Neural Networks and Deep Learning 3

-Introduction to Data Science-

Lecturer: Darren Homrighausen, PhD

Additional topics

- Dropout
- Activation function
- Types of layers

  (Fully connected, pooling, normalization, and convolutional)
- Representation learning

Dropout

Dropout

The key idea is to randomly drop hidden units (along with their connections) from the neural network during training

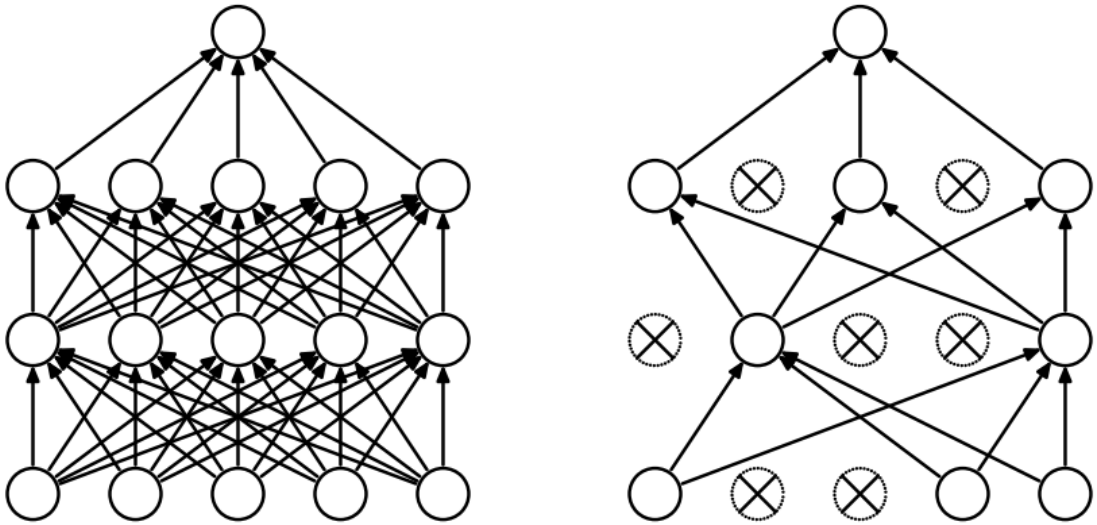
Effectively, this allows for the sampling from an exponential number of different "thinned" networks

When making predictions, we use the single "unthinned" network, but rescale all the weights

This approach has been shown to significantly reduce overfitting and gives major improvements over other regularization methods

(Srivastava et al. (2014))

Dropout



(a) Standard Neural Net

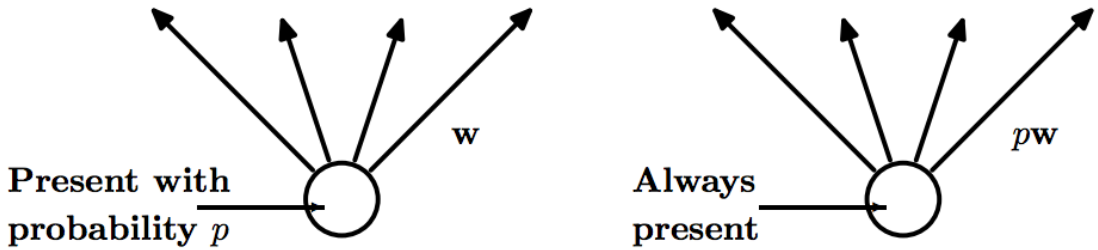
(b) After applying dropout.

Each unit is kept $i.i.d$ with probability p according to..

- $p = 0.5$ for upper layers
- $p \approx 1$ for first layer

There are exponentially many such networks $\rightarrow$ sampling from many "thinned" networks

Dropout



(a) At training time (b) At test time

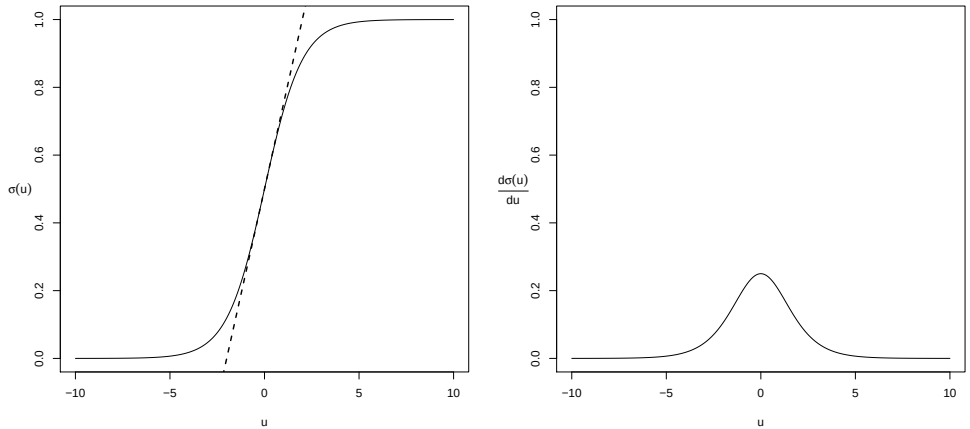
At test time, to mimic the averaging over all of the "thinned" networks, we...

- retain all the original units
- but rescale all the parameters w (known as weights) by the probability p

Activation function

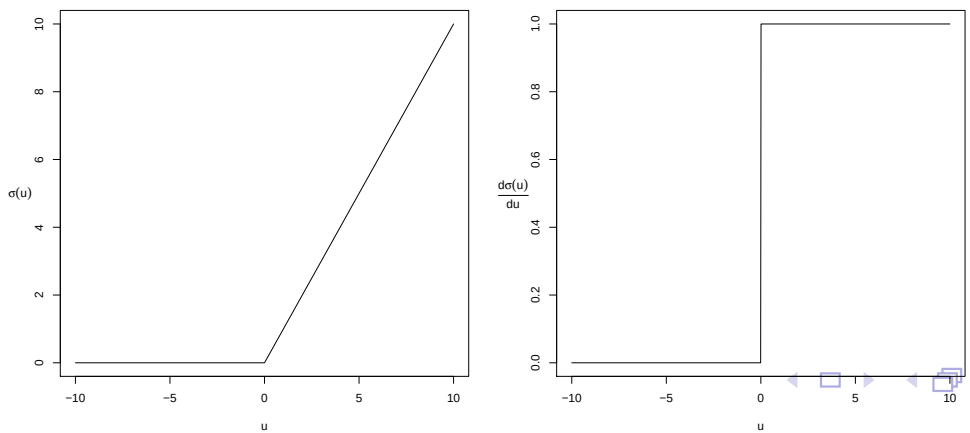
Activation function

The classical activation function is the sigmoid:



Modern neural networks use the rectilinear activation function

(Also known as the "Rectified Linear Unit" or (ReLU))



Rectilinear activation function

Relative to the sigmoid function, ReLU...

- tends to converge faster
- requires "lower level" computations

  (i.e. no exponentials/logarithms)
- Doesn't have the derivative problem in gradient descent

  (Derivative of $\sigma(u)$: $\sigma(u)(1-\sigma(u))$)
- can result in hidden units that are identically zero
- Scale invariant

  ($\max\{0, au\} = a\max\{0, u\}$)

There are variations on ReLU

- "leaky" ReLU

  (Adds a shallow slope to $u < 0$: $\sigma(u) = bu\mathbf{1}(u < 0) + u\mathbf{1}(u < 0)$, b very small)
- maxout ReLU

  (This one doesn't have the form $\sigma(\mathbb{Z}\alpha)$ but does generalize the various ReLU's)

Types of layers

Types of layers

We have generally discussed fully connected layers

Fully connected layers do not scale to large p

(For example: a $200 \times 200 \times 3$ image $\rightarrow 200 * 200 * 3 + 1 = 120,001$ parameters per hidden unit)

Moreover, we would want to have several such hidden units/layers

This much connectivity would quickly lead to overfitting and computational difficulties

Types of layers

If the features have a natural distance, then they can be locally smoothed

To be concrete, we will use image data as the running example/terminology

There are a few, commonly used smoothing steps

- Pooling
- Localization
- Normalization
- Convolution

Pooling

Perform a down-sampling step by taking a local {min, max, average, median, ℓ^2 norm}

This is can be interpreted as fixing the parameters α for a hidden unit at particular and not applying the activation function

(Though, technically, only the average can be expressed this way)

Localization

Assign hidden units to regions or channels in an image

Instead of allowing $\alpha \in \mathbb{R}^p$ to have all nonzero entries, only a patch of α_j's are nonzero

This dramatically reduces the number of parameters

Numerous choices still need to be made

- Boundaries: What do we do with patches that exit the image?
- Stride: How do we move the patches around?
- Patch size: What are the patch's dimensions?

Normalization

As previously noted, neural networks are very scale dependent

(e.g. learning rate)

Hence, it has been suggested that each layer should be renormalized to mean zero and variance 1

(Not just the original features)

Hence, we can scale each hidden unit over the training data:

$$Z_{ik} \leftarrow \frac{Z_{ik} - \overline{Z}_k}{sd(Z_k) + \epsilon}$$

(If using minibatch for the gradient updates, this is done over only the minibatch observations)

The ϵ is a fixed, tiny number that prevents division by zero

Normalization

In practice, this idea is usually pushed one step further: let the neural network learn the shift/scale as additional parameters

$$Z_{ik} \leftarrow \text{scale}_k \left(\frac{Z_{ik} - \overline{Z}_k}{sd(Z_k) + \epsilon} \right) + \text{shift}_k$$

(This allows the original parameterization as a special case (Ioffe, Szegedy (2015))

Convolutional

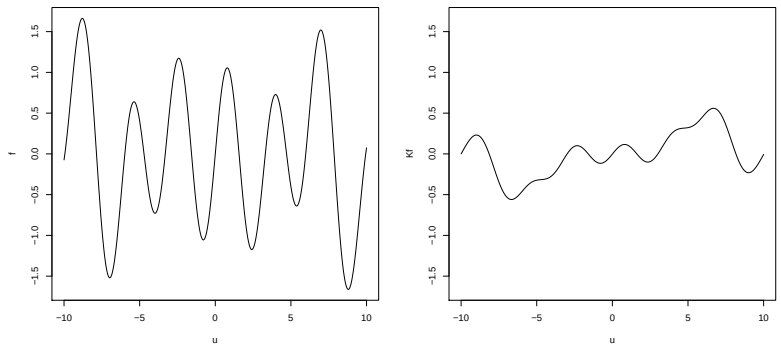
This looks a lot like the localization idea, with one change: the same patch is shared across the image

This even more dramatically reduces the number of parameters

The name comes from the forward pass now being a convolution

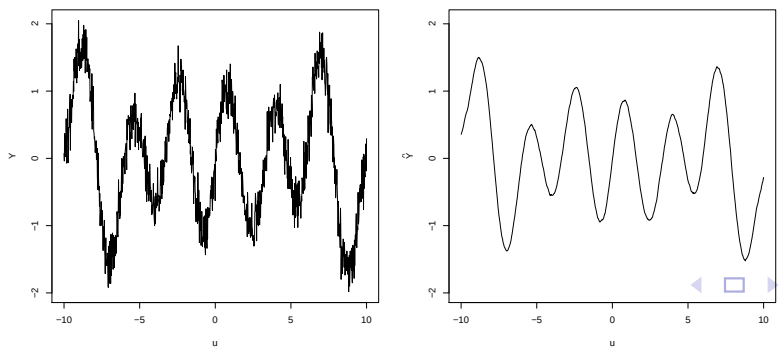
Convolutional

General convolution: $Kf(X) = (f * k)(X) = \int f(\tau)k(X-\tau)d\tau$

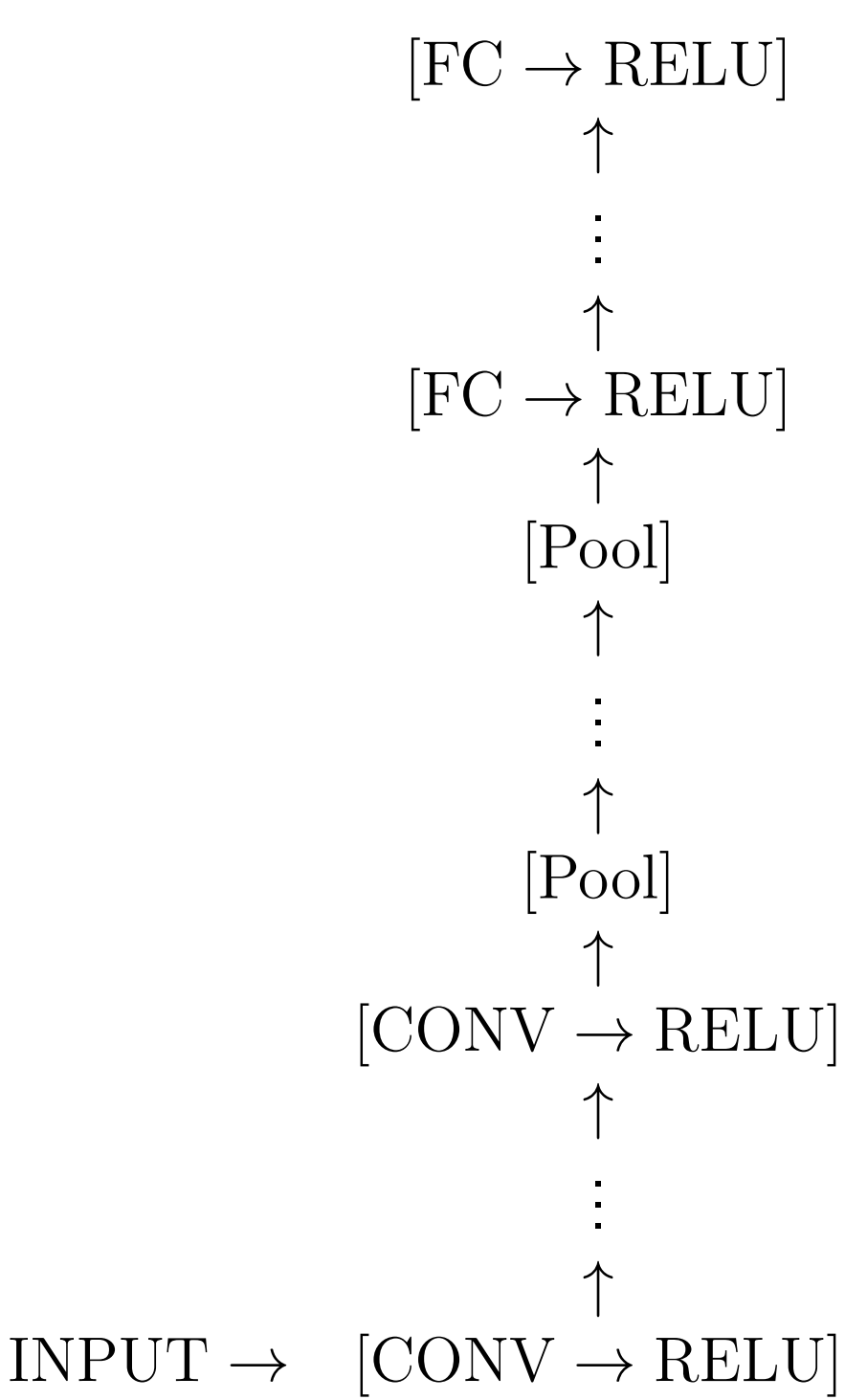


Example: $\hat{f}(X) = \frac{\sum_{i=1}^{n} Y_i k_h(X - X_i)}{\sum_{i=1}^{n} k_h(X - X_i)}$

(This is a kernel regression procedure (not to be confused with kernel ridge regression))



Putting it all together: A neural network architecture



Common Neural Network Architecture