokes

- For 4's and 5's we need to lift the pen off the paper in the middle of the trajectory. We do this by have some trajectory points that never produce ink.

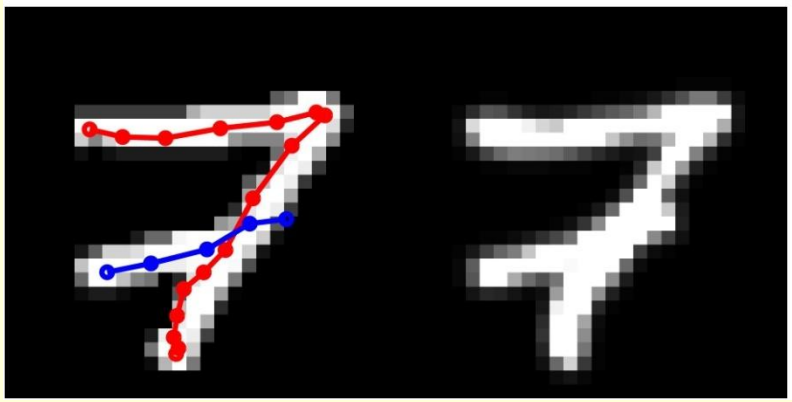


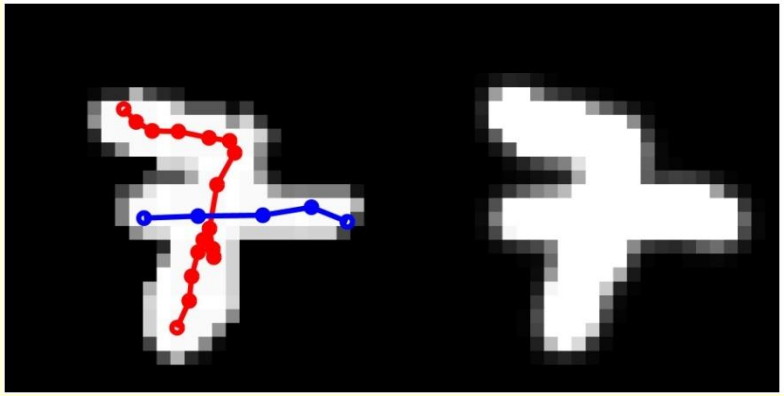


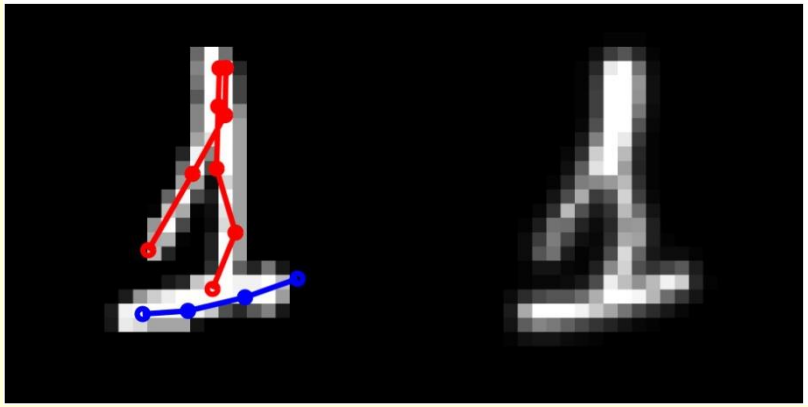


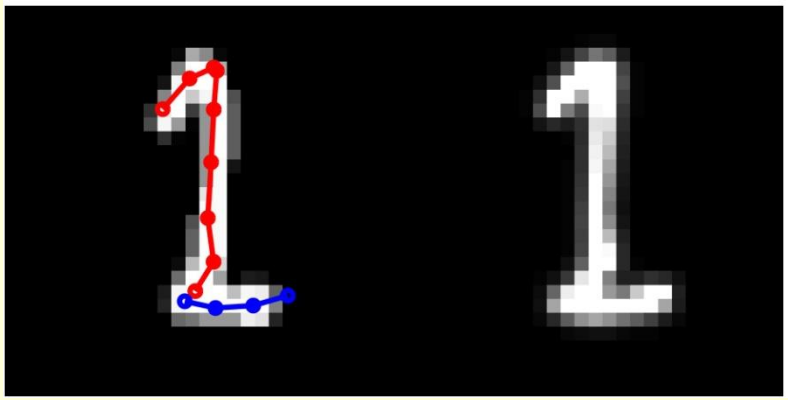


- For 7's and 1's we use an extra, optional stroke







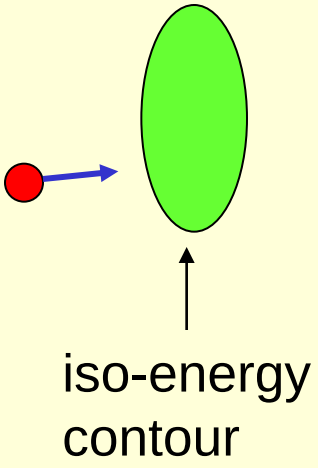


Performance of the final system

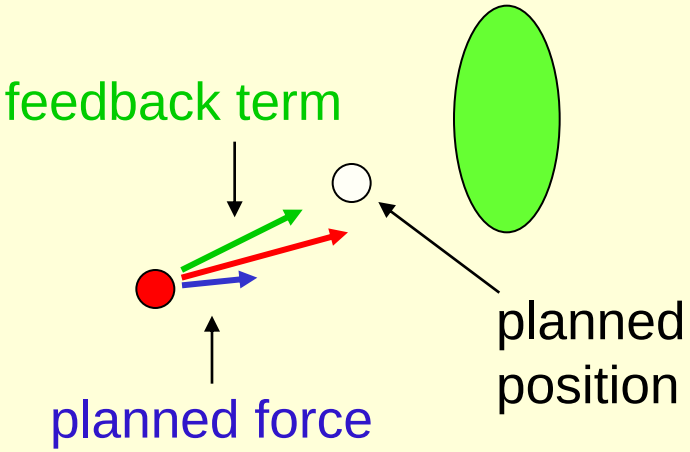
- On the MNIST test set we get 1.8% errors
  - Nearest neighbor 3.3%
  - Support Vector Machines 1.4%
  - Careful Backpropagation 1.5%

Why spring stiffnesses are a good language for motor control

- The four spring stiffnesses define a quadratic energy function. This function generates an acceleration. The acceleration depends on where the mass is.



- If the mass is not where we thought it was during planning, the force that is generated is the planned force plus a feedback term that pulls the mass back towards where it should have been



The original application

- Suppose we have a high quality speech synthesizer that produces speech from articulatory parameters.
  - These synthesizers already exist
- Can we solve the long-standing problem of recovering the state of the articulators from the speech wave?
  - This is the key to high-quality speech recognition

Another way to make use of motor programs

- Instead of using the motor program for recognition, we could just use noisy motor programs as a way of enhancing the dataset.
  - It's a much better distortion metric than anything we can do with operations on pixels
    - (e.g. it can change the topology).
  - Its trivial to apply affine transformations to the trajectories
    - Trajectories live in the right vector space.

Conclusion

- Elaborate priors are worth having if there is very little data.
- If there is quite a lot of data, the ability to fit a whole lot of parameters in a very flexible model is worth more than having a carefully hand-engineered and approximately correct prior.
  - It's a bad idea to have a tug-of-war with a steam engine.
- If you have a lot of computer power, use your prior knowledge to create more training data.