CPSC 340: Machine Learning and Data Mining

Even More Deep Learning

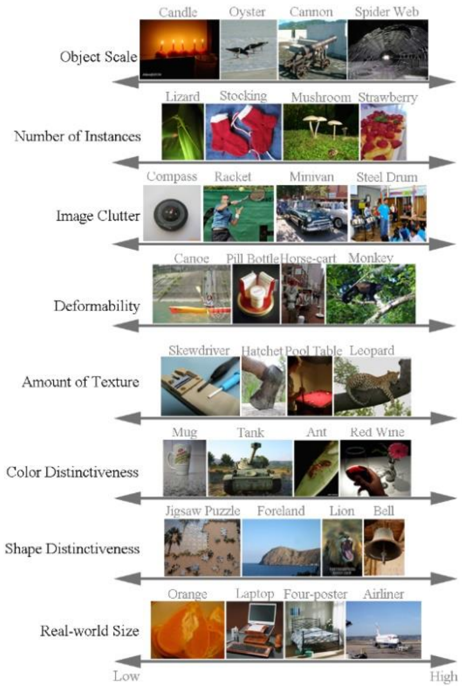
Fall 2018

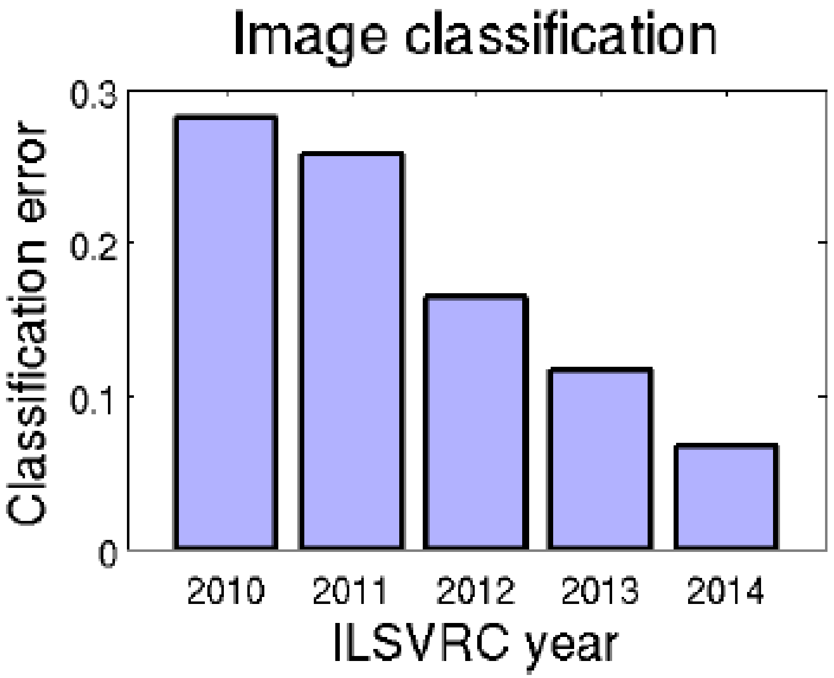
Last Lectures: Deep Learning

- We’ve been discussing neural network / deep learning models:

$$\hat{y}_i = v^T h(W^{(m)} h(W^{(m-1)} h(\cdots W^{(2)} h(W^{(1)} x_i))\cdots))$$

- We discussed unprecedented vision/speech performance.





https://arxiv.org/pdf/1409.0575v3.pdf

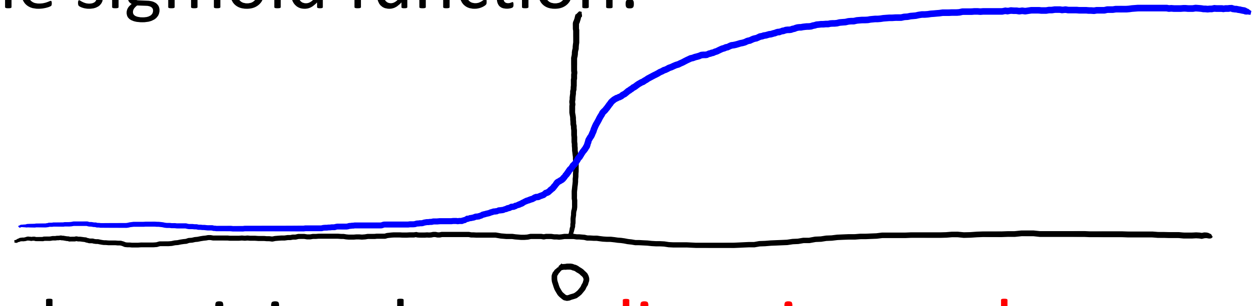
Setting the Step-Size

- Automatic method to set step size is Bottou trick:
  1. Grab a small set of training examples (maybe 5% of total).
  2. Do a binary search for a step size that works well on them.
  3. Use this step size for a long time (or slowly decrease it from there).

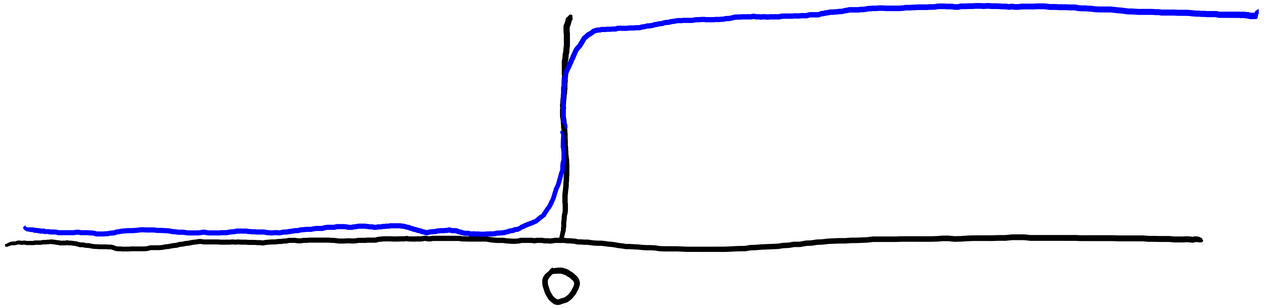
- Several recent methods using a step size for each variable:
  - AdaGrad, RMSprop, Adam (often work better "out of the box").
  - Seem to be losing popularity to stochastic gradient (often with momentum).
    - Often yields lower test error but this requires more tuning of step-size.
- Batch size (number of random examples) also influences results.
  - Bigger batch sizes often give faster convergence but maybe to worse solutions.
- Another recent trick is batch normalization:
  - Try to "standardize" the hidden units within the random samples as we go.
  - Held as example of deep learning "alchemy".
    - Sounds science-ey and often works but little theoretical justification/understanding.

Vanishing Gradient Problem

- Consider the sigmoid function:



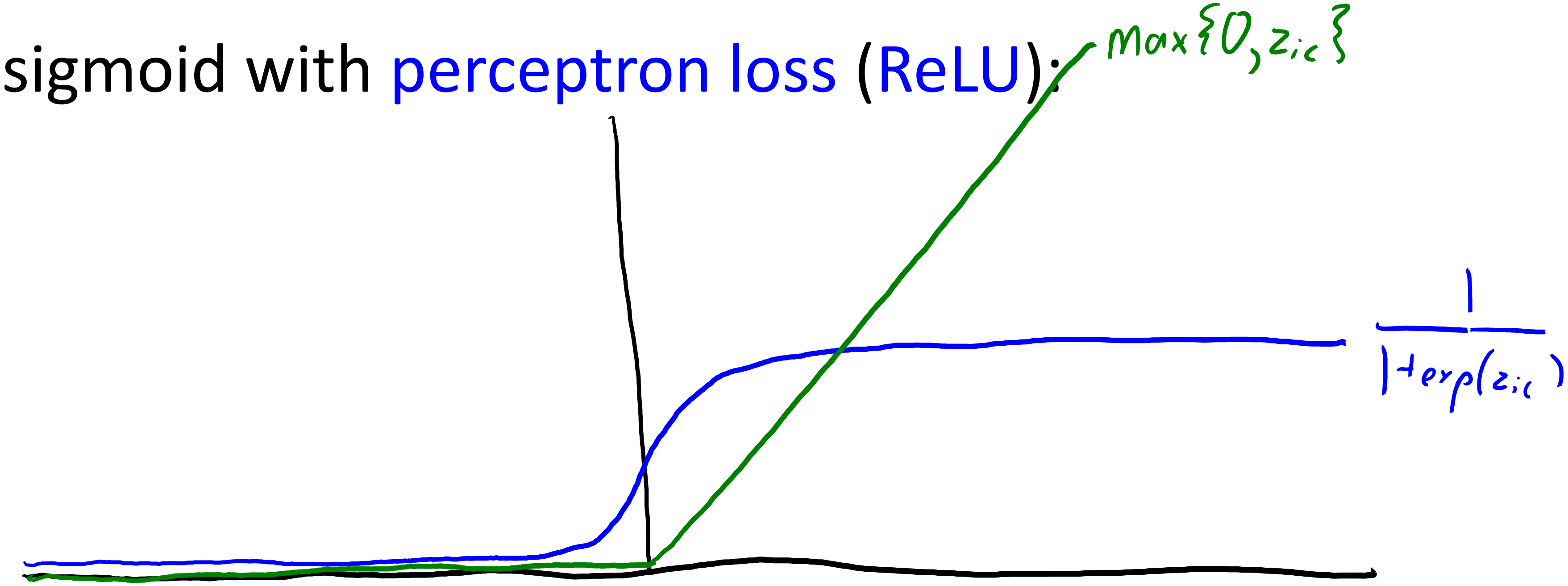
- Away from the origin, the gradient is nearly zero.
- The problem gets worse when you take the sigmoid of a sigmoid:



- In deep networks, many gradients can be nearly zero everywhere.

Rectified Linear Units (ReLU)

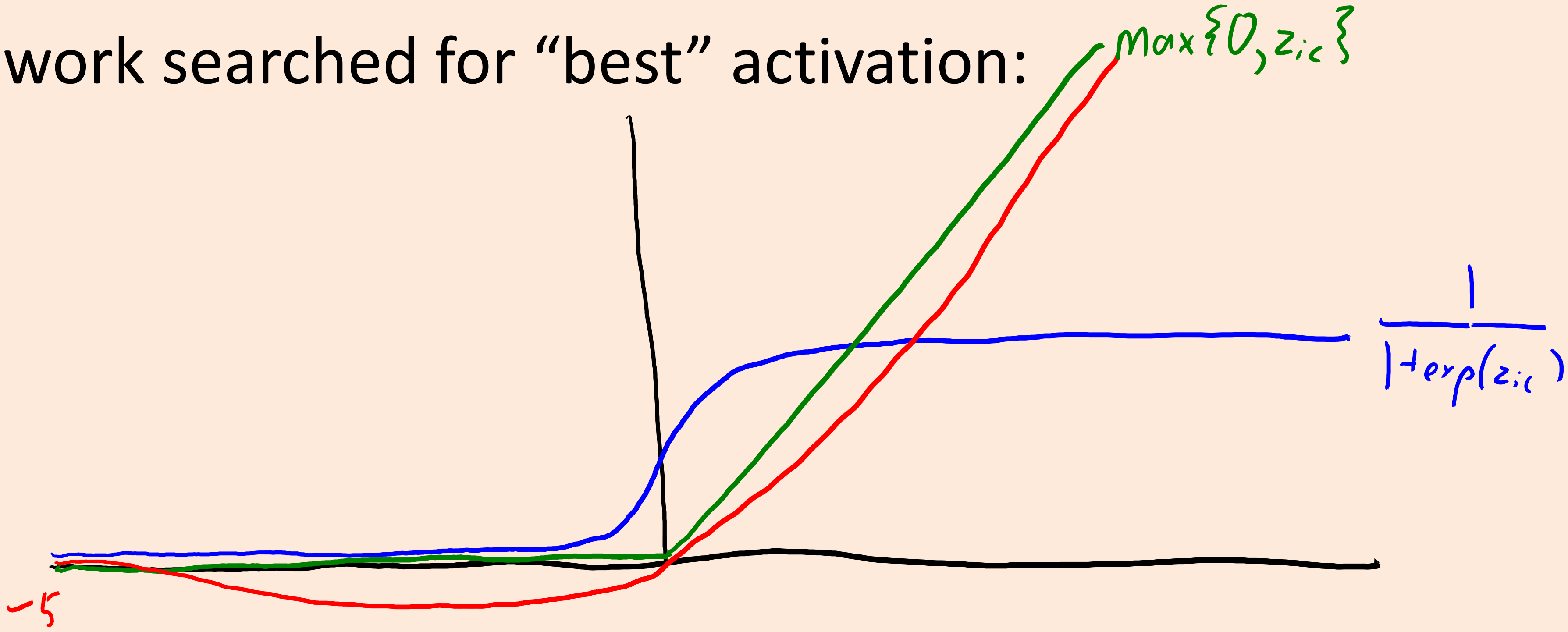
- Replace sigmoid with perceptron loss (ReLU):



- Just sets negative values z_{ic} to zero.
  - Fixes vanishing gradient problem.
  - Gives sparser activations.
  - Not really simulating binary signal, but could be simulating “rate coding”.

"Swish" Activation

- Recent work searched for "best" activation:

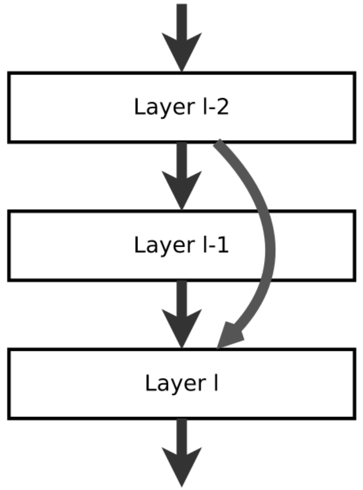


- Found that $z_{ic}/(1+\exp(-z_{ic}))$ worked best ("swish" function).
  - A bit weird because it allows negative values and is non-monotonic.
  - But basically the same as ReLU when not close to 0.

"Residual" Networks (ResNets)

- Suppose we fit a deep network to a linearly-separable dataset.
  - "All we need to do is look at the original features to solve the problem".
  - So with 'm' layers, the network needs to transform the features 'm' times.

- Situations like this have led to residual networks.
  - You can take previous (non-transformed) layer as input to current layer.
    - Also called "skip connections" or "highway networks".
  - Makes learning easier: "don't need to transform the input".
    - Non-linear part just "adds" non-linear information to a linear model.
  - This was a key idea behind first methods that used 100+ layers.
    - Evidence that biological networks have skip connections like this.



https://en.wikipedia.org/wiki/Residual_neural_network

DenseNet

- More recent variation is "DenseNets":
  - Each layer can see all the values from many previous layers.
  - Gets rid of vanishing gradients.

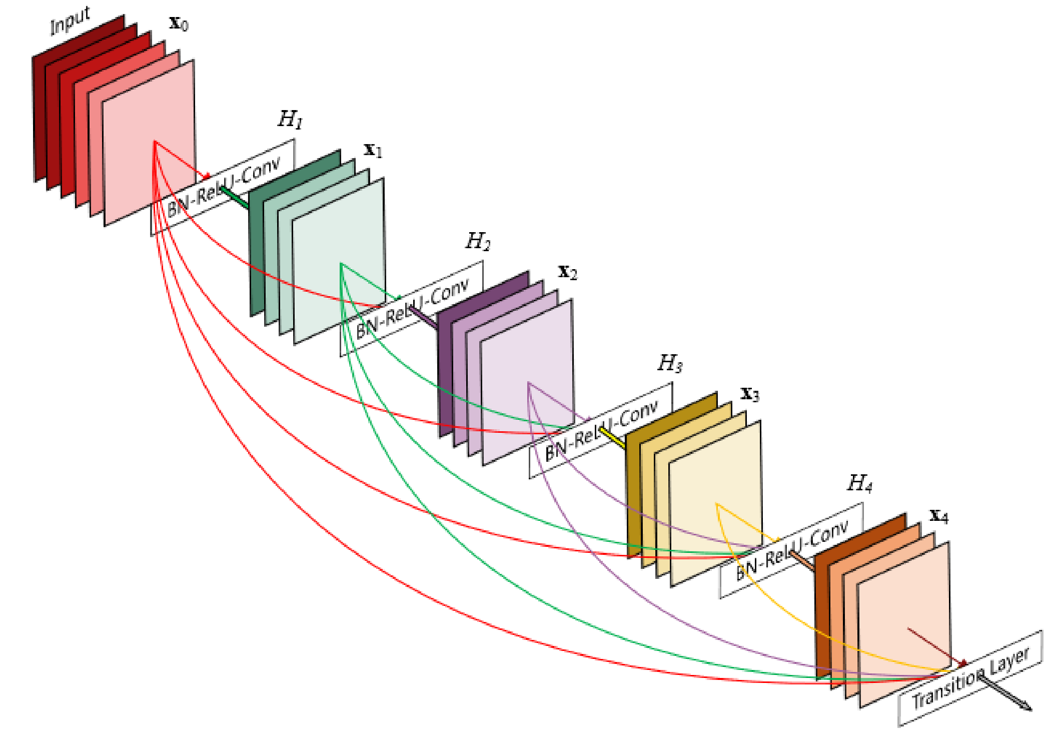


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

https://arxiv.org/pdf/1512.03385v1.pdf

Deep Learning and the Fundamental Trade-Off

- Neural networks are subject to the fundamental trade-off:
  - As we increase the depth, training error decreases.
  - As we increase the depth, training error no longer approximates test error.

- We want deep networks to model highly non-linear data.
  - But increasing the depth leads to overfitting.

- How could GoogLeNet use 22 layers?
  - Many forms of regularization and keeping model complexity under control.
  - Unlike linear models, typically use multiple types of regularization.

Standard Regularization

- We typically add our usual L2-regularizers:

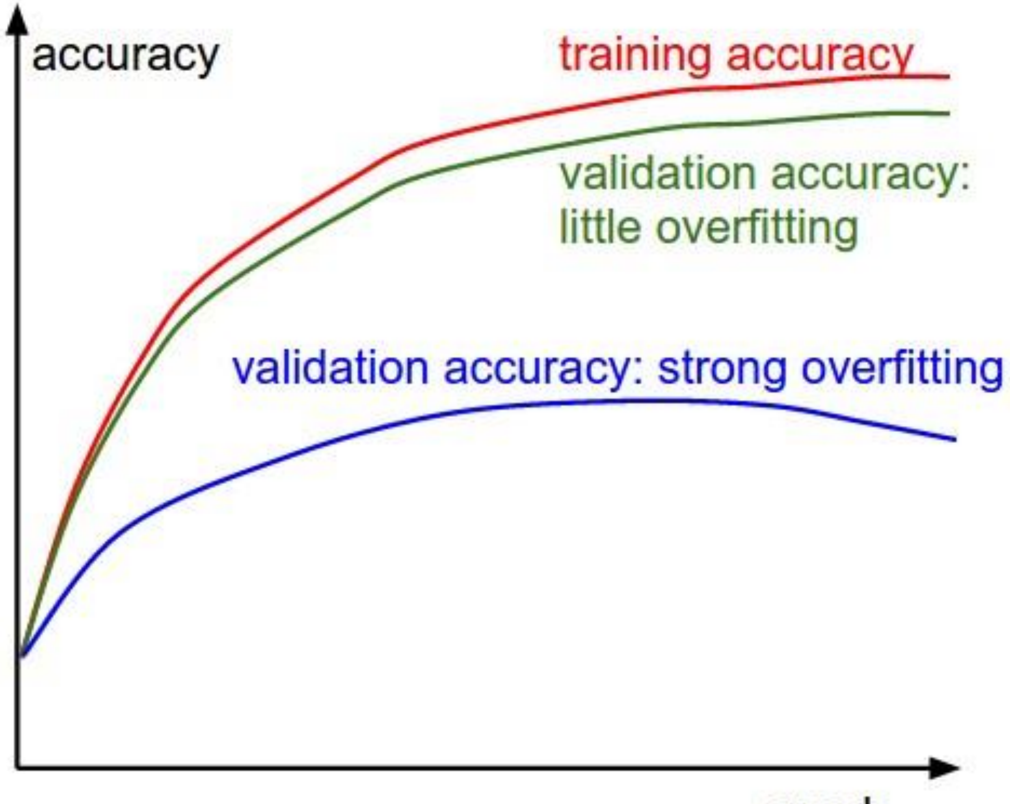
$$f(v, W^{(3)}, W^{(2)}, W^{(1)}) = \frac{1}{2}\sum_{i=1}^{n}(v^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) - y_i)^2 + \frac{\lambda_4}{2}\|v\|^2 + \frac{\lambda_3}{2}\|W^{(3)}\|_F^2 + \frac{\lambda_2}{2}\|W^{(2)}\|_F^2 + \frac{\lambda_1}{2}\|W^{(1)}\|_F^2$$

- L2-regularization is called "weight decay" in neural network papers.
  – Could also use L1-regularization.
- "Hyper-parameter" optimization:
  – Try to optimize validation error in terms of λ_1, λ_2, λ_3, λ_4.

- Recent result:
  – Adding a regularizer in this way creates bad local optima.

Early Stopping

- Second common type of regularization is "early stopping":
  - Monitor the validation error as we run stochastic gradient.
  - Stop the algorithm if validation error starts increasing.



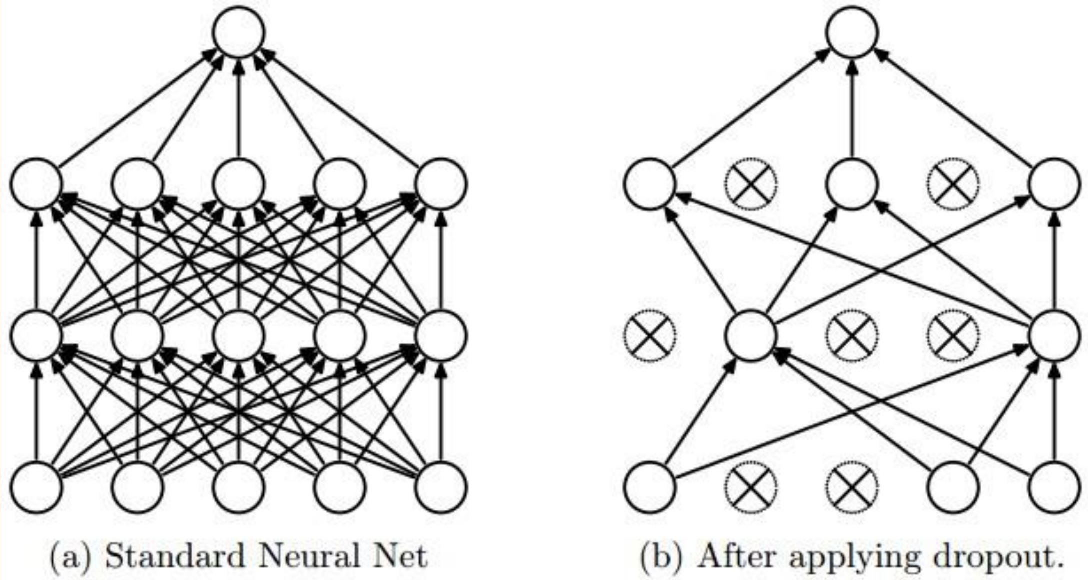
http://cs231n.github.io/neural-networks-3/

Unfortunately it might look more like

hopefully you don't stop here.

Dropout

- Dropout is a more recent form of regularization:
  – On each iteration, randomly set some x_i and z_i to zero (often use 50%).

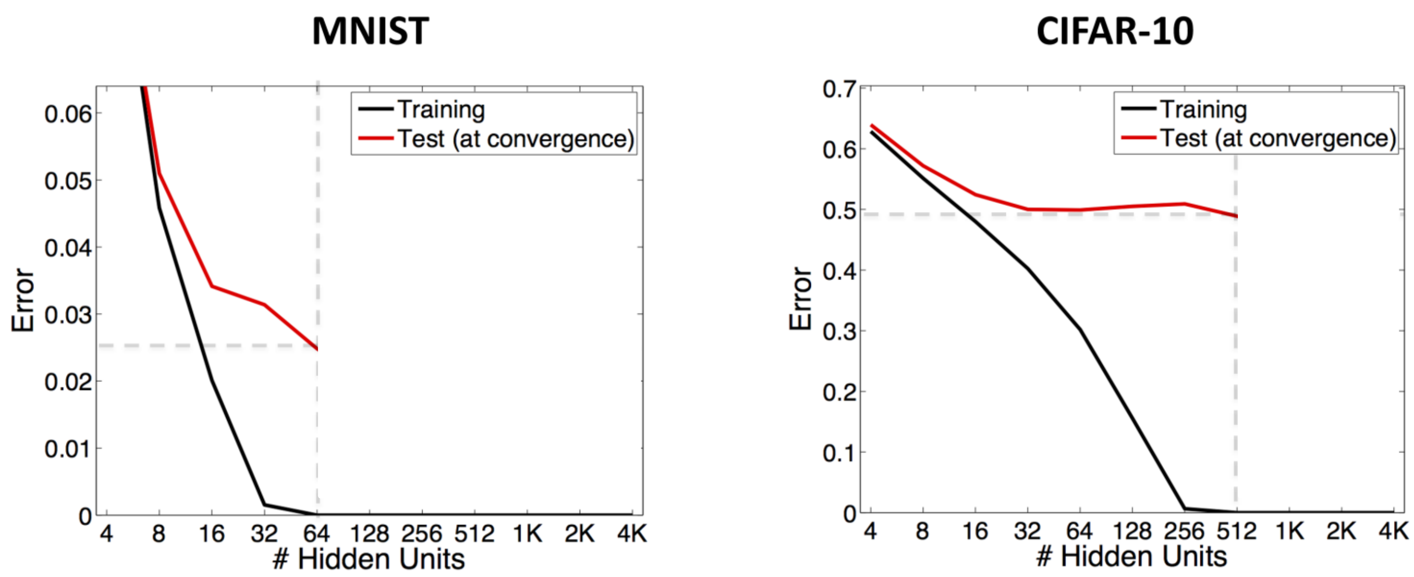


  – Encourages distributed representation rather than relying on specific z_i.
    - Alternately, you are adding invariance to missing inputs or latent factors.
  – After a lot of success, dropout may already be going out of fashion.

http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

“Hidden” Regularization in Neural Networks

- Fitting single-layer neural network with SGD and no regularization:



- Training goes to 0 with enough units: we’re finding a global min.

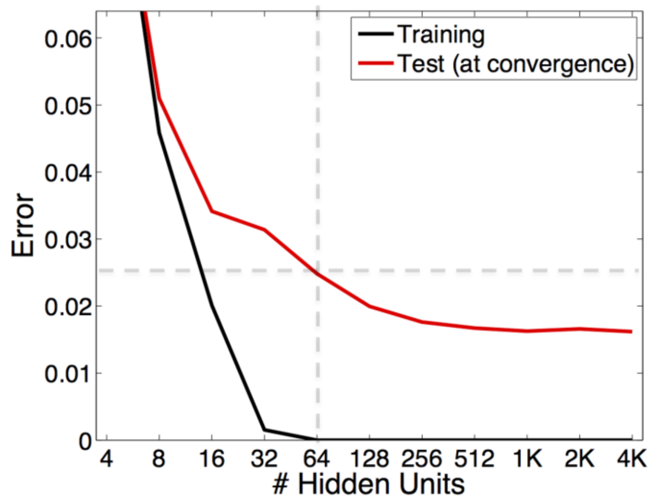
- What should happen to training and test error for larger #hidden?

https://www.neyshabur.net/papers/inductive_bias_poster.pdf

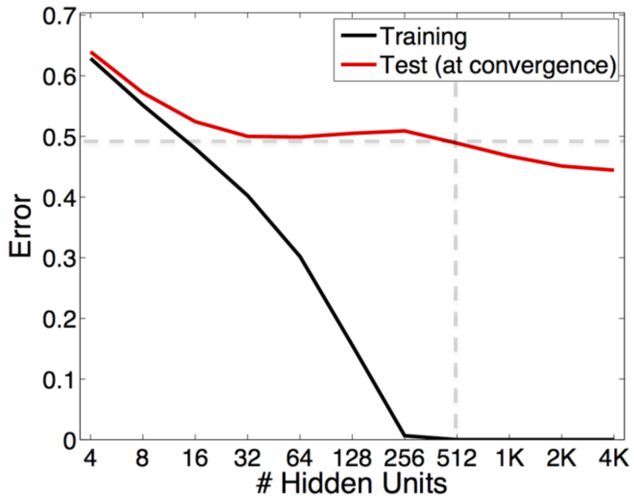
“Hidden” Regularization in Neural Networks

- Fitting single-layer neural network with SGD and no regularization:

MNIST



CIFAR-10



- Test error continues to go down!?! Where is fundamental trade-off??
- There exist global mins where large #hidden units have test accuracy 0.
  - But among the global minima, SGD is somehow converging to “good” ones.

https://www.neyshabur.net/papers/inductive_bias_poster.pdf

Implicit Regularization of SGD

- There is growing evidence that using SGD regularizes parameters.
- Beyond empirical evidence, we know this happens in simpler cases.
- Example:
  - Consider a least squares problem where there exists a 'w' where Xw=y.
    - Residuals are all zero, we fit the data exactly.
  - You run [stochastic] gradient descent starting from w=0.
  - Converges to solution w* of Xw=y that has the minimum L2-norm.
    - So using SGD is equivalent to L2-regularization here, but regularization is "implicit".

Implicit Regularization of SGD

- There is growing evidence that using SGD regularizes parameters.
- Beyond empirical evidence, we know this happens in simpler cases.
- Example:
  - Consider a logistic regression problem where data is linearly separable.
    - We can fit the data exactly.
  - You run [stochastic] gradient descent starting from w=0.
  - Converges to max-margin solution w* of the problem.
    - So using SGD is equivalent to encouraging large margin.

(pause)

Deep Learning “Tricks of the Trade”

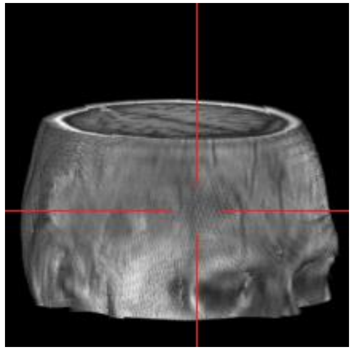
- We’ve discussed heuristics to make it work:
  - Parameter initialization and data transformations.
  - Setting the step size(s) in stochastic gradient.
  - Alternative non-linear functions like ReLU.
  - Different forms of regularization:
    - L2-regularization, early stopping, dropout, implicit regularization from SGD.

- These are often still not enough to get deep models working.

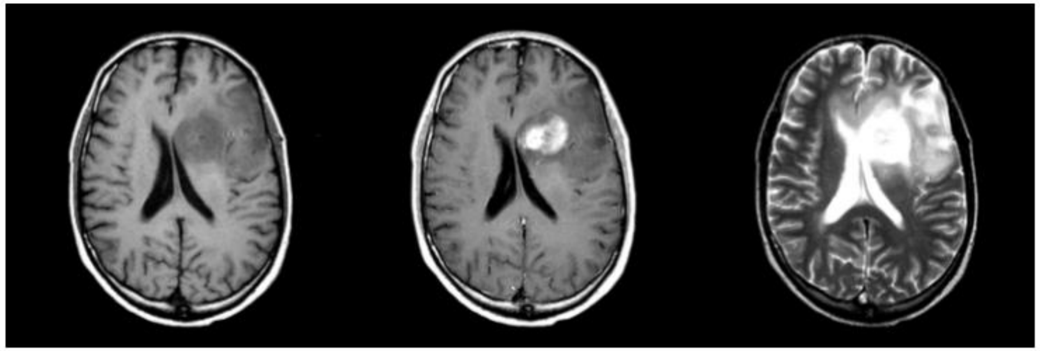
- Deep computer vision models are all convolutional neural networks:
  - The $W^{(m)}$ are very sparse and have repeated parameters (“tied weights”).
  - Drastically reduces number of parameters (speeds training, reduces overfitting).

Motivation: Automatic Brain Tumor Segmentation

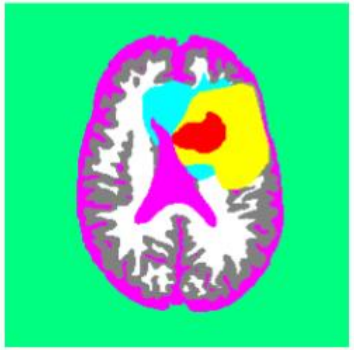
- Task: segmentation tumors and normal tissue in multi-modal MRI data.



Input:



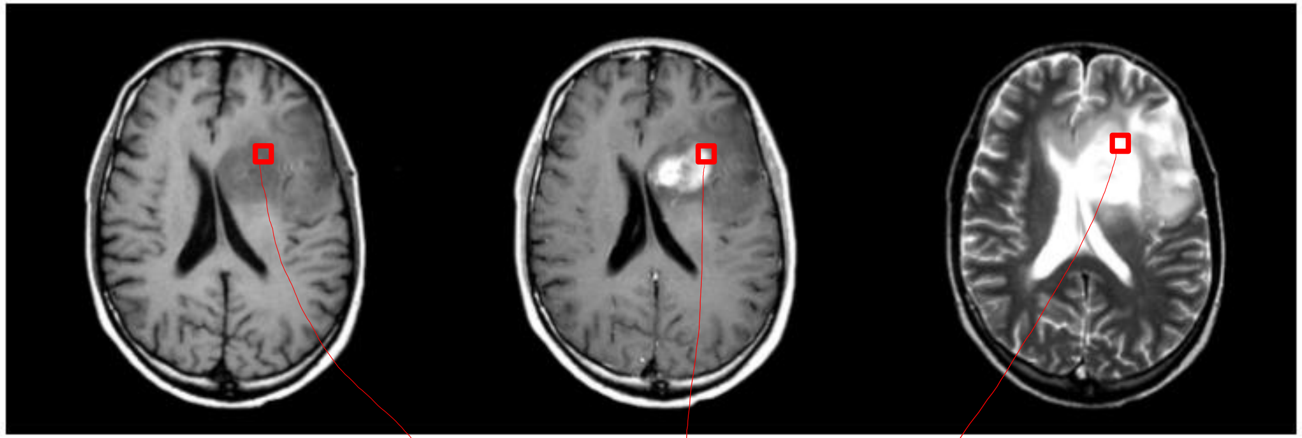
Output:



- Applications:
  - Radiation therapy target planning, quantifying treatment responses.
  - Mining growth patterns, image-guided surgery.
- Challenges:
  - Variety of tumor appearances, similarity to normal tissue.
  - “You are never going to solve this problem.”

Naïve Voxel-Level Classifier

- We could treat classifying a voxel as supervised learning:

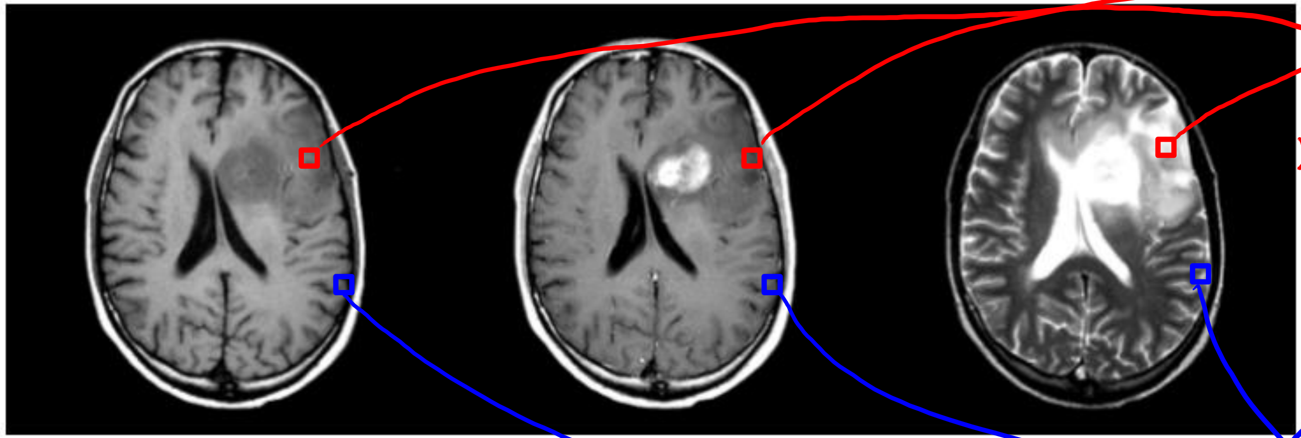


$x_i = (98, 187, 246)$ $\qquad$ $y_i = \text{"tumour"}$

- We can formulate predicting y_i given x_i as supervised learning.
- But it doesn't work at all with these features.

Need to Summarize Local Context

- The individual voxel values are almost meaningless:
  - This x_i could lead to different y_i.



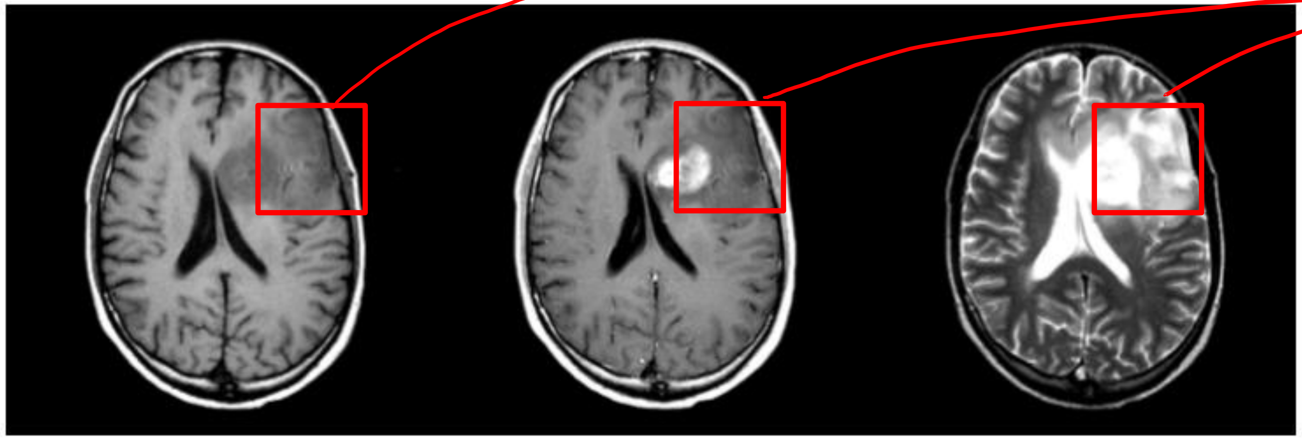
$x_i = (87, 54, 173)$ y_i = "tumour"

$x_i = (87, 54, 173)$ y_i = "normal"

- Intensities not standardized.
- Non-trivial overlap in signal for different tissue types.
- "Partial volume" effects at boundaries of tissue types.

Need to Summarize Local Context

- We need to represent the spatial “context” of the voxel.

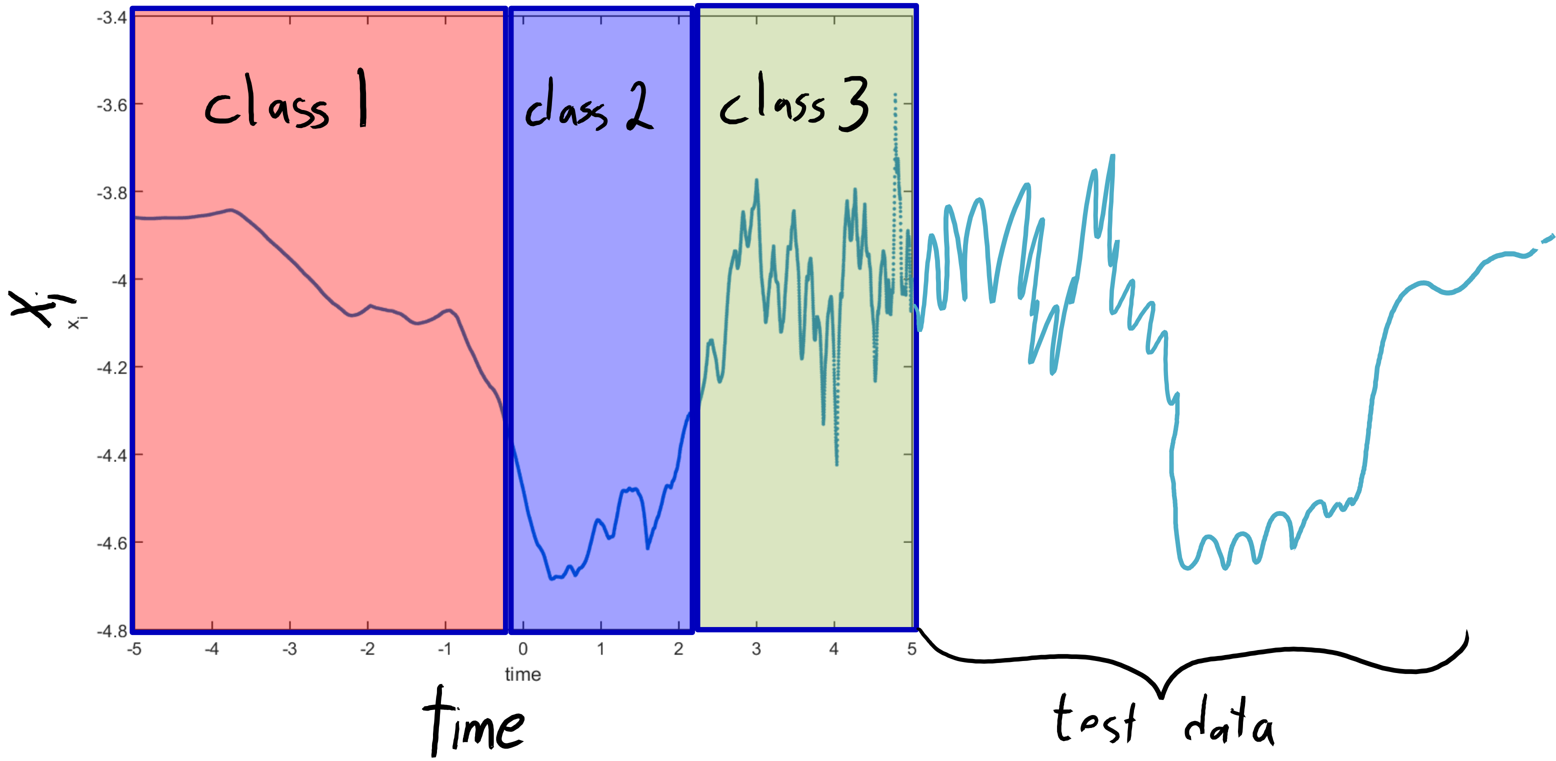


$x_i = (\ldots , \ldots , \ldots)$

  - Include all the values of neighbouring voxels?
    - Variation on coupon collection problem: requires lots of data to find patterns.
  - Measure neighbourhood summary statistics (mean, variance, histogram)?
    - Variation on bag of words problem: loses spatial information present in voxels.
  - Standard approach uses convolutions to represent neighbourhood.

Representing Neighbourhoods with Convolutions

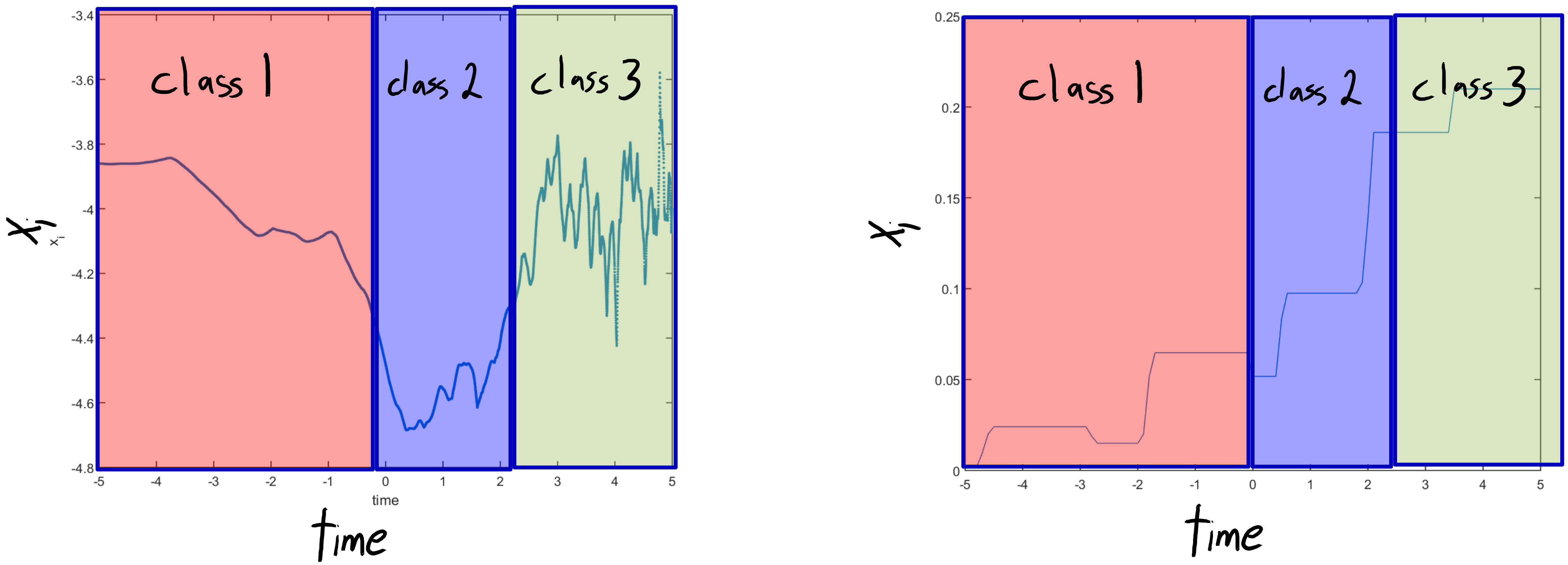
- Consider a 1D dataset:
  – Want to classify each time into y_i in {1,2,3}.
  – Example: speech data.



- Easy to distinguish class 2 from the other classes (x_i are smaller).
- Harder to distinguish between class 1 and class 3 (similar x_i range).
  – But convolutions can represent that class 3 is in "spiky" region.

Representing Neighbourhoods with Convolutions

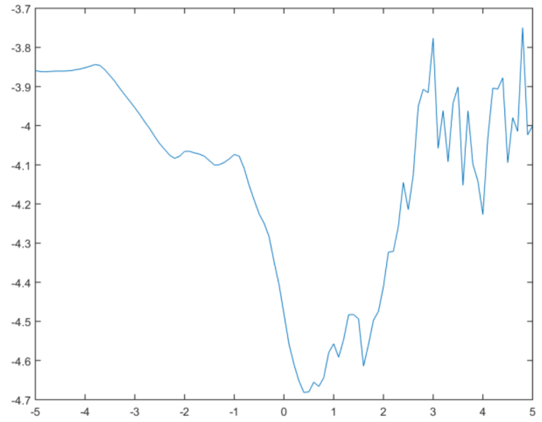
- Original features (left) and features from convolutions (right):



- Easy to distinguish the 3 classes with these 2 features.

1D Convolution Example

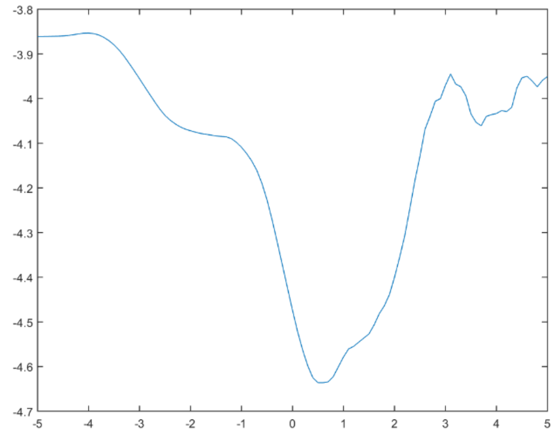
- Consider our original "signal":



- For each "time":
  - Compute dot-product of signal at surrounding times with a "filter".

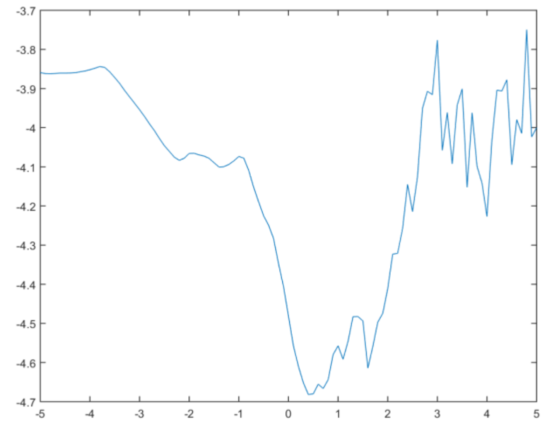
$$w = [\,1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9\,]$$

- This gives a new "signal":
  - Measures a property of "neighbourhood".
  - This particular filter shows a local "average" value.



1D Convolution Example

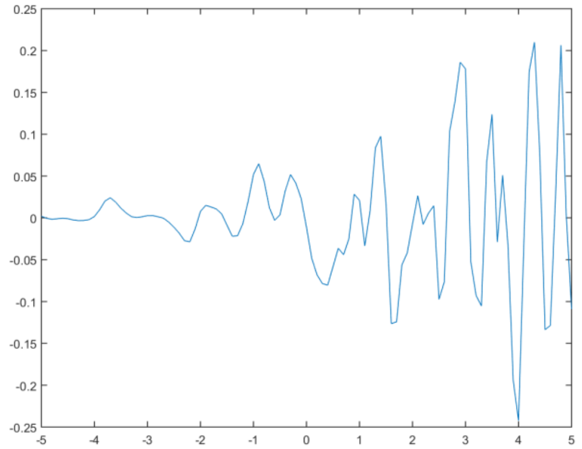
- Consider our original "signal":



- For each "time":
  - Compute dot-product of signal at surrounding times with a "filter".

$w = [-0.1416\ \ -0.1781\ \ -0.2746\ \ 0.1640\ \ 0.8607\ \ 0.1640\ \ -0.2746\ \ -0.1781\ \ -0.1416]$

- This gives a new "signal":
  - Measures a property of "neighbourhood".
  - This particular filter shows a local "how spiky" value.



1D Convolution (notation is specific to this lecture)

- 1D convolution input:
  - Signal 'x' which is a vector length 'n'.
    - Indexed by i=1,2,…,n.
  - Filter 'w' which is a vector of length '2m+1':
    - Indexed by i=-m,-m+1,…-2,0,1,2,…,m-1,m

$x = [0 \;\; 1 \;\; 1 \;\; 2 \;\; 3 \;\; 5 \;\; 8 \;\; 13]$

$$w = \begin{array}{ccccc} [0 & -1 & 2 & -1 & 0] \\ w_{-2} & w_{-1} & w_0 & w_1 & w_2 \end{array}$$

- Output is a vector of length 'n' with elements:

$$z_i = \sum_{j=-m}^{m} w_j x_{i+j}$$

  - You can think of this as centering w at position 'i', and taking a dot product of 'w' with that "part" x_i.

Summary

- ReLU and ResNets avoid “vanishing gradients”.
- Regularization is crucial to neural net performance:
  - L2-regularization, early stopping, dropout, implicit regularization of SGD.
- Convolutions are flexible class of signal/image transformations.

- Next time: convolutional neural networks.
  - The most important idea in computer vision?